CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

On the Role of Attention Maps in Visual Transformers—A Clustering Perspective

Master's thesis in Computer science and engineering

ERIC ANTTILA RYDERUP
YU-PING HSU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

On the Role of Attention Maps in Visual Transformers—A Clustering Perspective

ERIC ANTTILA RYDERUP
YU-PING HSU

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

On the Role of Attention Maps in Visual Transformers—A Clustering Perspective

ERIC ANTTILA RYDERUP
YU-PING HSU

Supervisor: Ashkan Panahi, Department of Computer Science and Engineering
Examiner: Rocío Mercado Oropeza, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

On the Role of Attention Maps in Visual Transformers—A Clustering Perspective

ERIC ANTTILA RYDERUP
YU-PING HSU
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis delves into a novel area of research, exploring whether attention maps from a single-layer Vision Transformer model exhibit a clustering structure. The discovery of such a structure would imply that tokens with similar semantic information tend to cluster together. We extract an attention map from a one-layer Vision Transformer model, which uses image patches as input data. Values below a set threshold are pruned from the attention map, and a graph is created from the remaining data. Various community detection algorithms are then applied to this graph and evaluated based on modularity. We visualize the patches belonging to each cluster and compare classification performance when removing salient and non-salient clusters. The method reveals a significant clustering structure, which was discovered by the Louvain algorithm. The tokens cluster to other objects with similar semantic information, effectively separating parts of the image. The classification logit values for specific images are improved when tokens belonging to unimportant clusters are removed while removing tokens from important clusters negatively impacts performance. This work suggests that a Vision Transformer's attention layer clusters tokens based on their semantic information, but further research is needed to confirm the generality of this result.

Keywords: Vision Transformer, attention layer, attention map, clustering, interpretability, Louvain, visualization.

Acknowledgements

We want to extend our heartfelt thanks to our examiner, Rocío Mercado Oropeza, for her excellent examination and invaluable corrections during the project. Her insight and suggestions significantly contributed to the quality of our thesis. Equally, we express our deep gratitude to our supervisor, Ashkan Panahi, for his patience, explanations, guidance, and ideas. His input was crucial for shaping and guiding the project towards a successful outcome. Lastly, we want to recognize the vital support from our families, friends, and relatives in this journey. Their encouragement have been a constant source of motivation. We are truly grateful for all the assistance and expertise we have received.

Eric Anttila Ryderup and Yu-Ping Hsu, Gothenburg, 2024-06-19

Contents

List of Figures

List of Tables

1

Introduction

Artificial intelligence (AI) and deep learning have recently seen a surge in popularity and complexity; chatbots can help with writing, satisfy the American bar exam and advanced placement exams, and even create computer code [1]. The transformer architecture, [2], [3], is the breakthrough that makes this possible. The architecture has also found success in other domains, where it is a part of the state-of-the-art: visual transformers [4], protein structure prediction [5], and image generation [6].

AI usage is expected to increase in the future. AI has already been implemented in office settings, where it summarizes meetings and generates emails [7]. As the transformer architecture—particularly the attention mechanism—currently plays an integral part in the state-of-the-art models, a deeper understanding of this mechanism is of great interest. However, understanding what happens when transformer models learn is limited [8], [9].

Transformer models, introduced by Vaswani et al. in their groundbreaking 2017 paper *Attention is All You Need* [10], have revolutionized the fields of Natural Language Processing (NLP) and image classification. These models use self-attention mechanisms to process input data, enabling them to capture complex dependencies and relationships within the data more effectively than traditional recurrent or convolutional neural networks.

In NLP, transformers have achieved state-of-the-art results in language translation, text generation, and sentiment analysis tasks. Their ability to handle long-range dependencies and parallelize training has led to the development of robust models like BERT, GPT, and T5, which have set new performance benchmarks across various applications.

Transformers have also shown their potential in image classification. Vision Transformers (ViTs) have demonstrated that, with sufficient data and computational resources, they can outperform conventional convolutional neural networks (CNNs) in image classification tasks. By treating image patches as sequences of tokens, ViTs use self-attention mechanisms to identify patterns and features within images, offering a novel approach to computer vision.

In this project, we focused on one of the most crucial components of the Vision Transformer model—the attention layer. By investigating the role of clustering in attention maps, we discovered a promising, but limited, approach to enhance the model's performance. This approach leverages the inherent structure within the

attention mechanisms to optimize information flow and representation, leading to more accurate and efficient model predictions.

1.1 Related Works

The understanding of how weights are allocated for a transformer is improving. In [11] and [12], it is shown that self-attention is equivalent to a hard-margin support vector machine that separates optimal and non-optimal tokens. Thus, the attention layer works as a selection mechanism. Clustering for output embeddings is considered for transformers in [13]. The authors consider a simplified model of attention and the changes to the input in each layer as neural differential equations. Because of the layer normalization, they picture the evolution of the input as movements on a hyper-dimensional sphere where tokens center on points — creating clusters on that sphere. For the model ALBERT XLarge v2 [14], they show that tokens become more similar to each other in later layers by calculating the inner product for each pair of tokens for each layer. In [8], the role of attention heads is considered, and the attention maps are analyzed. They identify three tasks that heads perform: association, manipulation, and broadcasting. Association globally attends similar tokens, manipulation shares dependencies to other nearby tokens, and broadcasting distributes values from special tokens such as class (`[CLS]`) or separate (`[SEP]`) tokens. To the best of our knowledge, considering clustering for attention maps is a novel concept, which has not been tried previously.

1.2 Aims

Understanding and optimizing the impact of attention weight clustering can significantly advance the performance and interpretability of Vision Transformer models. In this project, we implemented several models with different configurations. We studied how attention blocks affect model performance and investigated whether the number of attention heads influences classification accuracy. To understand the role of attention weights, we selected a dataset and used it to train the models. Various clustering algorithms were employed to cluster attention weights. We also examined how the manipulation of clustering results impacts classification predictions.

This project aims to delve into the impact of attention weight clustering on model performance, focusing on Vision Transformer (ViT) models. Both model performance and model interpretability can be improved by understanding how clustering within attention maps influences model outputs. Additionally, this project will explore how different clustering algorithms affect clustering outcomes and evaluate the effects of manipulating identified clusters, such as removing background clusters, on model performance.

1.3 Contributions

The project contributes to theoretical advancements by presenting a novel perspective on weight allocation. This perspective allows for new visualization and can lead to improved classification performance.

Visualizing attention maps can offer valuable insights into how Transformers focus on different input parts. One popular method for visualizing these maps is the heatmap, which overlays a color-coded map on the original input to indicate the intensity of attention at each patch. Another approach involves highlighting areas of the image on which the query token focuses. This project introduces a new technique that superimposes clusters derived from attention weights onto the original image. This method simplifies understanding the role and significance of clustering in attention weights by visualizing how the model segments the image.

In this project, we developed a method to analyze the clustering of attention weights, thereby enhancing the interpretability of the model and improving its predictive confidence. By examining the clusters in attention maps, we could identify areas of focus within the input data. This analysis allowed us to diagnose potential issues in model performance, such as the model concentrating on irrelevant parts of the input. Addressing these issues, we discovered that removing irrelevant patches not only increased the confidence of correct predictions but also had the potential to rectify incorrect predictions. In essence, refining the model's focus through attention-weight clustering could significantly enhance its predictive capabilities.

2

Theory

Machine-learning relies heavily on mathematical principles. Understanding fundamental mathematical concepts is crucial when modifying existing models or developing new ones. Analyzing attention mechanisms also necessitates familiarity with certain theories and algorithms. The theory chapter of this report serves as the bedrock, outlining the conceptual framework that supports our analysis and interpretations.

This chapter will explore essential theoretical constructs such as the Transformer model, graph theory, and various clustering algorithms. Furthermore, it acts as a guide, leading the reader through the landscape that shapes our research methodology and analytical approach. In particular, the chapter outlines the transformer, the graph structure used to analyze the attention map, and community detection algorithms used to find clustering structure in the graphs.

2.1 Image Classification

Image classification is a task in computer vision in which an image is categorized into predefined classes or categories based on visual content. It aims to teach a computer model to recognize and assign image labels accurately. This process typically involves training a machine-learning model using a labeled image dataset, where each image is associated with a specific class label. Many methods can be used for image classification. Traditional machine-learning methods for image classification typically involve extracting features from images and then training a classifier on these features.

Hand-crafted features are often used to capture relevant information about image content that can be used to discriminate between classes. These methods include (1) Histograms of Oriented Gradients, which computes the distribution of gradient orientations in an image to capture the shape and texture of objects [15]; (2) Local Binary Patterns, which describes the local texture patterns of an image by comparing each pixel with its neighboring pixels [16]; (3) Color Histogram, which represents the distribution of colors in an image to capture information about color composition; and (4) Scale-invariant feature transform, identify stable points in a separate scale space [17]. These methods capture relevant information about the image content that can be used to discriminate between classes. Once features are extracted from an image, they are encoded as a fixed-length vector that can be used as input for a

machine-learning model. After feature extraction and encoding, a machine-learning classifier is trained to map the input feature vectors to the corresponding labels.

Some popular traditional machine-learning algorithms for image classification include support vector machine (SVM), k-nearest neighbors (k-NN), decision trees, random forests, and the naive Bayes classifier. During training, the model learns the decision boundaries in the feature space and can separate different classes. The performance of a model is evaluated on a validation dataset. Evaluation metrics such as accuracy, recall, F1-score, and precision can be used to measure the model's performance. Techniques such as k-fold Cross-Validation may be used to obtain a more robust model. Once the model reaches the desired performance, or improvement has stalled, it can be deployed to classify a new and unseen dataset. The new dataset is transformed to feature space and then classified according to the learnt decision boundaries. The idea is that the decision boundaries express general properties unique to the class, not the dataset, and that the model generalizes.

Traditional machine-learning methods for image classification have been widely used. However, they have the following disadvantages compared to deep machine-learning: (1) HDBSCAN traditional machine-learning often requires hand-crafted feature engineering. (2) The traditional machine-learning models have problems with overfitting when the dataset is small or contains noise. (3) Linear traditional machine-learning models, such as linear and logistic regression, cannot capture non-linear relationships between features and the target. (4) Many traditional machine-learning models require hyperparameter tuning to achieve optimal performance, which can be time-consuming. (5) Many traditional machine-learning models, such as decision trees or support vector machines, may struggle to capture intricate relationships in high-dimensional or unstructured data.

Feature engineering is a vital component of traditional machine-learning, which require extraction of meaningful features from the original data to improve model performance. Vision Transformers (ViTs) reduce the necessity for manual feature extraction. These models can learn high-level representations directly from raw data, making them especially suitable for complex and large-scale image classification tasks.

2.2 Transformers

A transformer model is a deep learning model that uses an attention mechanism. The attention mechanism can assign different weights according to the importance of each part of the input data. This is intended to be similar to human attention and focus on important parts of the input rather than the entire input.

This subsection provides a detailed description of the various components of the Transformer model. In particular, the transformer is built by using a multi-head attention block connected to a multilayer perceptron, visualized in Fig. 3.2. The various parts of the Transformer model is presented in this section, which culminates in an explanation of how the model learns the optimized parameters.

2.2.1 Multi-head attention

The attention mechanism can find connections by using the entire input at once. Relevancy scores express the connections. Let the vector x designate the input, and the desired output be designated as O. First, transform x to a real-numbered vector $\hat{I} \in \mathbb{R}^{n \times d_{\text{model}}}$, e.g., by using a vocabulary method. Then, add positional information to $\hat{I}$ by using some positional encoding, e.g., learnable position embeddings [18], relative position embeddings [19], 2D positional embedding [20], or multi-scale position encodings [21]. Adding position embeddings corresponds to calculating $I = \hat{I} + E$, where E is the positional encoding generated earlier. The goal is to learn the three weight matrices $W^{(Q)} \in R^{d_{\text{model}} \times d_{\text{k}}}$, $W^{(K)} \in R^{d_{\text{model}} \times d_{\text{k}}}$, $W^{(V)} \in R^{d_{\text{model}} \times d_{\text{v}}}$, the query, key, and value weights, respectively. Calculate $Q = IW^{(Q)}$, $K = IW^{(K)}$, and $V = IW^{(V)}$ using these matrices. The matrices lead to the relevancy scores by computing $A = QK^T$, then applying row-wise softmax to create $\hat{A} = \text{softmax}(A \cdot d_k^{-\frac{1}{2}})$. Now, create the output, $O = \hat{A}V$, which connects to another layer of attention or any other structure.

An extension to the described attention mechanism is to use multiple heads. In this case, each head has its own $W^{(Q)}$, $W^{(K)}$, and $W^{(V)}$ matrices. Calculate O for each head, then concatenate the heads and right multiply with $W^{(O)} \in \mathbb{R}^{n_{\text{heads}} d_{\text{v}} \times d_{\text{model}}}$ to create $\text{concat}(O_0, O_1, \ldots, O_{n_{\text{heads}}})W^{(O)}$, where $W^{(O)}$ are additional learnable weights. The idea is that the heads will specialize in a specific connection type in the input data.

2.2.2 Multilayer perceptron

A multilayer perceptron (MLP) is a standard structure in deep neural networks. Notably, the Transformer uses an MLP after the multi-head attention. An MLP is a feed-forward layer that contains at least three layers with non-linear activation functions. Denoting the n-dimensional input as $x \in R^{1 \times n}$, the hidden weights as $W_1 \in R^{n+1 \times n_{\text{hidden}}}$, and output weights as $W_2 \in R^{n_{\text{hidden}+1} \times m}$, the output is given by $y = \text{concat}(1, \sigma(\text{concat}(1, x)W_1))W_2$, where sigma is the non-linear function. Some common choices for the non-linear functions are hyperbolic tangent tanh, RELU $\max(0, x)$, and GELU $x \cdot \Phi(x)$, where Φ is the cumulative distribution function for the Gaussian distribution.

2.2.3 Logistic regression

Logistic regression, used in machine-learning, is a supervised learning classifier. Like linear regression, the main difference lies in logistic regression's focus on finding a hyperplane boundary that can divide the data into two categories. In contrast, linear regression aims to find a hyperplane that minimizes the distance of each data point to that hyperplane. In other words, logistic regression is primarily used for binary classification, presenting probabilities ranging from 0 to 1. On the other hand, linear regression is mainly used for prediction and is applicable for predicting numerical values, such as price indices. For general linear regression, the dependent variable must be continuous. In contrast, the dependent variable of logistic regression

is categorical. For two categories, binary logistic regression is used; otherwise, multinomial logistic regression is used.

Softmax regression is a generalization of logistic regression. Logistic regression is suitable for binary classification problems, while softmax is suitable for multi-classification problems.

