Semantic Scene Change Detection

Evaluation through Classical & Machine Learning Algorithms

Master's thesis in Computer Science and Engineering

Jithinraj Sreekumar
Shreya Desai

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

MASTER'S THESIS 2021

Semantic Scene Change Detection

Evaluation through Classical & Machine Learning Algorithms

Jithinraj Sreekumar
Shreya Desai

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Semantic Scene Change Detection
Evaluation through Classical & Machine Learning Algorithms
Jithinraj Sreekumar
Shreya Desai

Supervisor: Peter Damaschke, Department of Computer Science and Engineering
Advisor: Per Nilsson Lundberg, CEVT AB
Examiner: Carl-Johan Seger, Department of Computer Science and Engineering

Master's Thesis 2021
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 2021

Semantic Scene Change Detection
Evaluation through Classical & Machine Learning Algorithms
Jithinraj Sreekumar
Shreya Desai
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Scene change detection helps to detect changes in a pair of multitemporal images of the same scene. We apply the concept of scene change detection to detect misplaced objects in a passenger vehicle. Deep learning neural networks have been extensively used in scene change detection. We study scene change detection using the classical Watershed algorithm and machine learning algorithms. In machine learning, we exploit the feature extraction capability of ResNet and Spatial Pyramid Pooling to predict the scene change. The performance of the classical and machine learning algorithms are also compared. The models are trained on a custom dataset and evaluated using the metrics, dice coefficient, mean intersection over union (mIoU) and pixel accuracy. We infer that the machine learning model significantly outperforms the classical model in terms of mIoU score.

Keywords: scene change detection, machine learning, semantic segmentation, convolutional neural network, residual neural network, siamese network, spatial pyramid pooling

Acknowledgements

We would like to express our sincere gratitude to our academic supervisor, Peter Damaschke, our advisor at CEVT AB, Per Nilsson Lundberg, and our examiner, Carl-Johan Seger, for their insightful input, guidance and support throughout the thesis.

We would also like to extend our gratitude to Anders Werner for giving us the opportunity to carry out our thesis at CEVT AB.

We are also grateful to our family and friends for their relentless encouragement.

Jithinraj Sreekumar and Shreya Desai, Gothenburg, February 2021

Contents

List of Figures

1
Introduction

Scene change detection is an appealing subject in computer vision. It is a process that detects change in two multitemporal images of the same scene, as shown in Figure 1.1.

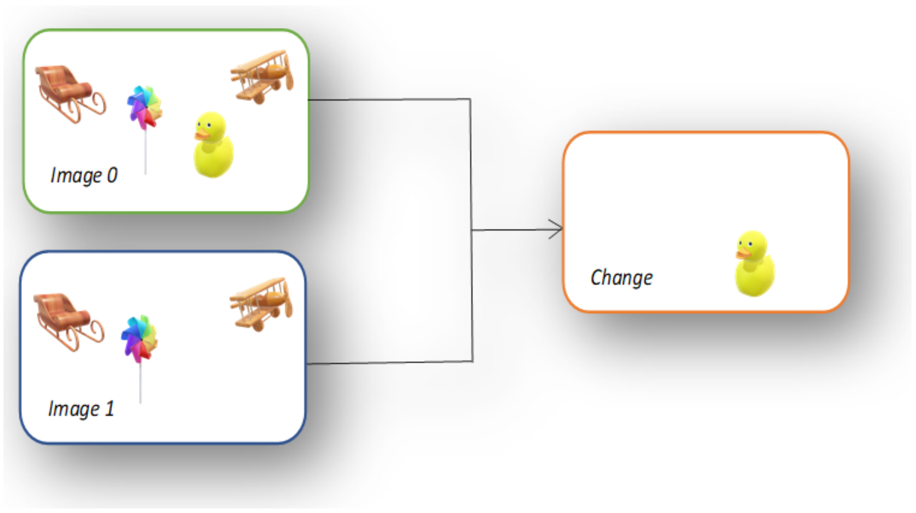


Figure 1.1: Scene change detection

The basic idea of scene change detection is to detect the change in an image. The following scenario of a passenger boarding a vehicle will illustrate the process of scene change detection. First, two different images are captured as the passenger enters and exits the vehicle. The images are captured using multiple cameras mounted at an angle above the back seat of the vehicle. The two images captured at different timestamps are used as input to an algorithm for detecting objects in the interior scene of the vehicle. The objects may be a bag, a mobile phone, keys, an umbrella, etc. The aim is to identify such objects that are not previously present in the vehicle. Also, the passenger can be notified about the misplaced objects. The algorithm should also ensure that no items belonging to the driver are detected if they have been placed in the back seat along with the passenger's items. We can apply this concept to taxis or other passenger vehicles.

Scene change detection using classical computer vision algorithms have been in practice for a long time [1, 2]. The ability to learn solutions from observational data makes machine learning an interesting technique to use for scene change detection. Machine learning concepts include powerful tools that can build complex applications, learn semantics, and extract useful features from images and videos. It addresses the limitations of classical techniques by taking on tasks associated with the human brain, which is capable of recognising objects and patterns, making visual classifications, and so on [3, 4, 5, 6]. Various scene change detection methods are studied and applied in remote sensing, street monitoring, etc.

To detect the changes in a scene, there should be a technique to distinguish the objects in images. It can be achieved by image segmentation. Image segmentation is the process of extracting meaningful information from an image by segregating the pixels. The pixels can be characterized based on texture, intensity level, shape, etc. Image segmentation can be further divided into semantic segmentation and instance segmentation. Semantic segmentation is a process that associates each pixel in an image with a class label, while instance segmentation is a process that assigns a unique ID to each object. Semantic segmentation is an important component of scene parsing. Figure 1.2 illustrates the process of semantic segmentation. Here, the object pixel (foreground area) in the image is associated with a class label.

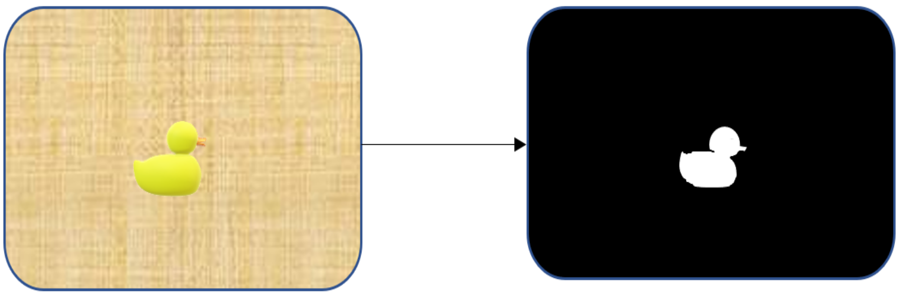


Figure 1.2: Semantic segmentation

A convolutional neural network is considered as the backbone for most image segmentation tasks. It forms a powerful architecture for feature extraction and classification. It is good at processing the data to create a large feature space from an input image which is encoded in the architecture. It allows the network to learn and self-train the appropriate features for a given task by itself that makes it suitable for image-focused tasks [7]. This ability of the convolutional neural network helps researchers to explore its use for image segmentation. There are several architectures in the field of convolutional neural networks, for example, GoogleNet [8], AlexNet [9], VGGNet [10], ResNet [11].

Convolutional neural network with deeper network architecture can provide more accurate prediction results. However, with deeper networks, it is more difficult to train the model and it is difficult to draw a conclusion about the accuracy of the model. Residual Neural Network (ResNet) attempts to mitigate this problem. ResNet is a kind of architecture in the field of convolutional neural networks. ResNet is a popular convolutional neural network. There are different versions of ResNet which contain 18, 34, 50, 101, 152 layers. The significance of layers and their selection will be discussed in the following chapters.

The main objectives of this study are:

1. Implement an existing classical algorithm and evaluate its performance on the custom (user-defined) dataset.

2. Implement new machine learning algorithms inspired by the existing studies and evaluate their performance.

The work is divided into six chapters. Chapter 1 briefly describes the basic concepts of scene change detection. Chapter 2 discusses the technical background of classical and machine learning algorithms. Chapter 3 reviews the existing studies dealing with scene change detection and semantic segmentation. Furthermore, Chapter 4 discusses the methodologies for implementing the classical and machine learning algorithms. Chapter 5 presents the results and compares the performance of both the algorithms. Finally, the decisions made for each algorithm implementation are discussed and the results are summarized in Chapter 6.

2

Technical Background

In this chapter, we will walk through the theoretical aspects of classical and machine learning algorithms for scene change detection. Section 2.1, briefly explains the theoretical concepts of the watershed algorithm. Section 2.2 briefly discusses the concepts of a neural network, weight initialization, convolutional neural networks, residual neural networks, global average pooling and 1x1 convolution. This section also explains the activation function, loss function, optimization algorithm for training the model, and the metrics for evaluating the performance of the model.

Subsection 2.2.12 explains the technical background for implementing ResNet50 based model. The feature extraction architecture of ResNet50 and the architecture for reconstructing a segmented scene change map are explained. Also, the ResNet50 based models developed for this study are presented in Subsection 2.2.12.3. Subsection 2.2.13 describes the technical background for the Spatial Pyramid Pooling model.

Section 2.1, Section 2.2, Subsection 2.2.2 - 2.2.11 can be skipped if the reader is familiar with the classical watershed algorithm and machine learning concepts.

2.1 Watershed Algorithm

The notion of the Watershed algorithm is that it considers an image as a topographic surface and is divided into multiple catchment basins or watershed basins [12]. It transforms an image such that the catchment basins are the objects that we want to identify. The Watershed algorithm was introduced in 1978 and further developed in 1982 by Serge Beucher [13]. It has been successfully used in image processing, especially in image segmentation [14, 15].

The Watershed algorithm works on a grayscale image, applying segmentation to the gradient of an image. It is a region-based algorithm that looks for the similarities between pixels and regions. Each region in the image is characterized by the gray levels of an image. And, any variation in the gray levels will result in small gradient values.

Assume the available image datasets are based on the RGB color model. An RGB image has three channels: Red(R), Green(G), Blue(B). Here, the red/green/blue channel is also referred to as a feature map. In image processing, a channel is the grayscale image of the same size as a color image, made of just one primary color (Red(R), Green(G), Blue(B)). It carries the intensity information about the image corresponding to a pixel value. A grayscale image has a single channel: It contains only one of the primary colors. Figure 2.1 shows two RGB based input images used for scene change detection (RGB values (R, G, B) are highlighted in yellow in the bottom left of the two images).

