CHALMERS

GÖTEBORGS UNIVERSITET

Maskininlärningstekniker för bildigenkänning

En jämförande analys och kombinering av PCA och neurala nätverk

Machine learning techniques for image recognition

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Enya Arvidsson

Hannes Björck

Joel Carlsson

Albin Edegran

Bekir Fazlija

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Maskininlärningstekniker för bildigenkänning

En jämförande analys och kombinering av PCA och neurala nätverk

Kandidatarbete i matematik inom civilingenjörsprogrammet Bioteknik vid Chalmers
Enya Arvidsson Albin Edegran

Kandidatarbete i matematik inom civilingenjörsprogrammet Maskinteknik vid Chalmers
Joel Carlsson Bekir Fazlija

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers
Hannes Björck

Handledare: Larisa Beilina

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Förord

Detta arbetet har resulterat i en kandidatuppsats som är skriven vid Institutionen för Matematiska vetenskaper på Chalmers tekniska högskola våren 2023. Gruppen vill rikta ett stort tack till vår handledare Larisa Beilina för vägledningen och den hjälp vi har fått med förståelsen av matematiken och skrivandet av rapporten. Vi i gruppen är tacksamma för den kunskap och lärdomar som detta arbetet gett oss. Slutligen vill vi skicka med en lärdom till er som läser denna rapport i syfte att bli inspirerade i ert egna kandidatarbete. Vår lärdom är att jobba mycket i grupp vid rapportskrivning, för det ökar kvalitén och produktiviteten, istället för att skriva alla delar individuellt. Samt att gemensam planering och organisation är en vital del av ett arbete.

En dokumentation av arbetet och varje individs bidrag och nedlagda tid ha skett genom en gemensam dagbok och individuella tidsloggar. Arbetet har delats upp jämnt mellan medlemmarna i gruppen, där Albin har haft ansvaret över programmeringen tillsammans med Joel och Bekir, Enya hållt majoriteten av kontakten med externa parter, samt Hannes och Bekir varit drivande i bearbetning av rapporten med hjälp av övriga gruppmedlemmar. Nedan presenteras skaparna av figurerna i rapporten, se Tabell 1. Efter det visas en bidragsrapport över de olika delarna i rapporten, där författaren är den som har skrivit första utkastet. Delarna har därefter omarbetats flertalet gånger av alla gruppmedlemmar, för att uppnå en sammanhängande och korrekt text, se Tabell 2.

Tabell 1: Skapare av figurerna i rapporten.

Avsnitt	***Figur***	***Skapad av***
2	*Figur 1*	*Bekir*
3.1.1	*Figur 2*	*Albin*
3.2.1	*Figur 3*	*Albin*
3.3	*Figur 4*	*Joel*
3.3.4	*Figur 5*	*Bekir, Hannes*
4.1	*Figur 6*	*Bekir*
4.2	*Figur 7*	*Joel*
4.2	*Figur 8*	*Joel*
4.3	*Figur 9*	*Hannes*
5	*Figur 10-18*	*Albin*

Tabell 2: Bidragsrapport.

Kapitel	***Rubrik***	***Författare***
	Förord	*Enya, Joel*
	Populärvetenskaplig presention	*Hannes*
	Sammandrag	*Enya, Joel*
	Abstract	*Hannes, Joel*
1	*Inledning*	*Bekir, Hannes*
2	*Problemformulering*	*Bekir, Hannes*
3	*Matematiska metoder*	*Bekir*
3.1	*Singulärvärdesuppdelning - SVD*	*Bekir*
3.1.1	*Bildkompremering med SVD*	*Albin*
3.2	*Principial komponent analys - PCA*	*Albin, Hannes*
3.2.1	*Bildigenkänning med PCA*	*Albin*
3.3	*Neurala nätverk - NN*	*Enya, Joel*
3.3.1	*Framåtpropagering*	*Joel*
3.3.2	*Förlustfunktion*	*Enya*
3.3.3	*Bakåtpropagering*	*Enya, Joel*
3.3.4	*Konvulutionellt neuralt nätverk - CNN*	*Enya, Joel*
4	*Implementation och Arkitektur*	*Joel*
4.1	*Dataset*	*Joel*
4.2	*Arkitekturen av neurala nätverk*	*Joel*
4.3	*Implementation*	*Joel*
5	*Numeriska tester*	*Albin*
5.1	*PCA*	*Albin*
5.2	*LNN*	*Albin*
5.3	*CNN*	*Albin*
5.4	*Illustration av tester*	*Albin*
6	*Diskussion*	*Albin*
6.1	*PCA*	*Albin*
6.2	*LNN*	*Albin*
6.3	*CNN*	*Albin*
6.4	*Jämförelse*	*Albin*
7	*Samhälleliga och etiska aspekter*	*Hannes*
	Appendix A – Ordlista	*Alla*
	Appendix B1 – Singulärvärdesuppdelning	*Bekir*
	Appendix B2 – Algoritmer för beräkning av SVD	*Albin*
	Appendix C – Källkod	*Albin*

Populärvetenskaplig presentation

Bildigenkänning är något vi gör i vardagen och som är trivialt för oss, men vi tänker sällan på att det är ett oerhört komplext koncept. När barn kommer till världen vet de ingenting om former och dess betydelser. Det är därför barn har svårt att förstå att de inte kan lägga en fyrkantig kub i ett runt hål. De lär sig detta genom att träna många gånger - misslyckas och lyckas om och om igen, och till slut förstår de att det är cylindern som passar i det runda hålet. På samma sätt är datorer barn som är nya till världen och måste lära sig från grunden vad som är vad.

Artificiell intelligens innebär maskiners förmåga att uppta, analysera och dra slutsatser [1]. Med den exponentiella utvecklingen av datorers kapacitet att beräkna ettor och nollor, växer deras förmåga att lösa komplicerade uppgifter. Syn kan ses som människans mest komplexa sinne, till exempel används en större del av vår hjärna till seende jämfört med andra sinnen [2]. Ett naturligt steg i att ge maskiner intelligens är därför att lära dem analysera visuell data. Att ge en maskin syn var nog otänkbart för några hundra år sedan, men som Arthur Clarke sa: ”Any sufficiently advanced technology is indistinguishable from magic.” [3]. Men med dagens matematik är det möjligt och faktiskt begripligt.

Detta arbete har använt singularvärderuppdeling (SVD), principalkomponentanalys (PCA) och neurala nätverk (NN). SVD är en uppdelning av en matris och kommer användas till att komprimera datan. PCA är ett statistiskt instrument för att minska dimensionen och analysera datan. Genom att se de mest särskiljande dragen i dataset kan metoden finna mönster och strukturer.

Neurala nätverk är likt ett barn när det kommer till läroprocessen. Nätverken tar emot data och analyserar den genom flera lager av artificiella neuroner och kommer slutligen fram till en slutsats [4]. Men, de måste tränas. På samma sätt som man påpekar att barnet lägger halvmånen i det rektangulära hålet, säger man till det neurala nätverket hur långt ifrån rätt svar det är. Då är förhoppningen att barnet och nätverket gör bättre vid nästa försök. Nätverket ändrar sedan på lite värden i förloppet och försöker igen. Denna process repeteras tills slutsatserna börjar stämma.

Så hur bra är dessa metoder på att känna igen handskrivna siffror? Och går det att kombinera dem för att skapa nya snabbare och bättre metoder? Dessa är frågorna vi ville söka svaret på. Genom att representera bilderna som matriser där varje värde representerar hur starkt upplyst en pixel är, kunde de matematiska modellerna analysera dem. Som riktmärke användes PCA-metoden, ett linjärt och ett konvolutionellt neuralt nätverk. De nya metoderna gick ut på att använda det linjära neurala nätverket på data som har genomgått delar av processerna från SVD och PCA.

Resultatet visade att originalmetoderna, alltså PCA, linjärt NN och konvolutionellt NN, fungerade bäst. Medan de kombinerade metoderna inte riktigt nådde upp till samma nivå. Den bästa kombinationsmetoden var PCA som kombinerades med linjärt NN genom poängvektor. Dock hade samtliga metoder olika styrkor och svagheter, vilket resulterade i varierande träffsäkerheter, träningstider och klassifieringstider. Val av metod beror därför i hög grad på den specifika tillämpningen och de krav som ställs på träffsäkerhet och tidsaspekterna. Vid bildklassificering inom sjukvården är träffsäkerheten av högst prioritet, då en felaktig klassificering kan vara livsavgörande. Medan vid automatiseringsprossesser i realtid kräver att processen sker snabbt och därmed är klassificeringstiden en viktig aspekt. Så vilken är egentligen den bästa maskininlärningstekniken? Mycket jobb har gjorts men mycket jobb finns kvar att göra innan man kan svara på den frågan.

Sammandrag

Dagens datorer är mer avancerade är någonsin och besitter kunskapen om att klassificera en bild med hjälp av matematiska metoder. Bildigenkänning används inom ett flertal områden, bland annat sjukvården och rättsväsendet. Syftet med arbetet var att analysera matematiska metoder inom bildigenkänning var för sig och slutligen kombinera dem för optimering. Metoderna som undersöktes var singulärvärdesuppdelning (SVD), principialkomponentsanalys (PCA) och neurala nätverk (NN). SVD är en matrisfaktorisering som används inom komprimering av bilder, medan PCA och NN är maskininlärningstekniker som används inom klassificering av bilder. PCA bygger på SVD, medan NN är inspirerad av nervcellerna i den mänskliga hjärnan.

Undersökningen gjordes genom att applicera maskininlärningsteknikerna på databasen *MNIST handwritten digits*. Implementationen programmerades i Python där biblioteksramverket PyTorch användes. Analysen av PCA gjordes med avseende på parametrarna *antalet tränings bilder*, *reduceringgrad* och *trunkeringsgrad*. För att kunna jämföra metoderna och erhålla ett tillförlitligt resultat definierades ett linjärt nätverk som användes till alla kombineringar. Ett konvolutionellt nätverk användes som referens vid jämförelse av metoderna och tränades på *originalbilder*. Det linjära neurala nätverket tränades på *originalbilder*, *trunkerad PCA på originalbilder*, *poängvektor*, U_k och V_k^T.

Arbetets slutsats kring resultaten visade att de etablerade metoderna gav en hög och tillförlitlig träffsäkerhet. Vid kombineringen av metoderna var poängvektorn från PCA tillsammans med NN den mest lyckade.

Abstract

In today's society, computers are more advanced than ever and are able to classify images by using mathematical methods. Image recognition techniques are used in several industries, eg. healthcare and the judiciary system. The project aimed to analyze different types of mathematical methods used in image recognition individually and combine them for optimization. The project included the following methods: singular value decomposition (SVD), principal components analysis (PCA), and neural networks (NN). SVD is a form of matrix factorization used within the storage of images while PCA and NN are machine learning techniques used in image recognition. PCA has its roots in SVD, and NN are inspired by the neurons in the brain.

The research was done by applying machine learning techniques on the dataset *MNIST handwritten digits*. The implementation was programmed in Python where the library PyTorch was utilized. PCA was analyzed with regards to the parameters *amount of training pictures*, *degree of reduction*, and *degree of truncation*. To be able to compare the methods and obtain a reliable result, one linear network was defined and used for all the combinations of methods. A convolutional neural network was also utilized with the original pictures, to create a reference used for comparison. The linear network was trained on *original pictures*, *truncated PCA on original pictures*, *score vector*, *original pictures*, U_k and V_k^T.

The results showed that the established methods gave a high and reliable precision. From the combination of methods, the one with the score vector from PCA incorporated with the linear network was the most successful.

Innehåll

1 Inledning **1**

2 Problemformulering **2**

3 Matematiska metoder **3**
3.1 Singulärvärdesuppdelning - SVD . . . 3
3.1.1 Bildkomprimering med singulärvärdesuppdelning . . . 4
3.2 Principalkomponentanalys - PCA . . . 4
3.2.1 Bildigenkänning med PCA . . . 6
3.3 Neurala nätverk - NN . . . 6
3.3.1 Framåtpropagering . . . 7
3.3.2 Förlustfunktion . . . 8
3.3.3 Bakåtpropagering . . . 8
3.3.4 Konvolutionellt neuralt nätverk - CNN . . . 9

4 Implementation och arkitektur **10**
4.1 Dataset . . . 10
4.2 Arkitekturen av neurala nätverk . . . 10
4.3 Implementation . . . 11

5 Numeriska tester **13**
5.1 PCA . . . 13
5.2 LNN . . . 14
5.3 CNN . . . 17
5.4 Illustration av tester . . . 17

6 Diskussion **18**
6.1 PCA . . . 18
6.2 LNN . . . 18
6.3 CNN . . . 19
6.4 Jämförelse . . . 19

7 Samhälleliga och etiska aspekter **20**

A Appendix – Ordlista **i**

B Appendix – Teori **ii**
B.1 Singulärvärdesuppdelning . . . ii
B.2 Algoritmer för beräkning av SVD . . . iv

C Appendix – Källkod **vi**
C.1 Bildkomprimering . . . vi
C.2 Klassificering med PCA . . . vi
C.3 Plottning av PCA klassificering . . . xii
C.4 Generering av modifierade dataset för NN . . . xiv
C.5 Modeller för NN . . . xvii
C.6 Träning och testning av NN . . . xviii
C.7 Plottning av NN . . . xxvii
C.8 Nedladdning av MNIST och hjälpfunktioner . . . xxxvi
C.9 Grafiskt användargränssnitt . . . xliii

1 Inledning

En tillbakablick på mänsklighetens historia visar att innovationer och ny teknik ständigt har förändrat världen och varit en utslagsgivande faktor till människans förbättrade levnadstandard. Den evolutionära utvecklingen av sofistikerad teknik har möjliggjort att datorer och maskiner idag uppvisar allt mer människoliknande egenskaper [5]. Maskininlärningstekniker för bildklassificering utgör en fundamental grundpelare för datorseende och är tekniker som tillskriver datorer mänskliga drag, samt gör det möjligt för maskiner att interagera intelligent med omgivningen [6]. Denna interaktion har stora möjligheter för framtida tekniker med tanke på dess potential.

Bildklassificering, eller bildigenkänning, omfattar en uppsättning metoder och algoritmer, som bygger på matematiska modeller, vilka bearbetar innehållet i bilder [7]. Dessa metoder tillåter datorer att analysera och tolka bilder med olika processer, exempelvis genom att utvärdera en bild i jämförelse med ett omfattande bibliotek [8]. Andra adaptiva metoder försöker kartlägga en relation mellan en uppsättning pixlar och ett objekt [9]. En digital bild kan representeras av en matris bestående av numeriska värden [10] som utgör indata till bildigenkänningsalgoritmer. Med hjälp av denna information kan systemet känna igen mönster och tolka innehållet i bilden [11].

Tekniker för bildigenkänning utvecklades strax efter de första elektroniskt programmerbara datorerna. Vissa tekniker försöker rekonstruera den mänskliga hjärnan och dess förmåga till logiskt tänkande. I och med att en stor andel av information som bearbetas av hjärnan är visuell, utgjorde bildklassificering den naturliga utveckling för datorer [2]. Ett av de tidigare vetenskapliga verken inom området var Lawrence Roberts doktorstes ”Machine perception of three-dimensional solids” från 1963, där han utvecklade en bildbehandlingsalgoritm för att extrahera objekts egenskaper från bilder [12]. Det stora genombrottet inträffade däremot några decennium senare, närmare bestämt år 1991, i samband med Matthew Turk och Alex Pentlands forskringsarbete där principalkomponentanalys användes för att detektera och identifiera ansikten [13]. Bildigenkänning har sedan dess utvecklats enormt och applicerat inom en rad olika områden och branscher [14].

Framsteg inom digital teknik har varit en drivkraft bakom utvecklingen av sociala medier, som ursprungligen var utformade som ett sätt att etablera och vårda personliga kontakter. Dessa har över tid utvecklats till en virtuell samlingsplats och handelsplattform, där användare delar en enorm mängd data i form av videor och bilder [15]. Detta är en faktor till att många olika branscher har tillgång till en stor mängd digital data som kan användas för olika syften, vilket gör bildklassificering till ett väldigt kraftfullt verktyg [16]. En av de viktigare tillämpningarna av bildigenkänning är inom sjukvården, där det används för att diagnostisera olika sjukdomar som exempelvis cancer [17]. Bildigenkänningsmetoder är även gynnsamma och förekommande inom statliga myndigheter, där polis och andra myndigheter i allmänhet använder denna teknik för att känna igen människor i videor eller bilder [18]. Dessutom har bildigenkänningstekniker en avgörande roll i datorseende som gör det möjligt att samla in och analysera data i realtid, vilket ligger till grund för automatiseringsprocesser [19].

Syftet med detta arbete är att undersöka olika maskininlärningstekniker för klassificering av bilder, genom att tillämpa dessa på en datamängd för att sedan utvärderas och jämföras för bedömning av dess effektivitet. Vidare ska maskinlärningsteknikerna kombineras i strävan efter optimering av effektivitet. I detta projekt analyseras och tillämpas metoderna singulärvärdesuppdelning (SVD), principalkomponentanalys (PCA) och neurala nätverk (NN) för bildklassificering. Kombinering av metoderna innefattar att ett linjärt neuralt nätverk tränas på utdatan från SVD och PCA.

2 Problemformulering

Bildigenkänning är en dators förmåga att identifiera vad en bild föreställer [20]. Problemet som studerades i detta arbete utgick från en databas av bilder som klassificerades med hjälp av olika matematiska metoder. Genom teorier kring matrisfaktorisering (SVD), dimensionreducering (PCA) samt neurala nätverk har precision, träningstid och testtid undersökts. Precisionen P syftar på andelen rättklassificerade bilder och beräknas enligt

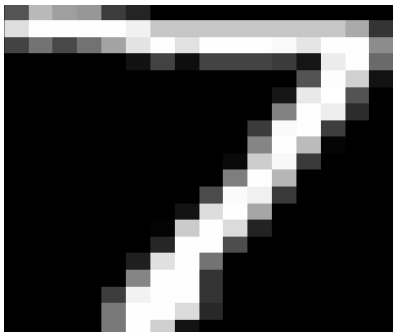
$$P = \frac{\sum_{i=1}^{K} N_{T,i}}{\sum_{i=1}^{K} \left(N_{T,i} + N_{F,i} \right)}, \tag{1}$$

där K är antalet klasser, $N_{T,i}$ och $N_{F,i}$ är antalet bilder av klassen i som är korrekt, respektive felaktigt klassificerade [21]. Träningstid avser den tid som krävs att träna en modell, medan testtid representerar tiden det tar för en tränad modell att klassificera en bild.

Digitala bilder är numeriska representationer av visuell data och kan representeras som matriser [10]. Varje element i matrisen korrelerar till en pixel i bilden, som motsvarar en viss färgs intensitet vid den specifika punkten. Pixlarna antar vanligtvis ett heltalsvärde mellan 0 och 255 och definieras av 8 bitar [22]. Det finns många olika modeller som representerar bilder, två modeller som använder just detta pixelintervall är gråskala och RGB. Gråskala är en monokrom illustration av en bild. Detta innebär att modellen endast innehåller information om ljusstyrkan, där 255 korrelerar till maximal ljusstyrka i form av vitt, medan 0 betyder att pixeln är svart [10]. Alla intensitetsvärden däremellan representerar olika nyanser av grått. En bild kan således illustreras i gråskala enligt Figur 1. Om bilden istället är i färg kan RGB-modellen användas som består av tre matriser vilka beskriver intensiteten för rött, grönt och blått. Matriserna kombineras sedan för att producera en fullständig färgbild.

84	185	159	151	60	36	0	0	0	0	0	0	0	0	0	0
222	254	254	254	254	241	198	198	198	198	198	198	198	198	170	52
67	114	72	114	163	227	254	225	254	254	254	250	229	254	254	140
0	0	0	0	0	17	66	14	67	67	67	59	21	236	254	106
0	0	0	0	0	0	0	0	0	0	0	0	83	253	209	18
0	0	0	0	0	0	0	0	0	0	0	22	233	255	83	0
0	0	0	0	0	0	0	0	0	0	0	129	254	238	44	0
0	0	0	0	0	0	0	0	0	0	59	249	254	62	0	0
0	0	0	0	0	0	0	0	0	0	133	254	187	5	0	0
0	0	0	0	0	0	0	0	0	9	205	248	58	0	0	0
0	0	0	0	0	0	0	0	0	126	254	182	0	0	0	0
0	0	0	0	0	0	0	0	75	251	240	57	0	0	0	0
0	0	0	0	0	0	0	19	221	254	166	0	0	0	0	0
0	0	0	0	0	0	3	203	254	219	35	0	0	0	0	0
0	0	0	0	0	0	38	254	254	77	0	0	0	0	0	0
0	0	0	0	0	31	224	254	115	1	0	0	0	0	0	0
0	0	0	0	0	133	254	254	52	0	0	0	0	0	0	0
0	0	0	0	61	242	254	254	52	0	0	0	0	0	0	0
0	0	0	0	121	254	254	219	40	0	0	0	0	0	0	0
0	0	0	0	121	254	207	18	0	0	0	0	0	0	0	0

(a) Matrisrepresentation



(b) Orginalbild

Figur 1: Handskriven siffra i gråskala och dess matrisrepresentation.

Betrakta en svartvit bild som representeras med en matris $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, se Figur 1a för exempel. Matrisen $\boldsymbol{A}$ kan faktoriseras med bland annat SVD, se sektion 3.1, där en approximation $\boldsymbol{A}_k$ tas fram för att minska datamängden, se Figur 2. Matriserna kan användas på flera sätt för bildigenkänning, bland annat genom principalkomponentanalys, se sektion 3.2, som skapar en poängmatris och vid jämförelse med träningsdatan kan klassificera bilder. Bilder kan även klassificeras med neurala nätverk, se sektion 3.3, som är en iterativ process för att hitta en funktion som bäst kan särskilja olika bilder. I arbetet har ovanstående metoder analyserats, det har även framtagits hybridmetoder som kombinerar SVD och PCA med neurala nätverk för optimering.

Det har satts avgränsningar på arbetet i syfte att undvika onödig komplexitet samt öka effektiviteten i utredningsarbetet. Bilderna som användes för klassificering begränsades till att komma från en databas, *MNIST handwritten digit database* [23]. Det är värt att notera att metoderna som användes i detta arbete inte är begränsade till en specifik databas. Kombineringen av metoderna SVD och PCA med neurala nätverk valdes att göras genom manipulation av indatan, och inte förändring av matematiken bakom neurala nätverket då det anses vara en avancerad process. De

valda metoderna har programmerats i Python där biblioteksramverket PyTorch [24] användes för implementering av neurala nätverk.

3 Matematiska metoder

I och med att en bild har möjligheten att representeras som en matris $\mathbf{A} \in \mathbb{R}^{m \times n}$ kan matematiska modeller för maskininlärning tillämpas. På så sätt kan modeller tränas till att utföra uppgifter som bildklassificering och objektdetektering. För att ge läsaren en förståelse redogörs den matematiska teorin bakom metoderna som användes för maskininlärning. Först introduceras SVD, vilket är en metod som kan användas till approximering av bilder [25]. Därefter presenteras PCA och slutligen beskrivs NN, som båda kan användas vid klassificering av bilder [4], [25]. För en sammanfattning av definitioner finns även en ordlista, se Appendix A.

3.1 Singulärvärdesuppdelning - SVD

SVD är en faktoriseringsmetod för matriser med tillämpningar inom flera områden, bland annat bildkomprimering och PCA [25]. Metoden bygger på att en matris kan faktoriseras till tre matriser som multiplicerade med varandra återskapar ursprungmatrisen [26]. Låt $\mathbf{A}$ vara en godtycklig matris sådan att $\mathbf{A} \in \mathbb{R}^{m \times n}$ där $m \geq n$, då kan dess singulärvärdesuppdelning skrivas som

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\boldsymbol{V}^T, \tag{2}$$

där $\mathbf{U} \in \mathbb{R}^{m \times n}$ och $\mathbf{V}^T \in \mathbb{R}^{n \times n}$ är ortogonala matriser, vilket betyder att $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ och $\mathbf{V}\mathbf{V}^T = \mathbf{I}$ där $\mathbf{I}$ är identitetsmatrisen. $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ är en diagonalmatris sådan att

$$\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n), \quad \text{där} \quad \sigma_1 \geq \sigma_2 \geq, \ldots, \geq \sigma_n \geq 0, \tag{3}$$

vars element σ_i kallas för singulärvärden till $\mathbf{A}$ och är ordnade efter storlek [25]. Detta ger således matriserna

$$\underset{m \times n}{\underset{A}{\begin{bmatrix} | & | & & | \\ a_1 & a_2 & \ldots & a_n \\ | & | & & | \end{bmatrix}}} = \underset{m \times n}{\underset{U}{\begin{bmatrix} | & | & & | \\ u_1 & u_2 & \ldots & u_n \\ | & | & & | \end{bmatrix}}} \underset{n \times n}{\underset{\Sigma}{\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}}} \underset{n \times n}{\underset{V^T}{\begin{bmatrix} — & v_1^T & — \\ — & v_2^T & — \\ \vdots & \vdots & \vdots \\ — & v_n^T & — \end{bmatrix}}}. \tag{4}$$

Singularvärdena är icke-negativa kvadratrötter av egenvärdena för $\boldsymbol{A}^T\boldsymbol{A}$ eller $\boldsymbol{A}\boldsymbol{A}^T$, se Appendix B.1 sats 2.1 och 2.2. Kolumnerna i $\mathbf{U}$ och $\mathbf{V}$ kallas för de vänstra respektive högra egenvektorerna till matrisen $\mathbf{A}$ då de är egenvektorerna till $\boldsymbol{A}\boldsymbol{A}^T$ respektive $\boldsymbol{A}^T\boldsymbol{A}$. Då singuarvärdena minskar i storlek minskar även betydelsen de har för återskapandet av originalmatrisen. Detta går att utnyttja för att reducera dimensionen på de faktoriserade matriserna genom att endast använda de k första singularvärdena och egenvektorerna. Denna metod kallas för trunkerad SVD och beskrivs enligt

$$\boldsymbol{A} \approx \boldsymbol{A}_k = \boldsymbol{U}_k\mathbf{\Sigma}_k\boldsymbol{V}_k^T, \tag{5}$$

där $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ och $\mathbf{V}_k^T \in \mathbb{R}^{k \times n}$ är de k första kolumnerna av $\mathbf{U}$ och $\mathbf{V}^T$, matrisen $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ är undermatrisen av $\mathbf{\Sigma}$ som innehåller de k största singularvärdena. Den trunkerade SVD:n är en optimal lågrangsapproximation av matrisen $\mathbf{A}$, det vill säga att felet vid rekonstruktion av $\mathbf{A}$ är det minsta möjliga med k stycken vektorer och singularvärden, se Appendix B.1, sats 2.4. För beräkning av SVD kan flera olika metoder användas, bland annat jacobimetoden, söndra och härska-algoritmen och QR-iteration, se Appendix B.2 för en förklaring av jacobimetoden.