Mathematically, the softmax function takes as input a vector $z = (z_1, z_2, \ldots, z_n)$ and gives an output vector $\sigma(z) = (\sigma(z_1), \sigma(z_2), \ldots, \sigma(z_n))$, where $\sigma(z_i)$ is calculated by using the following equation:

$$\sigma(z_i) = \frac{e^{z_j}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.1}$$

where e is the base of the natural logarithm. The denominator is the sum of the exponential values of all elements in the input vector. The softmax function converts arbitrary real-valued scores into a vector of probabilities that sum to 1.

2.2.4 Learning

In deep learning, optimizing weights works by considering a loss function, which measures how well the network performs. There are several different loss functions to handle different objectives. Given the possible input $\mathcal{X}$ and possible output $\mathcal{Y}$, the objective for classification is to find the function $f : \mathcal{X} \to \mathcal{Y}$ that best predicts the value $y \in \mathcal{Y}$ given input $x \in \mathcal{X}$. A loss function for classification typically quantifies the disparity between predictions from f and actual labels from a training set. One such function, for the set of classes C, is the Cross-Entropy loss:

$$l(f(x), y) = \log\left(\sigma(f(x)) \odot y\right) \tag{2.2}$$

Note that the network's output $f(x)$ is a vector of size $|C|$. The function σ is the softmax function, shown in Sec. 2.2.3. The operation $\odot$ denotes the Hadamard product or element-wise product. Note that the true label, y, uses one-hot encoding. For regression, two standard options are the mean absolute error (L1 loss) and mean square error (squared L2 norm). A reward function is used instead for specific problems, especially for reinforcement learning. It gives high values for desired output in some user-specified regard.

In practical applications, learning is usually performed batch-wise, which means that the same iteration of f evaluates several elements in the training set. There are multiple reasons to use batches instead of using a single element:

1. It speeds up training by utilizing parallelization on the graphics processing unit (GPU) and thus requires fewer iterations.
2. It reduces memory requirements as only the batch and model must be kept in memory while loading enough data to use the GPU effectively.
3. It may improve convergence [22].

Typically, the loss function is calculated and combined for the entire batch. Standard options to combine the output are to sum or to calculate the mean.

Learning the correct weights for a deep learning model is usually performed via gradient descent and the algorithm backpropagation (backward propagation of errors). Other optimization techniques, such as Ant Colony Optimization [23], can also be used to find optimal values for the parameter matrices. The essential part of the backpropagation algorithm is to use the chain rule to split the gradient calculation backward over the model's layers. Starting at the last layer avoids calculating intermediate terms for the gradients multiple times. Parallelization of the calculations is done to enable learning on large networks. Often, the parallelization is achieved via matrix multiplications performed on GPUs.

2.3 Graph

Graph theory is a branch of mathematics that studies the relationships and properties of graph structures. A graph consists of nodes and edges connecting the nodes. Graphs are often used to describe specific pair-wise relationships between certain entities. The nodes represent entities, and the edges represent the relationships between the nodes. A graph G is generally described by a set V of nodes and a set E of edges, denoted as $G(V, E)$, where V represents all the nodes $(v_1, v_2, \ldots, v_n)$ in the dataset.

Weighted graphs have weight values associated with the edges. The weight w_{ij} is defined as the weight between node v_i and node v_j. For two nodes, v_i and v_j, that are connected by an edge $w_{ij} > 0$ otherwise $w_{ij} = 0$.

Graphs can be either directed or undirected. An undirected graph is a graph where edges have no inherent direction. In other words, the relationship between nodes in an undirected graph is symmetric. If an edge connects node A to node B, there is also an edge from B to A. In an undirected graph, $w_{ij} = w_{ji}$. A directed graph is a type of graph in which edges have specific directions associated with them. Unlike undirected graphs, the relationships between nodes in a directed graph are asymmetric. The asymmetric relationship means that if there is an edge from node A to node B, there is not necessarily an edge from node B to node A.

A weighted directed graph is a type of directed graph in which each edge has an associated numerical value known as a weight. The main components of a weighted directed graph are as follows:

- Nodes: These are the fundamental units of the graph, representing patches in this project.
- Edges: These are the directed connections between pairs of nodes. Each edge has a direction, indicating a one-way relationship from a source node to a target node.
- Weights: Each directed edge has an associated weight. This project's weights are the relevancy scores gathered from the attention layer's parameter matrix.

- Adjacency: In a weighted directed graph, node u is adjacent to node v if there is a directed edge from u to v with a certain weight.

This project proposes the graph structure to represent connection between tokens in attention maps. Nodes represent tokens, edges represent connection between tokens, and weights represent relevancy scores between tokens.

2.4 Clustering and Community Detection

Clustering is a technique used to group a set of objects such that objects in the same group (also called a cluster) are more similar than objects in other groups. It is an unsupervised learning approach; the algorithm learns to group the data without prior knowledge of the structure of groups. The goal is to find inherent structure in the data by organizing it into groups or clusters based on similarity. Similarity is typically measured using distance metrics, where objects that are closer to each other, in some regard, are considered more similar. The principle used to establish similarity may differ, leading to various methods for clustering. These can mainly be divided into partition-based, density-based, and hierarchical methods.

Partition-based clustering methods are a category of clustering algorithms that partition the data into a set of disjoint clusters, where each data point belongs to exactly one cluster. These methods iteratively refine the data partitioning until specific convergence criteria are satisfied. One of the most well-known partition-based clustering algorithms is K-means.

Density-based clustering methods are a category of clustering algorithms that partition the data based on the density distribution of the data points in the feature space. Unlike partition-based methods like K-means, which rely on defining centroids and partitioning the data into distinct clusters, density-based methods identify regions of high density as clusters and separate them from low-density regions. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular density-based clustering algorithm.

Hierarchical clustering is a set of clustering methods that build a hierarchy of clusters by either iteratively merging smaller clusters into larger ones or splitting larger clusters into smaller ones. This hierarchical structure is represented as a tree-like diagram called a dendrogram.

Communities do not have a precise definition. One of the most widely accepted and used definitions is given by [24] as:

> A community is a subgraph containing nodes that are more densely linked to each other than to the rest of the graph, or equivalently, a graph has a community structure if the number of links into any subgraph is higher than the number of links between those subgraphs.

Community detection for the graph $G = G(V, E)$ is to determine the number of communities, $n_c \in R$ ($n_c \geq 1$), in the graph such that

$$C = \{C_1, C_2, ..., C_{n_c}\}$$

The node set of each community constitutes a coverage of V. If the intersection of the node sets of any two communities is empty, then C is called a disjoint community; otherwise, it is called an overlapping community.

2.4.1 K-means

K-means is one of the popular partition-based clustering algorithms used to partition a given dataset into K distinct, non-overlapping clusters. The "K" in K-means represents the number of clusters the algorithm seeks to create.

The algorithm starts by randomly choosing k data points from the dataset to serve as the initial cluster centroids. Each data point is assigned to the nearest centroid to form k clusters based on a distance metric. The distance metric can be Euclidean distance, Manhattan distance, Chebyshev distance, Cosine similarity, or Hamming distance. However, the typical distance metric is Euclidean distance. After all data points are assigned to clusters, update the centroids of the clusters. The centroids of the clusters are recalculated as the mean of all data points assigned to each cluster. Repeat the assignment and update the centroids until they no longer change or the maximum number of iterations is reached.

K-means is computationally efficient and widely used for clustering of large datasets. However, it is sensitive to initial centroid selection, and the number of clusters must be set beforehand. The algorithm may not converge to a global optimum with an unfortunate initialization, which means it must be executed multiple times to avoid a suboptimal solution. Another issue is that the number of clusters in a given data set often is unknown. Lastly, k-means performs poorly when clusters have different densities, for example, when clusters are not spherical or their sizes are inconsistent.

2.4.2 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is an algorithm for partitioning a dataset into the minimum number of clusters such that pairs of points in the clusters can be reached with the same density. DBSCAN is calculated by finding points with sufficient nearby points in the neighborhood, the hyperparameter `min_samples`, given by the distance hyperparameter ε. The points with enough density are considered core points and assigned to clusters. Each non-core point is assigned to a nearby cluster if it is within ε distance away from the cluster; otherwise, it is marked as noise. Setting the hyperparameters, `min_samples`, and ε is complex, and the algorithm only works if the density is the same in multiple clusters.

2.4.3 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) is an algorithm similar to DBSCAN, but it makes setting hyperparameters easier and works with varying densities. The first step is calculating a distance metric for each pair of data points. The value of this metric, called mutual reachability, is determined by considering the distance to the k-th closest

neighbor for each point and the distance between the points; the value is set to the largest of these distances [25]. If $d : \ P \times P \rightarrow R$ is some distance function and $d_k : \ P \rightarrow R$ is the distance function to the k-th closest point from the input. Then reachability $= \max(d_k(p_1), d_k(p_2), d(p_1, p_2)) \ \forall \ p_1, p_2 \in A$ where $A = P : p_1 \neq p_2 \wedge p_1 \in D \wedge p_2 \in D$. The idea is to find dense data, and the k-th closest neighbor works as a proxy for density. A graph can be found by considering data points as vertices and pair-wise mutual reachability as weighted edges between vertices [25]. A tree of connected nodes is then found by considering the graph's minimum spanning tree [25], which connects vertices to their closest neighbor. Kruskal's algorithm is then applied to the minimum spanning tree to find a dendrogram of the vertices. The dendrogram is then merged, top-down, by only allowing a split in the tree if each side has at least `min_number` vertices. The distance from the tree's root is recorded if a split is not allowed. The hyperparameter `min_number` differs for each data set and must be provided. Selecting clusters is performed by calculating the stability score for each point in the merged dendrogram. These points are at least `min_number` big. All bottom points in the dendrogram are selected, and then, working up the tree, if a point has a higher stability score, it is selected instead, and all descendants are unselected. When the root has been reached, all the clusters have been found. The stability score is calculated as

$$\sum_{v \in point} \frac{1}{d(v, root) - d(v_{start}, root)},$$

where d is the earlier distance measurement and v_{start} is the point where the cluster splits from its parent [25]. The set *point* is the set of points in the dendrogram.

2.4.4 Phase transitions in semidefinite relaxations

Regular clustering algorithms can be applied to graphs by using some transform to turn the graph into euclidean space embeddings. One method proposed in [26] was developed for an undirected and unweighted graph.

$$\begin{aligned} &\max \sum_{(i,j) \in E} X_{ij} \\ &s.t. X \succeq 0, X1 = 0, X_{ii} \forall i \in 1..n \end{aligned} \tag{2.3}$$

The method, [26], works by considering a problem similar to the semidefinite program eq. 2.3, from [26], which is a relaxation of the problem of estimating community memberships. The authors, [26], found this simplification by considering $x_{MLE} = argmax_{x \in 1, -1^n} (\sum_{i,j \in E}) x_i x_j : \sum_i x_i = 0$. The objective function will have a high value when x has the same sign, which means edges in clusters with tighter connections have the same sign. The sum given in the constraint ensures equal positive and negative numbers. However, since this program is demanding to solve, it can be rewritten as a semidefinite program eq. 2.3.

The authors, [26], considered the related non-convex optimization problem,

$$\begin{aligned} &\max \sum_{(i,j) \in E} \sigma_i \cdot \sigma_j - \frac{\gamma}{2} ||M||_2^2 \\ &s.t. ||\sigma_i||_2 = 1 \ \forall i \in 1..n. \end{aligned} \tag{2.4}$$

Where $\sigma_i \in R^m$ is vector spin on a node and $M = \sum_{i=1}^{n} \sigma_i$. All σ_i together create $\sigma = (\sigma_1, \sigma_1, ..\sigma_n) \in (R^m)^n$, which was derived from decomposing the positive semidefinite matrix X as $X = \sigma\sigma^*$. The hyperparameter m was set to 40, which was shown by [26] to work well even for large graphs. After solving eq. 2.4, an empirical covariance matrix can be created, $\widehat{\Sigma} = \frac{1}{n_{\text{tokens}}} \sum_{i=1}^{n} \sigma_i \sigma_i^T$. This covariance matrix shows how the m spin variables relate to each other. The eigenvectors and eigenvalues for X were calculated, and the first $m_{\text{needed}} \leq m$ eigenvectors were kept. The number to keep was decided based on the variance of the covariance matrix that they could express. Finally, the eigenvectors were used to create a lower-dimensional embedding with size $m_{\text{needed}} \leq m \leq n_{tokens}$. This embedding can then be used with regular clustering algorithms such as K-means, DBSCAN, or HDBSCAN.

2.4.5 Infomap

Infomap is an algorithm used in network science to detect community structures within complex networks [27]. The problem that Infomap initially aimed to solve concerns how to use the shortest code to describe the path generated by the random walk if the number of random steps is not limited. The original approach uses equal-length binary codes. Each node has its own code, and the encoding between different nodes differs. The advanced approach is to use Huffman encoding with one code for each node, but varying the code lengths. Nodes with higher access frequency are given shorter codes, and those with lower access frequency are given longer codes.

The method used in the Infomap algorithm involves a two-layer structure to divide different nodes into groups, requiring the encoding of two types of information [27]. The first type is the name of the group. Different groups have different name encodings. The second type is the nodes within each group. The name encodings of different nodes are unique within one group. However, the coding of nodes within different groups can be reused. After dividing the groups, there are relatively few internal nodes in each group, allowing for shorter coding. Multiplexing the coding of internal nodes in different groups can significantly reduce the length of the described information.

The Infomap algorithm provides the group name encoding and assigns a code to each group's exit action to distinguish the random walk from one group to another [27]. For example, for group A, the group name is given the code 111, and the code to exit the group is 0001. Thus, the Infomap algorithm ensures that describing a random walk path within a specific group always begins with the code of the group name and ends with the exit code.

The length of the quantization code is calculated as follows: Assuming there is a group division method M that divides nodes into m groups, the average number of bits per step describing the random walk can be measured by the following formula [27]:

$$L(M) = q \curvearrowright H(\mathcal{Q}) + \sum_{i=1}^{m} p_{\circlearrowright}^{i} H(P^i) \tag{2.5}$$

Where $q \curvearrowright$ is equal to $\sum_{i=1}^{m} q_i \curvearrowright$. $q_i \curvearrowright$ is the probability that the random walker switches modules on any given step. $\sum_{i=1}^{m} q_i \curvearrowright$ is the proportion of all codes

representing group names. $H(\mathcal{Q})$ is the frequency-weighted average length of codes in the index codebook. $H(P^i)$ is the frequency-weighted average length of codes in the module codebook i. $p^i_{\circlearrowleft}$ is the proportion of codes for all nodes belonging to group i (including jumping out nodes).

To calculate the values of the four variables in eq. 2.5, from [27], we need to know the visit probability of each node in the graph and the transition probability of each group. Infomap employs a PageRank-like, [28], approach to compute the visit probability.