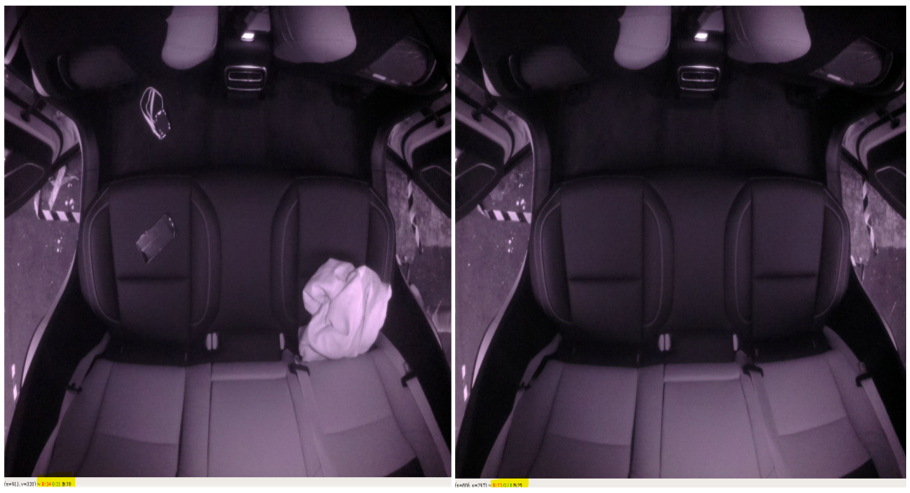


Figure 2.1: RGB image pair

Since the Watershed algorithm operates on a single grayscale image, the scene change is detected in two different images by first taking the absolute difference of the two RGB input images. The absolute difference image is then converted to a grayscale image, as shown in Figure 2.2 (grayscale value (L) is highlighted in yellow in the bottom left of the image).

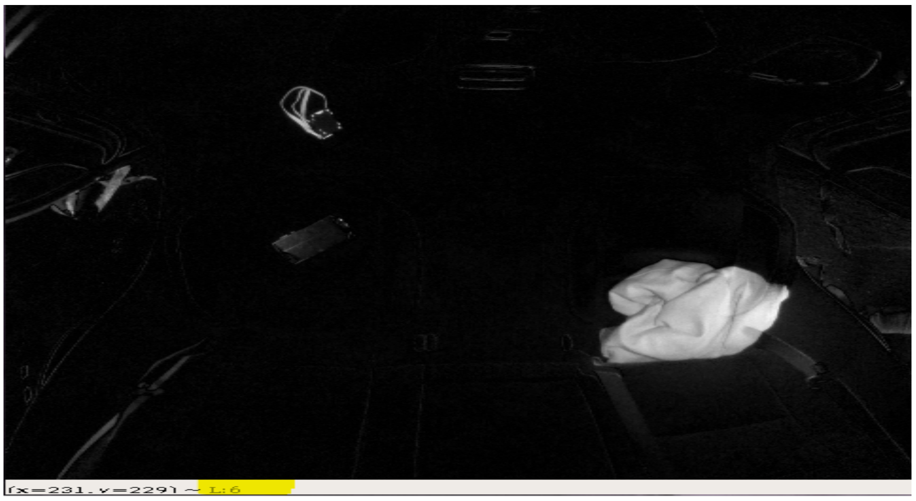


Figure 2.2: Absolute difference of RGB images (grayscale)

Image segmentation can also be done based on the shape of objects, using the distance transformation function. The distance transformation function calculates the distance between the object pixel and the nearest background pixel such that high intensity pixels are turned into catchment basins. It works with a binary image so that all object pixels are set to maximum intensity ‘255’ (white pixels), and the background pixels are set to lowest intensity ‘0’ (black pixels). The binary image is obtained by applying morphological transformation operators (discussed in section 2.1.1) and then thresholding. The Watershed algorithm works best with an image on which a distance transformation has already been applied. Image segmentation using the Watershed transform works better if the foreground objects can be marked well or defined from the background area. It helps in extracting the desired objects from the image.

2.1.1 Morphological Transformation

Morphological transformation is a powerful preprocessing step. It is essential to improve the quality of an image, highlight the required features and remove noise or distortion. It includes operators that are mainly used to analyze binary images to enhance the image, remove noise, detect edges, etc. It uses a kernel or structuring element to determine the type of transformation.

The following are the main operators of morphological transformation.

1. **Erosion**: The basic idea is to gradually chip away the boundaries of foreground objects. A pixel in the binary image (value 0’s or 1’s) is considered 1 only if all pixel values under the kernel are 1. Otherwise, the pixel will be considered 0 and eroded. This operation always tries to keep the sure foreground area (object pixel) in white and rest of the area in black.

2. **Dilation**: It is the reverse of erosion. In this case, if at least one-pixel value is 1, the pixel value is considered to be 1.

3. **Opening**: There are two operations performed by this operator usually in the following order: first is erosion, and then a dilation operation.

4. **Closing**: It is the reverse of opening.

2.2 Neural Networks

Neural networks are inspired by the capability of the biological neural system to process experimental data in the brain. The basic computational unit is a neuron. In the biological model, neurons receive and process information from their dendrites. The processed information is then transported along the axon to the terminal unit, the synapse. The output of this neuron forms the input for other neurons. The learning ability of the brain occurs through a series of activations of the neurons [16].

Figure 2.3, shows the computational model of a neuron.

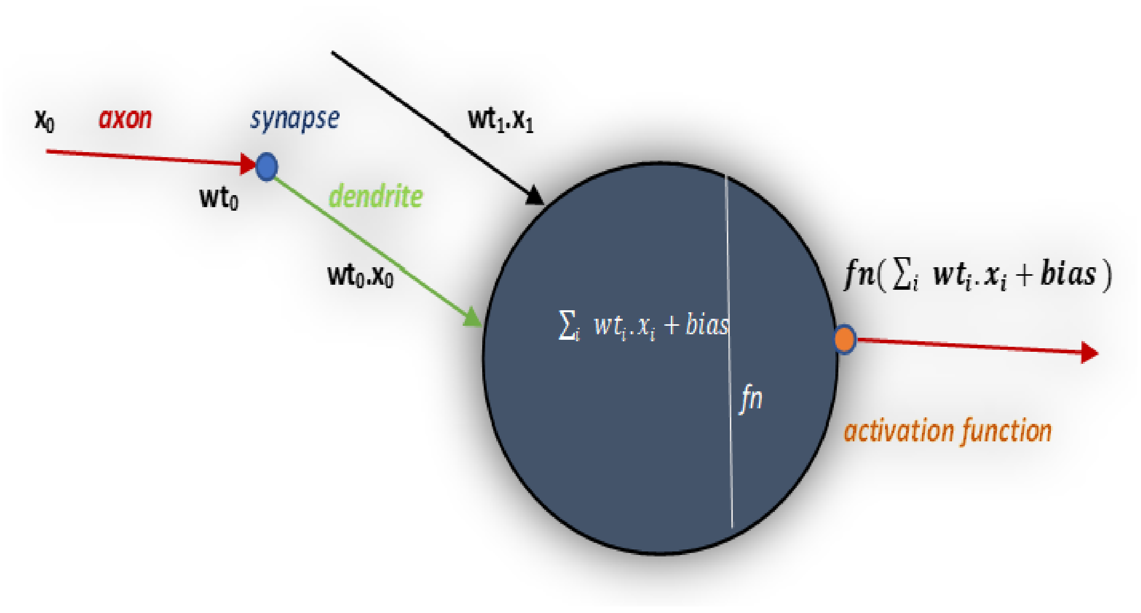


Figure 2.3: Neuron

Each neuron performs the following calculation: performs dot product with the input (x_i) and its weights (wt_i), then add the bias and apply a nonlinear function (fn).

Neural networks are a group of neurons. They are interconnected in the form of an acyclic graph. It receives input and goes through a series of hidden layers. The hidden layer consists of neurons and is independent of each other. The neurons in every layer are interconnected. In other words, the output of a neuron is a linear transformation of the previous layer combined with a nonlinear activation function.

2.2.1 Initialization

The purpose of weight initialization is to prevent the following problems:

1. Vanishing gradients: During training, the weights are updated based on the errors which are backpropagated. As we add more layers to the network, the errors during backpropagation fail to reach the initial layers. Therefore, the amount of gradient information decreases and eventually does not reach the initial layers. This is called vanishing gradient problem. The vanishing gradient problem can occur if the weight initialization values are very small.

2. Exploding gradients: The value of weights to be updated in each layer becomes large when larger values of gradient information gets accumulated in the layers. This is called exploding gradient problem. The exploding gradient problem can occur if the weight initialization values are very large.

Kaiming et al. (2015) [17] define the weight initialization using the following formula:

$$W \sim \mathcal{N}(0, 2/n_l) \tag{2.1}$$

where W is the initial weight matrix that depends on d (total number of filters). The weights of these filters are represented by every row in W (W is a matrix of size $d \times n$), $\mathcal{N}$ is the normal distribution with mean zero and variance $2/n_l$, n_l is the number of connections in a layer l. The standard deviation in the above equation is $\sqrt{2/n_l}$.

2.2.2 Convolutional Neural Networks

Convolutional neural networks are similar to neural networks in that they consist of layers. It may consist of many hidden layers and millions of parameters. When compared to the fully connected layers in the neural networks in Section 2.2, the output of the convolutional layer is the result of applying convolutions to a subset of the neurons in the previous layer.

Figure 2.4 shows the basic convolutional block, where a kernel/filter (represented as a matrix) in the convolutional layers, slides over the image (input image matrix) in the convolutional layers to create a feature map for the next layer. A kernel/filter is used to detect the essential features in an image. The feature map is the result of element-wise multiplication of the input image matrix and the kernel matrix. The kernel maps a subset of neurons from the previous layer to a single neuron in the next layer to create a feature map.

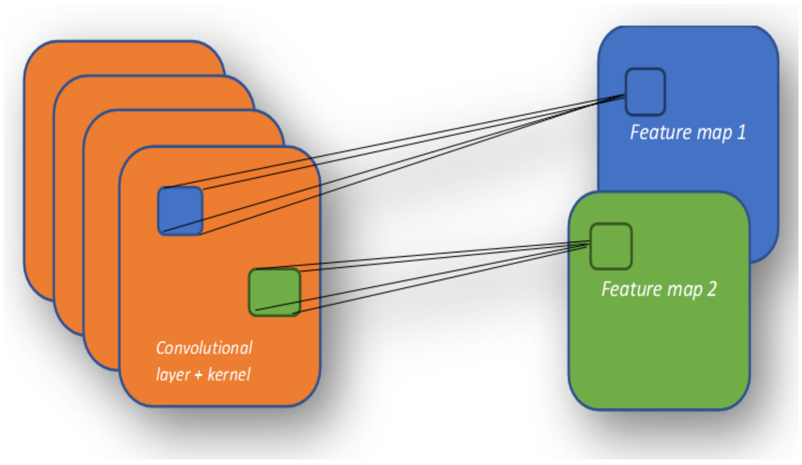


Figure 2.4: Convolutional block and feature maps

The mathematical concept of convolution [18] is explained by first defining convolution in one dimension. A convolution is an operation on two functions $f(x)$ and $g(x)$ that results in $(f(x) * g(x))$ at point x. Here, $f(x)$ and $g(x)$ represent one dimensional function. It is blending one function over the other. In mathematical terms, convolution is an integral that evaluates the overlap of a function $g(x)$ shifted over another function $f(x)$. It is evaluated for all shifts and thus yields a convolution function, $h(x)$. A one-dimensional convolution of two discrete functions $f(x)$ and $g(x)$ is given using the following formula:

$$h(x) = f(x) * g(x) = \sum_{-\infty}^{+\infty} f(a) \cdot g(x - a) \tag{2.2}$$

where ‘a’ is the shift.