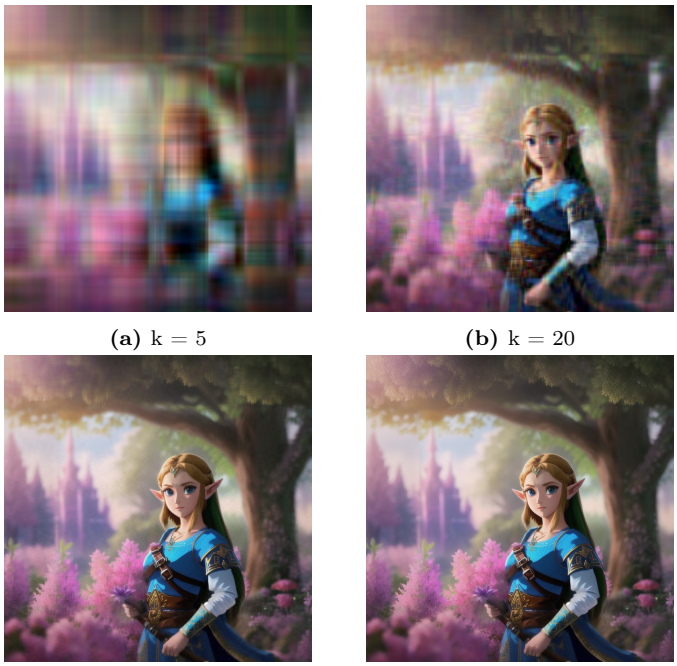
3.1.1 Bildkomprimering med singulärvärdesuppdelning

Ett användningsområde för trunkerad SVD är bildkomprimering [25]. För en färgbild med tre kanaler, röd, grön och blå, kan detta göras genom att låta bilden representeras av tre matriser, en för varje färgkanal. Genom att utföra trunkering på matriserna från SVD separat, kan varje kanal representeras av $\boldsymbol{U}_k$, $\boldsymbol{\Sigma}_k$ och $\boldsymbol{V}_k^T$. Då bilden ska rekonstrueras kan matriserna, för respektive färg, multipliceras enligt (5) till en approximation av orginalbilden. Kvaliteten av approximationen och den mängd lagringsutrymme som sparas beror på trunkeringsgraden k. Exempelvis, för en 784×784 pixlar stor bild med tre färgkanaler och 8-bitars färg kan lagringsutrymmet för en okomprimerad och en komprimerad bild, med trunkeringsgrad $k = 20$, beräknas som

$$S_0 = 784 \cdot 784 \cdot 3 \cdot 8 \,\text{bitar} \approx 1.84 \,\text{MB}, \tag{6}$$

$$S_{k=20} = 784 \cdot 20 \cdot 3 \cdot 32 \,\text{bitar} \cdot 2 + 20 \cdot 3 \cdot 32 \,\text{bitar} \approx 0.38 \,\text{MB}, \tag{7}$$

där S_0 är lagringsutrymmet som krävs för den okomprimerade bilden och $S_{k=20}$ är lagringsutrymmet som krävs för den trunkerade versionen. Värdena i de uppdelade matriserna $\mathbf{U}_k$, $\mathbf{\Sigma}_k$ och $\mathbf{V}_k^T$ antas lagras med 32 bitars flyttal. Figur 2 illustrerar bildkvaliteten för olika trunkeringsgrader.



(a) k = 5 **(b)** k = 20 **(c)** k = 100 **(d)** Original

Figur 2: Bilder (784×784 pixlar) komprimerade med hjälp av trunkerad SVD.

En bilds övergripande struktur fångas av de första singulärvärdena som identifierar de viktigaste egenskaperna och konturerna i bilden, detta är något som framgår i Figur 2 där bilden redan kan tydas för låga värden på k. För att de mindre detaljerna i bilden ska fångas krävs det att fler singularvärden inkluderas.

3.2 Principalkomponentanalys - PCA

PCA är en dimensionsreducerande metod som kan användas för att hitta mönster och samband i stora dataset [27]. Metoden transformerar en datamängd till ett hierarkiskt ortogonalt koordinatsystem som spänns upp av matrisens principalkomponenter (PK) [25]. Låt x_n, $n = 1, \ldots, N$ vara en datamängd där x_n är en vektor av ett visst prov med d variabler. Syftet med PCA är att projicera datamängden på ett rum med dimensionen $M < d$, samtidigt som variationen maximeras hos den projicerade datan. Metoden används för att identifiera de mest relevanta variablerna som representerar en större del av variationen av datamängden. Varje PK representerar en riktning i datamängden och väljs så att de beskriver den maximala variansen i datasetet i fallande ordning

[25], där den första komponenten representerar den riktning som beskriver den maximala variansen.

För illustration av beräkning av PCA, låt $M = 1$, vilket innebär att datamängden projiceras på ett endimensionell rum [28]. Först beräknas medelvärdet $\bar{x}$ av datamängden x enligt

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n. \tag{8}$$

För att erhålla information om hur variablerna i datamängden samspelar med varandra, beräknas kovariansmatrisen $\mathbf{C}$ av datamängden som

$$\boldsymbol{C} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T. \tag{9}$$

Låt u_1 vara enhetsvektorn som representerar det endimensionella rummet, där $u_1^T u_1 = 1$. Då kan varje prov x_n projiceras som en skalär, $u_1^T x_n$. Variansen av datan som projiceras blir

$$\frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = u_1^T \boldsymbol{C} u_1, \tag{10}$$

denna ska maximeras med avseende på u_1. Detta görs med en Lagrangemultiplikator, λ, därefter maximeras

$$L(u_1) = u_1^T \boldsymbol{C} u_1 + \lambda_1 (1 - u_1^T u_1). \tag{11}$$

Om $L'(u_1) = 0$ ansätts, fås

$$L'(u_1) = \boldsymbol{C} u_1 - \lambda u_1 = 0, \tag{12}$$

vilket betyder att

$$\boldsymbol{C} u_1 = \lambda_1 u_1. \tag{13}$$

Detta säger att λ_1 är ett egenvärde till $\mathbf{C}$ med korresponderande engenvektor u_1, och att $u_1^T \boldsymbol{C} u_1 = \lambda_1$. Därmed fås största variansen då u är egenvektor som tillhör det största egenvärdet, detta kallas den första principalkomponenten. De andra komponenterna hittas genom att titta på de andra egenvärdena. För ett M dimensionellt rum, utvärderas istället M egenvektorer och respektive egenvärden, som maximerar mängdens varians genom projicering.

PCA kan utföras med hjälp av SVD på en normaliserad datamängd $\mathbf{X}$ med olika variabler i kolumnerna och olika prover i raderna, då fås principalkomponenterna för $\mathbf{X}$ som de högra egenvektorerna till $\mathbf{X}$ [25]. Det vill säga att kolumnerna i $\mathbf{V}$ från SVD:n $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ är principalkomponenterna till $\mathbf{X}$. Ett vanligt tillvägagångssätt för att bestämma PCA är att först beräkna kovariansmatrisen $\mathbf{C}$ för datamängden $\mathbf{X}$ och sedan beräkna SVD. Då erhålls

$$\boldsymbol{C} = \frac{1}{N} \boldsymbol{X}\boldsymbol{X}^T = \frac{1}{N} (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T) = \frac{1}{N} \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \frac{1}{N} \boldsymbol{V}\boldsymbol{\Sigma}^2 \boldsymbol{V}^T, \tag{14}$$

varifrån det går att erhålla egenvektorer $\mathbf{V} = (\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_n})$ som motsvarar PK:er för $\mathbf{X}$. För att transformera datasetet till vektorrummet som spänns upp av principalkomponenterna projiceras de på PK:erna. Denna projektion kan enkelt beräknas som

$$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{V} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)\boldsymbol{V} = \boldsymbol{U}\boldsymbol{\Sigma}, \tag{15}$$

där matrisen $\mathbf{T}$ defineras som poängmatrisen för $\mathbf{X}$ och är projektionen av $\mathbf{X}$ på dess PK:er. Varje rad i poängmatrisen representerar projektionen av ett prov i $\mathbf{X}$ på PK:erna, låt en sådan

projektion kallas för provets poängvektor. För att minska dimensionen av datanmändgen trunkeras ofta poängmatrisen, likt trunkeringen för SVD. Den trunkerade poängmatrisen $\boldsymbol{T}_k$ kan beräknas enligt

$$\boldsymbol{T_k} = \boldsymbol{X}\boldsymbol{V}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k. \tag{16}$$

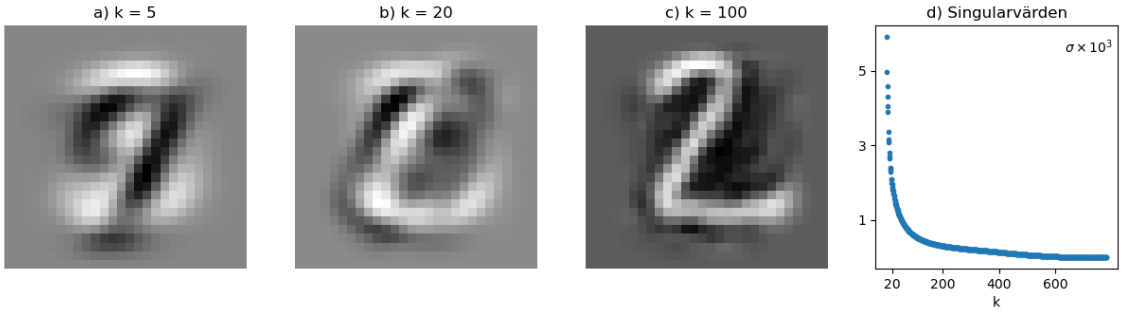
Där $\boldsymbol{V}_k$, $\boldsymbol{U}_k$ och $\boldsymbol{\Sigma}_k$ fås från trunkerad SVD enligt (5).

3.2.1 Bildigenkänning med PCA

För att genomföra bildigenkänning kan PCA utföras på en träningsmängd med bilder för att erhålla en poängmatris samt principalkomponenter för dessa. Träningsmängden ska innehålla bilder där varje pixel är en variabel och varje bild är ett prov. Bilderna plattas därför ut till en endimensionell vektor och staplas för att bilda en matris som PCA kan genomföras på. För att klassificera okända bilder i en testmängd kan även dessa projiceras på PK:erna för träningsmängden, varefter en poängvektor för testbilderna erhålls. Bilderna i testmängden kan därefter antas vara av samma klass som den bild i träningsmängden där normen av skillnaden mellan de två poängvektorerna är som minst. Matematiskt kan detta uttryckas som

$$\min \|\mathrm{T_k(test(i))} - \mathrm{T_k(träning(j))}\|_2^2. \tag{17}$$

En bild i i testmängden antas ha samma klass som bild j i träningsmängden om absolutbeloppet av skillnaden mellan deras poängvektorer är det minsta möjliga. Tiden det tar att klassificera en bild beror dels på hur många bilder som ingår i träningsmängden, och dels på hur många principalkomponenter som använts. Precisionen beror på samma parametrar, trunkeringsgraden då det påverkar hur mycket detaljer som sparas, och träningsmängdens storlek eftersom en större sådan medför en ökad sannolikhethet att innehålla en liknande bild som i träningsmängden. Ett exempel på hur bilder ser ut med olika antal principalkomponenter bevarade finns i Figur 3. Figuren innehåller även en plot på storleken av singularvärdena. I denna är det uppenbart att de första principalkomponenterna har störst påverkan på bilden.

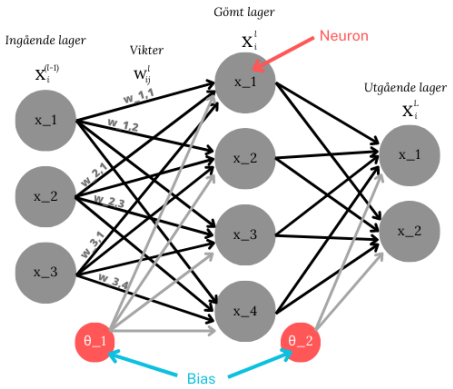


Figur 3: a), b), c) En handmålad tvåa återskapad med olika antal principalkomponenter. d) Storleken på singularvärdena för SVD på datasetet MNIST.

3.3 Neurala nätverk - NN

Neurala nätverk kan bland annat användas för att klassificera bilder [4]. För att förstå hur ett NN fungerar behöver först uppbyggnaden studeras. Det finns flertal olika NN modeller som har olika komplexitet. Linjära neurala nätverk är den enklaste typen och består av linjära lager, där alla neuroner i ett visst lager är genom vikter kopplade till nästkommande lager, se Figur 4. Det första lagret i ett neuralt nätverk, kallas för det ingående lagret och agerar som mottagare av indata. Det sista lagret, som är det utgående lagret, utgör nätverkets slutresultat. Mellan dessa två lager finns

ett eller flera gömda lager. Mellan varje lager används också en aktiveringsfunktion samt läggs en bias till, som innebär att en neurons värde kan förskjutas. Det som skiljer olika neurala nätverk åt är uppbyggnaden av de olika lagren, där vikternas kopplingar kan skilja.



Figur 4: Exempel på uppbyggnaden av ett linjärt neuralt nätverk, där det ingående lagret består av tre neuroner och det utgående lagret av två neuroner. Detta neurala nätverk består av ett gömt lager med fyra neuroner.

För att ett NN ska kunna klassificera till exempel bilder, behöver det först tränas [4]. Till det neurala nätverket ges en indata bestående av bilder och utdata bestående av bildernas klassificering, exempelvis träd eller boll. Nätverket försöker sedan att förutspå bildernas motiv, genom *framåtpropagering*. Hur säkert nätverket klassificerar olika bilder räknas sedan ut med en *förlustfunktion*. Efter det kommer nätverket att utföra så kallad *bakåtpropagering*, där uppdatering av vikter sker för att klassificeringen av bilderna ska få högre precision. Dessa tre steg kommer sedan att upprepas flera gånger om, tills dess att nätverket säkert kan klassificera bilderna. Under de följande tre rubriker presenteras matematiken bakom de olika stegen på ett linjärt NN, som ligger till grund för alla nätverk. Slutligen presenteras ett mer komplex, kallat konvolutionellt NN, som är presenterat i sektion 3.3.4.

3.3.1 Framåtpropagering

Framåtpropagering används för att bearbeta indata genom de olika lagren i det neurala nätverket för att slutligen generera en klassificering [29]. Denna metod ligger till grund för resterande del av matematiken då det beräknade värdet jämförs med det önskade. Varje lagers neuroner kan beskrivas som en vektor $\mathbf{x}$ och vikterna som en matris $\mathbf{W}$ [30]. Låt x_i^l vara en specifik neuron i lager l, på plats i. Denna neurons värde fås genom

$$x_i^l = f(\underbrace{\sum_{j=1}^{N}(\omega_{ji}^{(l)} x_j^{(l-1)} - \theta_i^l)}_{b_i^l}) \tag{18}$$

där N är antalet neuroner i lagret. ω_{ji}^l är vikten från neuronen j i lager $(l-1)$ till neuron i i lager l och θ_j^l är tillhörande bias till neuronen i. f är tillhörande akriveringsfunktion till lagret l.

Det kommer att användas två aktiveringsfunktioner, f, med olika syften. Den ena funktionen används för att introducera icke-linjära egenskaper till nätverket, vilket gör att det kan modellera komplexa funktioner. Den kallas ReLu (Rectified Linear Unit) och används efter varje lager på b_i^l för att beräkna x_i^l. Den fås från

$$\text{ReLu}(b_i) = \max(0, b_i) = \begin{cases} b_i, & \text{om} \quad b_i \geq 0 \\ 0, & \text{om} \quad b_i < 0. \end{cases} \tag{19}$$

Den andra aktiveringsfunktionen som används kallas för Softmax och beräknas genom

$$\text{Softmax}(b_i^L) = \frac{e^{b_i^L}}{\sum_k^M e^{b_k^L}}. \tag{20}$$

Den används på det utgående lagret, L, för att få det beräknade värdet som en sannolikhetsfördelning över alla möjliga klasser och representeras av ett tal mellan [0,1] som kan jämföras med det önskade värdet. Där i är den specifika neuronen och M är antalet neuronen i lagret.

3.3.2 Förlustfunktion

För att kunna utföra bakåtpropagering, som bygger på att jämföra skillnaden mellan det beräknade värdet och det önskade värdet, behövs en förlustfunktion [30]. Då Softmax används som aktiveringsfunktion efter det utgående lagret kommer förlustfunktionen *Binary Cross-Entropy Loss* att användas. Låt H vara det beräknade felet, för en specifik bild, mellan det beräknade värdet x_i^L och det önskade värdet y_i. Då kan felet beräknas genom

$$H = -\sum_{i=1}^{K} y_i \log(x_i^L), \tag{21}$$

där K är antalet neuroner i det utgående lagret, det vill säga antalet klasser för databasen. L betecknar det utgående lagret.

3.3.3 Bakåtpropagering

Bakåtpropagering är den matematiska delen i ett neuralt nätverk som uppdaterar alla vikter och bias. Det kan göras genom flera olika metoder. Den metod som används i detta arbete är ”stochastic gradient descent” som är en stokastik iterativ process där ett globalt minimum eftersöks [30]. Felet mellan det beräknade värdet från framåtpropageringen och det önskade värdet tas i beaktning då varje vikts påverkan på felet beräknas och uppdateras enligt *Hebbs's rule*, som är

$$\omega_{ij}^{\prime l} = \omega_{ij}^l + \delta\omega_{ij}^l. \tag{22}$$

Där $\omega_{ij}^{\prime l}$ är det nya värdet som ersätter det gamla värdet ω_{ij}^l, och $\delta\omega_{ij}^l$ är litet steg mot en minimipunkt och ges av

$$\delta\omega_{ij}^l = -\eta\frac{\partial H}{\partial w_{jk}^L} = \eta \cdot \delta_i^l \cdot x_j^{(l-1)}, \tag{23}$$

där η är inlärningsgraden som är en faktor vid stegvisa iterationen. δ_i^l är felet vid neuronen i i lagret l som beräknas med

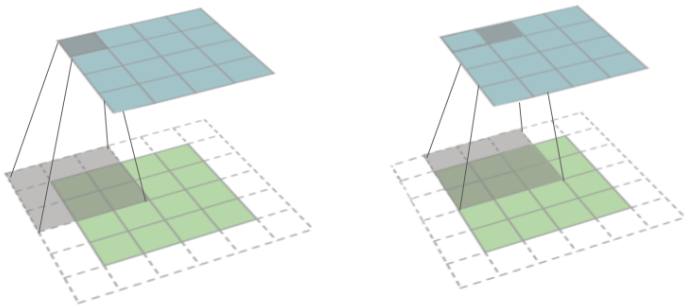
$$\delta_i^{l-1} = \begin{cases} \sum_j \delta_i^l \cdot \omega_{ij}^l \cdot f'(b_j^{(l-1)}) & l < L \\ \\ (y_i - x_i^L) \cdot f'(b_i^L) & l = L \end{cases}, \tag{24}$$

där f' är derivatan av aktiveringsfunktionen och y_i är det eftersökta värdet för neuronen x_i^L i det utgående lagret. På liknande sätt kan nya bias-värden $\theta_i^{\prime l}$ beräknas och uppdateras. När parametrarna är uppdaterade kommer träningsprocessen av det neurala nätverket att fortsätta genom framåtpropagering, förlustfunktion, bakåtpropagering och uppdatering av vikter samt bias, tills dess att det neurala nätverket presterar på den önskade nivån.

3.3.4 Konvolutionellt neuralt nätverk - CNN

I detta avsnitt beskrivs konvolutionella neurala nätverk och relaterade begrepp. Dessa nätverk presterar bra vid igenkänning av bilder eftersom bilder innehåller mönster [4]. Till exempel så ser ett objekt likadant ut oberoende av om det placeras högt eller lågt i en bild. CNN är uppbyggda av lager som har en längd, höjd och ett visst antal kanaler (djup), de är därmed tredimensionella, se Figur 8. Bilder känns igen genom att de första lagrena identifierar enkla strukturer och de senare lagrena hittar mer komplexa strukturer.

För att nätverken ska kunna identifiera dessa mönster har varje lager *filter* eller så kallade *kernels* [4]. Ett filter har en lägre dimension på längd och höjd än lagret den appliceras på, men har samma djup som lagret. Dimensionen av filtret kan skrivas som (k x k x C), där C är djupet och k betecknar storleken. En vanlig storlek av ett filter är $k = 3$. Filtret appliceras stegvis över lagret och utför skalärprodukten mellan värdena i filtret och de korrelerande värdena i lagret, detta ger värden i nästkommande lager. Varje filter som appliceras har olika parametrar och upptäcker därför olika mönster hos lagrena. Dimensionen på det nästkommande lagret påverkas av filtrets storlek, antalet filter som appliceras, stride och padding, se Figur 5. Stride, s, är det steg filtret tar vid varje ny beräkning av skalärprodukten. Padding, p, innebär att matrisen som filtret tillämpas på förstoras, det läggs på rader och kolumner med nollor, vilket gör att det inte tappas information vid hörn och kanter när ett filter används.



Figur 5: Illustration av ett filter för ett konvolutionellt lager. Indatan representeras av den gröna matrisen, där vita elementen motsvarar padding $p = 1$. Filtret i detta fall har storleken 3x3x1 och representeras av den gråa matrisen. Stride, med värde ett, innebär att filtret förflyttas med en steglängd ett över matrisen.

Ett vanligt problem med konvolutionella lager är att dimensionen av datan efter flera lager ökar och där med ökar träningstiden. Maxpooling är en vanlig lösning för att reducera storleken på utgående data från ett lager. Likt vid ett konvolutionellt lager används ett filter, där värdet istället fås från det maximala värdet inom filterområdet på matrisen i kanal C.

4 Implementation och arkitektur

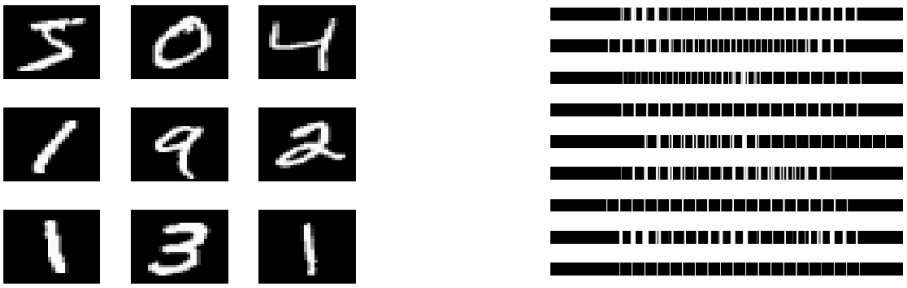
Innan en undersökning och analys av bildigenkänning genomfördes, bestämdes de metoder och parametrar som skulle användas. Detta gjordes genom informationsökning och matematisk analys. Vidare tillämpades en försöksprocess där olika metoder och strategier systematiskt testades till den mest effektiva identifierades. Arbetet implementerades i programeringspråket Python och är samlat på GitHub [31].

4.1 Dataset

Datamängden som användes genom arbetet är *MNIST Handwritten Digit Database* [23], som består av handskrivna svartvita siffror från noll till nio (0-9). Den består av 70 000 bilder, där 60 000 är träningsbilder och 10 000 är testbilder. Bilderna är representerade som en matris $\mathbf{X} \in \mathbb{R}^{m \times n}$ där $m = n = 28$, det vill säga att bilderna har 28 x 28 pixlar, och siffrorna är centrerade i bilderna, se Figur 6a. För att genomföra bland annat PCA krävs det en matris där varje kolumn representerar en variabel och varje rad representerar ett prov av variabler. Det är därför viktigt att notera att en bild representerad av en matris kan omvandlas till en vektor där dimensionen ändras som

$$\mathbb{R}^{m \times n} \implies \mathbb{R}^{1 \times m \cdot n}, \tag{25}$$

genom att varje rad från matrisen läggs efter varandra i en vektor. Sådana vektorer från flera bilder kan staplas, varefter det erhålls en matris där varje kolumn representerar en variabel, i detta fall ljusstyrkan på en pixel, och varje rad representerar ett prov av alla variabler, i detta fall en bild. För ett exempel se Figur 6b.