1. All nodes have uniform access probability initially.
2. In each iteration step, there are two ways to jump from one node to another.
   - Select one edge from the connecting edges. The probability of transition is $1 - r$. The probability of selecting each edge is proportional to the weight of the edge.
   - Randomly jump from node a to any other point on the graph with probability r.
3. Repeat step 2 until convergence.

When the visit probability converges, the visit probability of each node in the graph is obtained. The exit probability for module i can be calculated using the eq. 2.6, from [27].

$$q_i \curvearrowright = \tau \frac{n - n_i}{n - 1} \sum_{\alpha \in i} p_\alpha + (1 - \tau) \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha w_{\alpha\beta} \tag{2.6}$$

Where n_i is the number of nodes in module i. Every node teleports with probability $\tau \frac{n-n_i}{n-1}$ and guides a fraction $(1 - \tau) \sum_{\beta \notin i} w_{\alpha\beta}$ of its weight p_α to nodes outside of the group.

2.4.6 Louvain

The Louvain algorithm [29] is an iterative algorithm that optimizes modularity, Sec. 2.4.7, function to find communities that maximize the density of connections within communities while minimizing connections between communities. The Louvain algorithm operates as follows:

1. Initially, each node is assigned to its own community.
2. It iterates over each node and moves it to the community of each of its neighbors. If the move increases the network's modularity, it is accepted.
3. After each round of node movement, the communities found in the previous step are aggregated into single nodes, and a new network is constructed. The nodes of this new network represent the clusters found in the previous step.
4. Repeat steps 2 and 3 until no further improvement in modularity can be achieved.

2.4.7 Modularity

Modularity is a measure that quantifies the strength of the division of a network into modules or communities. It assesses how well a network is partitioned into distinct groups of nodes, with dense connections within modules and sparse connections between modules. A higher modularity score indicates a more substantial community structure within the network. Modularity (Q) score is defined as follows:

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \gamma \frac{k_i k_j}{2m}) \delta(c_i, c_j) \tag{2.7}$$

Where $A_{i,j}$ is the weight of the edge between nodes i and j. k_i and k_j denote the degrees of nodes i and j. m is the total edge weight in the network. γ is the resolution parameter. $\delta(c_i, c_j)$ is the Kronecker delta function. If node i and j belong to the same community (cluster), $\delta(c_i, c_j)$=1 and 0 otherwise.

2.4.8 Jaccard similarity index

The Jaccard similarity index, also known as the Jaccard similarity coefficient, measures the similarity between two data sets. It ranges from 0 to 1. A higher score indicates greater similarity between the two sets.

Given two sets, X and Y, their Jaccard Similarity is calculated with

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}, \tag{2.8}$$

where $|X \cap Y|$ is the size of the intersection of the sets, and $|X \cup Y|$ is the size of the union of the sets.

3

Methods

The methodology chapter of this report delineates the techniques employed to fulfill the research objectives and address the hypothesized questions. This chapter outlines the procedures, methodologies, and tools used in dataset selection, model construction, and attention weight analysis and interpretation. Fig. 3.1 shows the method workflow used in attention weights analysis. We started our project with model implementation. We trained one model with an attention layer and one without by using the CIFAR10 [30], CINIC10 [31], Fashion-MNIST [32], and TFFlower [33] datasets. Then we selected the TFFlower data set due to its high-resolution images and increase in model classification performance when using the model with an attention layer. Afterward, we extracted attention weights from the models with different structures, treated attention maps as graphs, and used different clustering algorithms to study if clustering exists within the graphs. Finally, we nullified output embeddings belonging to the communities or clusters we were not interested in and then studied whether nullifying impacted the model's performance.

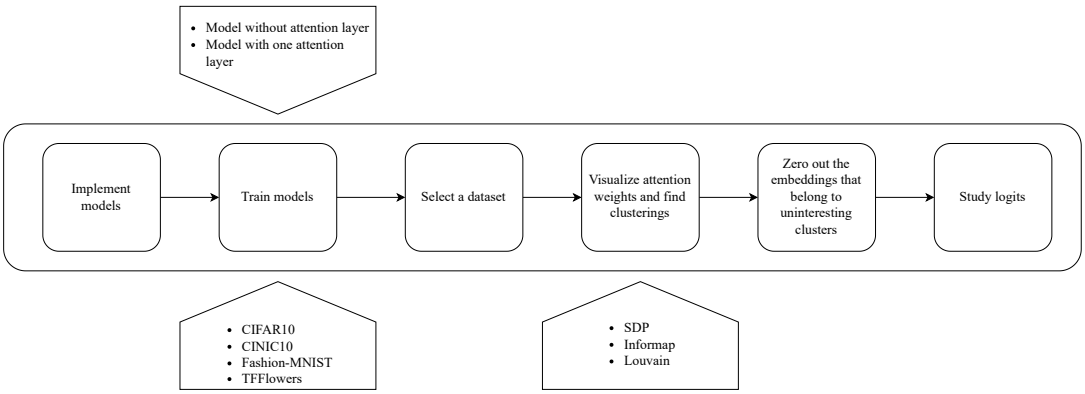


Figure 3.1: The workflow used in this project involves model implementation, dataset validation, attention weights visualization, clustering detection, and analysis. Two implemented models were trained using four different datasets with and without an attention layer. The attention weights from the selected dataset and images are extracted and analyzed to uncover clusters. Finally, the clusters are separated, and their impact on the output logits is considered.

3.1 Model Implementation

We implemented the model in Pytorch [34] and based it on the method described in the paper *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [35]. However, we restricted the number of transformer blocks to one. The structure of the models is shown in Fig. 3.2. The dashed line and solid line indicate the workflow for the model without and with using one attention layer, respectively.

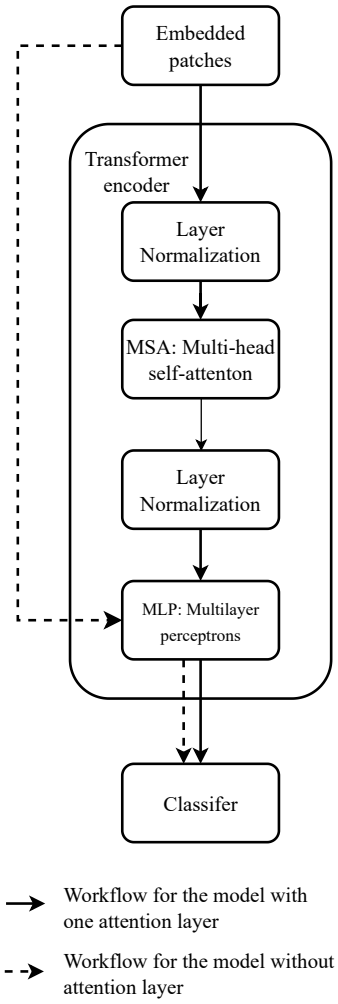


Figure 3.2: Overview of the implemented models. The solid line is the workflow of the simplified Vision Transformer model with one attention block. The overview showcases the sequential steps from image input to the final classification result, notably patch embedding, transformer encoder, and classification output. The dashed line is the workflow for the model without using the attention mechanism.

3.1.1 Embedded patches

We divided the image into fixed-size image patches, performed linear embedding, added position information to each image patch, and fed the resulting sequence of patch tokens into a simplified attention layer. The classification was done on the output from the attention layer or the output from patch embedding.

Fig. 3.3 illustrates the patch and position embedding process using a sample from an image. The example image size is 28×28 pixels, with three channels. Suppose the size of each patch is 7×7. Then, the image is divided into 16 non-overlapping patches. Each patch is flattened into a one-dimensional row vector with the shape $(1, 147)$. Other patch sizes and image sizes can also be used with a similar technique. Subsequently, each flattened patch is linearly projected using learnable weights. Finally, the patch embeddings are concatenated with the positional embeddings from the input sequences.

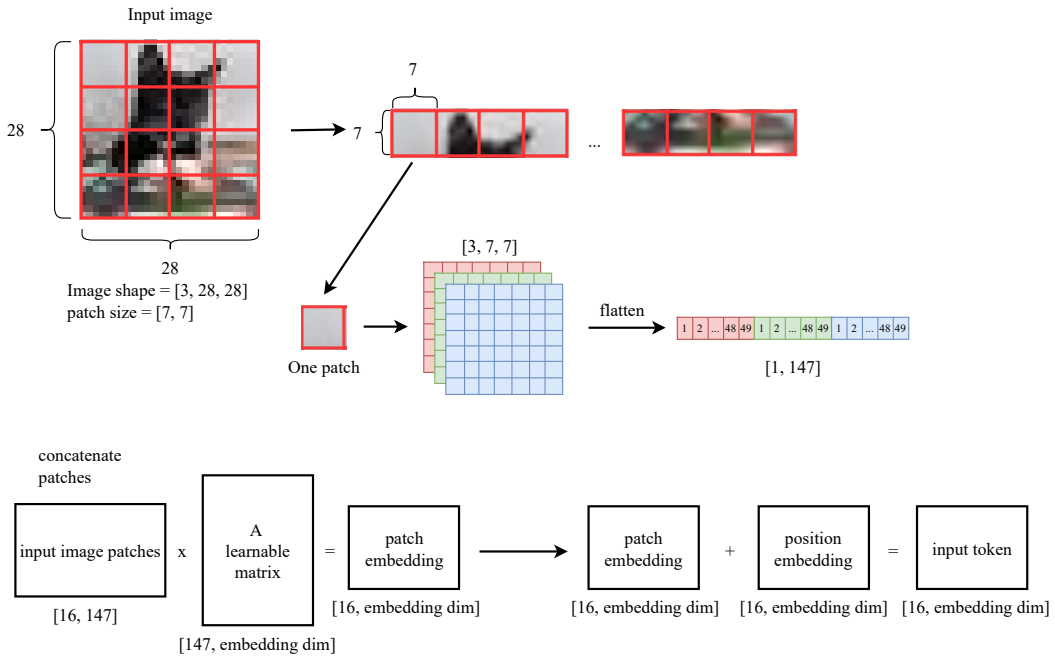


Figure 3.3: Illustration depicting the process of patch embedding, where an image is divided into patches and transformed into embeddings for a vision transformer model. The first step shows the separation into patches. The second step shows the flattening of a single patch. Finally, the bottom part of the figure shows the mathematical operations to create input tokens.

3.1.2 Attention layer

The multi-head self-attention block (MSA) contains two procedures (see Fig. 3.2). The first procedure is to apply normalization to the token embeddings. The second procedure captures dependencies between patches in the multi-head self-attention block (MSA). Fig. 3.4 shows the process of calculating attention weights in the model

with one attention layer.

3.1.2.1 Attention weights and attention output

We followed the following steps to obtain attention weights and attention output.

- Step 1: Calculated Query, Key, and Value matrices The query matrix $W^{(q)}$, the key matrix $W^{(k)}$, and the value matrix $W^{(v)}$ are typically initialized as trainable parameters of the model. Their shape depends on the dimensions of the input tokens and the embedding dimension. We simply multiplied the input token embeddings by the query matrix to compute the queries. This computation can be represented as:

$$Q = XW^{(q)} \tag{3.1}$$

  Where X is a matrix of shape batches $\times$ $n_{\text{tokens}} \times d_{model}$, n_{batches} is the batch size, n_{tokens} is the length of input tokens, and d_{model} is the dimensionality of the embeddings. $W^{(q)}$ is the query matrix of shape $d_{model} \times d_{model}$. Q is a matrix of shape $n_{\text{batches}} \times n_{\text{tokens}} \times d_{model}$, representing the queries for each input token in each sequence. We used the same way to obtain the Key matrix $W^{(k)}$, and the Value matrix $W^{(v)}$.

$$K = XW^{(k)} \tag{3.2}$$

$$V = XW^{(v)} \tag{3.3}$$

  Where $W^{(k)}$ and $W^{(k)}$ represent the Key and Value matrices, respectively. They both have a shape of $d_{model} \times d_{model}$. K and V are matrices of shape $n_{\text{batches}} \times n_{\text{tokens}} \times d_{model}$, representing the keys and values for each input token in each sequence. K and V have the same shape as Q.

- Step 2: Computed QK^T.
- Step 3: Scaled the QK^T by dividing $\sqrt{d_k}$, where d_k is the embedding dimension.
- Step 4: Took softmax of $\frac{QK^T}{\sqrt{d_k}}$.
- Step 5: Multiplied the result from step 4 with the value matrix V.

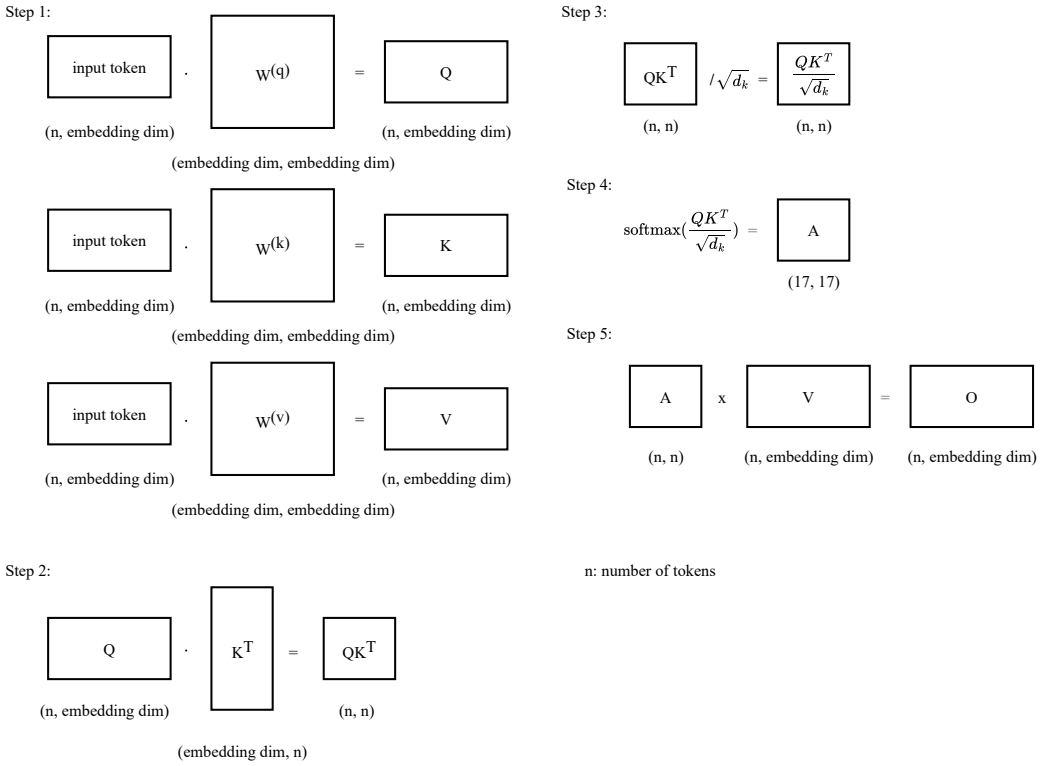


Figure 3.4: Detailed schematic of the intricate steps involved in deriving self-attention weights and self-attention output for the sample image shown in Fig. 3.3. It illustrates the pairwise comparison process between patch embeddings through dot-product attention, followed by scaling, softmax normalization, and weighted aggregation.