A convolution in two dimensions is used when the input is an image. Let $f(x, y)$ be the input image and $g(x, y)$ be the kernel function. Then a convolution of the input image and kernel function is given using the following formula:

$$h(x, y) = f(x, y) * g(x, y) = \sum_{a=-\infty}^{+\infty} \sum_{b=-\infty}^{+\infty} f(a, b) \cdot g(x - a, y - b) \tag{2.3}$$

where ‘a’ and ‘b’ are the shift.

A convolutional neural network [19] is independent of spatial dimension and takes an image with two/three dimensions as input. It also consists of learnable weights and biases. In general, a convolutional neural network transforms the input image pixels, layer by layer into a final class probability score in a classification task. Convolutional neural networks form the backbone in computer vision applications such as pattern recognition, image classification, object recognition, etc.

The convolutional neural network architecture can be divided into three main layers:

1. The first layer is the convolutional layer. It forms the basic block of convolutional neural networks. It takes an image as input to create a feature space by scanning each pixel. The convolutional layer is followed by the activation function, which is a nonlinear transformation operation.

2. The second layer is pooling, where the dimensionality of the feature space is reduced to extract more finer features. This is also known as downsampling. The pooling layer does not contain any parameters.

3. The final convolutional layer consists of vectors of feature maps. These vectors are passed to the fully connected layers, which form the third layer. Softmax regression often follows, such that the output of the network is a probability distribution with respect to the predicted classes. The fully connected layers help the network generalize its prediction by compiling the features extracted from the previous layers.

The parameters in the convolutional and fully connected layers are trainable. A training should ensure that the class likelihood score matches the ground truth image in the training dataset. Gradient descent is a common training method.

2.2.3 Residual Neural Network (ResNet)

In a simple convolutional neural network where the convolutional layers are stacked on top of each other. A convolutional neural network with more layers can provide more accurate results. There are two scenarios to consider when more layers are added to the network: the network learns new weights, or the network does not learn any new weights. When the network does not learn new weights, it could result in a state where the weights are not updated effectively during each training phase. Here, adding more layers would only increase the computational overhead, and no improvement in terms of accuracy.

ResNet [11], a kind of convolutional neural network architecture, is used to mitigate this problem. It uses a residual block. The residual block consists of a residual function $F(x)$ and an identity mapping function as illustrated in Figure 2.5. The residual function $F(x)$ is given using the following formula:

$$F(x) = H(x) - x \tag{2.4}$$

where x is the input to the residual block, $H(x)$ is the final output of the residual block. The residual function is the difference between the input and the output of the residual block. This allows the network to learn $F(x)$ instead of $H(x)$. Along with $F(x)$, the residual block also give our network the ability to learn the identity mapping of the input to the output. The above equation can be rearranged as follows:

$$H(x) = F(x) + x \tag{2.5}$$

$H(x)$ is passed to the following layers in the network.

The advantage of using residual function $F(x)$ is that the network learns new weights, say wt_1 and wt_2, from the convolutional layers ($Layer1$ and $Layer2$ depicted in Figure 2.5) in the residual block. In the worst case, when the network does not learn any new weights, i.e., when the weights in the residual block tend to zero, the identity mapping of the input layer is used. The identity mapping ensures that the previously learned layers are passed on to the subsequent layers. This approach helps to maintain the accuracy of the network.

Figure 2.5 represents the residual block with added weights and an activation function, ReLU. The layers in the residual blocks are connected in such a way that few layers are skipped between them. This is often referred to as skip connection or shortcut connection. The layers in the residual blocks are connected in such a way

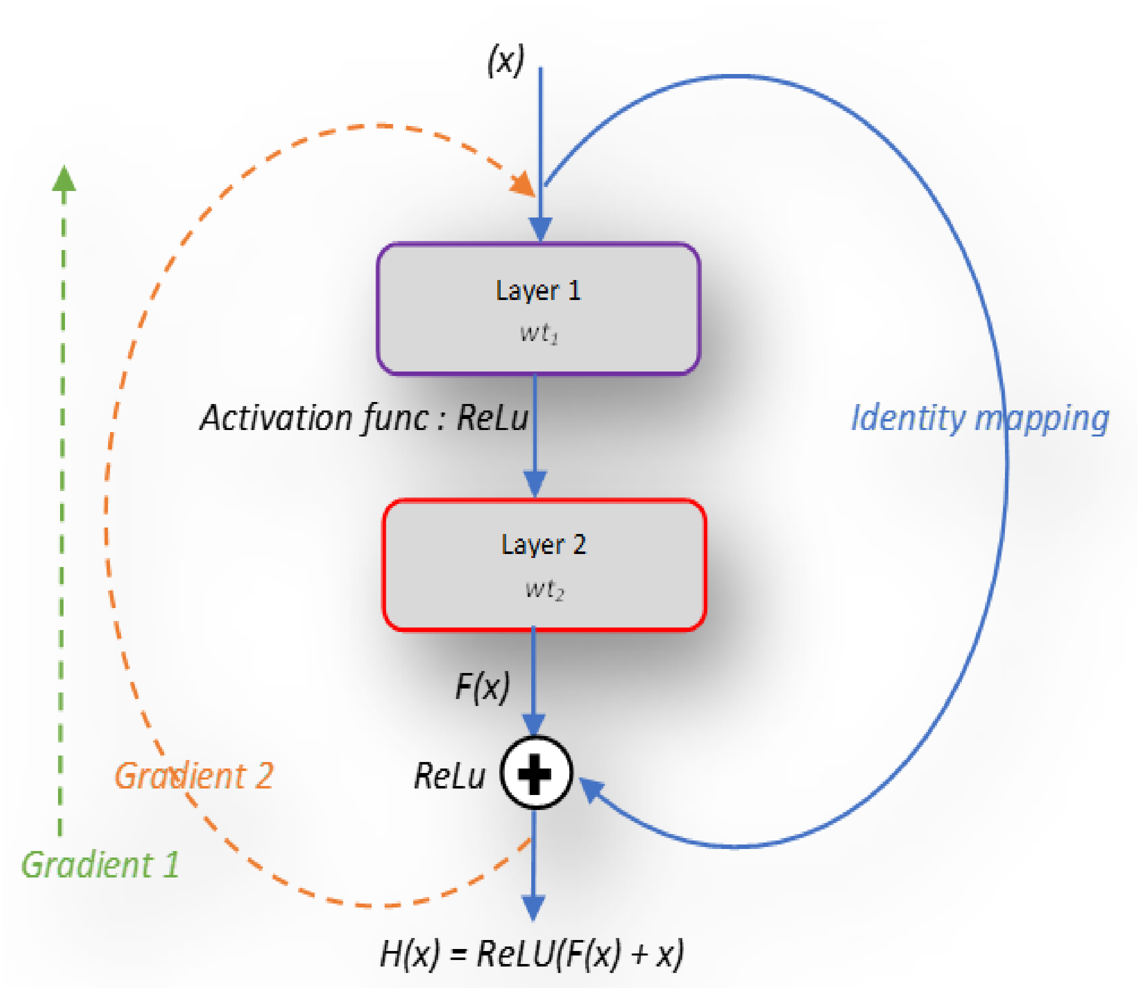


Figure 2.5: Residual block

that few layers are skipped between them. This is often referred to as skip connection or shortcut connection. The blue line in Figure 2.5 represents the skip connection where *Layer*1 and *Layer*2 are skipped. The skip connection is basically an identity mapping function. Here, the input from the previous layers is added to the output of another layer.

ResNet uses identity mapping function [20] to overcome the vanishing gradient problem and accuracy degradation problem when more layers are added. Applying identity mapping to the input will yield an output that is identical to the input. Let X be the input and I be the identity mapping, then their product is $XI = X$. In Figure 2.5, Gradient 1 (green dotted line) represents the gradient traversing back through the residual function during backpropagation, and Gradient 2 (orange dotted line) represents the gradient traversing back through the identity function during backpropagation. The new gradients are computed by updating the weights 'wt_1' and 'wt_2'. The gradients become smaller and may eventually vanish. Here, the gradients reach the initial layers by skipping the residual block. This ensures that the network updates and learns the correct weights.

2.2.4 Global Average Pooling (GAP)

A pooling operation is used to reduce the dimension of an input by encapsulating significant features of the feature map. Global Average Pooling (GAP) [21] is a pooling operation that computes the average of each feature map to encapsulate significant features in the input. It is also used to reduce the spatial dimension of an input by averaging the individual feature maps to output robust spatial information of the input using its pooling operation. In a convolutional neural network, GAP layer is commonly used after the last stage of convolution where there are large number of feature maps. GAP layer does not add any new parameters in the network. Therefore, it speeds up the training process of the network and minimizes the overfitting problem.

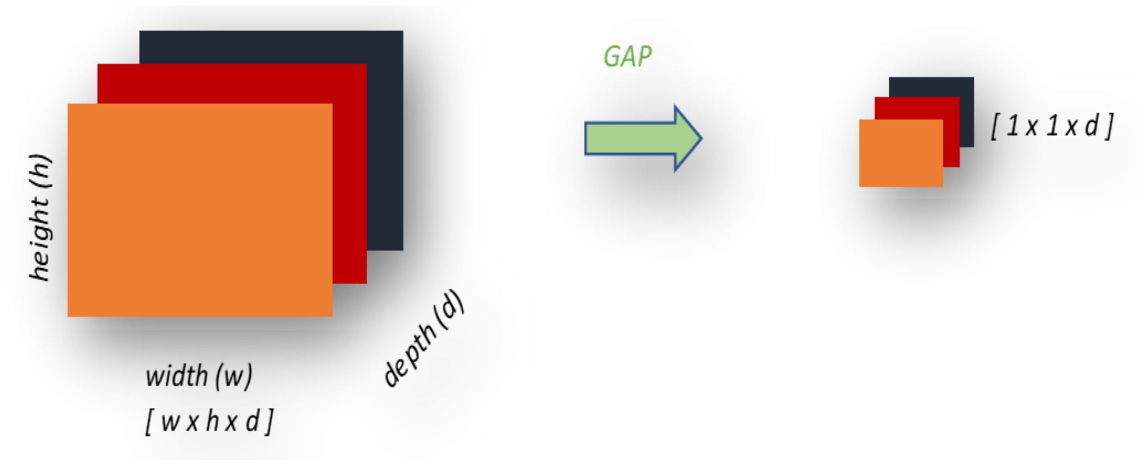


Figure 2.6: Global average pooling

Given a three-dimensional input (width,height,depth), GAP reduces the dimension to (1,1,depth), as shown in Figure 2.6. For example, let the output dimension of a convolutional layer be [wxhxd]:[15x20x2048]. The layer GAP takes the average over the [wxh] values, so it transforms [15x20x2048] to dimension [1x1x2048]. Here depth(d) is the range of values, e.g. 0-255, in a channel.

2.2.5 Conv 1x1

Conv 1x1 (1x1 convolution) is a convolution process using a single filter of size 1x1. It helps in reducing the dimension depth-wise, so that n channels are embedded into a single channel.