(a) Slumpmässigt utvalda bilder på handskrivna siffror.

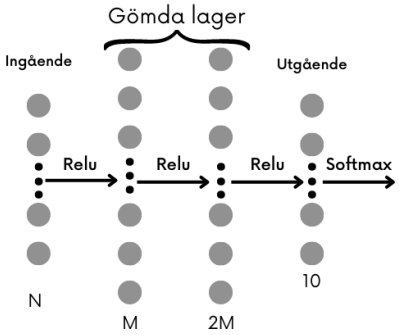
(b) Handskrivna siffror representerade som vektorer (prov) i en matris.

Figur 6: Illustration av utplattning av bilder ur *MNIST Handwritten Digit Database*.

4.2 Arkitekturen av neurala nätverk

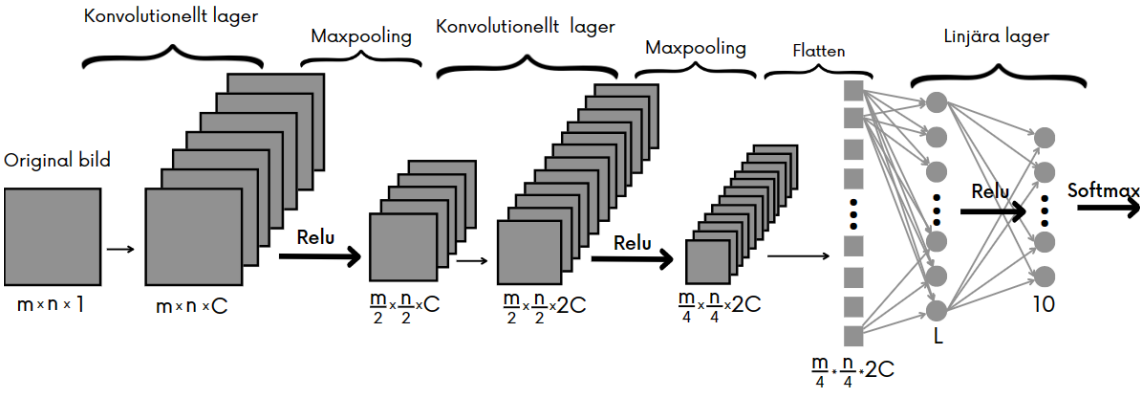
För att kunna jämföra och analysera SVD, PCA och NN med tillförlitliga resultat sattes ett ramverk för strukturen av de neurala nätverken. De olika arkitekturerna togs fram genom informationsökning och optimering efter egna undersökningar och analyser, men eftersom arbetets fokus var jämförelse av metoderna presenteras inte arkitekturens framtagning. I arbetet har ett linjärt och ett konvolutionellt nätverk tagits fram, se Figurer 7 och 8.

Linjära neurala nätverkets (LNN) arkitektur består av fyra fullt kopplade lager, där samtliga lager är uppbyggda på samma sätt med undantaget av att deras storlek varierar, se Figur 7. Det ingående lagret har samma storlek, N, som den ingående datan. Det första gömda lagret har storleken M, vilket är en variabel som undersöktes. Det andra gömda lagret är dubbelt så stort som det första, $2 \cdot M$. Storleken på det utgående lagret är 10, vilket motsvarar antalet klasser i datamängden (siffrorna 0-9). Efter varje lager används aktiveringsfunktionen ReLu, se (19), förutom efter det utgående lagret där aktiveringsfunktionen Softmax används istället, se (20).



Figur 7: Arkitektur av det linjära neurala nätverket som användes i detta arbete.

Konvolutionella neurala nätverkets (CNN) arkitektur, se Figur 8, består av fyra lager varav det ingående lagret och det första gömda lagret är konvolutionslager, som följs av ett gömt lager och ett utgående lager som är fullt kopplade (linjära lager). Båda konvolutionella lagrena har filterstorlek $k = 3$, stride $s = 1$ och padding $p = 1$. Det som skiljer lagrena åt är höjden, bredden och mängden kanaler, se Figur 8. Relu, se (19), används som aktiveringsfunktion efter de konvolutionella lagrena och det första linjära lagret. Efter det utgående används Softmax, se (20). För pooling har funktionen maxpooling används, det vill säga att det nya värdet tas från det maximala värdet. De värden som används är filterstorleken $k = 2$ och stride $s = 2$. Flatten plattar ut datan från det konvolutionella lagret till vektorform, se (25), så att det linjära lagret får korrekt dimension på indatan. Det första linjära lagret tar emot den utplattade datan till L neuroner som var en variabel i undersökningen, och det utgående lagret klassificerar slutligen dessa mellan dem tio olika siffrorna.



Figur 8: Arkitektur av det konvolutionella neurala nätverket som användes i detta arbete.

4.3 Implementation

Undersökningen genomfördes genom att analysera på vilket sätt olika bearbetningar av datan påverkade hur väl metoderna klassificerade bilder. De parametrar som studerades var *träffsäkerheten*, *träningstiden* och *klassificeringstiden*. Träffsäkerheten är procentsatsen för det korrekta antalet klassificeringar, och beräknades som antalet korrekt klassificerade bilder dividerat med antalet klassificerade bilder. Testtiden syftar på den tid det tar i genomsnitt för en bild att klassificeras och beräknades genom att dividera antalet klassificerade bilder med den totala tiden. Slutligen definieras träningstiden som tiden det tar att träna modellen till att bli redo att klassificera bilder. För NN innebär det träningsfasen och för PCA antas det vara tiden för beräkningarna fram till jämförelsen med poängmatrisen.

Analysen av principalkomponentsanalys gjordes genom att studera hur formateringen av träningsmängden och testmängden påverkade klassificeringen av bilder. De formateringar som undersöktes och analyserades var:

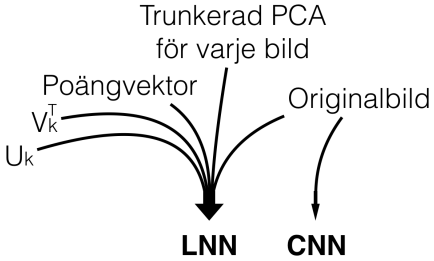
1. **Datamängdens storlek**. I och med att en datamängd består av ett stort antal träning- och testbilder är det inte alltid nödvändigt att träna en modell på samtliga bilder. För att undersöka hur storleken på träningssetet, och därmed poängmatrisen $\boldsymbol{T}_k$(träning), påverkade modellen testades olika storlekar.
2. **Reduceringen av datamängden**. Då datamängden består av många likartade bilder är det mycket information som upprepas och därmed gör metoden långsammare. Därför kan datamängden minskas genom att likartade bilder slås ihop. För att skapa den reducerade träningsmängden beräknades ett genomsnitt av de euklidiska normerna mellan alla bilders trunkerade poängvektorer. De vars skillnad var mindre än medelavståndet multiplicerat med en konstant a, kallad reduktionsgrad, slogs därefter ihop till en ny bild. De bilder som uppfyllde kriteriet kombinerades genom att medelvärdet för respektive pixel beräknades. Den kvarstående träningsmängden fick en mindre storlek och innehöll färre likartade bilder.
3. **Trunkeringsgraden**. Denna formatering innebär att bilderna i datamängden reducerades genom PCA med trunkeringsgraden k, se (5). Genom att justera trunkeringsgraden kan datamängden minskas, samtidigt som den mest värdefulla informationen bevaras.

Tabell 3: Den modifierade datan till PCA.

Formatatering	***Beskrivning***	***Beräkning***	***Variabel***
Datamängdens storlek	*Antalet bilder som modellen tränas på*	$\mathbf{X} \in \mathbb{R}^{m \times k}$ *(Ekvation 17)*	m
Reducerad datamängd	*Likartade bilder slås ihop*	*Se 2. Reducering av datamängden*	a
Trunkeringsgraden	*Datamängden har reducerats till trunkeringsgraden k*	$\mathbf{T}_k$ *(Ekvation 16)*	k

Kombineringen av metoderna SVD, PCA och NN innefattade enbart formateringen av ingående data, och ingen ändring i matematiken bakom ett neuralt nätverk undersöktes. Metodernas syfte var att minska dimensionen på datan in till det neurala nätverket, medan den viktigaste informationen sparades. Detta för att minska träningstiden och samtidigt behålla träffsäkerheten på modellen.

Det som undersöktes var resultaten från LNN då indatan var framtagen från SVD och PCA. Från SVD användes trunkerade versioner av $\mathbf{U}$ och $\mathbf{V}^T$, och från PCA användes poängmatrisen $\mathbf{T}_k$, se (16). Slutligen användes PCA på originalbilden med hänsyn på olika trunkeringsgrader k, som motsvarar antalet använda principalkomponenter. För att få ett resultat som basreferens testades även originalbilderna utan modifiering som ingående data både till LNN och CNN. I Figur 9 illustreras en sammanfattning av samtliga hybridmetoder och i Tabell 4 visas en förklaring av dem olika modifieringarna och dess dimensioner.



Figur 9: Sammanfattning av ingående data till de neurala nätverken.

Tabell 4: Sammanfattning av ingående data till de neurala nätverken, där dim(bild) = $m \times n$.

Formatering	***Erhålls genom***	***Dimension***
Orginalbild	*Matrisrepresentation av orginalbild*	$\mathbb{R}^{m \times n}$
Trunkerad PCA på orginalbild	*PCA på varje enskild bildmatris*	$\mathbb{R}^{m \times k}$
Poängvektor	*PCA på en matris där varje rad mosvarar en bild, se Figur 6b*	$\mathbb{R}^{1 \times k}$
$\boldsymbol{U}_k$	*Trunkerad SVD (ekvation 5)*	$\mathbb{R}^{m \times k}$
$\boldsymbol{V}_k^T$	*Trunkerad SVD (ekvation 5)*	$\mathbb{R}^{n \times k}$

5 Numeriska tester

I detta avsnitt presenteras de numeriska testerna som genomfördes för de olika klassificeringsmetoderna. Resultaten presenteras i figurform och de värden som användes för respektive test är presenterat i tabellform under varje underrubrik. Tre olika aspekter togs hänsyn till i utvärderingen av modellerna, precisionen, träningstiden samt testtiden. Resultaten är baserade på numeriska tester som genomfördes på MNIST-databasen, se sektion 4.1, med hjälp av en dator utrustad med en *Intel Core i7-13700K* processor och ett *NVIDIA RTX 4070 Ti* grafikkort. Träningen och testningen av de neurala nätverken gjordes med *CUDA*, en accelerator för beräkningar på grafikkort. Alla andra beräkningar, inklusive PCA och SVD, utfördes på processorn eftersom det är mycket mer komplext att implementera GPU-acceleration för dessa. Målet med testerna var att hitta de metoder som ger den bästa precisionen med så liten träning- och testtid som möjligt.

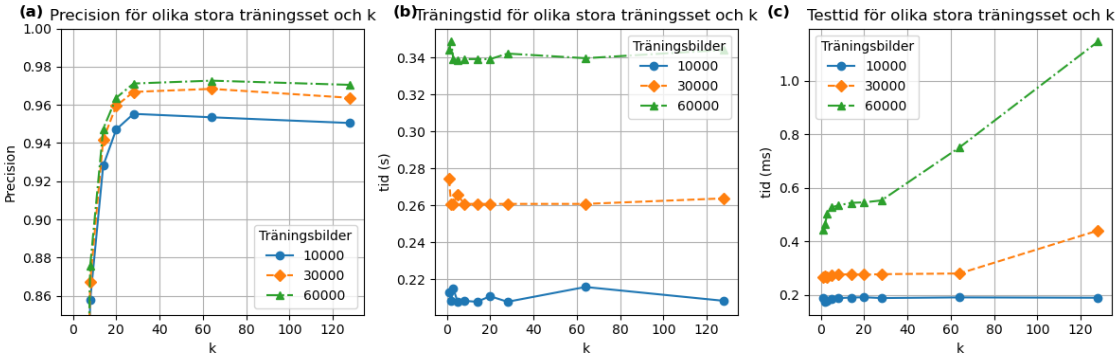
5.1 PCA

Klassificering med PCA utfördes på två olika sätt, se sektion 4.3. I den första metoden undersöktes hur träningmängdens storlek samt trunkeringsgraden påverkade klassificeringen av bilder. Den andra metoden undersökte hur reduceringen av antalet träningsbilder samt trunkeringsgraden påverkade resultatet. Vid högre reduceringsgrad ökade antalet ihopslagna bilder och träningsmängden minskade. De värden som användes i ovanstående metoder är presenterade i Tabell 5.

Tabell 5: Sammanfattning av de värden som användes i undersökningen av PCA.

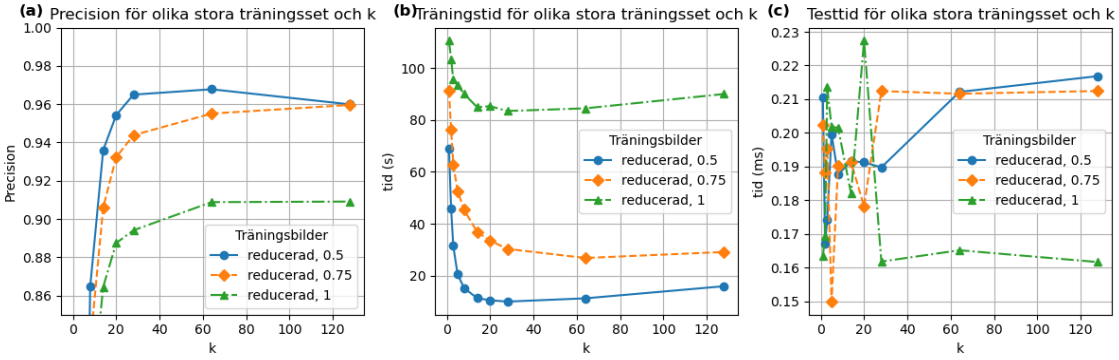
Parametrar	***Värden***
Metod 1: Antalet träningsbilder	*10 000, 30 000, 60 000*
Metod 2: Reduceringsgrad, a	*0.5, 0.75, 1*
Trunkeringsgrad k för båda metoderna	*1, 2, 4, 8, 16, 20, 28, 64, 128*

Figur 10 visar utfallet från undersökningen av den första metoden för PCA. Resultatet visar att samtliga undersökta parametrar ökade med antalet träningsbilder. Antalet principalkomponenter k hade en stor inverkan på precisionen och precisionen planade ut för $k \geq 28$. Trunkeringsgraden får en påverkan på testtiden först vid cirka $k \geq 28$ och har ingen tydlig påverkan på träningstiden.



Figur 10: a) Precision, b) träningstid [s], c) testtid [ms] för olika antal principalkomponenter k och olika antal bilder i träningssetet.

Resultat från den andra metoden illusterars i Figur 11, där varje graf representerar en viss reduceringsgrad. Det visas att testtiden stabiliserades vid $k \geq 20$ och precisionen ökade i takt med att reduceringsgraden minskade. Däremot ökade träningstiden med reduceringsgraden.



Figur 11: a) Precision, b) träningstid [s], c) testtid [ms] för olika antal principalkomponenter k och olika nivåer av reduktion.

Från Figur 10 och 11 noteras det att precisionen ökade hastigt för båda metoderna vid låga k och planade av vid högre. Träningstiden varierade med en faktor 100 mellan metoderna då algoritmen för reducering av datasetet tog lång tid. Vid $k = 28$ uppvisade både metoden med 30000 träningsbilder och metoden med en reduktionsfaktor på 0.5 en precision på mellan 96% och 97%. Testtiden var i detta fall däremot bättre för metoden med reducerad träningsmängd då den hade en tid på cirka 0.2 ms jämfört med 0.3 ms för den med 30000 bilder, vilket är en ökning med 50%. Träningstiden var däremot flera gånger större för den reducerade metoden.

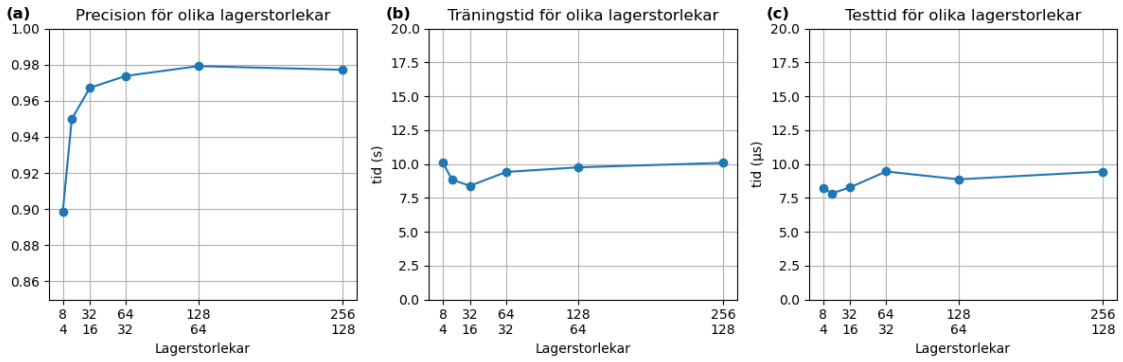
5.2 LNN

Det linjära neurala nätverket, se sektion 4.3, bestod av fyra fullt kopplade lager. Det ingående lagret hade storleken N och definierades av indatans storlek, som bland annat berodde på trunkeringsgraden k. Det första gömda lagret hade storleken M och det andra gömda lagret hade halva dess storlek. Värdena på M och k redovisas i Tabell 6.

Tabell 6: Sammanfattning av värden som användes i undersökningen av LNN och hybridmetoder.

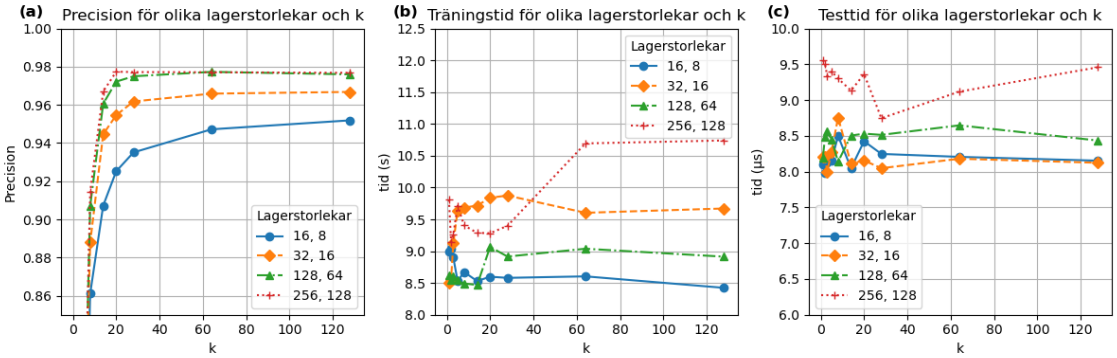
Indata	***Lagerstorlek, M***	***Trunkeringsinterval***
Originalbild	*8, 32, 64, 128 och 256*	–
Poängvektor	*16, 32, 128 och 256*	*1 - 128*
$\boldsymbol{V}_k^T$	*16, 64 och 256*	*1 - 20*
$\boldsymbol{U}_k$	*16, 64 och 256*	*1 - 20*
Trunkerad PCA på originalbild	*128 och 256*	*1 - 28*

Som basreferens användes en obehandlad bild som indata till det linjära nätverket. Ingångslagret bestod av en tillplattad bild, vilket var en vektor med längden 784. I figur 12 går det att se att främst precisionen ökade med stigande storlek av de gömda lagrena. Tränings- och testtiden hade en varians men överlag syns en svag ökning medan precisionens ökade snabbt till en början för att sedan nå en platå.



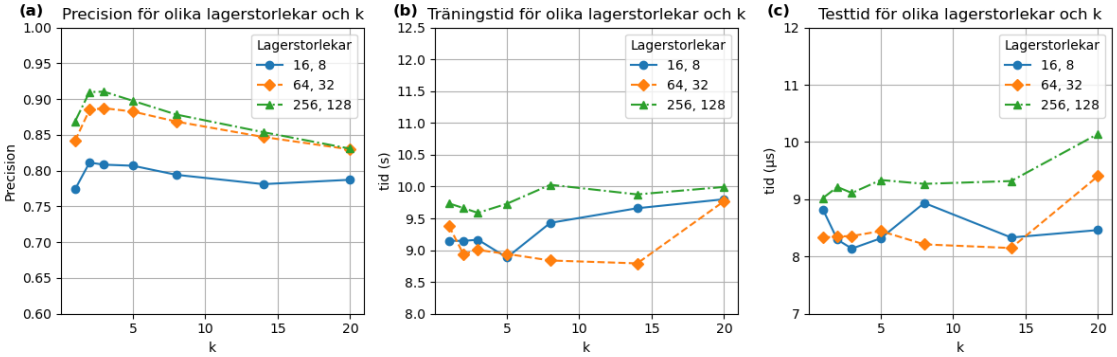
Figur 12: a) Precision, b) träningstid [s], c) testtid [μs] för olika lagerstorlekar i ett LNN. Det övre och undre numret på x-axeln anger storleken på det första respektive andra gömda lagret.

För metoden då poängmatrisen användes som indata till nätverket, berodde ingångslagret på poängvektorn. Denna erhölls genom projicering på träningssetets k första principalkomponenter, således var nätverkets ingångslager av storleken k. För de PCA-behandlade bilderna i Figur 13 gav LNN på poängvektorn liknande resultat som de obehandlade bilderna i Figur 12. Ungefär samma precision på 98% för höga k och en sämre precision då k var mindre. Träningstiden hade en viss varians men var generellt sett lägre för denna metod. För en lagerstorlek $M = 128$ var träningstiden cirka 9 sekunder, jämfört med 10 sekunder för den obehandlade bilden. Testtiden var liknande för båda metoderna.

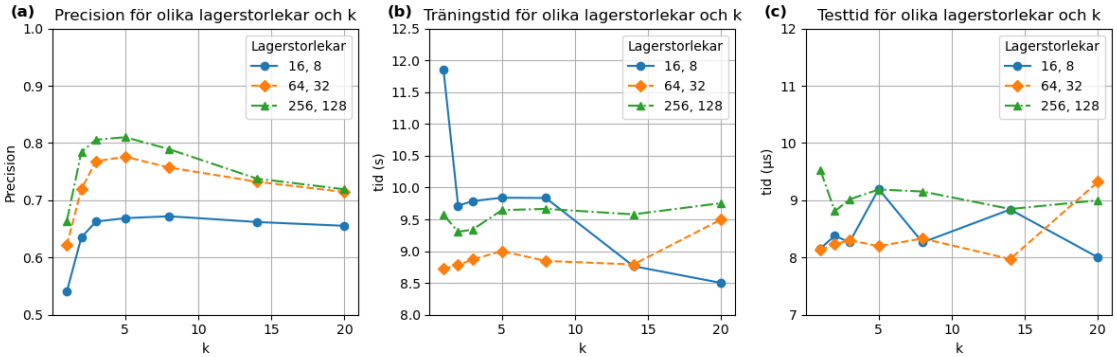


Figur 13: a) Precision, b) träningstid [s], c) testtid [μs] för LNN med olika lagerstorlekar tränade på poängvektorer från PCA med storleken k.

Vidare testades även att träna LNN med komponentmatriserna $\boldsymbol{U}_k$ och $\boldsymbol{V}_k^T$, från SVD. Detta gjordes med avseende på olika trunkeringsgrader k, vilket innebar att de k första kolumnerna av $\boldsymbol{U}$ och $\boldsymbol{V}^T$ användes. Matriserna plattades ut och ingånslagret blev därmed $m \cdot k$ respektive $n \cdot k$. Testerna för LNN tränade på egenvektorer, presenterade i Figur 14 och 15, gav sämre precision än basreferensen med orginalbilder i Figur 12. För dessa metoder nådde precisionen ett maximum då k var mellan 3 och 5 för att sedan minska. Test- och träningstid var dock relativt oförändrad i förhållande till referensen.



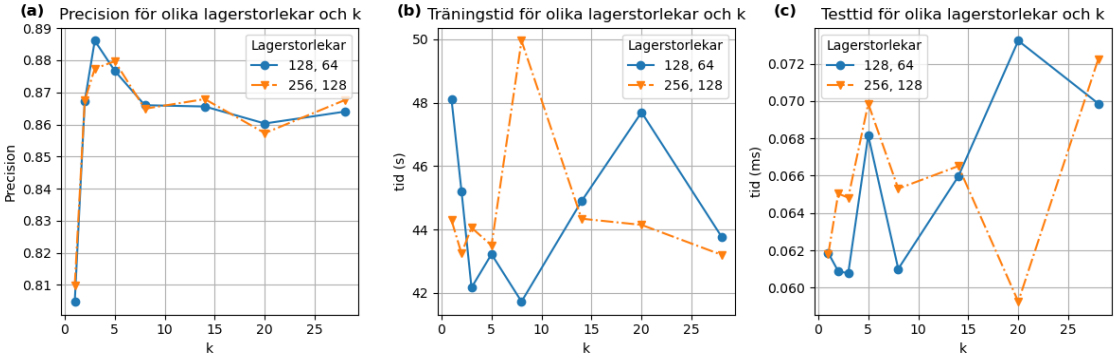
Figur 14: a) Precision, b) träningstid [s], c) testtid [μs] för LNN med olika lagerstorlekar tränade på de k första vänstra egenvektorerna $\boldsymbol{U}_k$.