The output of step 4 consists of attention weights for a model with one attention head. The output of step 5 is the attention output.

3.1.3 Attention weights for multi-head self-attention heads

According to the steps in the Sec. 3.1.2, the attention weight for each head can be calculated by the following equation:

$$A_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_v}}) \tag{3.4}$$

Where Q_i and K_i are Query and Key matrices for each head, d_v is equal to the embedding dimension divided by the number of heads. The split Q_i and K_i have shape $n_{\text{tokens}} \times d_v$, where n_{tokens} is the length of the input tokens and d_v is the embedding dimension divided by the number of heads.

The following equation was used to obtain the attention output,

$$O_i = A_i V_i. \tag{3.5}$$

Where V_i is a value matrix with shape $n_{\text{tokens}} \times d_v$.

We used eq. 3.4 to compute the attention weight for each attention head. However, each head had its own set of Query and Key matrices, resulting in multiple sets of attention weights. Fig. 3.5 illustrates calculating attention weights and output for a vision transformer model with three heads. The Q, K, and V matrices are split into Q_i, K_i, and V_i, where i is equal to 1, 2, and 3. For example, the attention weight A_1 for the first head is equal to $\text{softmax}(\frac{Q_1 K_1^T}{\sqrt{d_v}})$, and the attention output Z_1 for the first head is equal to $\text{softmax}(\frac{Q_1 K_1^T}{\sqrt{d_v}})V_i$. Attention works as a weighted average. Our project focused solely on each head's attention weight matrix (A_i).

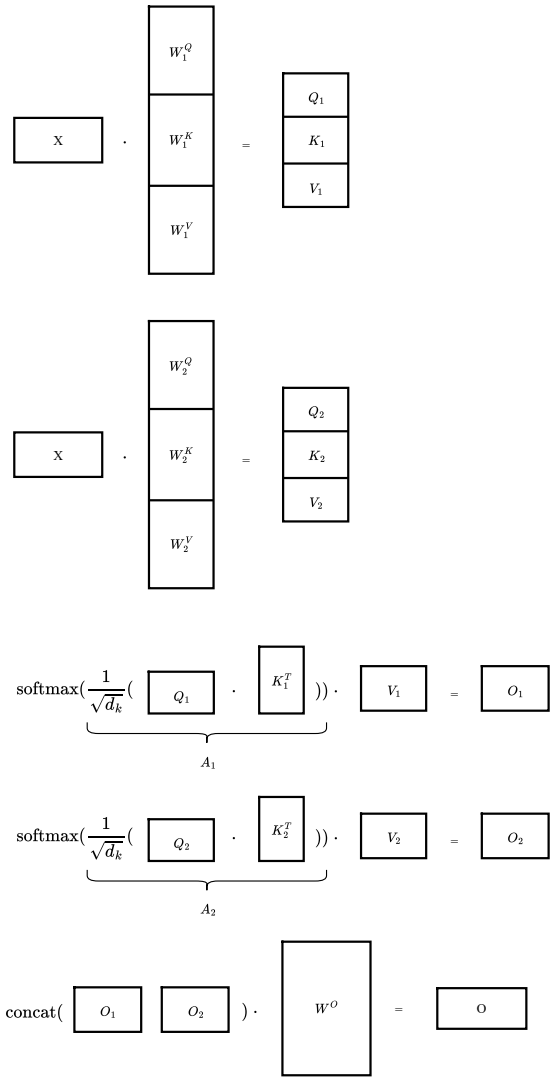


Figure 3.5: Comprehensive breakdown illustrating the intricate process of deriving self-attention weights and self-attention output for multiple heads in a Vision Transformer model. It elucidates the parallel computation of attention scores across multiple heads, showcasing the iterative comparison of patch embeddings, subsequent scaling, softmax normalization, and weighted aggregation for the attention output.

3.1.4 MLP: Multilayer perceptron

We implemented MLPs by multiplying the input with a learnable matrix and adding a learnable bias vector in the standard way. Then, GELU was used for each position in the resulting vector. Another matrix multiplication was then performed, and another bias vector was added. An MLP works on a vector, but the attention layer produces two-dimensional output. Reducing to one dimension was performed by running the MLP row-wise and concatenating the resulting vectors.

3.1.5 Classifier

The two-dimensional matrix was flattened to create a large vector to create an n-class image label prediction. Matrix multiplication was then performed to create an n-sized vector of logits. After that, Softmax was used to generate a probability vector. The probability vector was used with cross-entropy loss.

3.2 Data Acquisition

In our case, we aim to study the attention layer and whether semantic partitioning is vital to its effectiveness. Image data is a good fit. We identified four publicly available datasets: CIFAR10, CINIC10, TFFlowers, and Fashion-MNIST. We used these datasets to train both the model with and without attention.

The CIFAR10 [30] dataset consists of $60,000$ 32×32 color images in ten classes, with $6,000$ images per class. The dataset is divided into $50,000$ training images and $10,000$ testing images. All images in the CIFAR10 dataset are RGB (3 channels) and have a fixed size of 32×32 pixels. The ten classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The CINIC10 [31] dataset combines images from three different sources: CIFAR10, ImageNet, and unique images to CINIC10. It contains a total of $210,000$ images, with $60,000$ images from CIFAR10, $150,000$ images from ImageNet, and $10,000$ images that are unique to CINIC10. The dataset uses the same classes as CIFAR10 and has the same image size and number of channels.

Fashion-MNIST [32] consists of grayscale images, each 28×28 pixels in size. These images represent various types of clothing items. The dataset contains ten classes, each corresponding to a specific type of clothing item: t-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

The TFFlowers [33] dataset contains various flower images captured under different conditions, such as varying lighting, backgrounds, and angles. Each image typically depicts a single flower specimen. The dataset consists of $3,670$ RGB flower images with varying dimensions. The images were spread out over the classes: sunflowers, daisies, roses, tulips, and dandelions.

3.3 Model Training

To assess the performance of the attention mechanism for classification, we utilized the four datasets mentioned in Sec. 3.2 to train models with different structures. Table 3.1 shows the models with different setups.

Table 3.1: The simplest model settings for selecting a dataset and initial study

Model No.	Setup
Model 1	Patch embedding + classifier
Model 2	Patch embedding + One attention layer with 12 heads + MLP + classifier

Models 1 and 2 were trained, one with a single attention layer comprising 12 heads and another without an attention layer. Subsequently, we selected the dataset with high resolution and which exhibited high accuracy for the model with one attention layer. This dataset was TFFlowers.

We then trained a new model using a single attention head used for further analysis. We trained it for 512 epochs using the stochastic gradient descent with momentum algorithm, implemented in [34], with learning rate $\gamma = 3 \cdot 10^{-4}$ and momentum $\mu = 0.9$. We used a batch size 128 and patch size 16×16. For training we split the data set into train, test, and evaluation sets. The train set was the only data the model could see during the training phase. After each epoch the model's accuracy on the evaluation set was recorded; if the evaluation was higher than the previous record, the model weights were saved. This means that the model iteration with best performance on the unseen evaluation set was kept and not necessarily the last iteration. The test set was kept separate and used for later evaluation of the model. We trained the models on two systems, the first using an Intel Core i7-2600k central processing unit (CPU) and a Nvidia Geforce GTX 1060 6GB graphical processing unit (GPU), and the second using an Intel Core i7-12700H CPU and a Nvidia Geforce RTX 3070 8GB GPU. Training time was less than one day for all models.

3.4 Attention Weights Visualization

After extracting the attention weights for each attention head, we plotted the attention weights from one head as a heatmap. In the heatmap, each block corresponds to a particular token. Brighter blocks indicate greater attention, revealing which patches are more influential in understanding the content of other patches.

We also visualized a heatmap for each head by specifying a query patch ID and overlaying the heatmaps on the original image. Visualization was done to enhance readability with attention heatmaps.

3.5 Modify Model Structure

We varied the structure to find a good model for the task. We emphasized finding simpler models to decrease the analysis required.

3.5.1 Vary the number of attention heads and add multilayer perceptron

To investigate whether the number of attention heads affects classification accuracy, we varied the number of heads assigned to the model with a single attention layer. Subsequently, we trained the models using the selected dataset to obtain attention weights. Attention layers with one, two, four, six, eight, and twelve heads were chosen to train models. The accuracy of the test set for individual classes and the total was compiled and analyzed to select the number of heads to use.

3.6 Clustering

This section describes the clustering and community detection methods we employed to analyze attention weights. Fig. 3.6 presents four images from the TFFlower dataset utilized in the clustering analysis.

Figure 3.6: One sunflower, one tulip, and two rose images were used, all of which are from the TFFlowers [33] dataset. The file name for the **Sunflower** image is 1022552002_2b93faf9e7_n.jpg. The file name for the **Tulip** image is 14957470_6a8c272a87_m.jpg. The file names are 6241886381_cc722785af.jpg for **Rose 1** and 16051111039_0f0626a241_n.jpg for **Rose 2**.

We considered the attention maps as adjacency graphs, A_i is the attention weights for head i. Typically, all entries in the attention maps are non-zero. When these elements are non-zero, every node is connected, and no clusters exist since every

node is connected. Some edges must be pruned to avoid every node ending up in the same cluster. We pruned the graph by removing all edges less than a given threshold hyperparameter, $c_{\text{edge\_min}}$. For "SDP", the adjacency graph, A_i, was transformed into a symmetric and binary adjacency matrix using $\hat{A}_{i,j,k} = (A_{i,j,k} > c_{\text{edge\_min}}) \vee (A_{i,k,j} > c_{\text{edge\_min}})$. For Infomap and Louvain, all values less than the threshold were removed.

3.6.1 Graph clustering via phase transitions in semidefinite relaxations (SDP)

We recreated the algorithm in Sec. 2.4.4 using Python. To ensure correctness, the new version was compared to a C program written by the original authors of [26]. The program was reimplemented because the original implementation could not load and output Python structures. Modifying it would have required more time than reimplementing it. We used the embedding generated by the Python program to find clusters with regular HDBSCAN, see Sec. 2.4.3. For HDBSCAN, we used the scikit-learn, [36], library's implementation, as it is a well-known and proven library. Henceforth we call this combined technique "SDP".

3.6.2 Louvain

NetworkX [37] is a Python package designed for creating, manipulating, and studying complex networks' structure, dynamics, and functions. The Louvain function in the NetworkX package is used for community detection in adjacency matrices.

3.6.3 Infomap

We used the package Infomap [38], published by the original author for [27]. This package applies the Infomap algorithm by using an optimized C++ implementation.

3.7 Investigate the Role of Clustering in Attention Maps

To explore the role of clustering in attention maps, we employ community detection, or clustering algorithms, to identify communities and clusters within the attention maps. The study process is outlined as follows:

1. Obtain Attention Map: Generate the attention map for the input image.
2. Convert to Adjacency Matrix: Treat the attention map as an adjacency matrix.
3. Construct Directed Graph: Build a directed graph using the adjacency matrix.
4. Threshold Edge Weights: Remove edges with weights below a specified threshold.
5. Apply Clustering Algorithm: Use a community detection or clustering algorithm to partition the graph into clusters.

6. Calculate Modularity: Compute the modularity score for the resulting clusters.

   Repeat Steps 3-6 for different threshold values.

7. Visualize Modularity Scores: Plot the threshold values against the corresponding modularity scores and the number of clusters obtained.

8. Analyze Results: Study the plot to determine the optimal threshold for edge removal and the corresponding number of clusters that best represent the attention map.

Let $Q(t, k)$ be the modularity score for a given threshold t and number of clusters k. The optimal threshold is determined by using the equation 3.6.

$$t^* = \arg\max_t Q(t, k) \text{ in stable phase} \tag{3.6}$$

Here, t^* denotes the optimal threshold that maximizes the modularity score, Q, for each value in the stable phase. The stable phase is a period of stability where the values for the number of clusters remain constant. The period of stability was determined by manual inspection.

After determining the optimal threshold:

- Assign Cluster Labels: Assign cluster labels to each patch embedding based on the identified clusters.
- Visualize Clusters: Plot the clusters on the image to identify which clusters are less important in the image.
- Adjust Attention Map: nullify the output embedding belonging to uninteresting clusters in the attention map to refine its focus.
- Study logits: Compare logits for the nullified output embeddings with those for the original, unmodified output.

4

Results

This chapter presents the outcomes and findings obtained from our experimental investigations.

4.1 Dataset Selection

We used the four datasets mentioned in Sec. 3.2 to train Model 1 and Model 2; Model 1 did not use an attention layer, and Model 2 used one. The model with a single attention layer contained 12 heads. Table 4.1 displays the accuracy of the model for the datasets using both models.

Table 4.1: Classification accuracy on test portion of data sets when using an attention layer and when not using one. The architectures considered are patch embeddings → logistic regression and patch embeddings → attention layer → logistic regression for without and with, respectively.

Dataset	Accuracy without attention	Accuracy with attention (12 heads)
CIFAR10 [30]	61.08%	67.12%
CINIC10 [31]	56.22%	55.48%
TFFlower [33]	68.12%	72.21%
Fashion-MNIST [32]	90.08%	89.20%

Fig. 4.1 depicts the confusion matrix for the test set of the TFFlower dataset. The model with one attention layer best classifies the sunflower class (91.43%). There is a 4.29% probability for the sunflowers class that the model misclassifies a sample as belonging to the tulips class.