Assume an image of dimension [480x640x3] and a filter of size [1x1x3] as shown in Figure 2.7. We use a single filter (filter=1) to perform 1x1 convolution. After performing 1x1 convolution, the number of channel of the input image, 3 in this case, is reduced such that the output is of dimension [480x640x1].

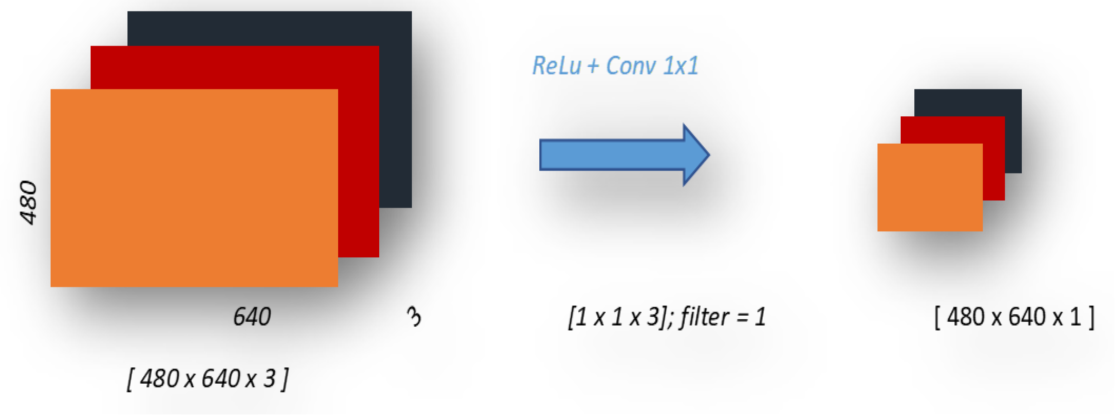


Figure 2.7: Conv 1x1

Conv 1x1 is used in the ResNets' bottleneck design (explained in Subsection 2.2.12). Conv 1x1 layer is usually added before an expensive convolutional layer, such as 3x3 or 5x5. Conv 1x1 [21], is often followed by a nonlinear activation function, such as ReLU, allowing the model to learn more deeply and adjust weights efficiently during backpropagation.

2.2.6 Rectified Linear Unit (ReLU)

ReLU function [22] is defined by the following formula:

$$S(x) = max(0, x) \tag{2.6}$$

It gives an output x, if x is positive and 0 otherwise.

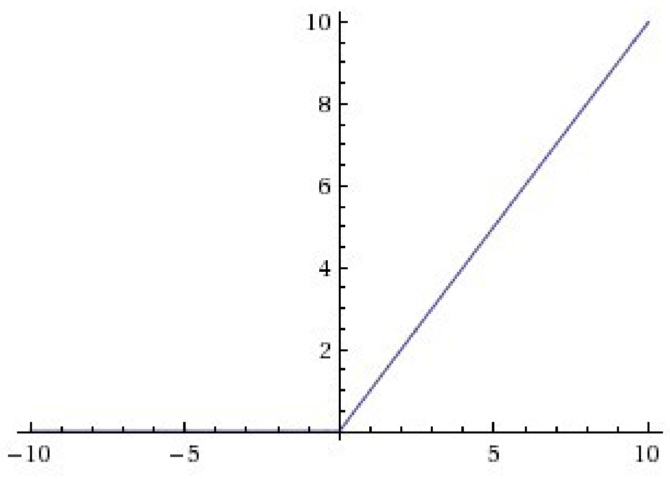


Figure 2.8: ReLU Function [23]

It is important to understand the problem with activation functions like the sigmoid and tanh functions. A general issue is that they saturate, which means that if the input values are large, then in the case of tanh and sigmoid, they tend to 1. If the values are not large they tend to -1 in terms of tanh and 0 for sigmoid. These properties lead to challenges especially in case of adapting the weights for improving the performance of an algorithm.

A solution to this problem is an activation function that helps an algorithm to learn the complex relations in a given data (nonlinear in nature) and at the same time behave like a linear function such that SGD can be used to train neural networks. Also, it should not easily saturate like the other activation functions. In such cases, the ReLU function is suitable and gives better performance. Kaiming initialization is often used with ReLU activation function.

A few advantages of ReLU are: It is computationally inexpensive compared to most of the other activation functions, such as sigmoid. ReLU is used to enable better training of deep neural networks by mitigating vanishing gradient problem.

2.2.7 Loss Function

A loss function is used to measure the degree of relationship between the prediction and the ground truth label. The total loss is taken as an average over a set of data and is calculated using the following formula:

$$L = \sum_{i=1}^{N} L_i / N \tag{2.7}$$

where N is the number of training or validation datasets; i is the i^{th} training sample in a dataset; L_i is the loss calculated for i^{th} data sample. We will now define L_i for binary cross entropy loss.

Binary Cross Entropy (BCE): Binary cross entropy loss is used in binary classification or segmentation problem, where an output predicted by the model is binary: 0 or 1. Binary cross entropy loss is defined using the following formula:

$$L_i = -[gt_i * \log(pred_i) + (1 - gt_i) * \log(1 - pred_i)] \tag{2.8}$$

where gt_i is the ground truth class value for the i^{th} training sample, 0 or 1; $pred_i$ is the predicted class value for the i^{th} training sample, 0 or 1.

For binary classification problems, the output layer uses a sigmoid[1] function followed by BCE loss. The sigmoid function is applied to the predicted output so that the resulting values are between 0 and 1. Given an input x, the sigmoid function is

[1]Sigmoid/Logistic function converts real numbers into probabilities in the range [0,1]

given by,

$$S(x) = 1/(1 + e^{-x}) \tag{2.9}$$

As x approaches infinity, $S(x)$ approaches 1 and as x approaches negative infinity, $S(x)$ approaches 0.

Dice Loss: The Sørensen–Dice coefficient [24] or the dice loss [25], is the harmonic mean of recall and precision.

Recall is the measure of actual features that are retrieved and is defined using the following formula:

$$Recall = TP/(TP + FN) \tag{2.10}$$

where TP (True Positive) is the number of cases in which the model predicts a class and the class is also present in the ground truth; FN (False Negative) is the number of cases where a class is present in the ground truth and the model does not predict it.

Precision is the measure of positive features among the actual features retrieved and is defined using the following formula:

$$Precision = TP/(TP + FP) \tag{2.11}$$

where FP (False Positive) is the number of cases in which the model predicts a class and the class is not present in the ground truth.

In terms of recall and precision, Dice coefficient is defined using the following formula:

$$Dice\ coefficient = (2 * Recall * Precision)/(Recall + Precision) \tag{2.12}$$

Dice coefficient attaches equal importance to false positives (FP) and false negatives (FN) as shown in Figure 2.9. Therefore, it is highly immune to class-imbalanced datasets.

As illustrated in Figure 2.9, the union of prediction mask and ground truth label is the number of pixels present in prediction mask, in ground truth or both in prediction mask and ground truth label. The intersection of prediction mask and ground truth label is the number of pixels present both in prediction mask and ground truth label.

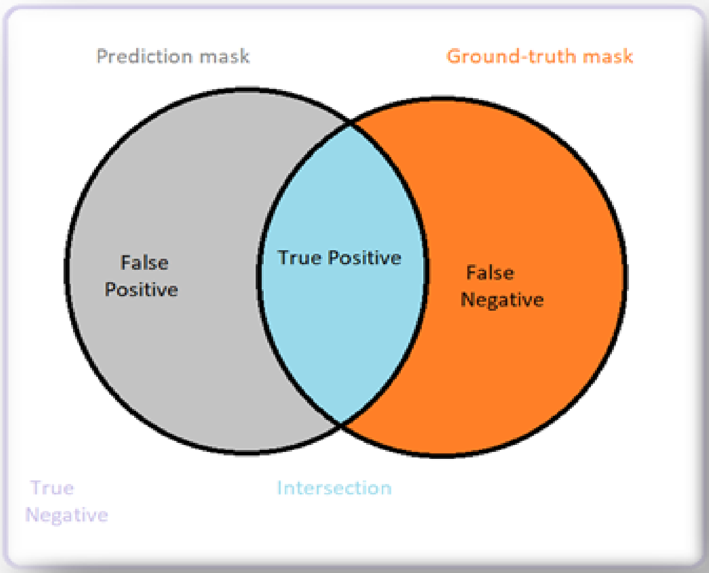


Figure 2.9: Dice coefficient

$$\begin{aligned} \textit{Dice coefficient} &= (2 * Intersection)/(Union + Intersection) \\ &= (2 * TP)/((2 * TP) + FN + FP) \end{aligned} \tag{2.13}$$

$$\textit{Dice loss} = 1 - \textit{Dice coefficient} \tag{2.14}$$

2.2.8 Optimizers

Adam (Adaptive moment estimation): Adam [26] is a method of first order stochastic gradient optimization. It is an extension of the classical stochastic gradient descent algorithm. It additionally performs step-size annealing and computes the learning rates adaptively from the estimates of the first and second moment of the gradients. In the pytorch library, the running means of the gradients and its square are computed from the coefficients, β_1 and β_2. The parameter updates depend on the value of the momentum. The momentum update ensures that the parameter updates are in the direction of the constant gradient of the loss function. This guarantees a better convergence of the network.

2.2.9 Learning Rate Scheduler

Reduce on Plateau (RoP): Learning rate schedulers are used to anneal the learning rate over time. RoP is a learning rate scheduler that dynamically reduces the learning rate based on the validation loss. Machine learning models often benefit from reducing the learning rate by a certain factor. For example, the learning rate is reduced every 5 epochs. In the Pytorch library, this is fulfilled by the 'patience' parameter. Once the patience value is set, the scheduler monitors a metric, such as validation loss, for improvements. If there are no improvements for the set patience value (number of epochs), the learning rate is reduced by a factor of 0.1 by default.

2.2.10 Metrics

There are several metrics that are commonly used to evaluate the accuracy of the semantic segmentation, such as pixel accuracy, mean intersection over union (mIoU), dice score, etc., [27, 28].

Pixel Accuracy: Pixel accuracy is the measure of number of correct pixels among the total number of predicted pixels. It allows us to compare each pixel of the predicted masks with the ground-truth label.

Intersection over Union (IoU): IoU or Jaccard's Index is the area of overlap (intersection) between the predicted segmentation and ground truth divided by the area of union of the predicted segmentation and ground truth, as shown in Figure 2.10.

IoU is a measure of the percentage of overlap. Therefore, IoU is a value between 0 and 1, where 0 represents no overlap and 1 represents that the prediction and ground truth overlap completely with each other. A higher value of IoU corresponds to more accurately predicted segmentation.