Figur 15: a) Precision, b) träningstid [s], c) testtid [μs] för LNN med olika lagerstorlekar tränade på de k första högra egenvektorerna $\boldsymbol{V}_k$.

För metoden med PCA utfört på varje bild enskilt, skapdes en poängmatris med dimensionen $28 \times k$.

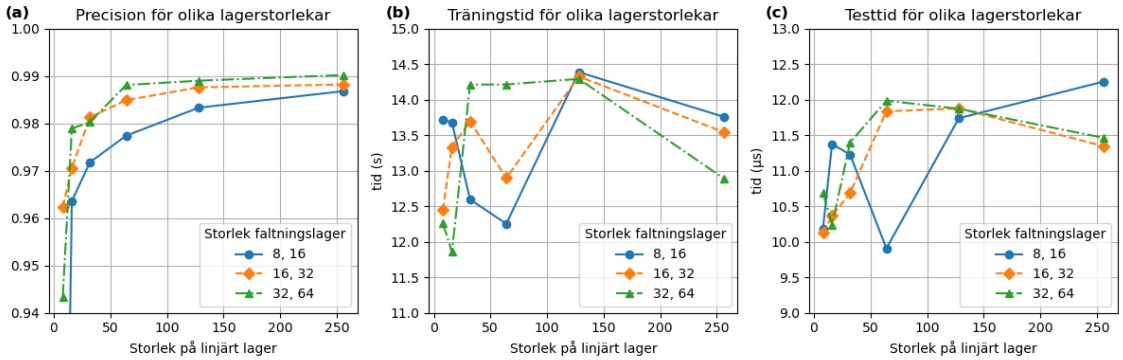
Poängmatrisen plattades ut till en vektor, med längden $28 \cdot k$, som utgjorde indatan till LNN. För metoden, vars resultat illustreras i Figur 16, var precisionen sämre än LNN på originalbilderna med värden på under 90%. På samma vis var både träningstiden och testtiden mycket längre.



Figur 16: a) Precision, b) träningstid [s], c) testtid [ms] för LNN med olika lagerstorlekar tränade på poängmatrisen från enskilda bilder med k principalkomponenter.

5.3 CNN

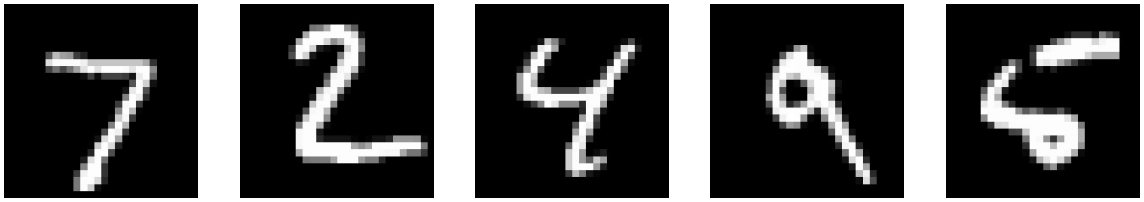
Det konvolutionella neurala nätverket bestod av två konvolutionella lager följt av två linjära lager. Det ingående lagret tog emot orginalbilden med dimensionen $28 \times 28 \times 1$ och producerade utdata som bestod av C antal kanaler, och andra lagret producerade $2 \cdot C$ kanaler. Dessa plattades sedan ut till en vektor för det första linjära lagret av storleken N. Nätverket testades för värdena $8, 16$ och 32 på variabeln C, samt $8, 16, 32, 64, 128$ och 256 för N. CNN-metoden, presenterad i Figur 17, hade vid de största lagerstorlekarna, $C = 32$ och $N = 256$ en precision på över 99%, vilket var den högsta av samtliga testade metoder. Träning- och testtiden för denna metod var däremot också en utav de längre bland metoderna med neurala nätverk. För de testade lagerstorlekarna syns en stor varians men generellt sätt ingen större skillnad på träningstid eller testtid.



Figur 17: a) Precision, b) träningstid [s], c) testtid [μs] för CNN med olika lagerstorlekar. Det övre och undre numret på x-axeln anger storleken på det första respektive andra gömda linjära lagret.

5.4 Illustration av tester

I denna del presenteras klassificering från de använda metoderna på fem slumpmässigt utvalda bilder från MNIST. Figur 18 visar att vissa metoder till exempel hade svårigheter att korrekt klassificera ett exempel på en otydligt skriven femma, medan ett tydligare exempel på en tvåa klassificerades rätt av samtliga metoder.



	(a)	(b)	(c)	(d)	(e)
PCA1	7	2	4	9	5
PCA2	7	2	4	**4**	**2**
LNN1	7	2	4	9	5
LNN2	7	2	4	9	**6**
LNN $\boldsymbol{V}_\mathrm{k}$	7	2	**9**	**4**	**8**
LNN $\boldsymbol{U}_\mathrm{k}$	7	2	4	**4**	**2**
LNN Poäng	7	2	4	9	**6**
CNN	7	2	4	9	5
Korrekt	7	2	4	9	5

Figur 18: Exempel på klassificeringar på 5 bilder från MNIST. PCA1 och PCA2 är klassificerade med 60000 bilder i träningsmängden och trunkerad till 30 respektive 10. LNN1 och LNN2 är tränade på orginalbilderna där de första lagerstorlekarna är 128 och 16 respektive. LNN $\boldsymbol{V}_\mathrm{k}$ är klassificerat av ett LNN tränat på $\boldsymbol{V}_\mathrm{k}$ med storleken 256 på första lagret och $k = 3$. LNN Poäng är ett LNN tränat på poängmatrisen med första lagerstorleken som 128 och $k = 30$. CNN är ett konvolutionsnätverk med storleken 128 på det linjära lagret och 16 på det första konvolutionslagret. Korrekt är siffran som bilden föreställer, med felaktiga klassificeringarna som fetmarkerade.

6 Diskussion

I denna del presenteras en analys och jämförelse av de olika metoderna utifrån de utförda testerna. Först diskuteras metoderna som enbart bygger på PCA, därefter analyseras LNN och dess hybridmetoder och till sist CNN. Slutligen jämförs samtliga metoder och slutsatser dras. En metod som maximerar precisionen och minimerar träning- samt testtiden ansågs vara optimal.

6.1 PCA

För PCA var det tydligt att de första principalkomponenterna innehåller den viktigaste informationen. Denna slutsats går att dra både från att precisionen ökade snabbt vid låga k och långsamt vid höga k i Figur 10, men också då storleken på singulärvärdena, och därmed mängden ny information per principalkomponent, minskade snabbt i Figur 3 d). Att inkludera fler principalkomponenter än cirka 30 gav ingen förbättring för klassificeringen av siffrorna från MNIST, utan är oönskat då testtiden ökar. Större träningsmängd resulterade i högre precision enligt Figur 10, detta förväntades eftersom sannolikheten att en bild liknande testbilden återfinns i träningsdatan ökade. Testtiden ökade vid stora k linjärt i takt med antalet principalkomponenter och visas i Figur 10, något som går att förklaras med hjälp av (17) där poängvektorns dimension ökar linjärt med k. Av samma anledning är det även förväntat att testtiden ökar med ökat antal träningsbilder vilket också går att se i resulatet. Att ökningen inte var linjär vid små k kan möjligtvis förklaras med att andra hårdvarubegränsningar då hade en större inverkan på tiden. Metoden som reducerade träningsmängden i Figur 11 hade en längre träningstid i förhållande till den andra, men uppvisade däremot en lägre testtid vid vissa precisionsnivåer.

6.2 LNN

För LNN var de bästa metoderna med avseende på precision LNN på originalbild och på poängvektor, som visas i Figur 12 och 13. Dessa hade som bäst en precision på cirka 98%, och varierade beroende på storleken på de gömda lagrena och antalet principalkomponenter i poängvektorn. Det visade sig också att vinsten i att öka lagerstorleken minskade för att sedan nå en platå vid en storlek på cirka 128 och 64 neuroner i respektive gömd lager. Det var förväntat att test- och träningstiden skulle öka med lagerstorlek och poängvektorns storlek då antalet beräkningar som måste göras

ökar med fler neuroner och principalkomponenter. Träningstiden och testtiden för dessa visade dock endast en svag ökning med ökande lagerstorlek och antal principalkomponenter.

Olika anledningar till varför ökningen inte var större kan vara bland annat att grafikkortet gjorde beräkningarna parallellt eller att det fanns någon annan faktor eller hårdvarubegränsning med större påverkan på tiden. Det är troligt att både lagerstorleken och antalet principalkomponenter hade haft en större påverkan om den bild som skulle klassificeras hade haft större dimensioner. Detta då tiden det hade tagit för datorn att utföra beräkningarna i så fall hade utgjort en större andel, och den slumpmässiga faktorn hade utgjort en mindre del. Det är också möjligt att det finns ett tröskelvärde som beror på graffikkortet där alla beräkningar inte längre kan ske parallellt och efter tröskelvärdet kan det finnas en skillnad i tidsåtgång. För att säkerställa varför lagerstorleken hade så liten påverkan och om påverkan är större för bilder med fler pixlar hade ytterligare test behövt utföras.

De andra testade metoderna med LNN gav avsevärt sämre precision och ingen förbättring av varken träning- eller testtid. En orsak till varför dessa presterade sämre skulle kunna vara att dimensionsreduceringen i dessa metoder gjordes individuellt på alla bilderna. Låt principalkomponenterna representera olika egenskaper i bilderna. Då PCA genomförs enskilt för varje bild kommer egenskapernas ordning sorteras efter singulärvärdenas storlek för den specifika bilden. I och med att olika egenskaper kommer ha olika stor betydelse för alla bilder så kommer också singulärvärdenas storlek och därmed egenskapernas ordning att variera. På grund av detta kan variablerna i ingångslagret till det linjära nätverket representera olika egenskaper beroende på vilken bild som testas. Detta förklarar varför precisionen för metoderna blev sämre, till skillnad från poängmatrisen och originalbilden där varje variabel alltid representerade samma egenskap.

6.3 CNN

Det konvolutionella nätverket presterade bäst av samtliga metoder med avseende på precisionen. Nätverket klassificerade 99% av testbilderna korrekt med tillräckligt stora gömda lager. Likt de linjära nätverken nådde precisionen en platå vid en storlek på cirka 128 neuroner i det gömda linjära lagret. Platån indikerar att det finns ett maximalt antal neuroner som kan ha en effekt på nätverket och om nätverket är överdimensionerat kommer överflödiga neuroner att få en väldigt liten påverkan och vikter nära noll. Vinsten i att öka storleken på konvolutionslagret var på samma sätt störst i början för att sedan minska och det är troligt att även denna var nära att nå en platå vid en storlek på cirka 32 kanaler. Att CNN lyckades så bra kan bero på att filtrerna har förmågan att detektera mönster och egenskaper, som bland annat kanter och hörn i bilderna, se sektion 3.3.4. Tidsåtgången för CNN var lite större än för LNN på den obehandlade bilden, både vad det gällde träningstid och testtid. Skillnaden var däremot relativt liten då det rörde sig om en ökning på träningstid från cirka 10 sekunder till 13 sekunder och en ökning av testtid från cirka 9 μs till cirka 11.5 μs.

6.4 Jämförelse

De testade metoderna visade alla olika styrkor och svagheter och den metod som bör användas beror därför på tillämpningsområdet. Överlag visade det konvolutionella nätverket den bästa precisionen för bildigenkänning av siffror i MNIST och då den inte heller visade någon större ökning i tidsåtgång jämfört med LNN är detta troligen den metod som är lämpligast för bildklassificering. LNN på både originalbild och poängmatris från PCA var också väl fungerande metoder med en precision som var något lägre än CNN. De hade dessutom något snabbare träning- och testtider. Användning av PCA för klassificering av bilder visade sig ha god prestanda vad gäller träningstid, men med en precision som var något sämre än CNN och LNN-metoderna. PCA hade mycket snabbare träningstider, upp till 50 gånger, dock också mycket långsammare testtider, cirka 30 gånger, jämfört med LNN-metoderna.

Metoderna med LNN på $\mathbf{U_k}$ och $\mathbf{V_k}$ gav ej bra resultat ur någon aspekt och de bör därför inte användas för bildklassificering. Det bör också kommenteras att resultaten i rapporten är kraftigt

påverkade av den hårdvara och de algoritmer som användes. Bland annat kunde träningen och testningen av NN göras med hjälp av ett grafikkort, vilket kraftigt minskade tiden för dessa beräkningar. Vid avsaknad av ett grafikkort hade resultaten för dessa troligtvis blivit långsammare. För PCA användes jämförelsevis bara processorn. Algoritmen för beräkningen av de reducerade datamängderna för PCA (se avsnitt 4.3) är inte heller optimal och det är troligt att tiden för denna metod hade kunnat minskas om en bättre algoritm utvecklades.

Slutligen har förmodligen även klassificeringsuppgiften stor påverkan på vilken metod som fungerade bäst. I denna undersökning testades metoderna enbart på ett dataset, MNIST, vilket är ett relativt enkelt dataset jämfört med vad som finns i många andra tillämpningsområden. För att tillämpa metoderna på mer komplexa uppgifter, exempelvis identifiering av hudcancer, skulle metoderna för varje bild behöva klara av att hantera miljontals pixlar och flera färgkanaler. En intressant uppföljande studie skulle därför vara att se om resultaten fortfarande håller för större bilder. Det skulle även vara lämpligt att undersöka om dimensionsreducering med PCA kan ge en större vinst för stora bilder. I en sådan undersökning hade det även kunnat testas om GPU-acceleration hade kunnat implementeras för PCA-metoden.

7 Samhälleliga och etiska aspekter

Datorseende har en stor potential att hjälpa mänskligheten i vår allt mer datoriserade och automatiserade värld [32]. Att låta datorer ta över monotona, tråkiga eller svåra uppgifter kommer kunna göra samhället bättre, men vi måste vara försiktiga i vår jakt på framtiden. Det kommer även med risker och faror när man ger vissa arbetsuppgifter till maskiner.

Anonymitet och integritet är viktigt för åtskilliga människor, att kunna ha sin identitet och vardag skild från omvärlden värderas högt. Med bildigenkänning som är tillräckligt kompetent finns möjligheten att identifiera individer från bilder och videor. Det kan vara väldigt användbart, som i fallet med mobiltelefoner som kan låsas upp genom att bara visa sitt ansikte, men det kan också användas på kränkande vis. Till exempel finns det möjlighet för ett rättsväsende att skapa ett nätverk av övervakningskameror i samhället som konstant filmar gator och torg, de kan med bildigenkänning spåra hur individer rör sig och interagerar med sin omgivning. Detta i syfte att finna kriminalitet med ändamålet att göra samhället tryggare. Ett konstant övervakande tar dock bort människors anonymitet och integritet. Det är viktigt skapa lagstiftning och praxis som säkerställer individens trygghet och integritet i fortsatt utveckling av bildigenkänning.

En tillämpning av AI är automatisering av uppgifter [33]. Det kan handla både om uppgifter som människor kan genomföra, men även lösa komplexa problem som tidigare varit svåra att lösa. Positiva konsekvenser kan vara att ge människor mer fritid och bättre användning av vardagen samt att med hjälp av bildigenkänning kunna lösa problem som varit ytterst invecklade eller omöjliga tidigare. Ett exempel på jobb som redan kan utföras av människor och som kan försvinna till konsekvens av automatisering är vägtransport. Denna form av automatisering kan ersätta många människor samt skapa sociala och ekonomiska svårigheter.

Referenser

[1] Lunds Universitet, *Allt du skulle vilja veta om AI | Lunds universitet*. URL: https://www.lu.se/artikel/allt-du-skulle-vilja-veta-om-ai (hämtad 2023-05-09).

[2] F. Hutmacher, "Why Is There So Much More Research on Vision Than on Any Other Sensory Modality?", *Frontiers in Psychology*, årg. 10, s. 2246, okt. 2019. DOI: 10.3389/fpsyg.2019.02246.

[3] A. C. Clarke, "Clarke's third law on ufo's", en, *Science*, årg. 159, nr 3812, s. 255–255, jan. 1968. DOI: 10.1126/science.159.3812.255.c.

[4] C. C. Aggarwal, *Neural networks and deep learning: a textbook*. Cham, Switzerland: Springer, 2018. DOI: https://doi.org/10.1007/978-3-319-94463-0.

[5] C. Pelau, D.-C. Dabija och I. Ene, "What makes an AI device human-like? The role of interaction quality, empathy and perceived psychological anthropomorphic characteristics in the acceptance of artificial intelligence in the service industry", *Computers in Human Behavior*, årg. 122, s. 106 855, 2021.

[6] C. H. Chen, *Handbook of pattern recognition and computer vision*. World Scientific, 2015.

[7] S. Russell och P. Norvig, *Artificial Intelligence: A Modern Approach*, 4. utg. Prentice Hall, 2020.

[8] H. Duan, R. Yan och K. Lin, "Research on Face Recognition Based on PCA", s. 29–32, 2008. DOI: 10.1109/FITME.2008.115.

[9] Y. Tian, "Artificial Intelligence Image Recognition Method Based on Convolutional Neural Network Algorithm", *IEEE Access*, årg. 8, s. 125 731–125 744, 2020. DOI: 10.1109/ACCESS.2020.3006097.

[10] V. Vučković, "Image and its Matrix, Matrix and its Image", *(NCD Review)*, årg. 12, s. 17–31, 2008.

[11] J. Wäldchen och P. Mäder, "Machine learning for image based species identification", *Methods in Ecology and Evolution*, årg. 9, nr 11, s. 2216–2225, 2018. DOI: https://doi.org/10.1111/2041-210X.13075.

[12] L. G. Roberts, "Machine perception of three-dimensional solids", diss., Massachusetts Institute of Technology, 1963.

[13] M. Turk och A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, årg. 3, nr 1, s. 71–86, 1991. DOI: 10.1162/jocn.1991.3.1.71.

[14] A. Pannu, "Artificial intelligence and its application in different areas", *Artificial Intelligence*, årg. 4, nr 10, s. 79–84, 2015.

[15] A. Bergström, "Personligt och privat i sociala medier", i *rapport nr.: SOM-rapport*, 50, 2010.

[16] J. Qiu, Q. Wu, G. Ding, Y. Xu och S. Feng, "A survey of machine learning for big data processing", *EURASIP Journal on Advances in Signal Processing*, årg. 2016, s. 1–16, 2016. DOI: 10.1186/s13634-016-0355-x.

[17] H. Fujita, "AI-based computer-aided diagnosis (AI-CAD): the latest review to read first", *Radiological physics and technology*, årg. 13, nr 1, s. 6–19, 2020. DOI: 10.1007/s12194-019-00552-4.

[18] D. F. Engstrom, D. E. Ho, C. M. Sharkey och M.-F. Cuéllar, "Government by algorithm: Artificial intelligence in federal administrative agencies", *NYU School of Law, Public Law Research Paper*, nr 20-54, 2020.

[19] H. Fujiyoshi, T. Hirakawa och T. Yamashita, "Deep learning-based image recognition for autonomous driving", *IATSS Research*, årg. 43, nr 4, s. 244–252, 2019. DOI: https://doi.org/10.1016/j.iatssr.2019.11.008.

[20] H. Bhardwaj, P. Tomar, A. Sakalle och U. Sharma, "Chapter 20 - Principles and Foundations of Artificial Intelligence and Internet of Things Technology", i *Artificial Intelligence to Solve Pervasive Internet of Things Issues*, G. Kaur, P. Tomar och M. Tanque, utg., Academic Press, 2021, s. 377–392. DOI: https://doi.org/10.1016/B978-0-12-818576-6.00020-4.

[21] M. Kubat, *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2017. DOI: 10.1007/978-3-319-63913-0.

[22] R. Fisher, S. Perkins, A. Walker och E. Wolfart, "Hypermedia image processing reference", *England: John Wiley & Sons Ltd*, s. 118–130, 1996.

[23] L. Deng, "The mnist database of handwritten digit images for machine learning research", *IEEE Signal Processing Magazine*, årg. 29, nr 6, s. 141–142, 2012.

[24] *PyTorch*. URL: https://www.pytorch.org (hämtad 2023-04-29).

[25] S. L. Brunton och J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[26] L. Beilina, E. Karchevskii och M. Karchevskii, *Numerical linear algebra: theory and applications*. New York, NY: Springer Berlin Heidelberg, 2017.

[27] I. T. Jolliffe, *Principal Component Analysis*. Springer New York, NY, 2002.

[28] C. M. Bishop och N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[29] A. Obuchowski. "Understanding neural networks 2: The math of neural networks in 3 equations". (2020), URL: https://becominghuman.ai/understanding-neural-networks-2-the-math-of-neural-networks-in-3-equations-6085fd3f09df.

[30] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge: Cambridge University Press, 2021. DOI: 10.1017/9781108860604.

[31] A. Edegran, B. Fazlija, J. Carlsson, E. Arvidsson och H. Björck, *kandidatarbete-ML*, version 1.0.0, maj 2023. URL: https://github.com/albined/kandidatarbete-ML.

[32] *Datorseende – en snabb och pålitlig superkraft*. URL: https://www.ri.se/sv/berattelser/datorseende-en-snabb-och-palitlig-superkraft (hämtad 2023-05-09).

[33] R. Acemoglu, "Artifi cial Intelligence, Automation, and Work", *University of Chicago Press*, s. 197–236, 2019.

A Appendix – Ordlista

Bildigenkänning En dators förmåga att klassificera bilderkänning. Finns många metoder som används för bildigen, bland annat PCA och NN.

Bit En binär enhet som kan ha ett av två möjliga värden, 0,1. Används för att representera information i dotorn genom binär kod.

Byte Är en enhet inom datorkunkap som består av 8 bitar. Används av oss mäniskor för att förstå information.

CNN Konvolutionellt nätverk (faltningsnätverk), mer komplicerad typ av neuralt nätverk som innehåller minst ett konvolutionellt lager (*eng. convolutional neural network*).

Filter Mindre matris som förs över större matris, för att generera utgående data. Kan användas inom konvolutionella lager, maxpooling, etc (*eng. kernel*).

Flatten En matris plattas ut till en endimensionell vektor genom att raderna i matrisen läggs efter varandra.

Flyttal Datatyp, är en beteckning inom datorprogramering och matematik för att representera decimaltal med hög precision (*eng. foating-point number*).

Fullt sammankopplade lager Linjära lager som ligger efter varandra i neurala nätverk, där varje neuron i ett lager är kopplad till varje neuron i nästkommande lager (*eng. fully connected layer*).

LNN Linjärt neuralt nätverk, enklaste typen av neuralt nätverk med linjära lager (*eng. linear neural network*).

Maxpooling En metod för att minska ner storleken på en matris. Ett filter förs över matrisen och största värdet från filterområdet tas vidare till utdatan.

NN Neuralt nätverk, en maskininlärningsmetod för klassificering som är inspirerad av hjärnans neuroner (*eng. neural network*).

Padding Antalet pixlar med värde noll som läggs till i kanten på en bild innan en operation med ett filter. Används innom konvolutionellet lager, maxpooling, etc.

PCA Principalkomponentanalys, en dimentionsreducerande metod som kan användas för bildklassificering och klustering (*eng. principal component analysis*).

PK Principalkomponent, en linjär kombination som beskriver en viss variation i en datamängd, byggstenar i PCA (*eng. principal component*).

Poängmatris Projektionen av en matris på dess principalkomponenter från PCA (*eng. score matrix*).

Poängvektor Projektionen av en vektor på principalkomponenterna från PCA (*eng. score vector*).

Stride Steglängden för varje operation av ett filter, vilket varierar beroende på filtret. Används i konvolutionellt lager eller maxpooling.

SVD Singulärvärdesuppdelning, en typ av matrisfaktorisering vanlig inom matematiken (*eng. singular value decomposition*).