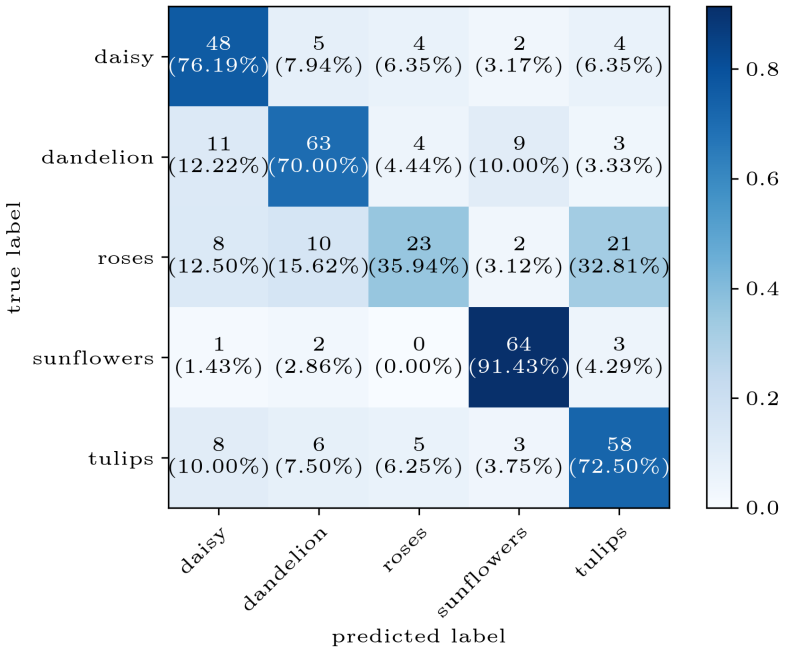


Figure 4.1: The confusion matrix provides a detailed model breakdown with one attention layer and 12 attention heads on the test set from the TFflower dataset. Each cell represents the count of instances classified by the model. Numbers in parentheses indicate percentages, offering a proportional breakdown of each class's performance.

4.2 Visualization

This section gives a view on how attention is distributed for patches in images with correct and incorrect classification results.

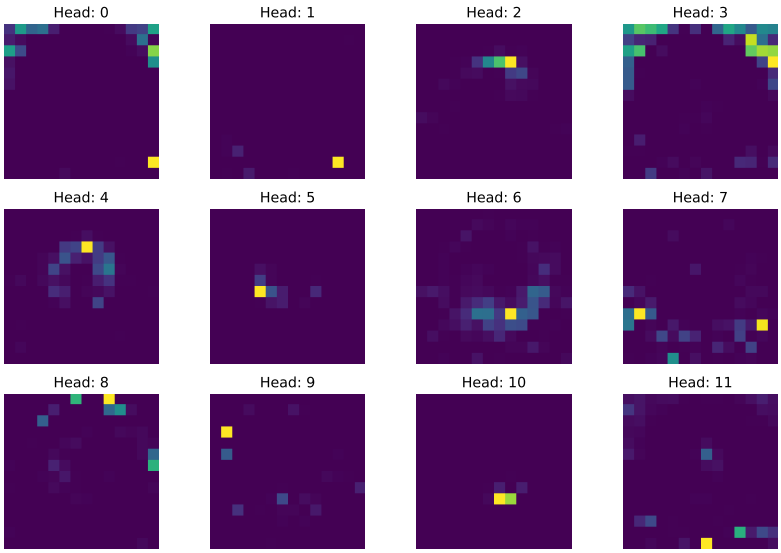
4.2.1 An image with a correct classification

We used the image in Fig. 4.2 as input to obtain attention weights. The red block in the image represents the query patch. Fig. 4.3a displays heatmap plots for 12 attention heads. To visualize which areas of the image the query token attends to, we superimposed the heatmaps on the original image (see Fig. 4.3b). Model 2 made a correct prediction for this image.

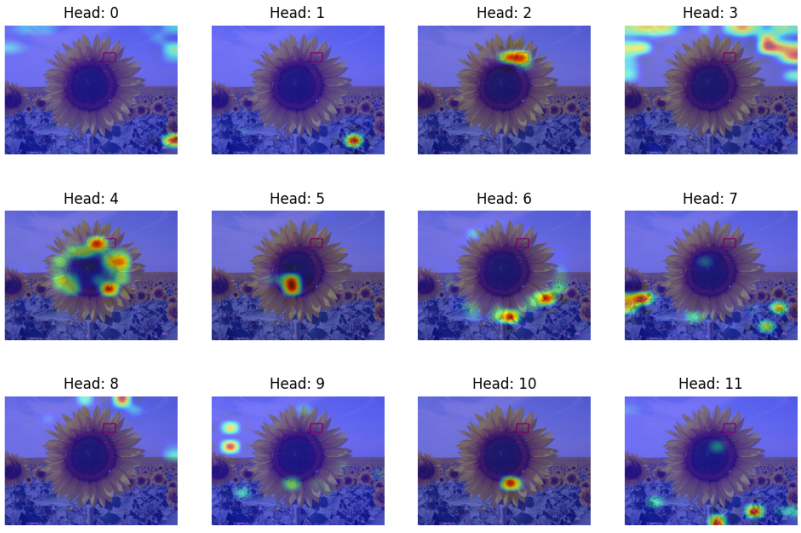
Figure 4.2: The **Sunflower** image with a query patch selected. The red box represents the selected patch (patch ID = 50).

4.2.2 An image with incorrect classification

We iterated through the test set and obtained a tuple of images with incorrect predictions to visualize the attention heatmaps and study the areas on which the given query patch focuses. The selected image is shown in Fig. 4.4, and the query patch ID was 31. The model assigned the class of dandelion for this tulip class image. Fig. 4.5b shows the heatmaps overlay with the original image.



(a) Attention maps from multiple heads depict the focus of each head on a query patch within the image shown in the **Sunflower** image. The visualization showcases the diverse attention patterns captured by individual heads, highlighting their collective contribution to understanding the context surrounding the query patch (patch ID = 50) in the model with one attention layer and 12 heads.



(b) Overlay of attention maps shown in above on the sunflower image in Fig. 4.2, revealing the areas of focus and importance as identified by the model with one attention layer.

Figure 4.3: Attention maps and overlay plot for the **Sunflower** image with correct classification

Figure 4.4: A tulips image with a selected query patch (patch id = 31).

4.2.3 Accuracy for models when varying the number of heads

Fig. 4.6 shows the classes' accuracy and the whole test set for models when varying the number of attention heads. All models use one attention layer.

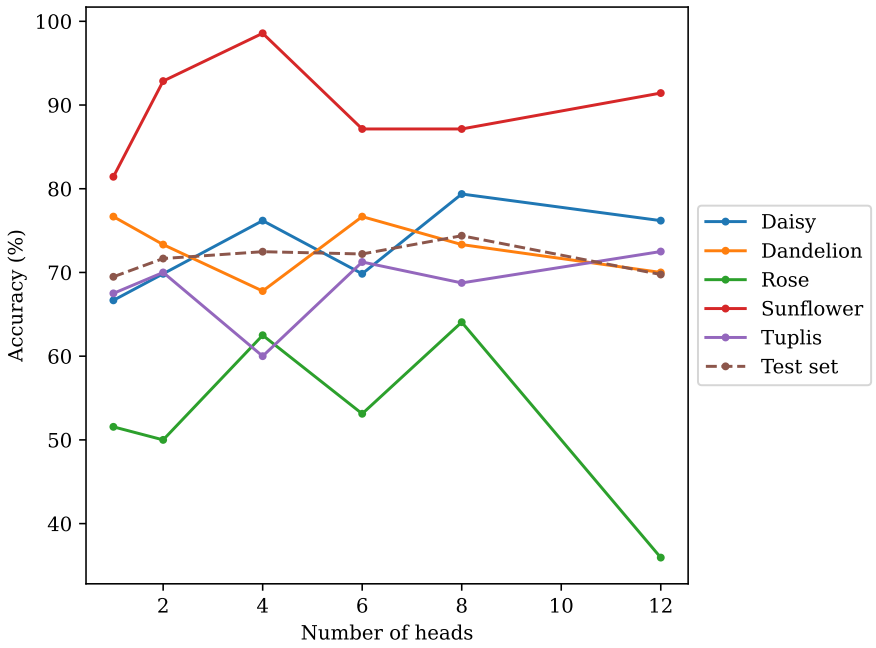
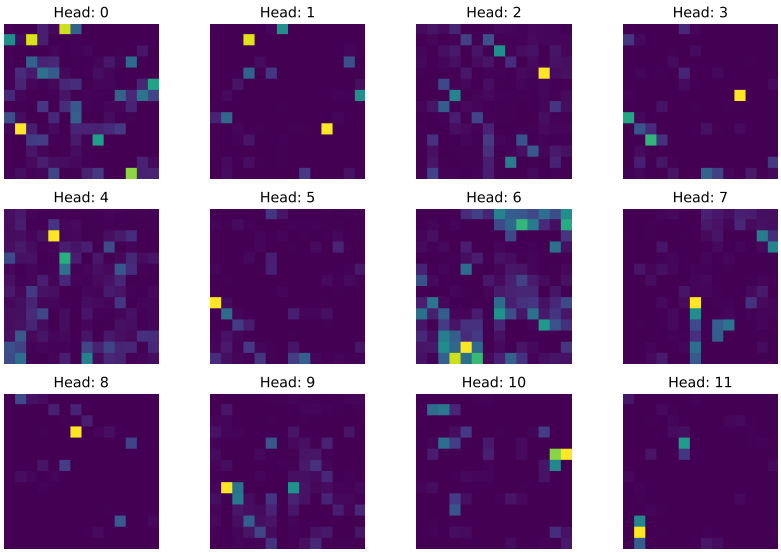
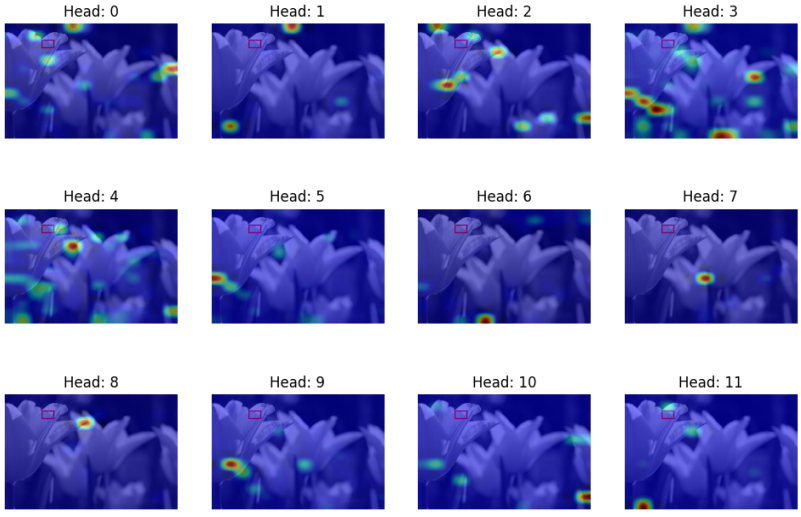


Figure 4.6: Plot for accuracy of the classes and the test set by using models with different numbers of attention heads.



(a) Attention maps from multiple heads illustrate the focus of each head on a query patch within the image depicted in Fig. 4.4. The visualization underscores the varied attention patterns captured by individual heads, emphasizing their collective contribution to comprehending the context surrounding the query patch in the model with one attention layer.



(b) Overlaying the attention maps depicted in the above figure onto the tulips image in Fig. 4.4 unveils the regions of focus and significance, as discerned by the single-layer attention model for the given query patch with ID number 31.

Figure 4.5: Attention maps and overlay plot for the image of tulips with incorrect classification

4.3 Class Prediction for Images

Table 4.2 displays the predictive probabilities for the various images. The predictive results indicate that the model correctly assigned class labels to the **Sunflower** and **Tulips** images. However, the predictions for the **Rose 1** and **Rose 2** images were incorrect, as the model assigned them the labels of Dandelion and Sunflower, respectively.

Table 4.2: The predictive probability (%) for each class.

Image Name	Daisy	Dandelion	Rose	Sunflower	Tulips
Sunflower	0.19	0.20	0.00	**99.33**	0.27
Tulips	0.01	0.00	7.57	0.08	**92.33**
Rose 1	1.57	**58.85**	8.79	25.12	5.66
Rose 2	1.37	5.59	37.04	**43.11**	12.89

4.4 Modularity Score, Number of Clusters, and Optimal Threshold

Using the Louvain algorithm, Fig. 4.7 shows the modularity value plot for the **Sunflower** image.

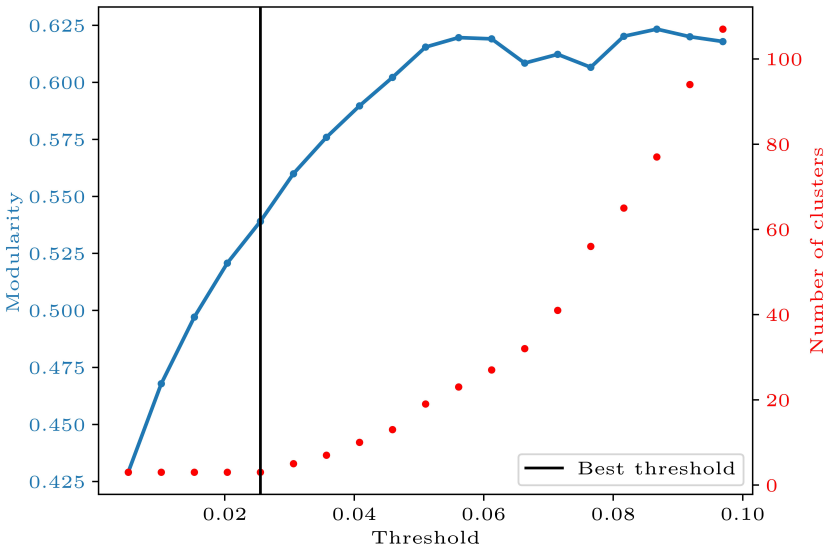


Figure 4.7: The plot of the modularity scores and the number of clusters was obtained by using different thresholds to ignore connections with weights less than the threshold. The threshold was tuned by finding the maximal modularity scores in the stable phase. The optimal threshold is 0.026. For this threshold, the modularity value is 0.539, and the number of clusters is 3. The input image was the **Sunflower** image shown in Fig. 3.6, and the Louvain algorithm was used.

Table 4.3 presents the optimal threshold, number of clusters, and modularity scores for different algorithms using various images. The optimal threshold was the best modularity score for a set number of clusters determined manually, see Sec. 3.7. The figure number for each modularity plot can be obtained from the remark column.

Table 4.3: The table lists the optimal thresholds, number of clusters, and modularity scores for the SPD, Infomap, and Louvain algorithms using various images. The remarks field indicates the figure numbers of each modularity plot in the appendix.

Algorithm	Image	Optimal threshold	Number of clusters	Modularity score	Remark
SDP	**Tulips**	0.002	4	0.357	Fig. A.1
Infomap	**Tulips**	0.003	4	0.505	Fig. A.2
Louvain	**Sunflower**	0.026	3	0.539	Fig. 4.7
Louvain	**Tulips**	0.041	4	0.612	Fig. A.3
Louvain	**Rose 1**	0.015	4	0.380	Fig. A.4
Louvain	**Rose 2**	0.010	3	0.337	Fig. A.5

Because of the dominance of the Louvain algorithm for modularity the following results use that clustering algorithm. In Table 4.3 SDP has a modularity of 0.357, Infomap 0.505, and Louvain 0.612.