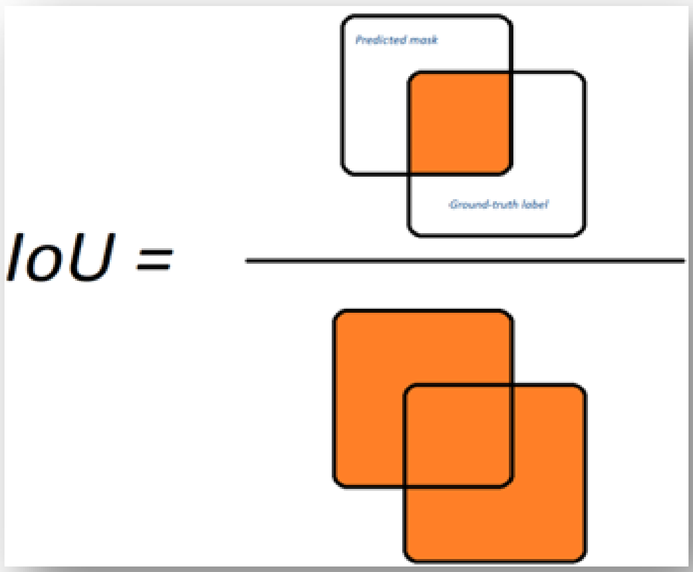


Figure 2.10: Intersection over Union (IoU)

IoU is defined using the following formula:

$$\begin{aligned} IoU &= Intersection/Union \\ &= TP/(TP + FN + FP) \end{aligned} \tag{2.15}$$

Dice score: The dice score or dice coefficient has already been covered in the section 2.2.7 when discussing about dice loss.

2.2.11 Transfer Learning

In machine learning, transfer learning is an approach that aims at gaining knowledge by solving a problem and using it to solve another related problem. There are two types of transfer learning [29] in convolutional neural networks: Feature extraction and fine-tuning. In feature extraction, a pre-trained convolutional neural network such as ResNet50, VGG16 etc. is chosen for feature extraction. The weights of the pre-trained model are frozen. A few layers can be added to the frozen layers and then trained. In fine-tuning, a pre-trained model is used to re-train the entire model by updating the weights through backpropagation. The pre-trained ResNet50 is trained on 1.2 million images with 1000 classes. The last fully connected layer is usually removed and the rest of the layers are trained for feature extraction.

The transfer learning process includes initializing a pre-trained model, choosing the fine-tuning or feature extraction, defining the optimization algorithm to update the correct weights. Finally, the model is trained, the model is validated to check if the model learns, and then tested to evaluate if the model has learned. It is computationally intensive and takes longer to train a convolutional neural network on a huge dataset like Imagenet. The Pytorch library model_zoo contains a list of pre-trained convolutional neural network architectures. As mentioned earlier, the model can be trained either from scratch, the lower layers or only the last layers. It is important to carefully select the pre-trained model based on the problem to achieve the desired prediction.

2.2.12 ResNet50-Siamese

2.2.12.1 Architecture of ResNet50

ResNet50 takes an input image of spatial dimension [height,width,channels]. The width of an image is multiple of 32. The number of channels should be 3. Each residual block consists of three convolutional layers.

ResNet50 has four main convolutional stages. The input size is halved and the channel width is doubled in each convolutional stage. The architecture of ResNet50 is depicted in Figure 2.2.12.1 [11].

There is an initial convolutional module performed with a 7x7 kernel size and 64 kernels, followed by max-pooling with 3x3 kernel sizes. A stride of 2 is used. For an input dimension [480,640,3], the output dimension of the initial convolution is [240,320,64] and the max-pooling layer will be [120,160,64].

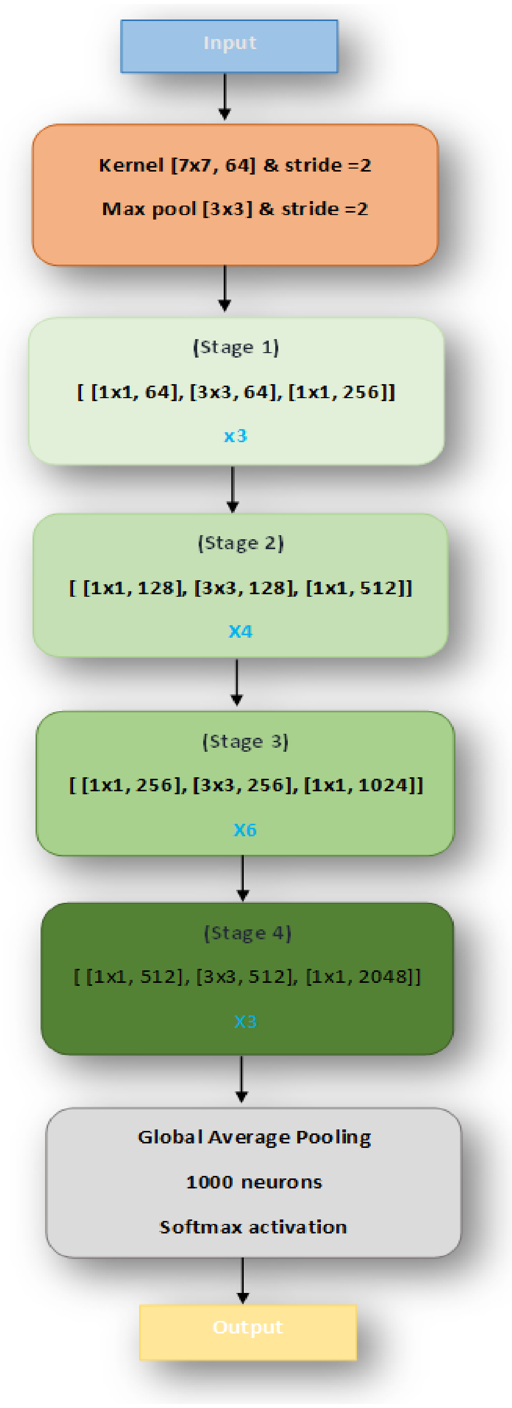

Input
Kernel [7x7, 64] & stride =2
Max pool [3x3] & stride =2
(Stage 1)
[[1x1, 64], [3x3, 64], [1x1, 256]]
x3
(Stage 2)
[[1x1, 128], [3x3, 128], [1x1, 512]]
X4
(Stage 3)
[[1x1, 256], [3x3, 256], [1x1, 1024]]
X6
(Stage 4)
[[1x1, 512], [3x3, 512], [1x1, 2048]]
X3
Global Average Pooling
1000 neurons
Softmax activation
Output

Figure 2.11: ResNet50 architecture

The first stage has [1x1, 64], [3x3, 64] and [1x1, 256] kernels, stacked together as the residual block. There are three residual blocks which results in 9 layers. Here, [1x1, 64] represents 1x1 convolution and 64 filters/kernels.

The second stage has [1x1, 128], [3x3, 128] and [1x1, 512] kernels. These kernels are stacked together and repeated four times, resulting in 12 layers.

The third stage has [1x1, 256], [3x3, 256] and [1x1, 1024] kernels, which are repeated six times to form 18 layers.

The fourth stage has [1x1, 512], [3x3, 512] and [1x1, 2048] kernels stacked together and repeated three times to form 9 layers.

Each residual block is stacked with three layers, namely 1x1, 3x3 and 1x1 convolution. These three layers form the bottleneck layer in ResNet architecture with 50 or more layers. The purpose of introducing bottleneck design is to reduce the computational cost in networks with more layers. As more layers are added and the network gets deeper, the 3x3 convolution becomes an expensive operation. Therefore, in a bottleneck design the dimension is reduced by a 1x1 convolution before performing a 3x3 convolution. Finally, the dimension is restored by a 1x1 convolution. The residual block in a ResNet with smaller number of layers (18 and 34 layers) consists of only two 3x3 convolution layers stacked on top of each other.

Each convolutional layer consists of batch normalization followed by an activation function, ReLU. Batch normalization [30] makes the network more stable and faster by mitigating the problem of internal covariate shift[2]. ResNet50 also uses a global average pooling layer, followed by a densely connected layer (having 1000 neurons corresponding to ImageNet class output) and softmax activation, resulting in a single layer.

2.2.12.2 Reconstruction Network

Recontruction Network is used to create an image from the feature maps obtained from ResNet50. It consists of a transposed convolutional layer (TransposeConv) and a bilinear upsampling layer (Bilinear). The transposed convolution helps in reconstructing the spatial dimension of the input.

The transposed convolutional part consists of a convolutional layer followed by transposed convolutional layers assuming input channels 16, 32, 64, kernel size of 1 and ReLU as activation function. Transposed convolution is used in semantic segmentation to up-sample the input feature maps from the convolutional stages of ResNet50 to a high-resolution feature map. It helps in learning its own parameters by updating the weights through backpropagation. Only the low-level features are passed as input to the transposed convolutional layer.

Bilinear upsampling uses a bilinear interpolation technique. It is generally a linear interpolation performed in two different directions. The feature maps from each stage of convolutional and the global average pooling layer are upsampled using

[2]The phenomenon of random distribution of input data across the layers of a neural network. For more details refer section 3.6.

bilinear upsampling, followed by convolution to preserve the spatial dimension of the image. The deep feature maps 256, 512, 1024, 2048 are directly upsampled using bilinear interpolation. Batch normalization is used to speed up the training of the deep network.

The kernel and output shape for each layer of ResNet50 and Reconstruction Network can be found in the appendix A.1.

2.2.12.3 Siamese Network

Siamese network [31, 32] forms a parallel network that share the same architecture. Siamese network shares the same set of weights and learn to differentiate the inputs rather than classify the inputs. Therefore, it learns the distinct image similarities.

The Siamese network forms the basis for scene change detection tasks. It is fed with a pair of images as input, for which the changes are to be detected. In this study, the ResNet50 and the Reconstruction network (represented as "TransponseConv_Bilinear_Net" in Figure 2.12 and Figure 2.13) are integrated to form the Siamese network. The network model presented here is inspired by the study of Varghese et. al. (2018) [33] and is an extension to it. It maps the feature space to the desired change map corresponding to the dimension of the input image. This is achieved by either merging the multilayer features from the parallel network or taking the absolute difference between the parallel networks. The weights of the ResNet50 are shared and the weights of the Reconstruction network are independent.