B Appendix – Teori

B.1 Singulärvärdesuppdelning

Låt $\mathbf{A}$ vara en godtycklig matris sådan att $\mathbf{A} \in \mathbb{R}^{m \times n}$ där $m \geq n$. Då kan dess singulärvärdesuppdelning skrivas som

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\boldsymbol{V}^T,$$

där $\mathbf{U} \in \mathbb{R}^{m \times n}$ och $\mathbf{V}^T \in \mathbb{R}^{n \times n}$ är ortogonala matriser, vilket betyder att $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ och $\mathbf{V}\mathbf{V}^T = \mathbf{I}$ där $\mathbf{I}$ är identitetsmatrisen. $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ är en diagonalmatris sådan att

$$\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n), \quad \text{där} \quad \sigma_1 \geq \sigma_2 \geq, \ldots, \geq \sigma_n \geq 0,$$

Då gäller följande satser [26]:

Sats 2.1 *Egenvärdena av den symmetriska matrisen A^TA är σ^2. Kolumnerna i matrisen $V = (v_1, v_2, \ldots, v_n)$ är motsvarande egenvektorer.*

Bevis. $A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma U^T U\Sigma V = V\Sigma^2 V^T$. Detta är ett egenvärdesproblem, där σ^2 är egenvärden och v_i är tillhörande ortonomerad egenvektor. □

Sats 2.2 *Egenvärdena av den symmetriska matrisen AA^T är σ^2 och m - n nollor. Kolumnerna i matrisen $U = (u_1, u_2, \ldots, u_n)$ är motsvarande egenvektorer.*

Bevis. Välj en godtycklig matris $\widetilde{U}$ sådan att $\dim(\tilde{U}) = m \times (m - n)$, så att $[U, \widetilde{U}]$ är kvadratisk och ortognonal. Då gäller följande

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T = \begin{bmatrix} U & \widetilde{U} \end{bmatrix} \cdot \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} U & \widetilde{U} \end{bmatrix}^T$$

Detta är ett egenvärdesproblem, där σ^2 är egenvärden och u_i är motsvarande ortonomerad egenvektor. □

Sats 2.3 *Om A har full rang, är $x = V\Sigma^{-1}U^Tb$ lösningen till $\min_x \|Ax - b\|_2$.*

Bevis. $\|Ax - b\|_2^2 = \|U\Sigma V^T x - b\|_2^2$. Då A och Σ har full rang. Låt $\tilde{U}$ vara en matris så att $\begin{bmatrix} U, \widetilde{U} \end{bmatrix}$ är kvadratisk och ortogonal.

$$\begin{aligned} \|U\Sigma V^T x - b\|_2^2 &= \left\| \begin{bmatrix} U^T \\ \widetilde{U}^T \end{bmatrix} (U\Sigma V^T x - b) \right\|_2^2 = \left\| \begin{bmatrix} \Sigma V^T x - U^T b \\ -\widetilde{U}^T b \end{bmatrix} \right\|_2^2 \\ &= \|\Sigma V^T x - U^T b\|_2^2 + \|\widetilde{U}^T b\|_2^2 \end{aligned}$$

För att minimera uttrycket ovan måste $\|\Sigma V^T x - U^T b\|_2^2 = 0$, vilket gäller för $x = V\Sigma^{-1}U^Tb$. □

Sats 2.4 *Låt $U = \begin{bmatrix} u_1, u_2, \ldots, u_n \end{bmatrix}$ och $V = \begin{bmatrix} v_1, v_2, \ldots, v_n \end{bmatrix}$ så $A = U\Sigma V^T = \sum_{i=1}^n \sigma_i u_i v_i^T$. Då är matrisen, av rang $k < n$, $A_k = \sum_{i=1}^k \sigma_i u_i v_i^T$ närmast A och $\|A - A_k\|_2 = \sigma_{k+1}$.*

Bevis. Normen av skillnaden mellan matriserna A och A_k kan skrivas enligt

$$\begin{aligned}\|A - A_k\|_2 &= \left\| \sum_{i=1}^{n} \sigma_i u_i v_i^T - \sum_{i=1}^{k} \sigma_i u_i v_i^T \right\|_2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \right\|_2 \\ &= \left\| U \begin{bmatrix} 0 & & & \\ & \sigma_{k+1} & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} V^T \right\|_2 = \sigma_{k+1}\end{aligned}$$

Det återstår att visa att det inte finns en matris av rang k som är närmre A än A_k. Låt B vara en godtycklig rang k matris, så att dess nollrum har dimensionen $n - k$. Rummet spänns upp av $\{v_1, \ldots, v_{k+1}\}$ och har dimensionen $k + 1$. Då summan av dimensionerna för nollrummet och rummet är $(n - k) + (k + 1) > n$, måste dessa överlappa varandra. Låt h vara en enhetsvektor i skärningspunkten. Då

$$\begin{aligned}\|A - B\|_2^2 &\geq \|(A - B)h\|_2^2 = \|Ah\|_2^2 = \left\|U\Sigma V^T h\right\|_2^2 = \left\|\Sigma(V^T h)\right\|_2^2 \\ &\geq \sigma_{k+1}^2 \left\|\Sigma V^T h\right\|_2^2 = \sigma_{k+1}^2\end{aligned}$$

□

B.2 Algoritmer för beräkning av SVD

Jacobialgoritmen är en iterativ numerisk metod beräkning för av SVD och bygger på en mer generell Jacobimetod som med hjälp av Jacobi-rotation stegvis diagonaliserar en matris A till en matris med egenvärdena av A på diagonalen [26]. Mer precist så skapar Jacobimetoden en serie $A_i, i = 0, \ldots, m$ ortogonalt lika matriser som konvergerar till en diagonalmatris A_m med egenvärdena till A på diagonalen. Matriserna A_i kan stegvis beräknas med ekvationen

$$A_{i+1} = J_i^T A_i J_i \tag{26}$$

där matrisen J_i är en speciell Jacobi-rotationsmatris. Om steget upprepas erhålls

$$A_m = J_{m-1}^T A_{m_1} J_{m-1} = J_{m-1}^T J_{m-2}^T \ldots J_0^T A_{m_1} J_0 \ldots J_{m-2} J_{m-1} = J^T A J. \tag{27}$$

Genom att välja J_i så att A_m konvergerar till en diagonal matris Λ för stora m kan ekvationen skrivas

$$\Lambda \approx J^T A J \tag{28}$$

$$J \Lambda J^T \approx A \tag{29}$$

varifrån det går att se att J och J^T approximerar egenvektorer till A. För att ovanstående ska stämma krävs det att J_i väljs så att matrisen iterativt blir mer diagonal. Detta görs genom att låta J_i vara en Givens-rotationsmatris där värdet på θ väljs så att $A_{j,k}$ och $A_{k,j}$ i matrisen A_{i+1} blir 0. Rotationsmatriserna har formen

$$J_i = R(j,k,\theta) \equiv \begin{array}{c} \\ \\ j \\ \\ k \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & j & & k & & & \end{array} \\ \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\theta & & -\sin\theta & & & \\ & & & & \ddots & & & & \\ & & & \sin\theta & & \cos\theta & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix} \end{array}. \tag{30}$$

Det går att visa att $\sin\theta$ och $\cos\theta$ kan beräknas med hjälp av följande ekvationer:

$$\tau = \frac{a_{jj}^i - a_{kk}^i}{2a_{jk}^i} \tag{31}$$

$$t = \frac{\operatorname{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}} \tag{32}$$

$$c = \cos\theta = \frac{1}{\sqrt{1+\tau^2}} \tag{33}$$

$$s = \sin\theta = tc \tag{34}$$

För härledningar av ovanstående ekvationer se [26]. Det kan även visas att matrisen A blir mer diagonaliserad för varje iteration med olika $(j,\ k)$ vilket i sin tur innebär att vi kan upprepa proceduren för olika $(j,\ k)$ tills det att A är tillräckligt diagonal. Ett exempel på en algoritm som uppnår detta är algoritm 1:

Algoritm 1 Ensidig-Jacobi

function ONE-SIDED-JACOBI-ROTATION(G, j, k)
  Compute $a_{jj} = (G^TG)_{jj}$, $a_{jk} = (G^TG)_{jk}$ and $a_{kk} = (G^TG)_{kk}$
  $\tau \leftarrow \dfrac{a^i_{jj} - a^i_{kk}}{2a^i_{jk}}$
  $t \leftarrow \dfrac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$
  $c \leftarrow \dfrac{1}{\sqrt{1+\tau^2}}$
  $s \leftarrow tc$
  $G \leftarrow G\,R(j,k,\theta)$ ▷ θ väljs så att $c = \cos\theta$ och $s = \sin\theta$
  $J \leftarrow J\,R(j,k,\theta)$
end function

while G^TG not diagonal enough **do**
  for $j = 1$ to $n-1$ **do**
    for $k = j+1$ to n **do**
      ONE-SIDED-JACOBI-ROTATION(G, j, k)
    end for
  end for
end while
$\sigma_i \leftarrow \|G(:,i)\|_2$ ▷ 2-normen av kolumn i i G
$U \leftarrow [u_1, \ldots, u_n]$ where $u_i = G(:,i)/\sigma_i$
$V \leftarrow J$ ▷ produkten av Jacobirotationerna

Algoritm 1 förutsätter att vi har en symmetrisk matris $A = G^TG$ och att G är av storleken $n \times n$. Algoritmen beräknar singularvärdena σ_i, den vänstra singularvektormatrisen U och den högra singularvektormatrisen V så att $G = U\Sigma V^T$ där $\Sigma = \text{diag}(\sigma_\text{i})$. Den gör detta genom att iterativt gå igenom olika par $(j,\ k)$ tills det att matrisen är tillräckligt diagonal. Med att vara tillräckligt diagonal menas det att de värdena i matrisen som inte ligger på diagonalen är försumbart små. Det går att visa att algoritm 1 kan beräkna SVD med hög precision. Se [26] för sats och bevis på detta.

C Appendix – Källkod

All källkod finns även tilgänglig på github [31].

C.1 Bildkomprimering

```
import os
from PIL import Image
import numpy as np
import extra_filer.extra_tools as tools


def read_image(path='zelda.png'):
    img = Image.open(path)
    img_array = np.array(img)
    return img_array


def compress_image(path, k):
    original_image = read_image(path)
    SVD_list = [np.linalg.svd(original_image[:, :, i], full_matrices=
        False) for i in range(3)]

    # Separate the U, S, and Vh components into their own lists
    U_list, S_list, Vh_list = list(map(list, zip(*SVD_list)))
    for idx, (u, s, v) in enumerate(zip(U_list, S_list, Vh_list)):
        U_list[idx] = u.astype(np.float32)
        S_list[idx] = s.astype(np.float32)
        Vh_list[idx] = v.astype(np.float32)

    # Multiply together only the k first singular values and vectors
        components to create a new image
    compressed_svd_list = [np.dot(U_list[i][:, :k] * S_list[i][:k],
        Vh_list[i][:k, :]) for i in range(3)]

    # Combine the color channels again and save it as a new image
    # Also clip the colors to 0 to 255 to prevent them from turning
        into a rainbow color.
    compressed_img_array = np.dstack(compressed_svd_list).clip(0, 255).
        astype('uint8')
    compressed_img = Image.fromarray(compressed_img_array)
    save_path = os.path.join(tools.get_project_root(), '
        svd_image_compression', 'compressed_images')
    name = os.path.basename(path)
    if not os.path.isdir(save_path):
        os.makedirs(save_path)
    save_name = os.path.join(save_path, f'compressed_{name.replace(".
        jpg", "").replace(".png", "")}_k({k}).png')
    compressed_img.save(save_name)

    return save_name
```

C.2 Klassificering med PCA

```
import numpy as np
from abc import ABC, abstractmethod
import extra_filer.extra_tools as tools
```

```
4  from sklearn.decomposition import PCA
5  import time
6  import os
7  import pandas as pd
8
9  tl = tools.EasyTimeLog()
10
11
12 class GeneralPCA(ABC):
13     """
14     Defines the structure and what methods a PCA calculating class must
           contain
15     """
16
17     def __init__(self, training_data, testing_data):
18         self.training_data = training_data
19         self.testing_data = testing_data
20         self.center_data()
21
22     @abstractmethod
23     def get_training_components(self, k):
24         pass
25
26     @abstractmethod
27     def get_testing_components(self, k):
28         pass
29
30     def get_components(self, k):
31         training_comp, training_time = self.get_training_components(k)
32         testing_comp, testin_time = self.get_testing_components(k)
33         return training_comp, training_time, testing_comp, testin_time
34
35     def __len__(self):
36         return len(self.testing_data)
37
38     def center_data(self):
39         mean = self.training_data.mean(axis=0)
40         self.training_data = self.training_data - mean
41         self.testing_data = self.testing_data - mean
42         total_mean = self.training_data.mean()
43         total_std = self.training_data.std()
44         self.training_data = (self.training_data - total_mean) /
               total_std
45         self.testing_data = (self.testing_data - total_mean) /
               total_std
46
47
48 class CovariancePCA(GeneralPCA):
49     """
50     Calculates PCA using SVD and a covariance matrix
51     """
52
53     def __init__(self, training_data, testing_data):
54         tt = tl.start_log(f'training_size: {len(training_data)}')
55         super().__init__(training_data, testing_data)
56         C = np.cov(training_data, rowvar=False)
```

```
57          self.V, _, _ = np.linalg.svd(C)
58          tt.stop()
59          self.initial_training_time = tt.time()
60
61      def get_training_components(self, k):
62          tt = tl.start_log(f'components: {k}, training_size: {len(self.
                training_data)}')
63          comp = self.training_data @ self.V[:, :k]
64          tt.stop()
65          return comp, tt.time() + self.initial_training_time
66
67      def get_testing_components(self, k):
68          tt = tl.start_log(f'components: {k}, testing_size: {len(self.
                testing_data)}')
69          comp = self.testing_data @ self.V[:, :k]
70          tt.stop()
71          return comp, tt.time()
72
73
74  class SvdPCA(GeneralPCA):
75      """
76      Calculates PCA using SVD on the original matrix
77      """
78
79      def __init__(self, training_data, testing_data):
80          tt = tl.start_log(f'training_size: {len(training_data)}')
81          super().__init__(training_data, testing_data)
82          U, S, Vt = np.linalg.svd(self.training_data)
83          self.U = U
84          self.V = Vt.T
85          self.S = S
86          tt.stop()
87          self.initial_training_time = tt.time()
88
89      def get_training_components(self, k):
90          tt = tl.start_log(f'components: {k}, training_size: {len(self.
                training_data)}')
91          # Uk = self.U[:, :k]
92          # Sk = np.diag(self.S[:k])
93          comp = self.training_data @ self.V[:, :k]
94          tt.stop()
95          return comp, tt.time() + self.initial_training_time
96
97      def get_testing_components(self, k):
98          tt = tl.start_log(f'components: {k}, testing_size: {len(self.
                testing_data)}')
99          comp = self.testing_data @ self.V[:, :k]
100         tt.stop()
101         return comp, tt.time()
102
103
104 class SklearnPCA(GeneralPCA):
105     """
106     Calculates PCA using the existing library sklearn
107     """
108
```

```
109     def __init__(self, training_data, testing_data):
110         tt = tl.start_log(f'training_size: {len(training_data)}')
111         super().__init__(training_data, testing_data)
112         tt.stop()
113 
114     def get_training_components(self, k):
115         tt = tl.start_log('TRAINING TIME', f'components: {k}, 
                training_size: {len(self.training_data)}')
116         pca = PCA(k)
117         pca.fit(self.training_data)
118         tt.stop()
119         tt = tl.start_log(f'components: {k}, training_size: {len(self.
                training_data)}')
120         comp = pca.transform(self.training_data)
121         tt.stop()
122         return comp, tt.time()
123 
124     def get_testing_components(self, k):
125         tt = tl.start_log('TRAINING TIME', f'components: {k}, 
                training_size: {len(self.training_data)}')
126         pca = PCA(k)
127         pca.fit(self.training_data)
128         tt.stop()
129         tt = tl.start_log(f'components: {k}, training_size: {len(self.
                training_data)}')
130         comp = pca.transform(self.training_data)
131         tt.stop()
132         return comp, tt.time()
133 
134     def get_components(self, k):
135         tt = tl.start_log('TRAINING TIME', f'components: {k}, 
                training_size: {len(self.training_data)}')
136         pca = PCA(k)
137         pca.fit(self.training_data)
138         comp_training = pca.transform(self.training_data)
139         tt.stop()
140         t1 = tt.time()
141 
142         tt = tl.start_log(f'testing testing', f'components: {k}, 
                training_size: {len(self.training_data)}')
143         comp_testing = pca.transform(self.testing_data)
144         tt.stop()
145         return comp_training, t1, comp_testing, tt.time()
146 
147 
148 class LoadPCA(GeneralPCA):
149     """
150     Laods a dataset in the same format as you'd get if you computed it
151     """
152 
153     def __init__(self):
154         """ Class for loading in previously calculated files returns 0 
                for the times """
155         super().__init__(np.array([0]), np.array([0]))
156 
157     def get_training_components(self, k):
```

```
158         train_array, _, _, _ = tools.load_pca_mnist(f'
                pca_transformed_MNIST_k={k}')
159         return train_array, 0
160
161     def get_testing_components(self, k):
162         _, _, test_array, _ = tools.load_pca_mnist(f'
                pca_transformed_MNIST_k={k}')
163         return test_array, 0
164
165     def get_components(self, k):
166         train_array, _, test_array, _ = tools.load_pca_mnist(f'
                pca_transformed_MNIST_k={k}')
167         return train_array, 0, test_array, 0
168
169
170 class ReducedSizePCA(GeneralPCA):
171     def __init__(self, pca: GeneralPCA, training_labels,
            reduction_factor=0.5):
172         super().__init__(pca.training_data, pca.testing_data)
173         self.pca = pca
174         self.training_labels = training_labels
175         self.new_training_labels = None
176         self.reduction_factor = reduction_factor
177
178     def get_training_components(self, k):
179         tt = tl.start_log(f'Reducing␣size␣from␣{len(self.training_data)
                }', f'reduction␣factor={self.reduction_factor}',
180                           f'k={k}')
181         training_components, training_time = self.pca.
                get_training_components(k)
182         new_components, new_labels = tools.reduce_pca_dataset(
                training_components, self.training_labels,
183                                                               self.
                                                                   reduction_factor
                                                                   )
184         self.new_training_labels = new_labels
185         tt.stop()
186         return new_components, tt.time() + training_time
187
188     def get_testing_components(self, k):
189         return self.pca.get_testing_components(k)
190
191     def get_new_labels(self):
192         return self.new_training_labels
193
194
195 def calculate_predictions(pca: GeneralPCA, labels_train, labels_test,
        k_list: list[int]):
196     accuracies = np.zeros(len(k_list))
197     training_times = np.zeros(len(k_list))
198     testing_times = np.zeros(len(k_list))
199     for i, k in enumerate(k_list):
200         training_scores, training_time, testing_scores,
                testing_preprocess_time = pca.get_components(k)
201         if isinstance(pca, ReducedSizePCA):
202             labels_train = pca.get_new_labels()  # if it is a reduced
```

```
                size pca then the labels need to be refreshed
203             print(f'k: {k}, len(training): {len(labels_train)}')
204             t0 = time.time()
205             for j, row in enumerate(testing_scores):
206                 if j % 200 == 0:
207                     print(j)
208                 distances = np.sum(np.square(training_scores - row), axis
                    =1)
209                 predicted_number = labels_train[np.argmin(distances)]
210                 if predicted_number == labels_test[j]:
211                     accuracies[i] += 1
212             testing_times[i] = time.time() - t0 + testing_preprocess_time
213             training_times[i] = training_time
214         accuracies /= len(pca)
215         testing_times /= len(pca)
216         return accuracies, training_times, testing_times
217
218
219     def test_pca(save=True, training_set_sizes=None, k_list=None):
220         # training_set_sizes = [-1, -2, -3, 1000, 5000, 10000, 30000,
                60000]
221         if training_set_sizes is None:
222             training_set_sizes = [999, 1000, 1001, 5000, 10000]
223         if k_list is None:
224             k_list = [1, 2, 3, 5, 8, 14, 20, 28, 64, 128]
225
226         accuracies_list = []
227         training_times_list = []
228         testing_times_list = []
229         test_data, test_labels = tools.load_mnist(False)
230         test_data = np.array(test_data)
231         test_labels = np.array(test_labels)
232         for i, size in enumerate(training_set_sizes):
233             if size == -1:
234                 train_data, train_labels = tools.load_mnist(True)
235                 train_data = np.array(train_data)
236                 train_labels = np.array(train_labels)
237                 pca = ReducedSizePCA(CovariancePCA(train_data, test_data),
                        train_labels, 0.5)
238             elif size == -2:
239                 train_data, train_labels = tools.load_mnist(True)
240                 train_data = np.array(train_data)
241                 train_labels = np.array(train_labels)
242                 pca = ReducedSizePCA(CovariancePCA(train_data, test_data),
                        train_labels, 0.75)
243             elif size == -3:
244                 train_data, train_labels = tools.load_mnist(True)
245                 train_data = np.array(train_data)
246                 train_labels = np.array(train_labels)
247                 pca = ReducedSizePCA(CovariancePCA(train_data, test_data),
                        train_labels, 1)
248             else:
249                 train_data, train_labels = tools.load_mnist(True, size)
250                 train_data = np.array(train_data)
251                 train_labels = np.array(train_labels)
252                 pca = CovariancePCA(train_data, test_data)
```

```
253             accuracy, train_time, test_time = calculate_predictions(pca,
                    train_labels, test_labels, k_list)
254             accuracies_list.append(accuracy)
255             training_times_list.append(train_time)
256             testing_times_list.append(test_time)
257 
258         if save:
259             save_dir = os.path.join(tools.get_project_root(), '
                    pca_image_classification', 'results')
260             if not os.path.exists(save_dir):
261                 os.makedirs(save_dir)
262 
263             pd.DataFrame(
264                 accuracies_list, columns=k_list, index=np.array(
                        training_set_sizes)).to_csv(
265                 os.path.join(save_dir, 'accuracy.csv'), sep=';')
266 
267             pd.DataFrame(
268                 training_times_list, columns=k_list, index=np.array(
                        training_set_sizes)).to_csv(
269                 os.path.join(save_dir, 'training_time.csv'), sep=';')
270 
271             pd.DataFrame(
272                 testing_times_list, columns=k_list, index=np.array(
                        training_set_sizes)).to_csv(
273                 os.path.join(save_dir, 'testing_time.csv'), sep=';')
274 
275 
276 if __name__ == '__main__':
277     test_pca(True)
```

C.3 Plottning av PCA klassificering

```
 1 import pandas as pd
 2 import os
 3 import extra_filer.extra_tools as tools
 4 import matplotlib.pyplot as plt
 5 
 6 
 7 def plot_results():
 8     line_styles = ['-', '--', '-.', ':', '',
 9                    '--', (0, (1, 1)), (0, (3, 1, 1, 1)), (0, (5, 1))]
10     marker_styles = ['o', 'D', '^', '+', '>', '1', '2', '3',
11                      '4', 's', 'p', '*', 'h', 'H', 'v', 'x', 'D', 'd']
12     testing_accuracy = pd.read_csv(
13         os.path.join(tools.get_project_root(), '
                pca_image_classification', 'results', f'accuracy.csv'), sep
               =';',
14         index_col=0)
15     testing_time = pd.read_csv(
16         os.path.join(tools.get_project_root(), '
                pca_image_classification', 'results', f'testing_time.csv'),
                 sep=';',
17         index_col=0)
18     training_time = pd.read_csv(
19         os.path.join(tools.get_project_root(), '
```

```
            pca_image_classification', 'results', f'training_time.csv')
            , sep=';',
20          index_col=0)
21
22      fig, axs = plt.subplots(1, 3, figsize=(12, 4))
23
24      x = [int(v) for v in testing_accuracy.columns]
25
26      special_labels = {
27          -1: 'reducerad, 0.5',
28          -2: 'reducerad, 0.75',
29          -3: 'reducerad, 1'
30      }
31      labels = [f'{n}' if n not in special_labels else special_labels[n]
            for n in testing_accuracy.index]
32
33      # Plotting the accuracy
34      for idx, line in testing_accuracy.reset_index().iterrows():
35          idx = int(idx)
36          line = line.values[1:]
37          axs[0].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
38      axs[0].set_ylabel('Precision')
39      # axs[0].set_ylim([0.5, 1])
40      axs[0].set_title('Precision f r olika stora tr ningsset och k')
41
42      # Plotting the training time
43      for idx, line in training_time.reset_index().iterrows():
44          idx = int(idx)
45          line = line.values[1:]
46          axs[1].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
47      axs[1].set_ylabel('tid (s)')
48      # axs[1].set_ylim([0, 4])
49      axs[1].set_title('Tr ningstid f r olika stora tr ningsset och k'
            )
50
51      # Plotting the testing time
52      testing_time = testing_time * 1000
53      for idx, line in testing_time.reset_index().iterrows():
54          idx = int(idx)
55          line = line.values[1:]
56          axs[2].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
57      axs[2].set_ylabel('tid (ms)')
58      # axs[2].set_ylim([0, 3])
59      axs[2].set_title('Testtid f r olika stora tr ningsset och k')
60
61      for ax in axs:
62          ax.set_xlabel('k')
63          ax.legend(labels, title='Tr ningsbilder')
64          ax.grid()
65
66      for i, ax in enumerate(axs):
67          ax.text(-0.14, 1.08, f"({chr(i + 97)})", transform=ax.transAxes
                ,
```

```
68                  fontweight='bold', fontsize=12, va='top', ha='left')
69
70      fig.tight_layout()
71      plt.show()
72
73
74  if __name__ == '__main__':
75      plot_results()
```