4.5 Clustering plots

The clustering plots produced by various algorithms are displayed in this subsection. The original image is enhanced with cluster labels for each patch in the cluster overlay plot. Each image highlights a single cluster in color, while patches belonging to other clusters are depicted in grayscale. Fig. 4.8 shows the patch labeled as the cluster for the **Sunflower** image using the Louvain algorithm. Figs. 4.9, 4.10, and 4.11 display the patches labeled as clusters for the **Tulips**, **Rose 1**, and **Rose 2** images, respectively, using also the Louvain algorithm. Fig. A.6 in Appendix A shows a cluster overlap plot for the **Tulips** image using the Infomap algorithm.

Figure 4.8: The input image is **Sunflower**. The Louvain algorithm was utilized to detect clusters, resulting in the identification of three clusters.

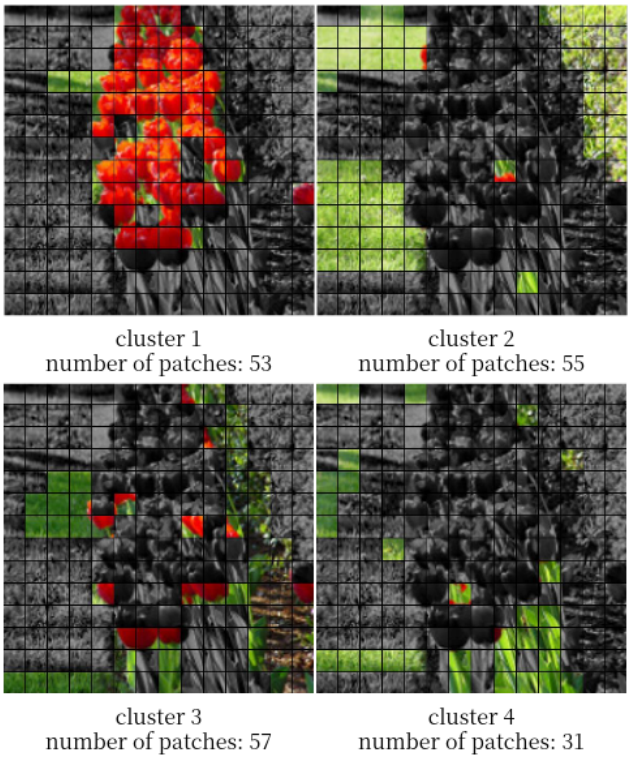


Figure 4.9: The input image is **Tulips**. The Louvain algorithm was utilized to detect clusters, resulting in the identification of four clusters.

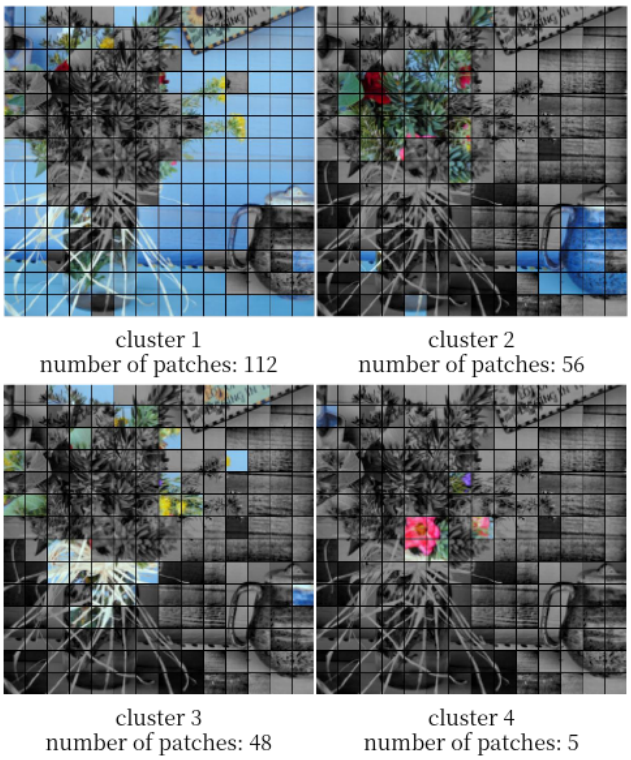


Figure 4.10: The input image is **Rose 1**. The Louvain algorithm was utilized to detect clusters, resulting in the identification of four clusters.

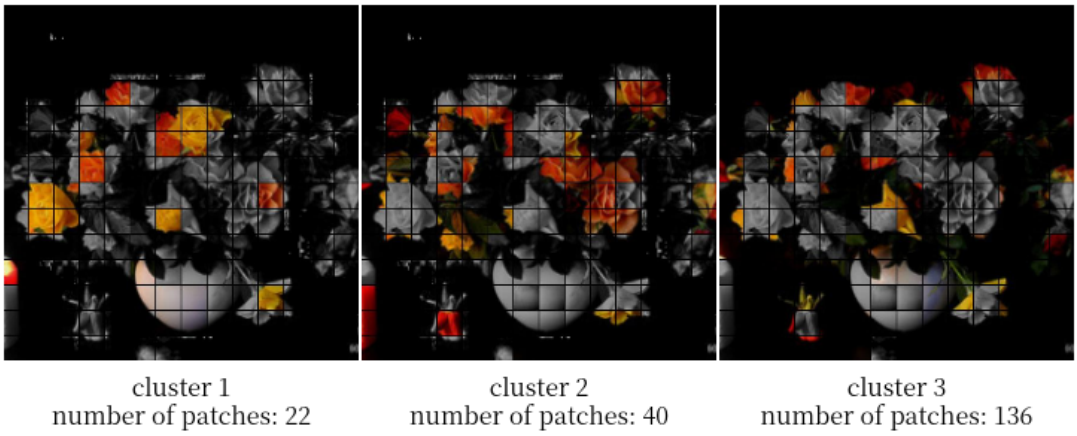


Figure 4.11: The input image is **Rose 2**. The Louvain algorithm was utilized to detect clusters, resulting in the identification of three clusters.

Fig. 4.12 shows two clusterings, one with three clusters and another with four. The Jaccard similarity for clusters 1 and 3 from the first clustering (k = 3) compared to the second clustering (k = 4) is 0.962 and 0.983, respectively. The Jaccard index for cluster 2 in the k = 3 clustering and the union of clusters 2 and 4 in the k = 4 clustering is 0.966. The algorithm used to detect clusters is Louvain.

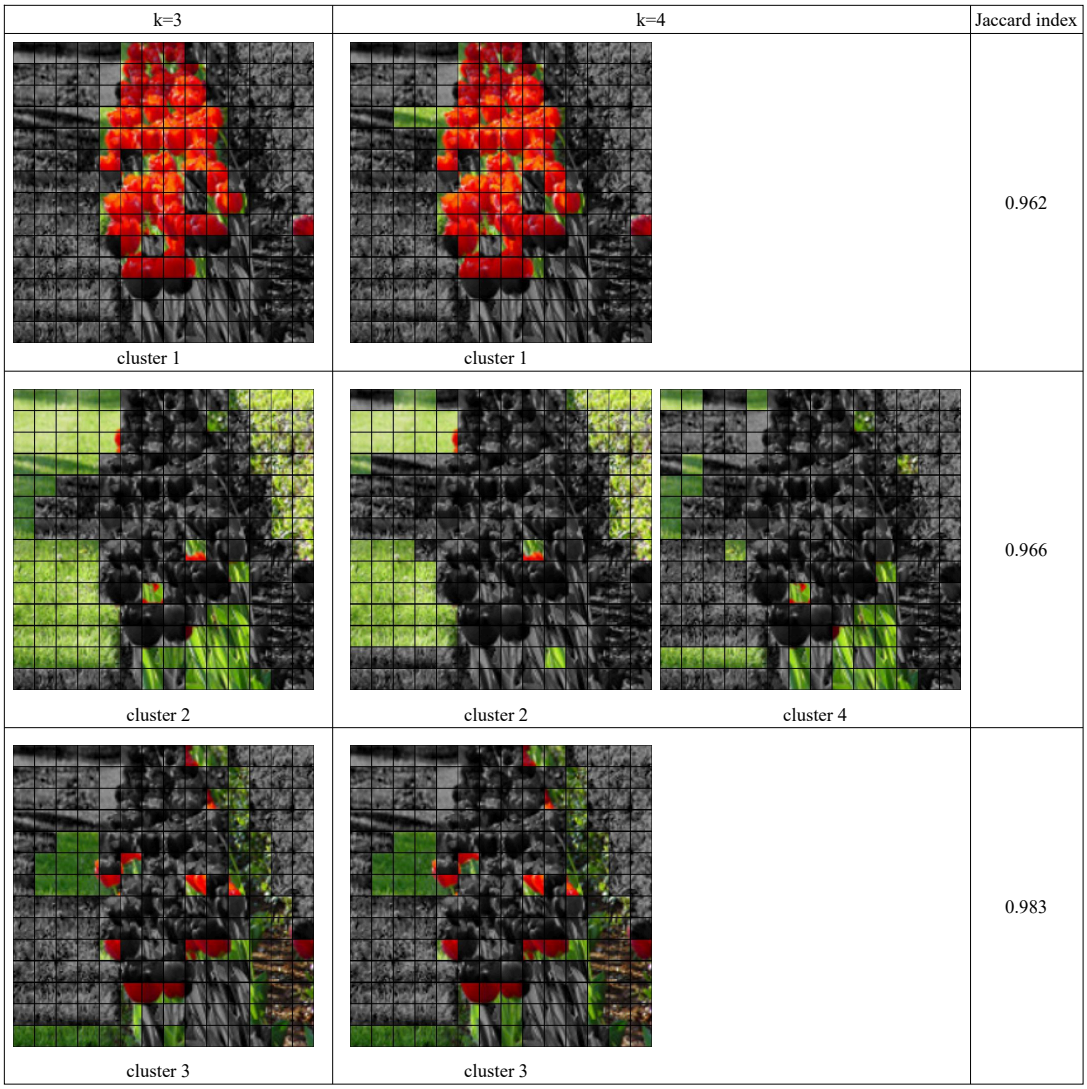


Figure 4.12: Overlay plots are presented for clustering with the number of clusters equal to 3 and 4, respectively. The clusters were detected using the Louvain algorithm. The input image is the **Tulips** image in Fig. 3.6. The Jaccard similarity for clusters in the two clusterings is included. Specifically, the Jaccard similarity for Cluster 1 and Cluster 3 between the k = 3 and k = 4 clusterings is 0.962 and 0.983, respectively. Additionally, the Jaccard index for Cluster 2 in the k = 3 clustering and the union of Clusters 2 and 4 in the k = 4 clustering is 0.966.

For an image of random pixels, see Fig, 4.13, the method did not find any meaningful clustering. The random pixels are given a value between 0 and 255 in each of the three channels. Because of the random assignment the image lacks natural clusters. In the modularity plot, Fig. 4.14, the highest modularity value is 0.175 with 7 clusters. The optimal threshold is 0.005 with modularity 0.035 and 3 clusters.

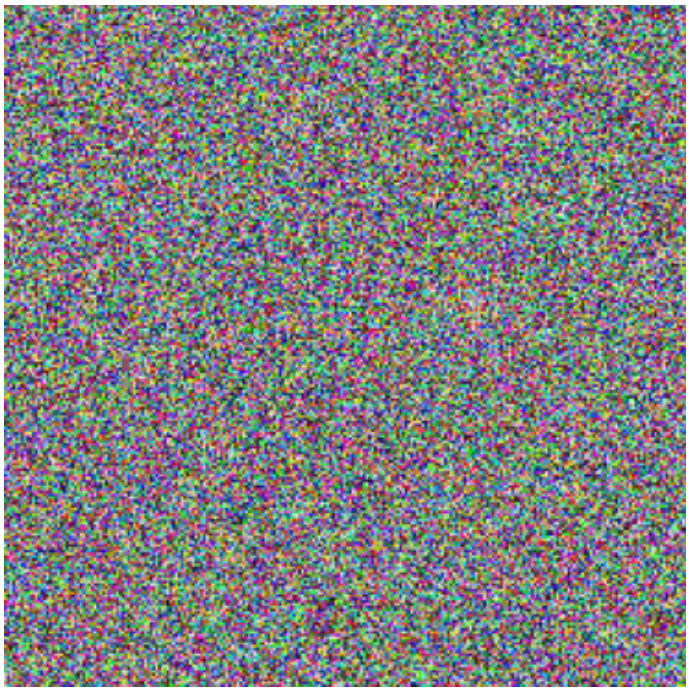


Figure 4.13: A random image generated by setting each pixel to a value between 0 and 255 uniformly for all channels. This image lacks natural clusters.

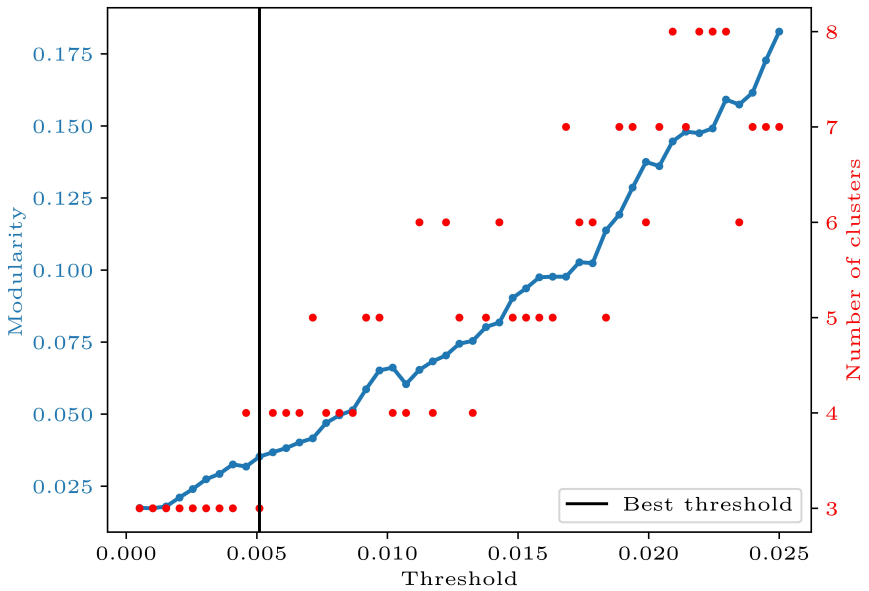


Figure 4.14: Plot of the modularity score obtained by using different thresholds to ignore connections with less weight than the threshold. The threshold was tuned by finding the maximal modularity scores in the stable period. The optimal threshold is 0.005. The modularity value and number of clusters are 0.035 and 3. The input image is shown in Fig. 4.13. The Louvain algorithm was used to detect clusters.