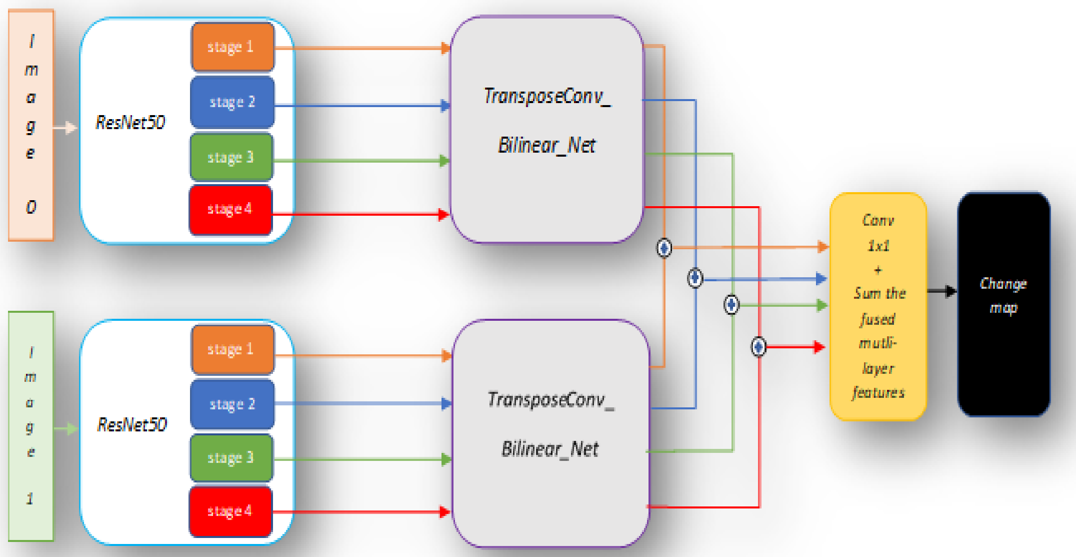


Figure 2.12: Siamese_Res50_Fuse_Net

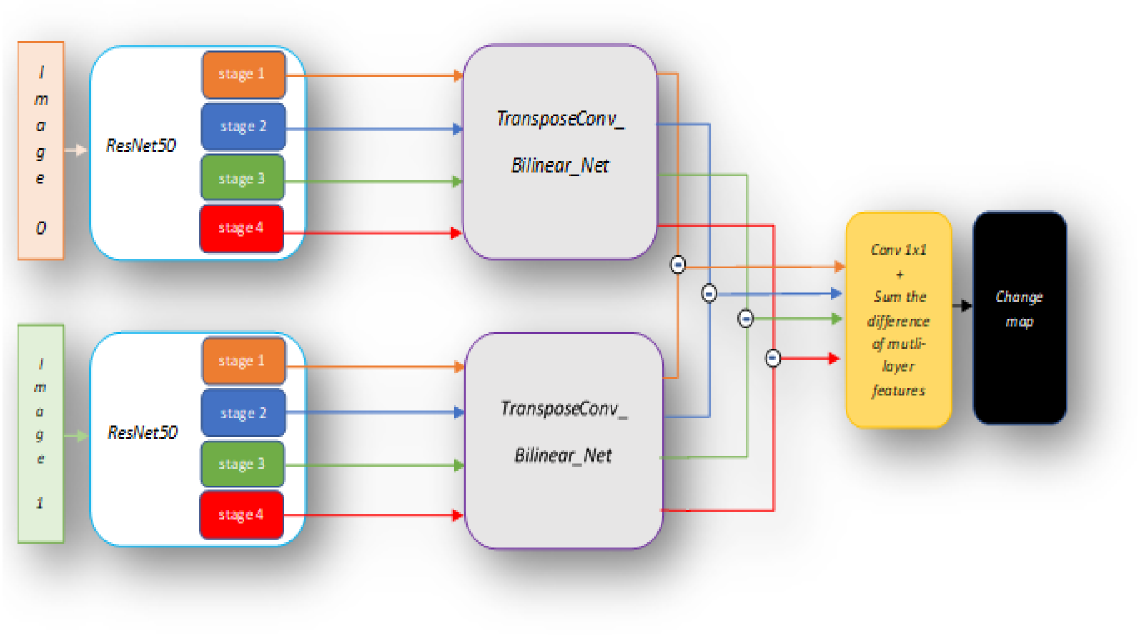


Image 0
ResNet50
stage 1
stage 2
stage 3
stage 4
TransposeConv_ Bilinear_Net
Image 1
ResNet50
stage 1
stage 2
stage 3
stage 4
TransposeConv_ Bilinear_Net
Conv 1x1 + Sum the difference of mutli-layer features
Change map

Figure 2.13: Siamese_Res50_Diff_Net

The model is called *Siamese_Res50_Fuse_Net* (Figure 2.12) when the outputs of the Siamese network are merged/ fused. Based on the difference operation on the output of the Siamese network, the model is called *Siamese_Res50_Diff_Net* (Figure 2.13).

2.2.13 Pyramid Pooling Module

CNNs are followed by fully connected layers that accept input of a fixed size thus making it unacceptable for the CNN to have varied size inputs. Therefore, images are first converted to a specific dimension before being fed into the CNN. This leads to another problem of image warping (where the image may be distorted in some way) as well as reduced resolution. Spatial Pyramid Pooling (SPP) helps to solve this particular problem. It manages the information in local bins (spatial). The number and size of these bins are fixed. The pooled responses of each filter are available in the bins.

The output feature map has 256 filters, as shown in Figure 2.14, and is of arbitrary size (depending on the input size). As seen in Figure 2.14, there are three pooling layers and the first one is similar to the global pooling operation, whose output is 256-d. The second pooling layer has 4 bins, which gives an output of 4*256 and similarly the third with 16 bins gives an output of 16*256.

A flattened and concatenated version of the output layer is obtained so that the dimension remains the same regardless of the size of the given input.

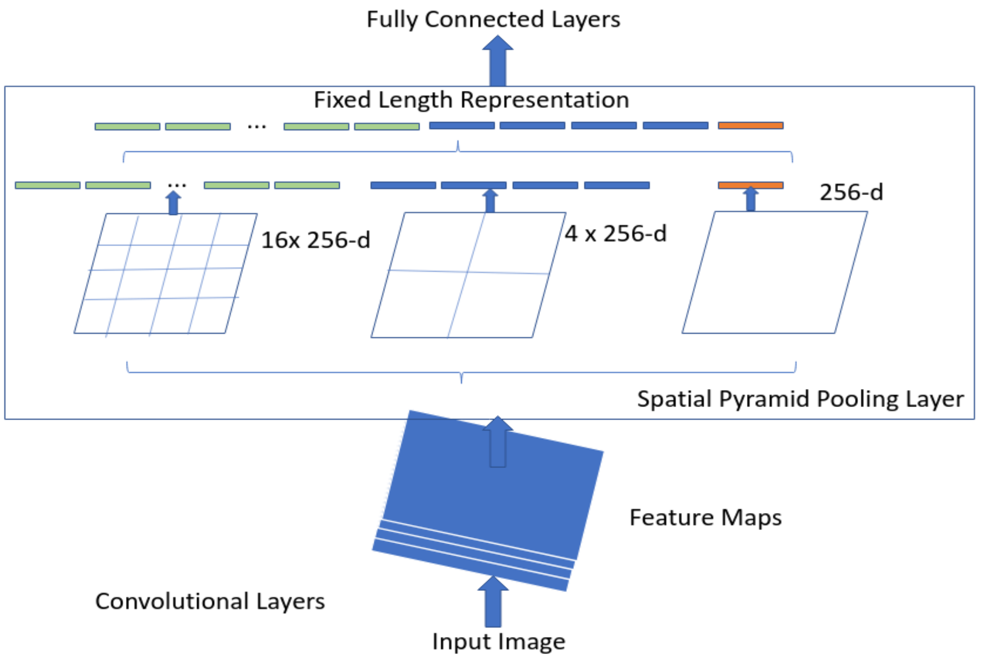


Figure 2.14: Example of SPP

3

Literature Review

This chapter will walk through the existing studies on machine learning algorithms for scene change detection. It will also discuss the studies on semantic segmentation, residual neural network, pyramid pooling, global average pooling, optimization algorithms and batch normalization.

3.1 Scene Change Detection

Jong et al. (2019) [34] study change detection using unsupervised learning with satellite images. The authors also address the need for future research in terms of improving the accuracy and noise resistance of a model. Sakurada et al. (2015) [31] detect the change externally for a given geographical area using the city landscapes. This paper proposes a method using a pair of its vehicular, omnidirectional (360°) images for detecting changes in a scene. The images are taken at different times and have temporal differences in illumination and photographing conditions. They make use of a fully convolutional Siamese network to overcome visual differences between image pairs. Zhao et al. (2019) [32] also use a Siamese encoder-decoder network for street-view change detection.

Varghese et al. (2018) [33], use a Siamese network to form a parallel network. ResNet50 forms the backbone of the network where the last three stages of convolutional layers are used to extract the features. The convolutional layer is then followed by a bilinear upsampling layer to upsample the features from the three convolutional layers. Finally, the feature maps are merged and fed to a softmax layer to generate the changes in image.

3.2 Semantic Segmentation

A semantic segmentation network is similar to an encoder-decoder network. The encoder is similar to any convolutional neural network model, like the ResNet, GoogleNet, etc. It is used to extract low and high resolution features. The decoder has a different mechanism that helps to recover the spatial information and produce the segmented prediction.

Ronneberger et al. (2015) [35] (U-Net) proposed a U-shaped architecture. It has a contracting (encoder) and expansive (decoder) path to perform semantic segmentation. The contracting path is like the convolution layer in CNN. It extracts the features from the image. The expansive path is similar to the transposed convolution (deconvolution/ upsampling) network. It takes the feature set from the contracting path and recovers the spatial information lost in the contracting path. Additionally, every step in the expanding path is concatenated with the high-resolution feature set from the contracting path. The combination of high-resolution features and the spatial information produce better segmentation result. The authors claim to train a network with very few images.

Zhao et al. (2017) [27] (PSPNet) is a scene parsing network. PSPNet has a CNN model to extract all the features of an image. Then, it exploits the capability of the global contextual information by feeding the pyramid pooling module with the feature space. The output from the pyramid pooling module is then upsampled and concatenated with the initial feature set from the convolution layer for pixel-wise prediction. It captures both local and global contextual information.

Chen et al. (2018) [36] (Atrous Spatial Convolution, ASPP) propose a powerful encoder module in the encoder-decoder model by applying several parallel atrous convolutions to capture higher semantic information and for a faster computation. Atrous or dilation rate is a parameter used to increase the receptive field of each layers. A larger kernel can be used for the same purpose. But the number of parameters increases with the size of kernels. Atrous is introduced in DeepLab as a tool to adjust/control effective field-of-view of the convolution. The model aggregates feature from the image at different scales. These models have shown success in several segmentation benchmarks.

3.3 Residual Neural Network

The network convergence can be hampered at an earlier stage due to the vanishing/ exploding gradient [37]. It is shown in the studies [38] that deeper networks can result in higher training error rate. Here, the network shows good performance at the beginning, but gradually the accuracy gets stagnant and degrades swiftly.

He et al. (2016) [11] propose a residual network to mitigate the vanishing gradient problem, to reduce the training error rate and to increase the performance of deeper networks.

The “Highway Networks” [39] is a similar technique to ResNet, which also uses a skip/ shortcut connection. However, the amount of information to pass through the skip connections are controlled by a parametric gate. Also, since the gates can be closed, the layers represent non-residual functions. Nevertheless, the identity shortcut connections in the ResNets are never closed.

3.4 Global Average Pooling (GAP)

GAP, first proposed in [21], is placed as the last layer which intake the feature maps of the last max pooling layer. The GAP layer output is a single-entry vector for each class in the classification task. The studies [40, 41], use global average pooling (GAP) to add global context information to their model framework. Zhou et al. [42] added a GAP layer with the convolutional neural networks for the purpose of object localization. This network is then trained for image classification. Hence, the object in the image can be detected using the convolutional neural network, and the addition of the GAP layers allows us to know where the object is contained in the image.