C.4 Generering av modifierade dataset för NN

```
1   from extra_filer.extra_tools import append_matrix_to_csv, load_mnist,
        EasyTimeLog
2   from sklearn.decomposition import PCA
3   import numpy as np
4   import os
5
6   timelog = EasyTimeLog()
7
8
9   def create_pca_library(train_set, label_train, test_set, label_test,
        principal_components, save_folder=None,
10                         dataset_name='', normalize=True):
11      # Define the pca, it's much easier if we just use a library
12      tl = timelog.start_log(f'PCA␣on␣the␣training␣set␣with␣{len(
            train_set)}␣images', f'k={principal_components}')
13      pca = PCA(n_components=principal_components)
14      pca.fit(train_set)
15
16      pca_train = pca.transform(train_set)
17      mean = 0
18      std = 1
19      if normalize:
20          mean = pca_train.mean()
21          std = pca_train.std()
22          pca_train = (pca_train - mean) / std
23      tl.stop()
24
25      tl = timelog.start_log(f'PCA␣projection␣of␣{len(test_set)}␣testing␣
            images', f'k={principal_components}')
26      pca_test = pca.transform(test_set)
27      if normalize:
28          print(f'normalizing␣mean:␣{mean},␣std:␣{std}')
29          pca_test = (pca_test - mean) / std
30      tl.stop()
31
32      if save_folder is not None:
33          append_matrix_to_csv(pca_train,
34                               f'pca_transformed_training_{dataset_name}
                                     _k={principal_components}.csv',
35                               relative_root=True,
36                               folder=os.path.join(save_folder,
37                                                   f'pca_transformed_{
                                                         dataset_name}_k={
                                                         principal_components
                                                         }'))
```

```
38          append_matrix_to_csv(pca_test,
39                               f'pca_transformed_testing_{dataset_name}_k
                                     ={principal_components}.csv',
40                               relative_root=True,
41                               folder=os.path.join(save_folder,
42                                                   f'pca_transformed_{
                                                         dataset_name}_k={
                                                         principal_components
                                                         }'))
43          append_matrix_to_csv(label_train,
44                               f'labels_svd_transformed_training_{
                                     dataset_name}_k={principal_components
                                     }.csv',
45                               relative_root=True,
46                               folder=os.path.join(save_folder,
47                                                   f'pca_transformed_{
                                                         dataset_name}_k={
                                                         principal_components
                                                         }'))
48          append_matrix_to_csv(label_test,
49                               f'labels_svd_transformed_testing_{
                                     dataset_name}_k={principal_components
                                     }.csv',
50                               relative_root=True,
51                               folder=os.path.join(save_folder,
52                                                   f'pca_transformed_{
                                                         dataset_name}_k={
                                                         principal_components
                                                         }'))
53
54
55  def create_svd_libaries(train_set, label_train, test_set, label_test,
        truncation: int, save_folder=None,
56                          dataset_name='', normalize=True):
57      U_train = np.zeros((train_set.shape[0], train_set.shape[1] *
            truncation))
58      Vt_train = np.zeros((train_set.shape[0], train_set.shape[2] *
            truncation))
59      k = truncation
60      tl = timelog.start_log(f'svd␣on␣the␣training␣set␣with␣{len(
            train_set)}␣images', f'k={k}')
61      Umean = 0
62      Vtmean = 0
63      Ustd = 1
64      Vtstd = 1
65      for i, img in enumerate(train_set):
66          u, _, vt = np.linalg.svd(img)
67          v = vt.T
68          u_k = u[:, :k]
69          v_k = v[:, :k]
70          U_train[i, :] = u_k.flatten()
71          Vt_train[i, :] = v_k.flatten()
72      if normalize:
73          Umean = U_train.mean()
74          Vtmean = Vt_train.mean()
75          Ustd = U_train.std()
```

```
76          Vtstd = Vt_train.std()
77          U_train = (U_train - Umean) / Ustd
78          Vt_train = (Vt_train - Vtmean) / Vtstd
79      tl.stop()
80
81      U_test = np.zeros((train_set.shape[0], train_set.shape[1] *
            truncation))
82      Vt_test = np.zeros((train_set.shape[0], train_set.shape[2] *
            truncation))
83      tl = timelog.start_log(f'svd␣on␣the␣testing␣set␣with␣{len(test_set)
            }␣images', f'k={k}')
84      for i, img in enumerate(test_set):
85          u, _, vt = np.linalg.svd(img)
86          v = vt.T
87          u_k = u[:, :k]
88          v_k = v[:, :k]
89          U_test[i, :] = u_k.flatten()
90          Vt_test[i, :] = v_k.flatten()
91      if normalize:
92          U_test = (U_test - Umean) / Ustd
93          Vt_test = (Vt_test - Vtmean) / Vtstd
94      tl.stop()
95
96      if save_folder is not None:
97          # Save for U
98          append_matrix_to_csv(U_train, f'U_SVD_transformed_training_{
                dataset_name}_k={k}.csv',
99                               relative_root=True,
100                              folder=os.path.join(save_folder, f'
                                    U_svd_transformed_{dataset_name}_k={k}
                                    '))
101         append_matrix_to_csv(U_test, f'U_SVD_transformed_testing_{
                dataset_name}_k={k}.csv',
102                              relative_root=True,
103                              folder=os.path.join(save_folder, f'
                                    U_svd_transformed_{dataset_name}_k={k}
                                    '))
104         append_matrix_to_csv(label_train, f'
                labels_SVD_transformed_training_{dataset_name}_k={k}.csv',
105                              relative_root=True,
106                              folder=os.path.join(save_folder, f'
                                    U_svd_transformed_{dataset_name}_k={k}
                                    '))
107         append_matrix_to_csv(label_test, f'
                labels_SVD_transformed_testing_{dataset_name}_k={k}.csv',
108                              relative_root=True,
109                              folder=os.path.join(save_folder, f'
                                    U_svd_transformed_{dataset_name}_k={k}
                                    '))
110         # Save for V
111         append_matrix_to_csv(Vt_train, f'Vt_SVD_transformed_training_{
                dataset_name}_k={k}.csv',
112                              relative_root=True,
113                              folder=os.path.join(save_folder, f'
                                    V_svd_transformed_{dataset_name}_k={k}
                                    '))
```

```
114         append_matrix_to_csv(Vt_test, f'Vt_SVD_transformed_testing_{
                dataset_name}_k={k}.csv',
115                              relative_root=True,
116                              folder=os.path.join(save_folder, f'
                                     V_svd_transformed_{dataset_name}_k={k}
                                     '))
117         append_matrix_to_csv(label_train, f'
                labels_SVD_transformed_training_{dataset_name}_k={k}.csv',
118                              relative_root=True,
119                              folder=os.path.join(save_folder, f'
                                     V_svd_transformed_{dataset_name}_k={k}
                                     '))
120         append_matrix_to_csv(label_test, f'
                labels_SVD_transformed_testing_{dataset_name}_k={k}.csv',
121                              relative_root=True,
122                              folder=os.path.join(save_folder, f'
                                     V_svd_transformed_{dataset_name}_k={k}
                                     '))
123
124
125 def pca_library_main(k, dataset_name='MNIST'):
126     n_test = -1  # all
127     pixels_train, labels_train = load_mnist(True, n_test)
128     pixels_test, labels_test = load_mnist(False, n_test)
129     create_pca_library(pixels_train, labels_train, pixels_test,
            labels_test, k,
130                        save_folder='Generated dataset variants',
                               dataset_name=dataset_name)
131
132
133 def svd_libraries_main(k, dataset_name='MNIST'):
134     n_test = -1  # all
135     pixels_train, labels_train = load_mnist(True, n_test, True)
136     pixels_test, labels_test = load_mnist(False, n_test, True)
137     create_svd_libaries(pixels_train, labels_train, pixels_test,
            labels_test, k,
138                         save_folder='Generated dataset variants',
                                dataset_name=dataset_name)
```

C.5 Modeller för NN

```
1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as functional
4
5
6  class LinearModule(nn.Module):
7      """A class represent a simple NN for digitalnumber MNIST"""
8
9      def __init__(self, input_size, hidden_layer_size) -> None:
10         super().__init__()
11         self.input_size = input_size
12         self.linear_1 = nn.Linear(self.input_size, hidden_layer_size)
13         self.linear_2 = nn.Linear(hidden_layer_size, hidden_layer_size
               // 2)
14         self.linear_3 = nn.Linear(hidden_layer_size // 2, 10)
```

```
15          # self.dropout = nn.Dropout(p=0.1)
16
17      def forward(self, x):
18          x = x.view(-1, self.input_size)  # Flattern
19          x = torch.relu(self.linear_1(x))
20          x = torch.relu(self.linear_2(x))
21          x = self.linear_3(x)
22          return functional.log_softmax(x, dim=1)
23
24
25  class ConvModule(nn.Module):
26      def __init__(self, fully_connected_size, convolutional_size):
27          super(ConvModule, self).__init__()
28          # Define the convolutional layers
29          self.conv1 = nn.Conv2d(1, convolutional_size, kernel_size=3,
                stride=1)
30          self.conv2 = nn.Conv2d(convolutional_size, convolutional_size *
                2, kernel_size=3, stride=1)
31
32          # Define the fully connected layers
33          self.fc1 = nn.Linear(convolutional_size * 2 * 5 * 5,
                fully_connected_size)
34          self.dropout = nn.Dropout(p=0.5)
35          self.fc2 = nn.Linear(fully_connected_size, 10)
36
37      def forward(self, x):
38          # Apply the convolutional layers
39          x = self.conv1(x)
40          x = nn.functional.relu(x)
41          x = nn.functional.max_pool2d(x, kernel_size=2)
42          x = self.conv2(x)
43          x = nn.functional.relu(x)
44          x = nn.functional.max_pool2d(x, kernel_size=2)
45
46          # Flatten the output from the convolutional layers
47          x = torch.flatten(x, 1)
48
49          # Apply the fully connected layers
50          x = self.fc1(x)
51          x = nn.functional.relu(x)
52          x = self.dropout(x)
53          x = self.fc2(x)
54          x = nn.functional.softmax(x, dim=1)
55          return x
```

C.6 Träning och testning av NN

```
1  import os.path
2  import warnings
3  import torch.optim as optim
4  from torchvision.datasets import MNIST
5  from torchvision import transforms
6  from torch.utils.data import DataLoader, Dataset
7  import numpy as np
8  import extra_filer.extra_tools as tools
9  from neural_network_image_classification.neural_network_models import *
```

```
10 import time
11 import torch.nn.functional as functional
12
13 timelog = tools.EasyTimeLog()
14 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
15
16
17 class SimpleDataset(Dataset):
18     """A class represent data, Are needed to make a dataloader to
          interate through when training and testing the model'
19     The "SimpleDataset" has __len__ and __getitem__ as abstractfunction
          -> therefore this classneed to have the same
20     """
21
22     def __init__(self, data, labels):
23         self.data = data
24         self.targets = labels
25         if len(data) != len(labels):
26             warnings.warn('The labels and the images are not the same
                  length')
27
28     def __len__(self):
29         return len(self.targets)
30
31     def __getitem__(self, idx):
32         data = self.data[idx]
33         label = self.targets[idx]
34         return torch.tensor(data, device=device), torch.tensor(int(
              label), device=device)
35
36
37 class ScoreDataset(Dataset):
38     """ This is a dataset for loading the pca dimensionality reduced
          image """
39
40     def __init__(self, k, datatype='train', dataset_name='MNIST'):
41         train_array, train_label_array, test_array, test_label_array =
              tools.load_pca_mnist(
42             f'pca_transformed_{dataset_name}_k={k}')
43         if datatype == 'train':
44             print(f'Loading training set pca_transformed_{dataset_name}
                  _k={k}')
45             self.data = torch.tensor(train_array, device=device)
46             self.label = torch.tensor(train_label_array, device=device)
47         else:
48             print(f'Loading testing set pca_transformed_{dataset_name}
                  _k={k}')
49             self.data = torch.tensor(test_array, device=device)
50             self.label = torch.tensor(test_label_array, device=device)
51
52     def __len__(self):
53         return len(self.label)
54
55     def __getitem__(self, idx):
56         data = self.data[idx]
57         label = self.label[idx]
```

```
58          return data, label
59
60
61  class SVDDataset(Dataset):
62      """ Parent for Uk and Vk dataset"""
63
64      def __init__(self, k, u_v, datatype='train', reshape=True,
            dataset_name='MNIST'):
65          train_array, train_label_array, test_array, test_label_array =
                tools.load_svd_mnist(
66              f'{u_v}_svd_transformed_{dataset_name}_k={k}', reshape=
                    reshape)
67          if datatype == 'train':
68              print(f'Loading training set pca_transformed_{dataset_name}
                    _k={k}')
69              self.data = torch.tensor(train_array, device=device)
70              self.label = torch.tensor(train_label_array, device=device)
71          else:
72              print(f'Loading testing set pca_transformed_{dataset_name}
                    _k={k}')
73              self.data = torch.tensor(test_array, device=device)
74              self.label = torch.tensor(test_label_array, device=device)
75          self.reshape = reshape
76
77      def __len__(self):
78          return len(self.label)
79
80      def __getitem__(self, idx):
81          data = self.data[idx]
82          if self.reshape:
83              data = data[None, :, :]
84          label = self.label[idx]
85          return data, label
86
87
88  class UkDataset(SVDDataset):
89      """ Dataset for loading in the Uk dataset """
90
91      def __init__(self, k, datatype='train', reshape=True, dataset_name=
            'MNIST'):
92          super().__init__(k, 'U', datatype=datatype, reshape=reshape,
                dataset_name=dataset_name)
93
94
95  class VkDataset(SVDDataset):
96      """ Dataset for loading in the Vk dataset """
97
98      def __init__(self, k, datatype='train', reshape=True, dataset_name=
            'MNIST'):
99          super().__init__(k, 'V', datatype=datatype, reshape=reshape,
                dataset_name=dataset_name)
100
101
102 class NormalMNISTDataset(Dataset):
103     """ Normal dataset just to compare the plots """
104
```

```
105     def __init__(self, datatype='train', square=False):
106         if datatype == 'train':
107             data, label = tools.load_mnist()
108         else:
109             data, label = tools.load_mnist(False)
110         if square:
111             data = data.reshape([-1, 1, 28, 28])
112         self.data = torch.tensor(data, device=device)
113         self.label = torch.tensor(label, device=device)
114 
115     def __len__(self):
116         return len(self.label)
117 
118     def __getitem__(self, idx):
119         data = self.data[idx]
120         label = self.label[idx]
121         return data, label
122 
123 
124 
125 
126 
127 def train_module(model: nn.modules, train_data: DataLoader, name: str,
        epochs: int):
128     """ A method which trains a module
129      Param: module = NN model, train_data = The trainset, name = the
             name of model that will be saved
130      Return: The total time for the training
131        """
132     # optimizer = optim.SGD(module.parameters(), lr=0.01)  # Stochastic
          Gradient Decent
133     optimizer = optim.Adam(model.parameters(), lr=0.001)
134     # loss = nn.CrossEntropyLoss()
135     time_1 = time.time()  # start time
136     for epoch in range(epochs):
137         print("Epoch: ", epoch)
138         for (x, y) in train_data:
139             y = torch.flatten(y)
140             optimizer.zero_grad(set_to_none=True)  # resets the
                  gradients before countinue to the next batch
141             prediction = model(x)  # predict and makes the forward
142             luss = functional.nll_loss(prediction, y)
143             luss.backward()  # compute the gradients
144             optimizer.step()  # updates the weights
145 
146     time_2 = time.time()  # end time
147     train_time = time_2 - time_1  # total time
148 
149     # Saves the model (assume you are in the big folder)
150     path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'saved_models', name + '.
            pth')
151     if not os.path.exists(os.path.dirname(path)):
152         os.makedirs(os.path.dirname(path))
153     torch.save(model.state_dict(), path)
154     return train_time  # the model don't need to be returned, it's
```

```
            already updated
155
156
157 def test_model(model: nn.Module, test_data: DataLoader):
158     """ Testing the model and calculate the prediction rate
159     param: model = the model to test, test_data = the test data
160     return: prediction rate
161     """
162     model = model.eval()  # sets the model in eveulation mode, because
            we dont want to update any weights
163     correct = 0
164     total_matrixes = 0
165
166     t0 = time.time()
167     for (data, labels) in test_data:
168         labels = labels.flatten()
169         outputs = model(data)
170         # Want to calculate max of each row inthe matrix, there for 1,
                (0 for colum)
171         # The output will be a tuple (value, index) but we are only
                intrested in the index because it can be compered to the
                labels
172         _, predicted = torch.max(outputs.data, 1)
173
174         # Calculate number of correct predictions
175         # (predicted == labels) -> Comperes the predicted valus with
                the labels and returns a vector with boolens
176         # .sum() -> sums the True boolen to a tensor
177         # .item() -> Transforms the tensor to an integer.
178         correct += (predicted == labels).sum().item()
179
180         # calculate the total number of images/matrixes in a batch
181         # Shape (x,y,z) -> 0 corresponds to batchsize
182         total_matrixes += labels.size(0)
183     time_taken = time.time() - t0
184
185     prediction_rate = correct / total_matrixes
186
187     return prediction_rate, time_taken
188
189 def load_model(model: nn.Module, name: str):
190     """ A method which loads the saved parameters into a model.
191      Param: module = NN model, name = the name of model the saved model
            """
192     # loads the model (assume you are in the big folder)
193     path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'saved_models', name + '.
            pth')
194     if os.path.exists(path):
195         model.load_state_dict(torch.load(path))
196         model.to(device)
197         return True
198     else:
199         raise FileNotFoundError(f'Model doesn\'t exist: {os.path.
                basename(path)}')
200
```

```
202
203
204 def train_score_nn(k, epochs=10, hidden_layer_size=128):
205     """
206     Fast way to train a score-matrix dataset
207     Trains a Neural network with only linear layers
208     """
209     dataset = ScoreDataset(k)
210     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
211     module = LinearModule(k, hidden_layer_size)
212     module.to(device)
213     training_time = train_module(module, dataloader,
214                                  f'linear_model_score_epoch_{epochs}_k
                                         ={k}_layersize_{hidden_layer_size}
                                         ',
215                                  epochs=epochs)
216     timelog.log(training_time, f'k={k}', f'hidden layer size {
            hidden_layer_size}', f'length dataset: {len(dataset)}',
217                 f'epochs={epochs}')
218     return training_time
219
220
221 def test_score_nn(k, epochs=10, hidden_layer_size=128, tests=1):
222     module = LinearModule(k, hidden_layer_size)
223     module.to(device)
224     load_model(module, f'linear_model_score_epoch_{epochs}_k={k}
            _layersize_{hidden_layer_size}')
225     dataset = ScoreDataset(k, datatype='test')
226     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
227     result_array = np.array([test_model(module, dataloader) for _ in
            range(tests)])
228     accuracy, testing_time = result_array.mean(axis=0)
229     testing_time /= len(dataset)
230     timelog.log(testing_time, f'k={k}', f'hidden layer size {
            hidden_layer_size}', f'length dataset: {len(dataset)}',
231                 f'epochs={epochs}', f'accuracy {accuracy}')
232     return accuracy, testing_time
233
234
235 def train_svd_nn(k, u_v, epochs=10, hidden_layer_size=128):
236     """ Trains the U or V dataset on a linear neural network """
237     dataset = SVDDataset(k, u_v, reshape=False)
238     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
239     module = LinearModule(k * 28, hidden_layer_size)  # We have k*28
            input values
240     module.to(device)
241     training_time = train_module(module, dataloader,
242                                  f'linear_model_{u_v}_epochs={epochs}_k
                                         ={k}_layersize={hidden_layer_size}
                                         ',
243                                  epochs=epochs)
244     timelog.log(training_time, f'k={k}', f'type: {u_v}', f'hidden layer
            size {hidden_layer_size}',
245                 f'length dataset: {len(dataset)}', f'epochs={epochs}')
246     return training_time
247
```

```
248
249 def test_svd_nn(k, u_v, epochs=10, hidden_layer_size=128, tests=1):
250     module = LinearModule(k * 28, hidden_layer_size)
251     module.to(device)
252     load_model(module, f'linear_model_{u_v}_epochs={epochs}_k={k}
        _layersize={hidden_layer_size}')
253     dataset = SVDDataset(k, u_v, reshape=False, datatype='test')
254     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
255     result_array = np.array([test_model(module, dataloader) for _ in
        range(tests)])
256     accuracy, testing_time = result_array.mean(axis=0)
257     testing_time /= len(dataset)
258     timelog.log(testing_time, f'k={k}', f'type: {u_v}', f'hidden layer
        size {hidden_layer_size}',
259                 f'length dataset: {len(dataset)}', f'epochs={epochs}',
                    f'accuracy {accuracy}')
260     return accuracy, testing_time
261
262
263 def train_normal_mnist_nn(epochs=10, hidden_layer_size=128):
264     """ Trains the U or V dataset on a linear neural network """
265     torch.no_grad()
266     dataset = NormalMNISTDataset()
267     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
268     module = LinearModule(784, hidden_layer_size)  # We have k*28 input
            values
269     module.to(device)
270     training_time = train_module(module, dataloader,
271                                  f'normal_mnist_linear_epochs={epochs}
                                        _layersize={hidden_layer_size}',
                                     epochs=epochs)
272     timelog.log(training_time, f'hidden layer size {hidden_layer_size}'
        , f'length dataset: {len(dataset)}',
273                 f'epochs={epochs}')
274     return training_time
275
276
277 def test_normal_mnist_nn(epochs=10, hidden_layer_size=128, tests=1):
278     module = LinearModule(784, hidden_layer_size)
279     module.to(device)
280     load_model(module, f'normal_mnist_linear_epochs={epochs}_layersize
        ={hidden_layer_size}')
281     dataset = NormalMNISTDataset(datatype='test')
282     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
283     result_array = np.array([test_model(module, dataloader) for _ in
        range(tests)])
284     accuracy, testing_time = result_array.mean(axis=0)
285     testing_time /= len(dataset)
286     timelog.log(testing_time, f'hidden layer size {hidden_layer_size}',
            f'length dataset: {len(dataset)}',
287                 f'epochs={epochs}', f'accuracy {accuracy}')
288     return accuracy, testing_time
289
290
291 def train_cnn_mnist_nn(epochs=10, linear_layer_size=128,
        convolutional_layer_size=16):
```

```
292     """ Trains the U or V dataset on a linear neural network """
293     torch.no_grad()
294     dataset = NormalMNISTDataset(square=True)
295     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
296     module = ConvModule(linear_layer_size, convolutional_layer_size)
297     module.to(device)
298     training_time = train_module(module, dataloader,
299                                  f'cnn_mnist_epochs={epochs}_layersize
                                         ={linear_layer_size}_cnn{
                                         convolutional_layer_size}',
300                                  epochs=epochs)
301     timelog.log(training_time, f'hidden layer size {linear_layer_size}'
            ,
302                 f'length dataset: {len(dataset)} and {
                        convolutional_layer_size}', f'epochs={epochs}')
303     return training_time
304
305
306 def test_cnn_mnist_nn(epochs=10, linear_layer_size=128,
        convolutional_layer_size=16, tests=1):
307     module = ConvModule(linear_layer_size, convolutional_layer_size)
308     module.to(device)
309     load_model(module, f'cnn_mnist_epochs={epochs}_layersize={
            linear_layer_size}_cnn{convolutional_layer_size}')
310     dataset = NormalMNISTDataset(datatype='test', square=True)
311     dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
312     result_array = np.array([test_model(module, dataloader) for _ in
            range(tests)])
313     accuracy, testing_time = result_array.mean(axis=0)
314     testing_time /= len(dataset)
315     timelog.log(testing_time, f'hidden layer size {linear_layer_size}
            and {convolutional_layer_size}',
316                 f'length dataset: {len(dataset)}', f'epochs={epochs}',
                        f'accuracy {accuracy}')
317     return accuracy, testing_time
318
319
320 def test_joel_pca_nn(k, epochs: int, hidden_layer_size: int, train:
        torch.Tensor, test: torch.Tensor):
321     name = 'linear_model_pca,k=' + str(k) + ',layer=' + str(
            hidden_layer_size)
322     # formating data
323     T_k_train = joel_pca_matrix(train.data, k)
324     T_k_test = joel_pca_matrix(test.data, k)
325     train_shape = T_k_train.shape[2] * T_k_train.shape[3]
326     T_k_train = SimpleDataset(T_k_train, train.targets)
327     T_k_test = SimpleDataset(T_k_test, test.targets)
328     # Makes the Dataloader
329     data_train = DataLoader(T_k_train, batch_size=64, shuffle=True)
330     data_test = DataLoader(T_k_test, batch_size=64, shuffle=True)
331     # defines the nn
332     lnn = LinearModule(train_shape, hidden_layer_size)
333     lnn.to(device)
334     # train
335     train_time = train_module(lnn, data_train, name, epochs)
336     # test
```

```
337     pre_rate, pre_time = test_model(lnn, data_test)
338     pre_time = pre_time / len(T_k_test)
339
340     return pre_rate, train_time, pre_time
341
342
343
344 def joel_get_MNIST():
345     """A methods which will return the MNIST data in a form of train
            and test data as tensors"""
346     mnist_train = MNIST('./data', train=True, download=True, transform=
            transforms.ToTensor())
347     mnist_test = MNIST('./data', train=False, download=True, transform=
            transforms.ToTensor())
348     return mnist_train, mnist_test
349
350
351 def joel_truncated_svd(A, k: int, SVD_k=False):
352     """ A method for truncated svd, which return a the svd, U, Z and V
            truncated with rang k
353     Param: A = data, k = Rang, SVD_k = if you want the svd_k
354     Return: U, Z, V and mabe svd all trunkated with rang k
355     """
356     U, Z, V = np.linalg.svd(A, full_matrices=True)  # Note: Z is not a
            diagonal matrix, only a vector
357     # Checks the dimension on the input and then Calculate truncated U,
            Z and V
358     if A.ndim == 3:
359         U_k = U[:, :, :k]  # shape (60000,28,k)
360         Z_k = Z[:, :k]  # shape (60000,k)
361         V_k = V[:, :k, :]  # shape (60000,k,28)
362         if SVD_k is True:
363             # Calculate the truncated SVD
364             # must use np.einsum because np.array matrix has a limit on
                    the shape, np.einsum will multiply elementwise in
                    shape of 'ijk,ik,ikl->ijl'
365             svd_k = np.einsum('ijk,ik,ikl->ijl', U_k, Z_k, V_k)  #
                    shape (60000,28,28)
366             return U_k, Z_k, V_k, svd_k
367     elif A.ndim == 2:
368         U_k = U[:, :k]  # shape (28,k)
369         Z_k = Z[:k]  # shape (k)
370         V_k = V[:k, :]  # shape (k,28)
371         if SVD_k is True:
372             # Calculate the truncated SVD
373             svd_k = np.matrix(U_k) * np.diag(Z_k) * np.matrix(V_k)  #
                    shape (28,28)
374             return U_k, Z_k, V_k, svd_k
375     else:
376         raise ValueError("Input matrix should be 2D or 3D.")
377     return U_k, Z_k, V_k
378
379
380 def joel_pca_matrix(A, k):
381     """ Calculates the score matrix by truncated svd and given rang
382     param: A = matrix, k = rang
```

```
383         return: score matris with rang k as a tensor
384         """
385         A_mean = torch.mean(A.float(), dim=2, keepdim=True)
386         A = A - A_mean
387         U_k, Z_k, _ = joel_truncated_svd(A, k, SVD_k=False)
388         # Calculation the score matix
389         T_k_nparray = np.einsum('ijk,ik->ijk', U_k, Z_k)  # np.einsum will
                multiply elementwise in shape of 'ijk,ik->ijk'
390         T_k_tensor = torch.from_numpy(T_k_nparray)
391         T_k_tensor = T_k_tensor.unsqueeze(1)
392         return T_k_tensor
```