4.6 Logits for Nullifying Specific Clusters

Table 4.4 presents the logits obtained from two models: one nullifies output embeddings for patches assigned to cluster label 2 in Fig. 4.8 using the Louvain algorithm, and the other leaves patches unaltered. The Sunflower image was used as input.

Table 4.4: The **Sunflower** image was used to get logits from the model with and without nullifying the output embeddings that belong to cluster label 3. For the logits (with nullifying), the output embeddings of patches assigned to cluster number 3 in Fig. 4.8 were nullified.

Class	Daisy	Dandelion	Rose	Sunflower	Tulips
Logits (normal)	-0.312	-0.267	-5.419	**5.933**	0.041
Logits (patches in cluster 3 set to zero)	-0.470	-0.183	-3.197	**3.655**	0.083

Table 4.5 displays the logits for non-zeroing and zeroing output embeddings, with patches assigned to various cluster labels using the Louvain algorithm. The Tulips image in Fig. 3.6 was used as input.

Table 4.5: The logits for **Tulips** image (see Fig. 3.6) with different output embedding were zeroed or non-zeroed. The cluster label overlay images refer to Fig. 4.9.

Class	Daisy	Dandelion	Rose	Sunflower	Tulips
Logits (normal)	-3.167	-4.502	3.599	-0.928	**6.101**
Logits (patches in cluster 2 and 4 set to 0)	-2.636	-5.054	3.453	-1.871	**6.552**
Logits (patches in cluster 1 set to 0)	-0.553	-2.781	1.125	0.754	**2.235**
Logits (patches in cluster 1 and 3 set to 0)	-0.411	0.518	0.199	**0.834**	-0.450

Table 4.6 provides the logits for zeroing output embeddings, with patches assigned to clusters 1, 2, and 3 (see Fig. 4.10) and the logits for non-zeroing. The **Rose 1** image was used as input.

Table 4.6: The logits for the **Rose 1** image with non-zeroed and zeroed output embeddings, where patches are assigned to clusters 1, 2, and 3 in Fig. 4.10.

Class	Daisy	Dandelion	Rose	Sunflower	Tulips
Logits (normal)	-1.884	1.738	-0.164	**0.886**	-0.604
Logits (patches in cluster 1, 2 and 3 set to 0)	-0.402	-0.031	**0.330**	-0.294	0.102

5

Discussion

This chapter discusses the project. First, a data set is selected, followed by a model, and then clustering and its importance. Further, limitations for the project are presented, and suggestions for future work are raised.

5.1 Dataset Selection

Table 4.1 shows the classification accuracies when using attention and when not using it. The increases in accuracy when using attention are -0.880 percentage points for Fashion-MNIST, -0.749 points for CINIC10, 4.087 points for TFFlower, and 6.040 points for CIFAR10. The Fashion-MNIST dataset has the highest accuracy in models with and without attention. However, using the self-attention mechanism does not significantly affect the accuracy of this model. For the CINIC10 dataset, the accuracy is slightly decreased when using an attention layer. CIFAR10 and TFFlower data sets improve when using accuracy. Because the resolution in pixels for the CIFAR10 dataset is 32×32, the details and clarity of the images are limited. For some images in the CIFAR10 dataset, even the naked eye cannot distinguish the classes.

There are three reasons why we selected the TFFlower dataset for this project. One is that it shows that using a model with an attention layer improves accuracy. The improvement in accuracy is essential because it means attention effectively transforms the input. The transformation is crucial for the existence of clusters. Without an increase, the attention layer could work as an identity function. The second reason is the increased patch size. In ViT models, the input image is divided into fixed-size patches. The patch size determines the spatial resolution of these patches. Larger patch sizes result in fewer patches covering the entire image, while smaller patches lead to more patches. The patch size also affects the spatial information captured by each patch. Larger patch sizes potentially allow the model to better understand spatial relationships between different image parts. However, smaller patch sizes might capture finer details. Of the four datasets, the TFFlower dataset has the largest image size. The larger image size allows division into relatively large patches, thus potentially allowing the model to understand spatial relationships better. A better understanding could lead to improved clustering. Finally, the third reason is that the larger image allows using the typical 224×224 image size used in previous work on Vision Transformers.

5.2 Confusion Matrix

The confusion matrix of the test set for the model with one attention layer is shown in Fig. 4.1. The model is confident in most classes' results, but the roses class is an exception. For rose images, only 35.94% are correctly classified; notably, 32.81% of rose images are classified as tulips. The total misclassification rate was 64.06%. The poor classification means that the model did not learn the classification boundary between roses and tulips well. The two considered images from the rose class exhibited poor clustering results (modularity of 0.380 and 0.337).

Interestingly, tulips are not likely to be classified as roses, with only 6.25% misclassified. The considered tulip image exhibited clustering with a high modularity score (0.612). The predictive outcome within the rose class should be refined to improve the model's performance.

Out of the 70 samples for the sunflower class, 64 were correctly predicted. Consequently, the overall accuracy for this class is 91.43%. The model's prediction is always distinct from the rose class for sunflower samples.

5.3 Visualization

Attention maps illustrate how each patch focuses on other patches within the image. Brighter blocks in the attention maps signify higher attention weights. Fig. 4.3b demonstrates that patch ID 50 predominantly attends to adjacent patches in head 2. In head 4, this patch exhibits strong attention around the disk floret of the sunflower. This intense attention to the disk floret means there is a strong relation from this patch. However, the attention layer primarily concentrates on the image's background in heads 0 and 3. In head 6, attention is directed towards the edge of the ray florets. These findings indicate that not all attention heads contribute equally to the classification task. Heads 1, 7, 8, and 11 present challenges in determining which area of the given patch they attend to. Improper identification may lead to focusing on irrelevant parts of the image. Fig. 4.5b highlights that patch ID 31 directs attention toward unrelated patches.

5.4 Number of Attention Heads

The accuracy plot in Fig. 4.6 indicates that varying the number of attention heads has minimal impact on the overall test set accuracy. Nonetheless, the accuracy for each category varies with the number of heads. This result is unexpected as the original implementation used 12 heads. It seems that the one-layer Vision Transformer is not sensitive to the number of heads. In light of these findings, we reduced the number of heads we use to one as a simpler model was desired.

5.5 Clustering

Clustering the attention matrix with the SDP approach had several issues. When pruning edges, the SDP program eventually failed to find clusterings due to many unconnected edges. Other techniques exhibited this flaw to a lesser degree. When clusters were found, they had low modularity scores. In the case of the **Tulips** image, the score was 0.357. The original implementation considered bisection. While the technique does not limit itself to two clusters, it may be more suited to the bisection case. The technique does not consider directed graphs, which may have led to poor performance. Nevertheless, the found clustering showed a meaningful community structure.

Fig. 4.8 illustrates three clusters in the Sunflower image (Fig. 3.6) at a threshold of 0.026. The Louvain algorithm separates the image into ray florets, disk florets, and background, achieving a modularity score of 0.539.

For the **Tulips** image, the Louvain algorithm identified four clusters at a threshold of 0.041. The images showing the cluster labels overlaid on the original image are presented in Fig. 4.9, with a modularity score of 0.527. Using the Infomap algorithm, four partitions were also detected for the same **Tulips** image after removing edges below a threshold of 0.015, resulting in a modularity score of 0.505.

Modularity scores range from -1 to 1, with scores near 1 indicating a strong community structure and near 0 indicating no significant community structure. Negative values imply that the division is worse than random. In practical applications, modularity scores above 0.3 are generally considered meaningful, indicating a discernible community structure, while scores above 0.5 suggest a strong structure [39]. The Louvain and Infomap algorithms demonstrated effective clustering, whereas the SDP algorithm encountered issues, underscoring the effectiveness of community detection algorithms for clustering attention maps. If we use modularity as the evaluation criterion, the clustering obtained by the Louvain algorithm for **Tulips** images shows relatively high modularity. However, compared to the Infomap algorithm, the Louvain algorithm requires a higher threshold, meaning it needs to remove more edges to find the clusters.

5.6 Artificial Clusters

Clusters can still be identified for an image composed of random values (see Fig. 4.13). Clustering algorithms, such as the Louvain algorithm, will always try to group data into clusters based on some structure, even if the data is random. The algorithm might detect patterns or groupings that appear meaningful but are actually just artifacts of the random data. The modularity plot in Fig. 4.14 shows that the modularity score (0.035) for the optimal threshold of the random image is low and that no significant community structure exists.

5.7 Importance of Clusters for Classification

A high logit for a specific class indicates that the network strongly believes the input belongs to that class. The higher the logit for a class, the more confident the model is that this class is the correct label for the given input compared to other classes. Table 4.5 provides logit values for different combinations of cluster labels, with and without zeroing the output embeddings for the **Tulips** image. The results show that zeroing the patches belonging to the background (clusters 2 and 3) increases the logit value from 6.104 to 6.552. The result means that when the background patches were zeroed, the model had higher confidence when classifying the input image as tulips.

Conversely, when the patches assigned to cluster 1 (tulip petals) were zeroed, the logit value for the tulip class decreased from 6.101 to 2.235, indicating that the model was less confident in classifying the input image as tulips. Additionally, when the patches containing tulip petals (cluster 1) and tulip petals with some background (cluster 3) were zeroed, the model assigned a relatively high raw score to the sunflower class, reflecting strong confidence in that class being the correct label for the **Tulips** image. In other words, nullifying patches that significantly influence classification adversely affects the model's accuracy.

The logit values in Table 4.6 indicate that the model was more confident in labeling the **Rose 1** image as a dandelion. However, when the output embeddings for patches belonging to clusters 1, 2, and 3 in Fig. 4.10 - which did not help to identify the rose- were zeroed, the logit value for the rose class increased. Removing the irrelevant output embeddings made the model more confident in classifying the image as a rose, resulting in a correct prediction.

The clustering result of the **Rose 2** image (see Fig. 4.11) shows that the Louvain algorithm does not effectively cluster the attention weights for this image. For this model, the correct classification rate for the rose class is only 35.94% on the test set, indicating that the model has not learned to distinguish roses well. The attention weights for the rose class do not accurately represent the similarities between patches. For this particular rose image, the model predicts a 37.04% probability of it being a rose and a 43.11% probability of it being a sunflower. Consequently, finding a good clustering in the attention map for this incorrectly predicted image is challenging due to these issues.

Clustering is a machine-learning technique that groups similar data points (nodes) into clusters based on their attributes. While clustering can be applied to networks by creating embeddings, it is an unsupervised learning method that handles data with various features. In contrast, community detection is specifically designed for network analysis, focusing on the attributes of the edges. Clustering algorithms often separate individual peripheral nodes from the communities they should belong to.

In this project, we treat the attention map as an adjacency matrix. The nodes represent patches, and each element in the attention map indicates the similarity (weight) between two patches. Community detection methods like the Louvain and Infomap algorithms provide better clustering results than traditional methods like K-means or DBSCAN on embeddings generated with the SDP technique.

5.8 Limitation

The difficulty of this project is the trade-off between simplicity and complexity of the model. Simple models are easier to implement, train, interpret, and understand. However, they cannot capture complex patterns in the data. Complex models may perform better but are challenging to interpret. Additionally, they require more computational resources during training. We need to find a balance between simplicity and complexity to have a model that is simple enough to analyze and yet complex enough to perform well in capturing the complex patterns of images.

The optimal threshold for clustering is the one that maximizes modularity during a stable phase, aiming to achieve the highest modularity with the fewest edges removed. This threshold correlates with the number of clusters, illustrated by Fig. 3.6, showing a plateau in the modularity curve at a threshold ≤ 0.041, resulting in a modularity score of 0.612 with 4 clusters. However, determining the optimal threshold is challenging, as increasing the number of clusters beyond $k = 3$ may not necessarily improve clustering efficacy, as shown in Fig. 4.12. For instance, the Jaccard index between cluster 2 in $k = 3$ and the union of cluster 2 and 4 in $k = 4$ is 0.966, indicating similarity. Increasing the threshold from 0.005 to 0.041 resulted in splitting a relatively large cluster (cluster 2 in $k = 3$) into two smaller clusters (clusters 2 and 4 in $k = 4$), suggesting that adding another cluster to the Tulip image may not enhance clustering. Determining the optimal number of clusters is challenging due to the absence of prior knowledge or ground truth, necessitating the identification of an optimal threshold through manual analysis of cluster overlay plots.

This project only considers one data set and only a few pictures in the TFFlowers data set. This was done for two reasons. The first reason was that presentation of similar graphs would make the report harder to read for little benefit. We chose images that were representative of the images we looked at for the report. The other reason was because it was labor-intensive to generate the images since the threshold must be set manually. This greatly limited the number of images. It's possible that the model learns different structure of the attention map for other data sets. In particular, Vision Transformers and NLP Transformers differ greatly in the structure of their attention maps. These other structures may not have the clustering found for the one-layer Vision Transformer.

5.9 Future Work

In this project, we examined the clustering of attention weights from a transformer model featuring only one attention layer and one attention head using the TFFlower dataset. We achieved promising clustering outcomes using the Louvain algorithm. However, the results were only examined for a small number of the available classes. An automatic way of extracting the clustering structure would be a great improvement to verify that a clustering structure exists for all images.

Whether this analysis can be extended to other image datasets for clustering remains

uncertain. Future research endeavors should explore the applicability of clustering analysis to attention weights across various datasets. Furthermore, investigating how the clustering of attention weights evolves across layers in models with multiple attention layers could be a valuable direction for future studies. Additionally, exploring the impact of employing multiple attention heads and discerning the differences in clustering patterns among them could offer further insights.

6

Conclusion

Attention maps exhibit significant community structure. Modularity scores were used to evaluate clustering effectiveness, with higher scores indicating more robust community structures. The Louvain algorithm produced superior results to other clustering methods and achieved a meaningful community/clustering structure.

Analyzing the removal of tokens from the attention layer's output embedding indicated that some clusters are more important than others. In particular, tokens belonging to the image's background hinder classification performance. Keeping only foreground tokens improved performance.

The choice of pruning threshold significantly affects clustering outcomes, with higher thresholds potentially generating more clusters but a smaller number often yielding more meaningful results. However, determining the optimal threshold is challenging and requires regularization to prevent excessive clustering.

Bibliography

[1] K. Roose, "How should i use a.i. chatbots like chatgpt?" *New York Times*, Mar. 2023. [Online]. Available: https://www.nytimes.com/2023/03/30/technology/ai-chatbot-chatgpt-uses-work-life.html (visited on 01/17/2024).

[2] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[3] OpenAI, *Gpt-4 technical report*, 2023. arXiv: 2303.08774 [cs.CL].