3.5 Optimizers

The gradient descent optimization algorithms [43, 44] are used in the field of deep learning neural network to optimize the neural networks. Gradient descent [45] is an optimization algorithm used to obtain the best set of weights in a network by finding local minima of a function. Stochastic gradient descent algorithm [46, 47, 48, 49] is an iterative method for optimizing gradient descent. The algorithm iterates through the training set. An adaptive learning rate (a parameter to tune an optimization algorithm that determines the step size for finding the minima) in conjunction with shuffled dataset during each training can help the algorithm converge better. Adam [26] is an extension of stochastic gradient descent.

3.6 Batch Normalization

The input data and parameters across each layer of a neural network can influence the training process. The parameters of the current layer can change the input data distribution of the succeeding layer. This phenomenon of random distribution of input data across the layers is described as internal covariate shift. Batch normalization [50, 30] mitigates the problem of internal covariate shift and makes the neural network stable and faster.

In a nutshell, Siamese network is a great architecture in a scenario where two different images are to be processed simultaneously. By the nature of its architecture, it can help in reducing the number of parameters and the memory footprint while training. ASPP, PSPNet, etc., models for semantic segmentation are trained on a deeper network. They use ResNet with 101, 152 layers for feature extraction and the entire model is trained using multiple GPUs. Therefore, we try to implement a model which can be computationally efficient and also yield accurate predictions. Inspired by the study of Varghese et al. (2018) [33], we exploit the feature extraction capability of ResNet using 50 layers and the siamese network architecture for processing two different images and to produce the desired semantic change map.

3.7 Spatial Pyramid Pooling

Zhang et al. (2015) [51] illustrate the effectiveness of using Spatial Pyramid Pooling (SPP) techniques in deep learning and visual recognition. The existing deep CNN model needs images of a fixed size to process and predict the objects and scenes. For this purpose, input images need to be either cropped or resized. While cropping, some of the objects in the images disappear. When resized, the dimension changes, and the resolution also changes which reduces the clarity of an image.

If this particular image is given as an input, it becomes difficult for the system to predict the exact scenes or objects. To mitigate these issues, a spatial pyramid pooling technique has been introduced. In this concept images of any size and specifications could be given as input. SPP layer is placed on top of the last layer as seen below.

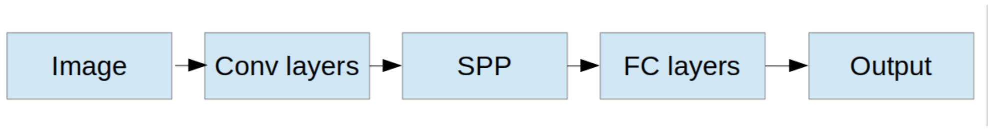


Figure 3.1: Spatial Pyramid Pooling presented in the paper

SPP is very significant in object detection. When the same model is tested for two other datasets namely Pascal VOC 2007 and Caltech1 01, it yields the best results and high accuracy scores compared to the other models. For Pascal VOC 2007 dataset, SPP model yields an accuracy score of 82.44 percent which is high than the previous high accuracy score of 81.58 percent and when the model has used for Caltech101 the model yields output scores of 93.42 percent which is very much higher than the previous best (88.54 percent). SPP has turned out to be one of the milestones in deep learning techniques in recent times. Overall, SPP is a better solution for handling images at different scales and sizes to yield approximate predictions.

In the study [27], the Pyramid Scene Parsing Network (PSPNet) method is introduced to predict the scenes and object in the given images(datasets) with better accuracy. It is a kind of deep neural network that primarily uses a Convolutional Neural Network (CNN). This paper initially suggested using Fully Convolution Network (FCN) but later decided to invoke PSPNet (Pyramid Scene Parsing Network) into the model to overcome some drawbacks of using FCN. FCN could not recognize some of the objects and scenes in real-time, whereas authors claim that the PSPNet analyzed the same scene with utmost accuracy to find the exact object.

FCN (Fully Convolutional Network) sometimes wrongly identified the objects (car instead of a boat, skyscraper instead of buildings, etc.). To avoid these kind of drawbacks the spatial pooling system and Pyramid Scene Parsing techniques have been introduced. Global average pooling is used as a baseline model. Using spatial pooling increased the accuracy scores and turned out to be more efficient. ADK dataset and few other datasets have been tested through this method, resulting in higher accuracy scores compared to other methods.

PSPNet is a pixel prediction framework that authors claim is ideal for applications like driving, robot sensing, etc. In PSPNet each pixel in the image is assigned with a category label. It clearly understands the scene and predicts the objects based on the scene. The input is first sent to CNN then the resultant of which is sent to the pooling layer. After this, the obtained output is sent to different layers of the pooling system.

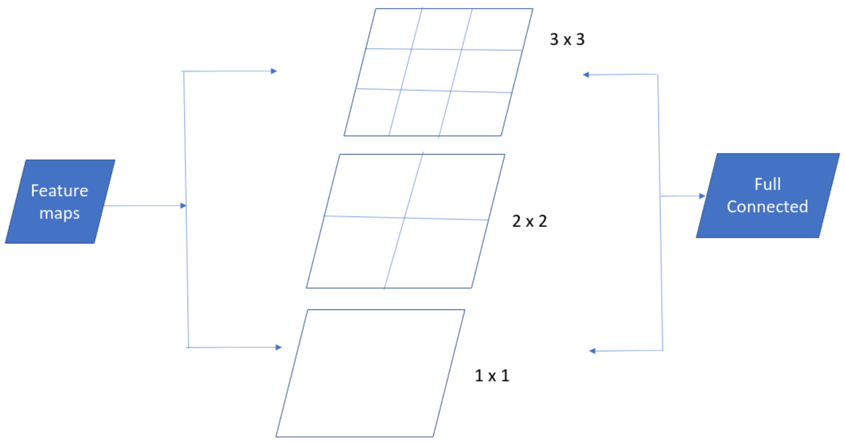


Figure 3.2: Example of Spatial Pyramid Pooling

In Figure , each layer of the SPP network performs the unique job assigned to these layers, and all these outputs are finally concatenated to get the final output which is predicted. For any model, the number of pyramid level and size of each level depends on the type of dataset used and modified. In the ADK dataset, the PSPNet model proposal is a remarkable achievement and solves all the common problems faced in the FCN model. Furthermore, the PSPNet model is used for the Cityscapes dataset to check for accuracy in predicting change, and it secures the best accuracy scores compared to other techniques. It also finds its applications in the field of military intelligence, where the prediction of correct objects and scenes is highly crucial, which is successfully executed using the PSPNet method.

When PSPNet models are pre-trained and then used, it further increases the accuracy scores of the model. In some cases, FCN recognizes two different objects as the same objects (mostly since both are in the same color), but PSPNet correctly identifies them as different objects. Cost usage of PSPNet and FCN networks are almost similar (the computational cost is not too high for using this model). PSPNet stands out in its ability to capture diverse scenes and unrestricted vocabulary.

3.8 Activation Function

In recent times computer vision and natural language processing have become so popular and widely used worldwide. These models are more powerful and effective and could be used for large-scale datasets containing even millions of data.

The authors in [52] primarily deals in activating the neurons in deep neural networks. The activation of neurons and loss functions play a vital role in the deep learning model. The calculation of loss functions and activation of neurons highly influence the efficiency and accuracy scores of the model. Each model has several biological neurons. These artificial neurons are arranged in an orderly manner. Each of these neurons is activated by electrical signals which are sent by the previous neurons. If these electrical signals are big enough to stimulate the neuron then these neurons go to an excited state. If not, they will remain in an inactive state.

The activation process can be carried out in techniques like the sigmoid function and hyperbolic tangent function. In the first technique sigmoid function, the activation of activated neurons is carried out as a saturated process. The activation process consists of several steps that are time-consuming as well as requires high cost. The neurons are activated by a mechanism of sigmoid function only in some instances of Deep learning.

The next technique is hyperbolic tangent function. Here, based on sine and cosine values the tangent values are calculated. The neurons are activated in certain steps and the loss values obtained in this process are comparatively negligible than the sigmoid function. It consumes a bit less time and cost as well, so it is widely used.

The authors in paper [52] suggests an activation function called Rectifier Linear Unit (ReLU) to activate the neurons. This is an unsaturated and supervised model that requires very little cost and less time.

The authors claim that this function can activate the neurons simultaneously. The computational cost is much cheaper than sigmoid and hyperbolic tangent functions. ReLU activation technique performs activation much faster and makes the network easily activated. This method can be carried out even without using any unsupervised or supervised learning methods.

The ReLU method could be carried out in three different methods such as LReLU, PReLU (Parameterised ReLU), and RReLU (Randomised ReLU). Each of these techniques differs only in some aspect while the baseline model remains the same.

The baseline model in the paper consists of five convolutional layers followed by two pooling layers and one fully connected layer. The given dataset consists of 60000 samples and 10000 training samples. When the dataset is given as input to deep convolutional neural networks and all these techniques are used separately to activate neurons the following results are obtained.

When a sigmoid function is used to activate the neurons the obtained error percentage (deviation in original value from predicted) is 1.15 percentage. For hyperbolic tangent, it is around 1.1 percentage whereas for ReLU function it is just 0.8 percentage making it the best among all the techniques. Hence for activating neurons ReLU is the best technique which is being followed for every deep learning method in recent past.

3.9 Loss Function

Nie et al. (2018)[53] in their research primarily focus on loss functions involved in deep learning and machine learning techniques. Loss functions are one of the important factors which influence the whole efficiency and accuracy of the model. Hence, loss functions must be given higher priority, and suitable methods must be chosen by the type of model.

In the deep learning mechanism, there are two different models namely the regression and classification model. In the Regression model, the values will be continuous whereas in the classification model the values will be discrete. Regression is about predicting the quantity whereas classification is about predicting the label. Suitable loss functions must be chosen for any model to get the best results.

There are two types of loss functions namely bi-lateral loss functions and unilateral loss functions. The deviation in predicted value and the original obtained value is known as a hyperplane. If the obtained value is lower the method is best suitable and higher the value the method is not suitable.

In the case of bi-lateral loss functions, the function is calculated for both regression and classification models and the loss (hyperplane) values are obtained. The loss value obtained for the regression model is less than 1 whereas the classification model is greater than 1. So, for the classification model, the values need to be punished (since it is greater than 1). Nevertheless, for the regression model, the loss values need not be punished (since it is less than 1). In this paper, the authors conclude that the bilateral loss functions are most suitable for regression models and less suitable for classification models.

In the case of unilateral loss functions, the loss values are calculated for both regression and classification models. The value obtained for the regression model is greater than 1 whereas for the classification model is less than 1. Hence, for the classification model, the values need not be punished (since it is less than 1) and for the regression model, the hyperplane values must be punished (since it is greater than 1). So, the model concludes that unilateral loss is suitable for the classification model whereas it is not suitable for the regression model. Hence choosing a suitable loss function is necessary for any algorithm.