C.7 Plottning av NN

```
1  import numpy as np
2  import pandas as pd
3  import extra_filer.extra_tools as tools
4  import os
5  import matplotlib.pyplot as plt
6  import neural_network_image_classification.neural_network_training as
       neural_network_training
7  import os
8
9
10 def test_score_matrix(onlytest=False, hidden_layer_sizes=None, k_list=
       None):
11     if hidden_layer_sizes is None:
12         hidden_layer_sizes = [8, 16, 32, 64, 128, 256]
13     if k_list is None:
14         k_list = [1, 2, 3, 5, 8, 14, 20, 28, 64, 128]
15
16     training_time_array = np.zeros([len(hidden_layer_sizes), len(k_list
           )])
17     testing_time_array = np.zeros([len(hidden_layer_sizes), len(k_list)
           ])
18     testing_accuracy_array = np.zeros([len(hidden_layer_sizes), len(
           k_list)])
19
20     for i, hidden_layer_size in enumerate(hidden_layer_sizes):
21         for j, k in enumerate(k_list):
22             print(f'Testing network with k={k}, hidden_layer_size={
                   hidden_layer_size}')
23             if not onlytest:
24                 training_time = neural_network_training.train_score_nn(
                       k, hidden_layer_size=hidden_layer_size)
25             testing_accuracy, testing_time = neural_network_training.
                   test_score_nn(k,
26                                                                          hid
27                                                                          tes
28
```

```
29              if not onlytest:
30                  training_time_array[i, j] = training_time
31              testing_time_array[i, j] = testing_time
32              testing_accuracy_array[i, j] = testing_accuracy
33
34      save_path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results')
35      if not os.path.exists(save_path):
36          os.makedirs(save_path)
37
38      if not onlytest:
39          pd.DataFrame(training_time_array, columns=k_list, index=np.
                array(hidden_layer_sizes)).to_csv(
40              os.path.join(tools.get_project_root(), '
                    neural_network_image_classification', 'results',
41                                 'score_training_time.csv'), sep=';')
42      pd.DataFrame(testing_time_array, columns=k_list, index=np.array(
            hidden_layer_sizes)).to_csv(
43          os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
44                             'score_testing_time.csv'), sep=';')
45      pd.DataFrame(testing_accuracy_array, columns=k_list, index=np.array
            (hidden_layer_sizes)).to_csv(
46          os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
47                             'score_testing_accuracy.csv'), sep=';')
48
49
50  def test_u_v_matrix(u_v, onlytest=False, hidden_layer_sizes=None,
        k_list=None):
51      if hidden_layer_sizes is None:
52          hidden_layer_sizes = [16, 64, 256]
53      if k_list is None:
54          k_list = [1, 2, 3, 5, 8, 14, 20]
55
56      training_time_array = np.zeros([len(hidden_layer_sizes), len(k_list
            )])
57      testing_time_array = np.zeros([len(hidden_layer_sizes), len(k_list)
            ])
58      testing_accuracy_array = np.zeros([len(hidden_layer_sizes), len(
            k_list)])
59
60      for i, hidden_layer_size in enumerate(hidden_layer_sizes):
61          for j, k in enumerate(k_list):
62              print(f'Testing␣network␣with␣k={k},␣hidden_layer_size={
                    hidden_layer_size}')
63              if not onlytest:
64                  training_time = neural_network_training.train_svd_nn(k,
                        u_v, hidden_layer_size=hidden_layer_size)
65              testing_accuracy, testing_time = \
66                  neural_network_training.test_svd_nn(k, u_v,
                        hidden_layer_size=hidden_layer_size, tests=10)
67              if not onlytest:
68                  training_time_array[i, j] = training_time
69              testing_time_array[i, j] = testing_time
70              testing_accuracy_array[i, j] = testing_accuracy
```

```
71
72      save_path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results')
73      if not os.path.exists(save_path):
74          os.makedirs(save_path)
75
76      if not onlytest:
77          pd.DataFrame(training_time_array, index=np.array(
                hidden_layer_sizes)).to_csv(
78              os.path.join(save_path, f'{u_v.lower()}_training_time.csv')
                    , sep=';')
79      pd.DataFrame(testing_time_array, index=np.array(hidden_layer_sizes)
            ).to_csv(
80          os.path.join(save_path, f'{u_v.lower()}_testing_time.csv'), sep
                =';')
81      pd.DataFrame(testing_accuracy_array, index=np.array(
            hidden_layer_sizes)).to_csv(
82          os.path.join(save_path, f'{u_v.lower()}_testing_accuracy.csv'),
                sep=';')
83
84
85  def test_normal_matrix(onlytest=False, hidden_layer_sizes=None):
86      if hidden_layer_sizes is None:
87          hidden_layer_sizes = [8, 16, 32, 64, 128, 256]
88
89      training_time_array = np.zeros([len(hidden_layer_sizes)])
90      testing_time_array = np.zeros([len(hidden_layer_sizes)])
91      testing_accuracy_array = np.zeros([len(hidden_layer_sizes)])
92
93      for i, hidden_layer_size in enumerate(hidden_layer_sizes):
94          print(f'Testing␣network␣with␣hidden_layer_size={
                hidden_layer_size}')
95          if not onlytest:
96              training_time = neural_network_training.
                    train_normal_mnist_nn(hidden_layer_size=
                    hidden_layer_size)
97          testing_accuracy, testing_time = neural_network_training.
                test_normal_mnist_nn(
98              hidden_layer_size=hidden_layer_size, tests=10)
99
100         if not onlytest:
101             training_time_array[i] = training_time
102         testing_time_array[i] = testing_time
103         testing_accuracy_array[i] = testing_accuracy
104
105     save_path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results')
106     if not os.path.exists(save_path):
107         os.makedirs(save_path)
108
109     if not onlytest:
110         pd.DataFrame(training_time_array, index=np.array(
                hidden_layer_sizes)).to_csv(
111             os.path.join(save_path, 'normal_training_time.csv'), sep=';
                    ')
112     pd.DataFrame(testing_time_array, index=np.array(hidden_layer_sizes)
```

```
        ).to_csv(
113         os.path.join(save_path, 'normal_testing_time.csv'), sep=';')
114     pd.DataFrame(testing_accuracy_array, index=np.array(
            hidden_layer_sizes)).to_csv(
115         os.path.join(save_path, 'normal_testing_accuracy.csv'), sep=';'
                )
116
117
118 def test_cnn_matrix(onlytest=False, linear_layer_sizes=None,
        convolutional_layer_sizes=None):
119     if linear_layer_sizes is None:
120         linear_layer_sizes = [8, 16, 32, 64, 128, 256]
121     if convolutional_layer_sizes is None:
122         convolutional_layer_sizes = [8, 16, 32]
123
124     training_time_array = np.zeros([len(linear_layer_sizes), len(
            convolutional_layer_sizes)])
125     testing_time_array = np.zeros([len(linear_layer_sizes), len(
            convolutional_layer_sizes)])
126     testing_accuracy_array = np.zeros([len(linear_layer_sizes), len(
            convolutional_layer_sizes)])
127
128     for i, linear_layer_size in enumerate(linear_layer_sizes):
129         for j, convolutional_layer_size in enumerate(
                convolutional_layer_sizes):
130             print(f'Testing network with fcc={linear_layer_size}, cn2d
                    ={convolutional_layer_size}')
131             if not onlytest:
132                 training_time = \
133                     neural_network_training.train_cnn_mnist_nn(
134                         linear_layer_size=linear_layer_size,
                                convolutional_layer_size=
                                convolutional_layer_size)
135             testing_accuracy, testing_time = neural_network_training.
                    test_cnn_mnist_nn(
136                 linear_layer_size=linear_layer_size,
                        convolutional_layer_size=convolutional_layer_size,
                        tests=10)
137
138             if not onlytest:
139                 training_time_array[i, j] = training_time
140             testing_time_array[i, j] = testing_time
141             testing_accuracy_array[i, j] = testing_accuracy
142
143     save_path = os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results')
144     if not os.path.exists(save_path):
145         os.makedirs(save_path)
146
147     if not onlytest:
148         pd.DataFrame(training_time_array, columns=
                convolutional_layer_sizes, index=np.array(
                linear_layer_sizes)).to_csv(
149             os.path.join(save_path, 'cnn_training_time.csv'), sep=';')
150     pd.DataFrame(testing_time_array, columns=convolutional_layer_sizes,
            index=np.array(linear_layer_sizes)).to_csv(
```

```
151         os.path.join(save_path, 'cnn_testing_time.csv'), sep=';')
152     pd.DataFrame(testing_accuracy_array, columns=
            convolutional_layer_sizes, index=np.array(linear_layer_sizes)).
            to_csv(
153         os.path.join(save_path, 'cnn_testing_accuracy.csv'), sep=';')
154
155
156 def plot_result_matrix(test_name='score'):
157     line_styles = ['-', '--', '-.', ':',
158                    '--', (0, (1, 1)), (0, (3, 1, 1, 1)), (0, (5, 1))]
159     marker_styles = ['o', 'D', '^', '+', '>', '1', '2', '3',
160                      '4', 's', 'p', '*', 'h', 'H', 'v', 'x', 'D', 'd']
161     testing_accuracy = pd.read_csv(
162         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
163                      f'{test_name}_testing_accuracy.csv'), sep=';',
                         index_col=0)
164     testing_time = pd.read_csv(os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results',
165                                             f'{test_name}_testing_time.
                                                csv'), sep=';',
                                                index_col=0)
166     training_time = pd.read_csv(os.path.join(tools.get_project_root(),
            'neural_network_image_classification', 'results',
167                                              f'{test_name}
                                                 _training_time.csv'),
                                                 sep=';', index_col=0)
168
169     # We don't need to plot all the rows/ different layer sizes
170     # selected_layer_sizes = [16, 32, 128, 256]
171     # selected_layer_sizes = [16, 64, 256]
172     # testing_accuracy = testing_accuracy.loc[selected_layer_sizes]
173     # testing_time = testing_time.loc[selected_layer_sizes]
174     # training_time = training_time.loc[selected_layer_sizes]
175
176     fig, axs = plt.subplots(1, 3, figsize=(12, 4))
177
178     x = [int(v) for v in testing_accuracy.columns]
179
180     labels = [f'{int(l)},␣{int(l)␣//␣2}' for l in testing_accuracy.
            index]
181
182     # Plotting the accuracy
183     for idx, line in testing_accuracy.reset_index().iterrows():
184         idx = int(idx)
185         line = line.values[1:]
186         axs[0].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
187     axs[0].set_ylabel('Precision')
188     # axs[0].set_ylim([0.5, 1])
189     axs[0].set_title('Precision␣f␣r␣olika␣lagerstorlekar␣och␣k')
190
191     # Plotting the training time
192     for idx, line in training_time.reset_index().iterrows():
193         idx = int(idx)
194         line = line.values[1:]
```

```
195         axs[1].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
196     axs[1].set_ylabel('tid (s)')
197     # axs[1].set_ylim([8, 12.5])
198     axs[1].set_title('Tr ningstid f r olika lagerstorlekar och k')
199
200     # Plotting the testing time
201     testing_time = testing_time * 10 ** 6
202     for idx, line in testing_time.reset_index().iterrows():
203         idx = int(idx)
204         line = line.values[1:]
205         axs[2].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx])
206     axs[2].set_ylabel('tid( s )')
207     # axs[2].set_ylim([7, 12])
208     axs[2].set_title('Testtid f r olika lagerstorlekar och k')
209
210     for ax in axs:
211         ax.set_xlabel('k')
212         ax.legend(labels, title='Lagerstorlekar')
213         ax.grid()
214
215     for i, ax in enumerate(axs):
216         ax.text(-0.14, 1.08, f"({chr(i + 97)})", transform=ax.transAxes
                ,
217                 fontweight='bold', fontsize=12, va='top', ha='left')
218
219     fig.tight_layout()
220     plt.show()
221
222
223 def plot_normal_result():
224     testing_accuracy = pd.read_csv(
225         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
226                      f'{"normal"}_testing_accuracy.csv'), sep=';',
                         index_col=0)
227     testing_time = pd.read_csv(os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results',
228                                             f'{"normal"}_testing_time.
                                                csv'), sep=';',
                                                index_col=0)
229     training_time = pd.read_csv(os.path.join(tools.get_project_root(),
            'neural_network_image_classification', 'results',
230                                              f'{"normal"}_training_time
                                                 .csv'), sep=';',
                                                 index_col=0)
231
232     fig, axs = plt.subplots(1, 3, figsize=(12, 4))
233
234     x = testing_time.index
235     xl = x.delete(1)
236     xticklabels = [f'{l}\n{l // 2}' for l in xl]
237
238     # Plotting the accuracy
239     axs[0].plot(x, testing_accuracy.T.iloc[0], '-o')
```

```
240     axs[1].set_ylabel('precision')
241     # axs[0].set_ylim([0.85, 1])
242     axs[0].set_title('Precision f r olika lagerstorlekar')
243
244     # Plotting the training time
245     axs[1].plot(x, training_time.T.iloc[0], '-o')
246     axs[1].set_ylabel('tid (s)')
247     # axs[1].set_ylim([0, 20])
248     axs[1].set_title('Tr ningstid f r olika lagerstorlekar')
249
250     # Plotting the testing time
251     testing_time = testing_time * 10 ** 6
252     axs[2].plot(x, testing_time.T.iloc[0], '-o')
253     axs[2].set_ylabel('tid ( s )')
254     # axs[2].set_ylim([0, 20])
255     axs[2].set_title('Testtid f r olika lagerstorlekar')
256
257     for ax in axs:
258         ax.set_xlabel('Lagerstorlekar')
259         ax.set_xticks(xl, xticklabels)
260         ax.grid()
261
262     for i, ax in enumerate(axs):
263         ax.text(-0.14, 1.08, f"({chr(i + 97)})", transform=ax.transAxes
                ,
264                 fontweight='bold', fontsize=12, va='top', ha='left')
265
266     fig.tight_layout()
267     plt.show()
268
269
270 def plot_cnn_result():
271     line_styles = ['-', '--', '-.', ':',
272                    '--', (0, (1, 1)), (0, (3, 1, 1, 1)), (0, (5, 1))]
273     marker_styles = ['o', 'D', '^', '+', '>', '1', '2', '3',
274                      '4', 's', 'p', '*', 'h', 'H', 'v', 'x', 'D', 'd']
275     testing_accuracy = pd.read_csv(
276         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
277                      f'{"cnn"}_testing_accuracy.csv'), sep=';',
                         index_col=0).T
278     testing_time = pd.read_csv(os.path.join(tools.get_project_root(), '
            neural_network_image_classification', 'results',
279                                             f'{"cnn"}_testing_time.csv'
                                                ), sep=';', index_col
                                                =0).T
280     training_time = pd.read_csv(os.path.join(tools.get_project_root(),
            'neural_network_image_classification', 'results',
281                                              f'{"cnn"}_training_time.
                                                 csv'), sep=';',
                                                 index_col=0).T
282
283     fig, axs = plt.subplots(1, 3, figsize=(12, 4))
284
285     x = [int(v) for v in testing_accuracy.columns]
286     labels = [f'{n}, {int(n) * 2}' for n in testing_accuracy.index]
```

```
287
288     # Plotting the accuracy
289     for idx, line in testing_accuracy.reset_index().iterrows():
290         idx = int(idx)
291         line = line.values[1:]
292         axs[0].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx], label=labels[idx])
293     # axs[0].plot(x, testing_accuracy.T.iloc[0], '-o')
294     axs[1].set_ylabel('precision')
295     # axs[0].set_ylim([0.94, 1])
296     axs[0].set_title('Precision f r olika lagerstorlekar')
297
298     # Plotting the training time
299     for idx, line in training_time.reset_index().iterrows():
300         idx = int(idx)
301         line = line.values[1:]
302         axs[1].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx], label=labels[idx])
303     # axs[1].plot(x, training_time.T.iloc[0], '-o')
304     axs[1].set_ylabel('tid (s)')
305     # axs[1].set_ylim([11, 15])
306     axs[1].set_title('Tr ningstid f r olika lagerstorlekar')
307
308     # Plotting the testing time
309     testing_time = testing_time * 10 ** 6
310     for idx, line in testing_time.reset_index().iterrows():
311         idx = int(idx)
312         line = line.values[1:]
313         axs[2].plot(x, line, linestyle=line_styles[idx], marker=
                marker_styles[idx], label=labels[idx])
314     # axs[2].plot(x, testing_time.T.iloc[0], '-o')
315     axs[2].set_ylabel('tid ( s )')
316     # axs[2].set_ylim([9, 13])
317     axs[2].set_title('Testtid f r olika lagerstorlekar')
318
319     for ax in axs:
320         ax.set_xlabel('Storlek p linj rt lager')
321         # ax.set_xticks(xl, xticklabels)
322         ax.grid()
323         ax.legend(title='Storlek faltningslager')
324
325     for i, ax in enumerate(axs):
326         ax.text(-0.14, 1.08, f"({chr(i + 97)})", transform=ax.transAxes
                ,
327                 fontweight='bold', fontsize=12, va='top', ha='left')
328
329     fig.tight_layout()
330     plt.show()
331
332
333 def test_joel_pca_matrix():
334     """ A funtion thats trains and tests the models
335         -> it is testing diffrent models depending on the parameters
336         -> Saves the results in csv files
337     """
338     # Sets the parameters
```

```
339     hidden_layer_sizes = [128, 256]
340     k_list = [1, 2, 3, 5, 8, 14, 20, 28]
341     # Defines varibles
342     training_time_array = np.zeros([len(hidden_layer_sizes), len(k_list
            )])
343     testing_time_array = np.zeros([len(hidden_layer_sizes), len(k_list)
            ])
344     testing_accuracy_array = np.zeros([len(hidden_layer_sizes), len(
            k_list)])
345     # Loads the original data
346     train_data, test_data = neural_network_training.joel_get_MNIST()
347
348     for i, hidden_layer_size in enumerate(hidden_layer_sizes):
349         for j, k in enumerate(k_list):
350             print(f'Training network with k={k}, hidden_layer_size={
                    hidden_layer_size}')
351             # Calls on test_pca_nn for train and test the nn
352             testing_accuracy, training_time, testing_time = \
353                 neural_network_training.test_joel_pca_nn(k, epochs=10,
354                                                          hidden_layer_size
                                                                 =
                                                                 hidden_layer_size
                                                                 ,
355                                                          train=
                                                                 train_data
                                                                 , test=
                                                                 test_data)
356             # Stores the results to later save in a csv file
357             training_time_array[i, j] = training_time
358             testing_time_array[i, j] = testing_time
359             testing_accuracy_array[i, j] = testing_accuracy
360     # Writes to a csv file to save the data
361     pd.DataFrame(training_time_array, columns=k_list, index=np.array(
            hidden_layer_sizes)).to_csv(
362         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
363                      'training_time_pca.csv'), sep=';')
364     pd.DataFrame(testing_time_array, columns=k_list, index=np.array(
            hidden_layer_sizes)).to_csv(
365         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
366                      'testing_time_pca.csv'), sep=';')
367     pd.DataFrame(testing_accuracy_array, columns=k_list, index=np.array
            (hidden_layer_sizes)).to_csv(
368         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
369                      'testing_accuracy_pca.csv'), sep=';')
370
371
372 def plot_joel_pca_matrix():
373     """ taken from "plot_score_matrix" change som eparameters to fit
            the pca matrix
374     -> plots the data
375     """
376     # Loads the already existing results from csv files
377     testing_accuracy = pd.read_csv(
```

```
378         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
379                     'testing_accuracy_pca.csv'), sep=';', index_col=0)
380     testing_time = pd.read_csv(
381         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
382                     'testing_time_pca.csv'), sep=';', index_col=0)
383     training_time = pd.read_csv(
384         os.path.join(tools.get_project_root(), '
                neural_network_image_classification', 'results',
385                     'training_time_pca.csv'), sep=';', index_col=0)
386     # Creates the figure to plot in
387     fig, axs = plt.subplots(1, 3, figsize=(12, 4))
388     # Takes out the k values from the loaded results
389     x = [int(v) for v in testing_accuracy.columns]
390     # Takes out the layersize values from the loaded results
391     labels = [f'{int(l)}, {int(l) // 2}' for l in testing_accuracy.
            index]
392
393     # Plotting the accuracy
394     axs[0].plot(x, testing_accuracy.iloc[0], '-o')
395     axs[0].plot(x, testing_accuracy.iloc[1], '-.v')
396     axs[0].set_ylabel('Precision')
397     axs[0].set_title('Precision f r olika lagerstorlekar och k')
398
399     # Plotting the training time
400     axs[1].plot(x, training_time.iloc[0], '-o')
401     axs[1].plot(x, training_time.iloc[1], '-.v')
402     axs[1].set_ylabel('tid (s)')
403     axs[1].set_title('Tr ningstid f r olika lagerstorlekar och k')
404
405     # Plotting the testing time
406     testing_time = testing_time * 1000
407     axs[2].plot(x, testing_time.iloc[0], '-o')
408     axs[2].plot(x, testing_time.iloc[1], '-.v')
409     axs[2].set_ylabel('tid (ms)')
410     axs[2].set_title('Testtid f r olika lagerstorlekar och k')
411
412     for ax in axs:
413         ax.set_xlabel('k')
414         ax.legend(labels, title='Lagerstorlekar')
415         ax.grid()
416
417     for i, ax in enumerate(axs):
418         ax.text(-0.14, 1.08, f"({chr(i + 97)})", transform=ax.transAxes
                ,
419                 fontweight='bold', fontsize=12, va='top', ha='left')
420
421     fig.tight_layout()
422     plt.show()
```