[4] P. Zhang, X. Dai, J. Yang, *et al.*, "Multi-scale vision longformer: A new vision transformer for high-resolution image encoding," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 2978–2988. DOI: 10.1109/ICCV48922.2021.00299.

[5] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021. DOI: 10.1038/s41586-021-03819-2.

[6] A. Ramesh, M. Pavlov, G. Goh, *et al.*, *Zero-shot text-to-image generation*, 2021. arXiv: 2102.12092 [cs.CV].

[7] C. Metz, "What's the future for a.i.?" *New York Times*, Mar. 2023. [Online]. Available: https://www.nytimes.com/2023/03/31/technology/ai-chatbots-benefits-dangers.html (visited on 01/17/2024).

[8] Y. Zhang, A. Backurs, S. Bubeck, R. Eldan, S. Gunasekar, and T. Wagner, *Unveiling transformers with lego: A synthetic reasoning task*, 2023. arXiv: 2206.04301 [cs.LG].

[9] H. Fu, T. Guo, Y. Bai, and S. Mei, *What can a single attention layer learn? a study through the random features lens*, 2023. arXiv: 2307.11353 [cs.LG].

[10] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. arXiv: 1706.03762 [cs.CL].

[11] D. A. Tarzanagh, Y. Li, X. Zhang, and S. Oymak, *Max-margin token selection in attention mechanism*, 2023. arXiv: 2306.13596 [cs.LG].

[12] D. A. Tarzanagh, Y. Li, C. Thrampoulidis, and S. Oymak, *Transformers as support vector machines*, 2023. arXiv: 2308.16898 [cs.LG].

[13] B. Geshkovski, C. Letrouit, Y. Polyanskiy, and P. Rigollet, *A mathematical perspective on transformers*, 2024. arXiv: 2312.10794 [cs.LG].

[14] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, *Albert: A lite bert for self-supervised learning of language representations*, 2020. arXiv: 1909.11942.

[15] C. Tomasi, "Histograms of oriented gradients," *Computer Vision Sampler*, pp. 1–6, 2017. [Online]. Available: https://courses.cs.duke.edu/fall17/compsci527/notes/hog.pdf (visited on 06/05/2024).

[16] M. Pietikäinen, "Local Binary Patterns," *Scholarpedia*, vol. 5, no. 3, p. 9775, 2010, revision #200631. DOI: 10.4249/scholarpedia.9775.

[17] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[18] G. Wang, Y. Lu, L. Cui, T. Lv, D. Florencio, and C. Zhang, "A simple yet effective learnable positional encoding method for improving document transformer model," in *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, 2022, pp. 453–463. [Online]. Available: https://aclanthology.org/2022.findings-aacl.42 (visited on 01/26/2024).

[19] Y. Ma and R. Wang, "Relative-position embedding based spatially and temporally decoupled transformer for action recognition," *Pattern Recognition*, vol. 145, p. 109 905, 2024. DOI: https://doi.org/10.1016/j.patcog.2023.109905.

[20] Z. Raisi, M. A. Naiel, P. Fieguth, S. Wardell, and J. Zelek, "2d positional embedding-based transformer for scene text recognition," *Journal of Computational Vision and Imaging Systems*, vol. 6, no. 1, pp. 1–4, 2020. DOI: 10.15353/jcvis.v6i1.3533.

[21] P. Zhang, X. Dai, J. Yang, *et al.*, "Multi-scale vision longformer: A new vision transformer for high-resolution image encoding," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 2998–3008. arXiv: 2103.15358 [cs.CV].

[22] D. Masters and C. Luschi, *Revisiting small batch training for deep neural networks*, 2018. arXiv: 1804.07612 [cs.LG].

[23] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training," *Neural Computing and Applications*, vol. 16, no. 3, pp. 235–247, May 2007. DOI: 10.1007/s00521-007-0084-z.

[24] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 69, no. 2, p. 026 113, 2004. DOI: 10.1103/PhysRevE.69.026113.

[25] S. A. Leland McInnes John Healy, *What can a single attention layer learn? a study through the random features lens*, HDBSCAN, 2016. [Online]. Available: https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html (visited on 04/23/2024).

[26] A. Javanmard, A. Montanari, and F. Ricci-Tersenghi, "Phase transitions in semidefinite relaxations," *Proceedings of the National Academy of Sciences*, vol. 113, no. 16, E2218–E2223, 2016. DOI: 10.1073/pnas.1523097113.

[27] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the national academy of sciences*, vol. 105, no. 4, pp. 1118–1123, 2008. DOI: 10.1073/pnas.0706851105.

[28] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 107–117,

1998, Proceedings of the Seventh International World Wide Web Conference, ISSN: 0169-7552. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016975529800110X.

[29] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, P10008, 2008. DOI: 10.1088/1742-5468/2008/10/P10008.

[30] A. Krizhevsky, V. Nair, and G. Hinton, *The CIFAR-10 dataset*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html (visited on 01/26/2024).

[31] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, *CINIC-10 is not ImageNet or CIFAR-10*, 2018. arXiv: 1810.03505 [cs.CV].

[32] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," 2017. DOI: 10.48550/arXiv.1708.07747.

[33] The TensorFlow Team. "Flowers." (Jan. 2019), [Online]. Available: http://download.tensorflow.org/example_images/flower_photos.tgz (visited on 06/05/2024).

[34] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (visited on 01/26/2024).

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011. [Online]. Available: https://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf (visited on 06/02/2024).

[37] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," Jan. 2008. [Online]. Available: https://www.osti.gov/biblio/960616 (visited on 06/05/2024).

[38] D. Edler, A. Holmgren, and M. Rosvall. "Infomap python documentation." (2020), [Online]. Available: https://mapequation.github.io/infomap/python/infomap.html (visited on 05/16/2013).

[39] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, p. 066 111, 6 Dec. 2004. DOI: 10.1103/PhysRevE.70.066111.

A

Appendix 1

A.1 Modularity Plots

In this section, plots showcasing the modularity scores obtained through the utilization of different thresholds to disregard connections with weights lower than the threshold are presented using various algorithms. The threshold was fine-tuned by identifying the maximum modularity scores during the stable phase.

A.1.1 Modularity Plot for Tulips image by using SDP

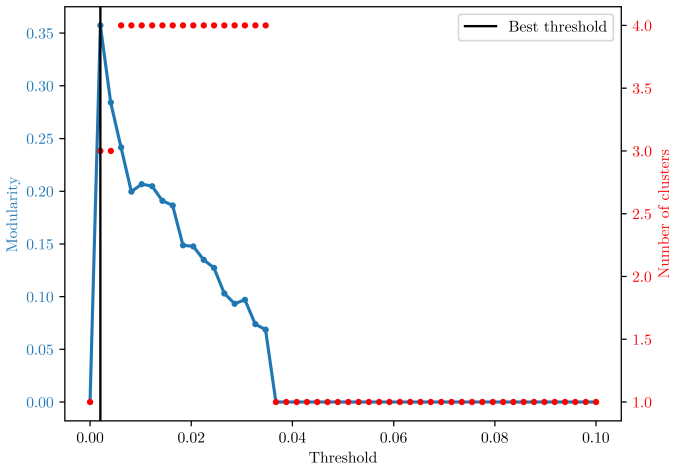


Figure A.1: Threshold selected according to the outlined heuristic. The algorithm used for extracting clusters is SDP with HDBSCAN. Notice that the peak modularity, 0.357, occurs at a low level of pruning, 0.0020. The **Tulips** image was used.

A.1.2 Modularity Plot for Tulips image by using Infomap

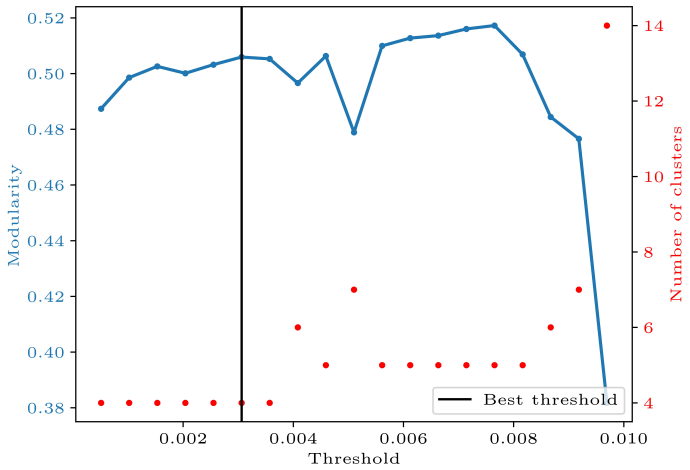


Figure A.2: The optimal threshold and modularity are 0.003 and 0.505, respectively. The number of clusters is 4. The input image used was the **Tulips**. The clustering method employed was Infomap.

A.1.3 Modularity Plot for Tulips image by using Louvain

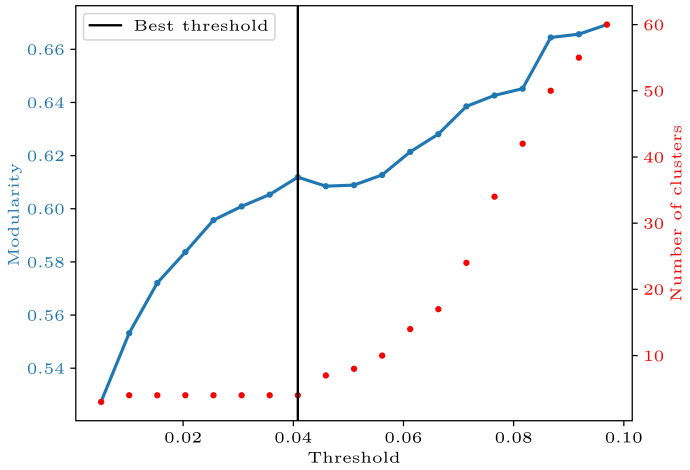


Figure A.3: The modularity value is 0.612, with 4 clusters identified. The input image file is the **Tulips**, and the Louvain algorithm was employed.

A.1.4 Modularity Plot for Rose 1 image by using Louvain

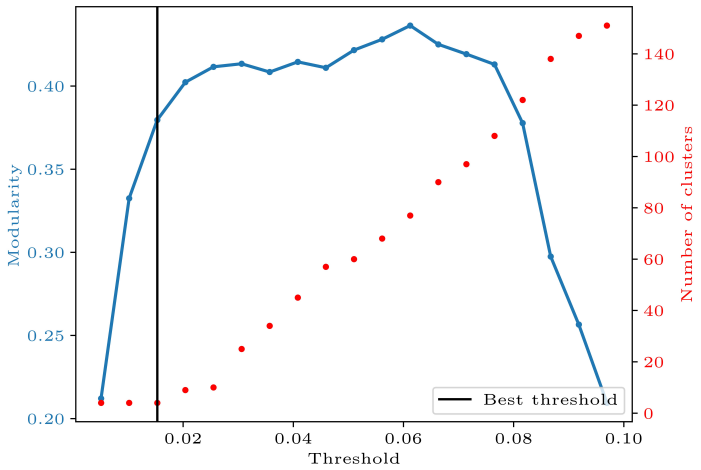


Figure A.4: The clusters were detected using the Louvain algorithm, with the best threshold identified as 0.0153. The modularity value attained is 0.380. The input used was the **Rose 1** image in Figure 3.6

A.1.5 Modularity Plot for Rose 2 image by using Louvain

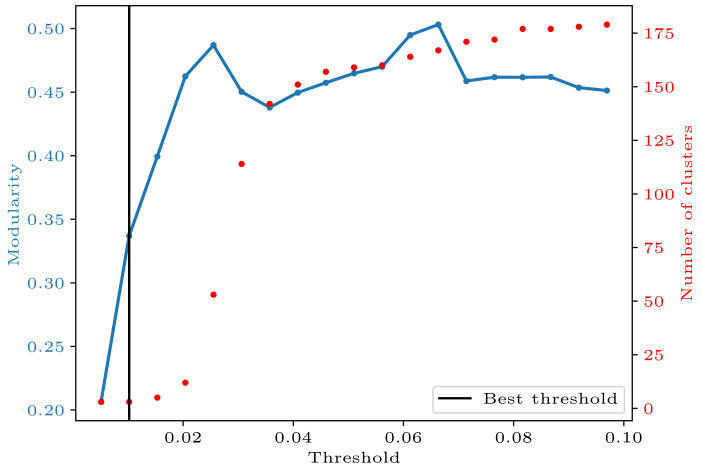


Figure A.5: The optimal threshold is 0.010, yielding a modularity value of 0.337. The **Rose 2** image served as the input for the Louvain algorithm to detect clusters.

A.2 Overlay plot for Tulips image by using Infomap

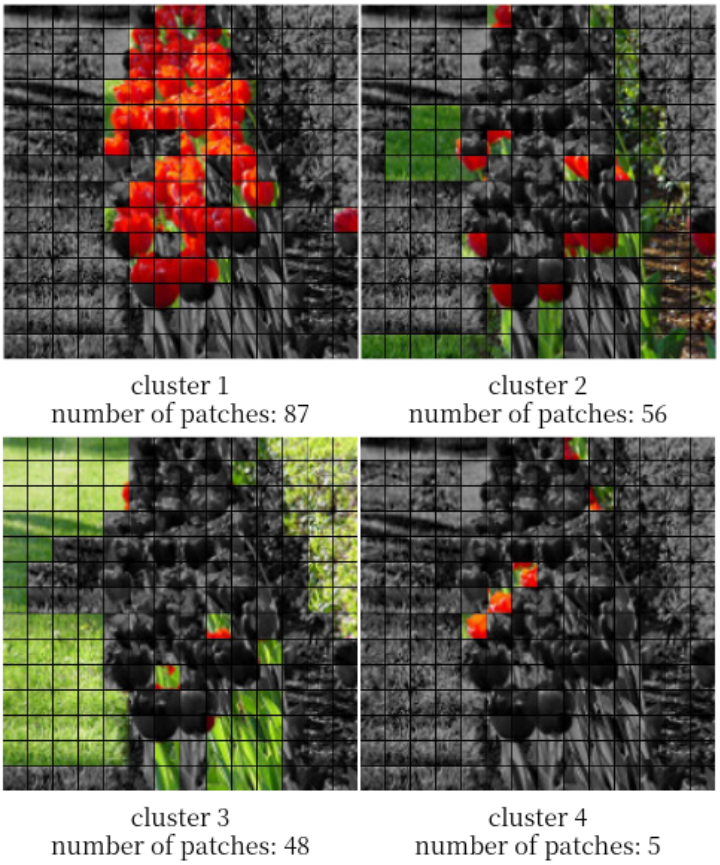


Figure A.6: The input image is **Tulips**. The **Infomap** algorithm was utilized to detect clusters, resulting in the identification of four clusters.