C.8 Nedladdning av MNIST och hjälpfunktioner

```
1 import datetime
2 import glob
3 import os
```

```
4  from collections import defaultdict
5  import numpy as np
6  import pandas as pd
7  from scipy.spatial.distance import cdist
8  from torchvision import datasets
9  import time
10 import inspect
11
12
13 class EasyTimeLog:
14     """
15     Creates an easy way to log the time it takes for different
           functions and such to execute
16     no parameters
17
18     Use log(time, model, etc)
19
20     Use start_log('model' ... etc) to start  a new log.
21     The start_log returns a TimeTracker object
22     call timetracker.stop() to stop the time tracker
23     This logs the time it took for the function to execute in a tidslog
           .txt file in the folder extra_filer
24     """
25
26     def __init__(self):
27         self.path = os.path.join(get_project_root(), 'extra_filer', '
               tidslog.txt')
28         if not os.path.exists(self.path):
29             # Create file if it does not exist
30             with open(self.path, 'a'):
31                 pass
32
33     def start_log(self, *args):
34         other = [str(arg) for arg in args]
35         caller = inspect.stack()[1][3]
36         s = '␣␣␣␣'.join([f'Date:␣{datetime.datetime.now()}', f'From:␣{
               caller}', *other, 'Time:␣{}']) + '\n'
37         return TimeTracker(s, self.path)
38
39     def log(self, log_time=None, *args):
40         other = [str(arg) for arg in args]
41         caller = inspect.stack()[1][3]
42         s = '␣␣␣␣'.join([f'Date:␣{datetime.datetime.now()}', f'From:␣{
               caller}', *other, f'Time:␣{log_time}']) + '\n'
43         with open(self.path, 'a') as f:
44             f.write(s)
45
46
47 class TimeTracker:
48     def __init__(self, log_string, path):
49         self.start_time = time.time()
50         self.log_string = log_string
51         self.path = path
52
53     def stop(self):
54         """ Stops current time and logs the time the activity took """
```

```
55          with open(self.path, 'a') as f:
56              f.write(self.log_string.format(time.time() - self.
                    start_time))
57
58      def time(self):
59          """ Gets the time since the object was created """
60          return time.time() - self.start_time
61
62
63  def get_project_root():
64      current_file = os.path.abspath(__file__)
65      current_directory = os.path.dirname(current_file)
66      project_root = os.path.dirname(current_directory)
67      return project_root
68
69
70  def append_matrix_to_csv(array_to_append, save_path, relative_root=
        False, folder=None):
71      if folder is not None:
72          if relative_root:
73              folder_path = os.path.join(get_project_root(), folder)
74              if not os.path.exists(folder_path):
75                  os.makedirs(folder_path)
76
77      if relative_root:
78          save_path = os.path.join(get_project_root(), folder, save_path)
79
80      if not os.path.exists(save_path):
81          # Create file if it does not exist
82          with open(save_path, 'a'):
83              pass
84
85      df = pd.DataFrame(array_to_append)
86      df.to_csv(save_path, index=False, header=False, mode='a', sep=';')
87      print(f'Saved file to {save_path}')
88
89
90  def load_mnist(training=True, max_number=-1, keep_square=False,
        normalize=True, dtype='float32'):
91      """
92      Loads the MNIST dataset as NumPy arrays.
93      args:
94          training (bool): Whether to load the training data (True) or
                the test data (False).
95          max_number (int): The maximum digit label to include in the
                dataset (default is 9).
96          keep_square (bool): Whether to reshape the data to n x 784 (
                False) or keep it at n x 28 x 28 (True)
97      returns:A tuple containing the data and labels as NumPy arrays.
98      """
99      save_path = os.path.join(get_project_root(), 'MNIST')
100     if not os.path.exists(save_path):
101         os.makedirs(save_path)
102
103     # Load the MNIST dataset as NumPy arrays
104     data_training = datasets.MNIST(root=save_path, train=True, download
```

```
        =True).train_data.numpy()
105     labels_training = datasets.MNIST(root=save_path, train=True,
            download=True).train_labels.numpy()
106     data_testing = datasets.MNIST(root=save_path, train=False, download
            =True).test_data.numpy()
107     labels_testing = datasets.MNIST(root=save_path, train=False,
            download=True).test_labels.numpy()
108
109     if normalize:
110         data_training = data_training / 255
111         data_testing = data_testing / 255
112         training_mean = 0.1306604762738429  # data_training.mean()
113         training_std = 0.3081078038564622  # data_training.std()
114         data_training = (data_training - training_mean) / training_std
115         data_testing = (data_testing - training_mean) / training_std
116
117     if not keep_square:
118         data_training = data_training.reshape(-1, 28 ** 2)
119         data_testing = data_testing.reshape(-1, 28 ** 2)
120
121     if training:
122         data = data_training
123         labels = labels_training
124     else:
125         data = data_testing
126         labels = labels_testing
127
128     if not keep_square:
129         data = data.reshape(-1, 28 ** 2)
130
131     if max_number > 0:
132         data = data[:max_number]
133         labels = labels[:max_number]
134
135     data = data.astype(dtype)
136
137     return data, labels
138
139
140 def load_pca_mnist(directory_name, create_if_nonexistent=True, dtype='
        float32'):
141     """
142     Easy way to load the PCA modified dataset.
143     :param dtype: -
144     :param create_if_nonexistent: -
145     :param directory_name: Directory name inside "Generated dataset
            variants" ex "pca_transformed_MNIST_k=10"
146     :return: training set, training labels, testing set, testing labels
147     """
148     k = None
149     if k is None and 'k=' in directory_name:
150         k = int(directory_name.split('k=')[-1])
151
152     load_path = os.path.join(get_project_root(), 'Generated dataset
            variants', directory_name)
153     if not os.path.exists(load_path):
```

```
154             if create_if_nonexistent:
155                 from extra_filer.dataset_variant_generator import
                        pca_library_main
156                 pca_library_main(k)
157             else:
158                 raise Exception(f"The specified directory does not exist at
                        : {load_path}")
159 
160         files = glob.glob(os.path.join(load_path, '*'))
161         test_label_path = None
162         train_label_path = None
163         test_path = None
164         train_path = None
165         for file_path in files:
166             label = False
167             test = False
168             if 'label' in file_path:
169                 label = True
170             if 'test' in file_path:
171                 test = True
172             if label and test:
173                 test_label_path = file_path
174             elif label:
175                 train_label_path = file_path
176             elif test:
177                 test_path = file_path
178             else:
179                 train_path = file_path
180         if test_path is None or train_path is None or test_label_path is
                None or train_label_path is None:
181             raise Exception(f"The specified directory does not contain all
                    necessary files: {load_path}")
182         test_label_array = pd.read_csv(test_label_path, sep=';', header=
                None).to_numpy()
183         train_label_array = pd.read_csv(train_label_path, sep=';', header=
                None).to_numpy()
184         test_array = pd.read_csv(test_path, sep=';', header=None).to_numpy
                ()
185         train_array = pd.read_csv(train_path, sep=';', header=None).
                to_numpy()
186 
187         train_array = train_array.astype(dtype)
188         test_array = test_array.astype(dtype)
189 
190         return train_array, train_label_array, test_array, test_label_array
191 
192 
193 def load_svd_mnist(directory_name, reshape=False, k=None,
        create_if_nonexistent=True, dtype='float32'):
194         """
195         Easy way to load the modified V or U datasets. Can return them both
                as 2d matrix of vectors or as a flattened version
196         :param dtype: -
197         :param create_if_nonexistent: creates the dataset if it doesn't
                exist
198         :param directory_name: Directory of the folder inside "Generated
```

```
        dataset variants"
199     :param reshape: Reshapes each row into 2d matrixes for each
            eigenvector
200     :param k: the truncation number
201     :return: training set, training labels, testing set, testing labels
202     """
203     if k is None and 'k=' in directory_name:
204         k = int(directory_name.split('k=')[-1])
205     elif reshape:
206         raise Exception(
207             f'The directory either has to contain the number of
                principal components k, or it has to be specified if
                reshape is true')
208
209     load_path = os.path.join(get_project_root(), 'Generated dataset
            variants', directory_name)
210     if not os.path.exists(load_path):
211         if create_if_nonexistent:
212             from extra_filer.dataset_variant_generator import
                svd_libraries_main
213             svd_libraries_main(k)
214         else:
215             raise Exception(f"The specified directory does not exist at
                : {load_path}")
216     files = glob.glob(os.path.join(load_path, '*'))
217     test_label_path = None
218     train_label_path = None
219     test_path = None
220     train_path = None
221     for file_path in files:
222         label = False
223         test = False
224         if 'label' in file_path:
225             label = True
226         if 'test' in file_path:
227             test = True
228         if label and test:
229             test_label_path = file_path
230         elif label:
231             train_label_path = file_path
232         elif test:
233             test_path = file_path
234         else:
235             train_path = file_path
236     if test_path is None or train_path is None or test_label_path is
            None or train_label_path is None:
237         raise Exception(f"The specified directory does not contain all
                necessary files: {load_path}")
238     test_label_array = pd.read_csv(test_label_path, sep=';', header=
            None).to_numpy()
239     train_label_array = pd.read_csv(train_label_path, sep=';', header=
            None).to_numpy()
240     test_array = pd.read_csv(test_path, sep=';', header=None).to_numpy
            ()
241     train_array = pd.read_csv(train_path, sep=';', header=None).
            to_numpy()
```

```
242
243     if reshape:
244         test_array = test_array.reshape((len(test_array), -1, k))
245         train_array = train_array.reshape((len(test_array), -1, k))
246
247     train_array = train_array.astype(dtype)
248     test_array = test_array.astype(dtype)
249
250     return train_array, train_label_array, test_array, test_label_array
251
252
253 def reduce_pca_dataset(dtrain, train_labels, threshold_mult=0.5, dtype=
        'float32'):
254     """
255     Combines similar images in the training set.
256     If the
257     threshold_mult: If the distance between two images is smaller than
            distance_mult*average_distance then they will be
258                     combined
259     """
260     # dtrain=dtrain.get()
261     # train_labels = train_labels.get()
262     dtrain_reduced = []
263     dlabels_reduced = []
264     for n in range(10):
265         print(f'Reducing: {n}/9')
266         dnum_new = []
267         dnum = dtrain[(train_labels == n).reshape([-1])]
268         # Compute the average distance for the first 10 occurences of
                each number since
269         # that should be close enough
270         average_distance = sum([np.sum(np.square(dnum - dnum[i]), axis
                =1).mean() for i in range(10)]) / 10
271         threshold = average_distance * (threshold_mult ** 2)
272         distances = np.array(cdist(dnum, dnum, metric='sqeuclidean'))
273         pairs = np.argwhere(distances < threshold)
274         pairs = [(i, j) for (i, j) in pairs if i < j]  # remove
                duplicates and self-matches
275         pair_dict = defaultdict(set)
276         used_set = set()
277         for idx, (i, j) in enumerate(pairs):
278             if idx % 1000 == 0:
279                 pass
280                 # print(f'idx: {idx} out of: {len(pairs)}')
281             if i not in used_set:
282                 used_set.add(j)
283                 pair_dict[i].add(i)
284                 pair_dict[i].add(j)
285         for key, item in pair_dict.items():
286             dnum_new.append(dnum[list(item)].mean(axis=0))
287         dtrain_reduced += dnum_new
288         dlabels_reduced += [n] * len(dnum_new)
289
290     dtrain_reduced = np.array(dtrain_reduced)
291     dlabels_reduced = np.array(dlabels_reduced)
292
```

```
293     dtrain_reduced = dtrain_reduced.astype(dtype)
294
295     return dtrain_reduced, dlabels_reduced
```

C.9 Grafiskt användargränssnitt

```
1  import os
2  import sys
3  import warnings
4  from PyQt5.QtGui import QIntValidator
5  from PyQt5.QtCore import QThread, pyqtSignal, Qt, QSize
6  from PyQt5.QtWidgets import QApplication, QWidget, QCheckBox, QLabel,
       QLineEdit, QVBoxLayout, QHBoxLayout, QPushButton, \
7       QFileDialog, QGridLayout, QSpacerItem, QSizePolicy, QMessageBox,
            QScrollArea, QMainWindow
8  from PyQt5.QtGui import QPixmap
9  from PyQt5 import QtCore
10 import extra_filer.extra_tools as tools
11 import svd_image_compression.image_compression as svd_imagecompression
12 import neural_network_image_classification.neural_network_testing as
       nn_testing
13 import pca_image_classification.pca_classification as
       pca_classification
14 import pca_image_classification.pca_classification_plotter as
       pca_plotter
15
16
17 class MainWindow(QMainWindow):
18     def __init__(self):
19         super().__init__()
20
21         self.svd_image_path = None
22
23         self.setWindowTitle("Kandidatarbete")
24         self.setStyleSheet("background-color: #333333; color: white;")
25
26         self.nn_label = QLabel("Neural network image classification")
27         self.nn_label.setStyleSheet("font-size: 20px; font-weight: bold
               ; margin-bottom: 10px;")
28
29         self.nn_input1_label = QLabel("Input 1:")
30         self.nn_input1 = QLineEdit()
31         self.nn_input2_label = QLabel("Input 2:")
32         self.nn_input2 = QLineEdit()
33
34         self.nn_button_train = QPushButton("Train and Test")
35         self.nn_button_train.clicked.connect(self.train_nn)
36
37         self.nn_button_test = QPushButton("Only Test")
38         self.nn_button_test.clicked.connect(self.test_nn)
39
40         self.nn_checkboxes = [QCheckBox("Linear"), QCheckBox("CNN"),
               QCheckBox("Score"), QCheckBox("Uk"),
41                               QCheckBox("Vk")]
42         for i, checkbox in enumerate(self.nn_checkboxes):
43             checkbox.clicked.connect(lambda state, n=i: self.
```

```
                checkbox_clicked(n))
44
45          self.nn_description = QLabel("Fill the input fields with comma
                separated values: x1, x2, ...")
46          self.nn_description.setStyleSheet("font-size: 12px;")
47
48          self.nn_layout = QVBoxLayout()
49          self.nn_layout.addWidget(self.nn_label)
50          for checkbox in self.nn_checkboxes:
51              self.nn_layout.addWidget(checkbox)
52          self.nn_layout.addWidget(self.nn_description)
53          self.nn_layout.addWidget(self.nn_input1_label)
54          self.nn_layout.addWidget(self.nn_input1)
55          self.nn_layout.addWidget(self.nn_input2_label)
56          self.nn_layout.addWidget(self.nn_input2)
57          self.nn_layout.addWidget(self.nn_button_train)
58          self.nn_layout.addWidget(self.nn_button_test)
59
60          self.pca_label = QLabel("PCA image classification")
61          self.pca_label.setStyleSheet("font-size: 20px; font-weight:
                bold; margin-bottom: 10px;")
62
63          self.pca_input1_label = QLabel("Training set size. Use -1, -2
                and -3 for reduced sets 0.5, 0.75 and 1")
64          self.pca_input1 = QLineEdit()
65          self.pca_input1.setText("1000, 5000, 10000, 30000, 60000")
66          self.pca_input2_label = QLabel("score matrix size (k)")
67          self.pca_input2 = QLineEdit()
68          self.pca_input2.setText("2, 8, 32, 128, 512")
69          self.pca_button = QPushButton("Test")
70          self.pca_button.clicked.connect(self.run_pca)
71
72          self.pca_layout = QVBoxLayout()
73          self.pca_layout.addWidget(self.pca_label)
74          self.pca_layout.addWidget(self.pca_input1_label)
75          self.pca_layout.addWidget(self.pca_input1)
76          self.pca_layout.addWidget(self.pca_input2_label)
77          self.pca_layout.addWidget(self.pca_input2)
78          self.pca_layout.addWidget(self.pca_button)
79
80          self.svd_label = QLabel("SVD image compression")
81          self.svd_label.setStyleSheet("font-size: 20px; font-weight:
                bold; margin-bottom: 10px;")
82          self.svd_description = QLabel("Enter the number of singular
                values to keep (k)")
83          self.svd_description.setStyleSheet("font-size: 12px;")
84
85          self.svd_input_label = QLabel("Input:")
86          self.svd_input = MyLineEdit()
87          # self.svd_input.setReadOnly(True)
88          self.svd_import_button = QPushButton("Import")
89          self.svd_import_button.clicked.connect(self.import_image)
90          self.svd_run_button = QPushButton("Run")
91          self.svd_run_button.clicked.connect(self.run_svd)
92          self.svd_image_label = QLabel()
93          self.svd_image_label.setFixedSize(200, 200)
```

```
94          self.svd_image_label.setAlignment(QtCore.Qt.AlignCenter)
95          self.svd_image_label.setStyleSheet("border: 1px solid white;")
96
97          self.svd_image_output_label = QLabel()
98          self.svd_image_output_label.setFixedSize(200, 200)
99          self.svd_image_output_label.setAlignment(QtCore.Qt.AlignCenter)
100         self.svd_image_output_label.setStyleSheet("border: 1px solid
                white;")
101
102         self.svd_layout = QGridLayout()
103         self.svd_layout.addWidget(self.svd_label, 0, 0, 1, 4)
104         self.svd_layout.addWidget(self.svd_description, 1, 0, 1, 4)
105         self.svd_layout.addWidget(self.svd_input_label, 2, 0, )
106         self.svd_layout.addWidget(self.svd_input, 2, 1, 1, 3)
107         self.svd_layout.addWidget(self.svd_import_button, 3, 0)
108         self.svd_layout.addWidget(self.svd_run_button, 3, 1, 1, 3)
109         self.svd_layout.addWidget(self.svd_image_label, 4, 0, 1, 2)
110         self.svd_layout.addWidget(self.svd_image_output_label, 4, 2)
111         self.svd_layout.setColumnStretch(1, 1)
112
113         self.layout = QVBoxLayout()
114         self.layout.addLayout(self.nn_layout)
115         self.layout.addLayout(self.pca_layout)
116         self.layout.addLayout(self.svd_layout)
117
118         self.scroll = QScrollArea()
119         self.widget = QWidget()
120         self.widget.setLayout(self.layout)
121         self.scroll.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
122         self.scroll.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
123         self.scroll.setWidgetResizable(False)
124         self.scroll.setWidget(self.widget)
125
126         self.setCentralWidget(self.scroll)
127         self.setGeometry(600, 100, 445, 820)
128         self.setWindowTitle('Demo')
129         self.show()
130
131         # self.setLayout(self.layout)
132         self.import_image(False, os.path.join(tools.get_project_root(),
                'svd_image_compression', 'zelda.png'))
133
134         self.show()
135
136     def train_nn(self):
137         self.test_train_nn(onlytest=False)
138
139     def test_nn(self):
140         print('Testing NN')
141         self.test_train_nn(onlytest=True)
142
143     def test_train_nn(self, onlytest):
144         print('Training NN')
145         input1 = self.nn_input1.text()
146         input2 = self.nn_input2.text()
147
```

```
148         checked = [checkbox.checkState() for checkbox in self.
                nn_checkboxes]
149
150         train_test_func = None
151         plot_func = None
152
153         if checked[0]:
154             input1 = self.check_input(input1, 1024)
155             if input1 is None:
156                 return
157             train_test_func = lambda: nn_testing.test_normal_matrix(
                    onlytest=onlytest, hidden_layer_sizes=input1)
158             plot_func = lambda: nn_testing.plot_normal_result()
159         elif checked[1]:
160             input1 = self.check_input(input1, 1024)
161             input2 = self.check_input(input2, 128)
162             if input1 is None or input2 is None:
163                 return
164             train_test_func = lambda: nn_testing.test_cnn_matrix(
                    onlytest=onlytest, linear_layer_sizes=input1,
165                                                                 convolutional_layer_
                                                                    =
                                                                    input2
                                                                    )
166             plot_func = lambda: nn_testing.plot_cnn_result()
167         elif checked[2]:
168             input1 = self.check_input(input1, 1024)
169             input2 = self.check_input(input2, 784)
170             if input1 is None or input2 is None:
171                 return
172             train_test_func = lambda: nn_testing.test_score_matrix(
                    onlytest=onlytest, hidden_layer_sizes=input1,
173                                                                 k_list
                                                                    =
                                                                    input2
                                                                    )

174             plot_func = lambda: nn_testing.plot_result_matrix('score')
175         elif checked[3]:
176             input1 = self.check_input(input1, 1024)
177             input2 = self.check_input(input2, 28)
178             if input1 is None or input2 is None:
179                 return
180             train_test_func = lambda: nn_testing.test_u_v_matrix('U',
                    onlytest=onlytest, hidden_layer_sizes=input1,
181                                                                 k_list
                                                                    =
                                                                    input2
                                                                    )
182             plot_func = lambda: nn_testing.plot_result_matrix('U')
183         elif checked[4]:
184             input1 = self.check_input(input1, 1024)
185             input2 = self.check_input(input2, 28)
186             if input1 is None or input2 is None:
187                 return
188             train_test_func = lambda: nn_testing.test_u_v_matrix('V',
```

```
                onlytest=onlytest, hidden_layer_sizes=input1,
189                                                                 k_list
                                                                       =
                                                                       input2
                                                                       )
190             plot_func = lambda: nn_testing.plot_result_matrix('V')
191
192         # thread = threading.Thread(target=train_test_func)
193         # thread.start()
194         self.worker_nn = NormalWorker(train_test_func)
195         self.worker_nn.finished.connect(plot_func)
196         self.worker_nn.finished.connect(self.worker_nn.deleteLater)
197         self.worker_nn.start()
198
199     def run_pca(self):
200         input1 = self.pca_input1.text()
201         input2 = self.pca_input2.text()
202         input1 = self.check_input(input1, 60001, lower_bound=-3)
203         input2 = self.check_input(input2, 784)
204         if input1 is None or input2 is None:
205             return
206         classify = lambda: pca_classification.test_pca(
                training_set_sizes=input1, k_list=input2)
207         plot_results = lambda: pca_plotter.plot_results()
208
209         self.worker_pca = NormalWorker(classify)
210         self.worker_pca.finished.connect(plot_results)
211         self.worker_pca.finished.connect(self.worker_pca.deleteLater)
212         self.worker_pca.start()
213
214     def import_image(self, state, file_name=None):
215         if file_name is None:
216             file_name, _ = QFileDialog.getOpenFileName(self, "Open␣
                    Image", "", "Image␣Files␣(*.png␣*.jpg␣*.bmp)")
217         if file_name:
218             # self.svd_input.setText(file_name)
219             self.svd_image_path = file_name
220             pixmap = QPixmap(file_name)
221             self.svd_image_label.setPixmap(
222                 pixmap.scaled(self.svd_image_label.size(), QtCore.Qt.
                        KeepAspectRatio, QtCore.Qt.SmoothTransformation))
223
224     def run_svd(self):
225         input_path = self.svd_image_path
226         k = self.svd_input.text()
227         if k is None or len(k) == 0:
228             warnings.warn('Input␣field␣can\'t␣be␣empty')
229             return
230         k = int(k)
231         self.worker_svd = SVDWorker(svd_imagecompression.compress_image
                , input_path, k)
232         self.worker_svd.finished.connect(self.update_result_image)
233         self.worker_svd.finished.connect(self.worker_svd.deleteLater)
234         self.worker_svd.start()
235         # call the function that handles the SVD image compression
236
```

```
237     def update_result_image(self, path):
238         pixmap = QPixmap(path)
239         self.svd_image_output_label.setPixmap(
240             pixmap.scaled(self.svd_image_label.size(), QtCore.Qt.
                KeepAspectRatio, QtCore.Qt.SmoothTransformation))
241
242     def checkbox_clicked(self, n):
243         for idx, checkbox in enumerate(self.nn_checkboxes):
244             if idx != n:
245                 checkbox.setChecked(False)
246             else:
247                 checkbox.setChecked(True)
248         input1_titles = [
249             "Layer sizes (x, x/2)",
250             "Linear layer size (x)",
251             "Linear layer sizes (x, x/2)",
252             "Linear layer sizes (x, x/2)",
253             "Linear layer sizes (x, x/2)"
254         ]
255         input2_titles = [
256             "Leave this empty",
257             "Convolutional layer sizes (x, x*2)",
258             "k (x)",
259             "k (x)",
260             "k (x)"
261         ]
262         input1_defaults = [
263             "8, 16, 32, 64, 128, 256",
264             "8, 16, 32, 64, 128, 256",
265             "8, 16, 32, 64, 128, 256",
266             "16, 64, 256",
267             "16, 64, 256"
268         ]
269         input2_defaults = [
270             "",
271             "8, 16, 32",
272             "1, 2, 3, 5, 8, 14, 20, 28, 64, 128",
273             "1, 2, 3, 5, 8, 14, 20",
274             "1, 2, 3, 5, 8, 14, 20"
275         ]
276
277         self.nn_input1_label.setText(input1_titles[n])
278         self.nn_input2_label.setText(input2_titles[n])
279         self.nn_input1.setText(input1_defaults[n])
280         self.nn_input2.setText(input2_defaults[n])
281
282     def check_input(self, input: str, upper_bound: int, lower_bound=0):
283         # Split the input string into a list of strings
284         input_list = input.split(',')
285
286         # Check that all the elements in the list are integers
287         try:
288             input_list = [int(x) for x in input_list]
289         except ValueError:
290             print("Invalid input: not all elements are integers")
291             return None
```

```
292
293         # Check that all the integers are between 0 and upper_bound
294         for x in input_list:
295             if x <= lower_bound or x >= upper_bound:
296                 print("Invalid input: integer out of range")
297                 return None
298
299         # Check that there is at least one element in the list
300         if len(input_list) == 0:
301             print("Invalid input: empty list")
302             return None
303
304         # Return the list of integers if everything checks out
305         return input_list
306
307
308 class SVDWorker(QThread):
309     finished = pyqtSignal(str)
310
311     def __init__(self, func, *args, **kwargs):
312         super().__init__()
313         self.func = func
314         self.args = args
315         self.kwargs = kwargs
316
317     def run(self):
318         try:
319             result = self.func(*self.args, **self.kwargs)
320         except Exception as e:
321             sys.excepthook(e.__class__, e, sys.exc_info()[2])
322             self.exit()
323         self.finished.emit(result)
324
325
326 class NormalWorker(QThread):
327     finished = pyqtSignal()
328
329     def __init__(self, func, *args, **kwargs):
330         super().__init__()
331         self.func = func
332         self.args = args
333         self.kwargs = kwargs
334
335     def run(self):
336         try:
337             self.func(*self.args, **self.kwargs)
338         except Exception as e:
339             sys.excepthook(e.__class__, e, sys.exc_info()[2])
340             self.exit()
341             self.finished.emit()
342         self.finished.emit()
343
344
345 class MyLineEdit(QLineEdit):
346     def __init__(self, *args, **kwargs):
347         super().__init__(*args, **kwargs)
```

```
348             self.setValidator(QIntValidator(1, 500))
349
350
351 if __name__ == "__main__":
352     app = QApplication(sys.argv)
353     window = MainWindow()
354     sys.exit(app.exec_())
```