4 Methods

Based on the literature reviewed in Chapter 3, our technical contributions in the field of scene change detection are discussed in this chapter. Section 4.2 explains the methodologies for the Watershed algorithm. Section 4.3 and Section 4.4 elaborates the methodologies for implementing ResNet50 based model and Spatial Pyramid Pooling model respectively.

4.1 Dataset and Dataloader

The dataset consists of pairs of images, captured by multiple cameras mounted inside the car. The images, originally with a resolution of [960x1280], are downsized to a resolution of [480x640]. The dataset is composed of around six thousand raw images which are then randomly transformed into about twenty thousand image pairs. The dataset is split 80:10:10 ratio during training for the machine learning part which corresponds to training, validation and test datasets. They are trained on two different training datasets, about two thousand (smaller training dataset) and sixteen thousand (full training dataset) respectively. The validation dataset and the test dataset are 10 percent of the full dataset. This includes about two hundred samples for the smaller training dataset and about two thousand samples for the full training dataset. A random seed is used to ensure that the models are tested on the same samples of the dataset. The data loader also provides functions to select subsets of the dataset if necessary. The dataset is then loaded using a data loader.

A Pytorch data loader is used to load the dataset. It allows iteration over the dataset during training. The data loader also provides batched samples by defining the required batch size. During training, a batch size of 4 is chosen. The training and validation dataset is always shuffled.

The training process is accelerated using a CUDA-enabled GPU (Tesla K80). When loading the data, the memory is pinned to enable faster data transfer from the hosts to the GPU.

4.2 Watershed Algorithm

The Watershed algorithm works with grayscale images. There are several preprocessing steps before the difference image is fed to the watershed algorithm. The two color coded RGB images (Red,Blue,Green), are shown in Figure 4.1 and Figure 4.2 respectively.

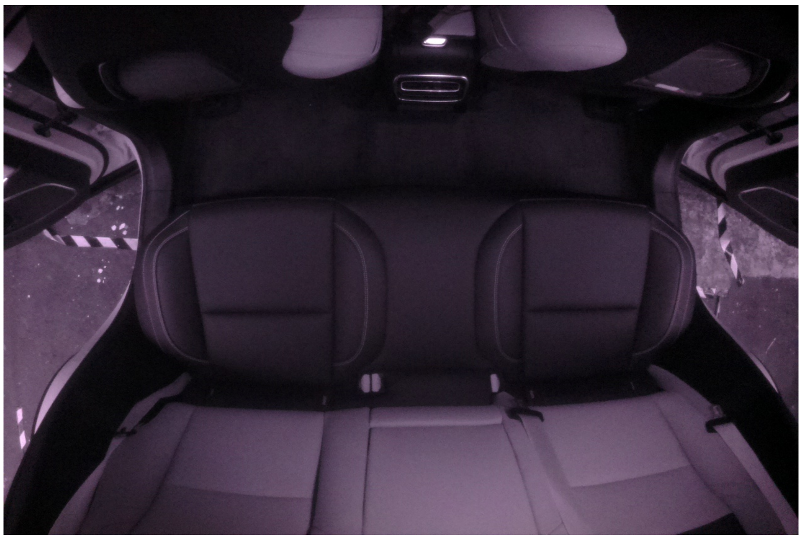


Figure 4.1: RGB input image 1

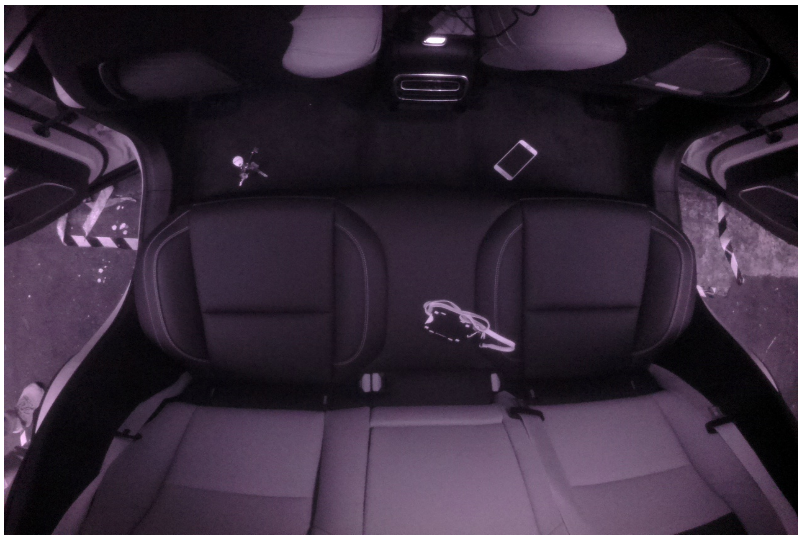


Figure 4.2: RGB input image 2

The difference of two RGB images is generated by finding the absolute difference between the pixel values of two images and then converting it to a grayscale image, as shown in Figure 4.3. This helps us to extract only the pixels that have changed.

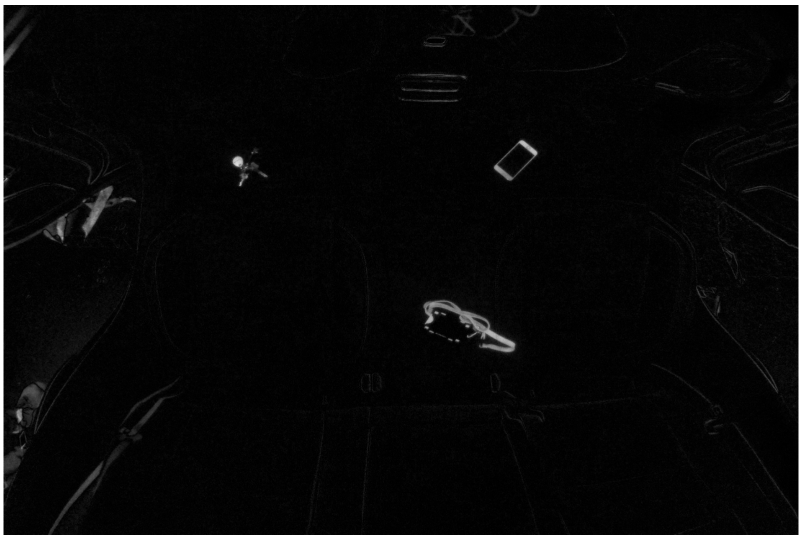


Figure 4.3: Difference image

Thresholding is a key operation in image processing. OTSU [54] is chosen to threshold the difference image. It makes the object appear more prominent. OTSU is a binary thresholding operation. It returns the optimal value of the pixel for thresholding and the array of thresholded image. All thresholded pixels are assigned the value '255'. Here the output is a binary image.

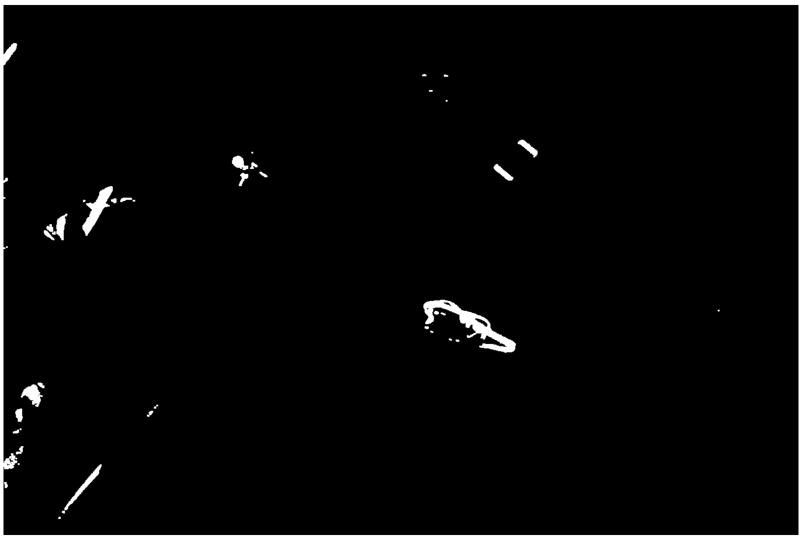


Figure 4.4: Morphological transformation

It is necessary to remove the noise after image thresholding. Then, we apply the morphological transformation, such as dilation and opening, to the binary image to remove the noise. First, the dilation operator is applied to expand the regions of the object pixels (sure foreground region, i.e., the region with maximum intensity, '255'). This is followed by an opening operator to erode the boundary of the dilated object pixel. The noise around the boundary is also removed. A well-defined foreground region, i.e., the object pixels, is now identified. The output of the morphological transformation is shown in Figure 4.4.

Further, distance transformation is applied to the noise free binary input. It operates on the binary image, such that all object pixels are set to a maximum intensity value ‘255’ and the background pixels are set to the lowest intensity value ‘0’.

At this stage, the sure foreground and background are known. A marker is now created. The marker based approach allows us to label the region-of-interest. It is an array that is the same size as the image to label all the regions. The marker is applied to the sure foreground region to label the object pixels with the value ‘255’ (white pixels in the image). All other region are marked with a value ‘0’ (black pixels in the image).

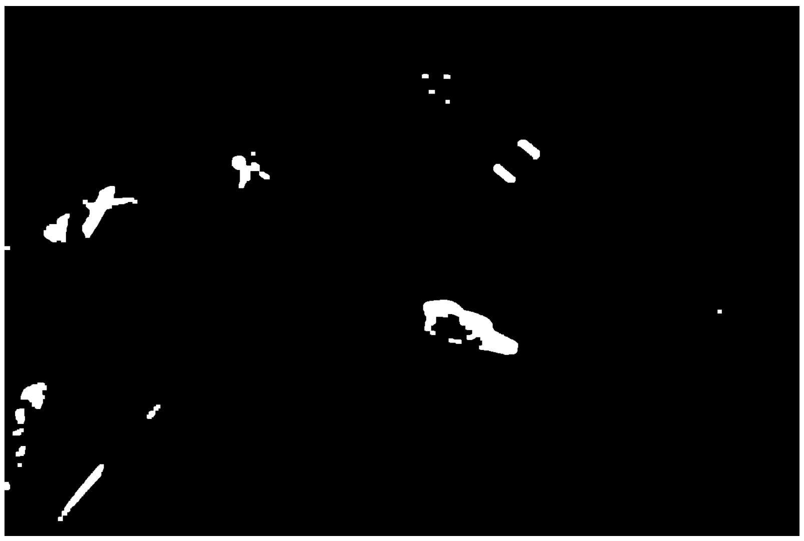


Figure 4.5: Watershed final output

Finally, the watershed is applied on the image along with the marker, resulting in the segmented difference image, as shown in Figure 4.5. We use a combination of morphological transformation and marker controlled watershed to segment the objects in an image.

4.3 ResNet50-Siamese

We use two different techniques in the Resnet50-Siamese based architecture for training. These are the fusing technique (Siamese_Res50_Fuse_Net) and the difference technique (Siamese_Res50_Diff_Net). The fusing technique combines the multi-layer features from the Siamese network. The difference technique takes the difference of the multi-layer features from the Siamese network. The architecture of these two techniques is explained in Section 2.2.12.3 and visually illustrated using Figure 2.12 and Figure 2.13.

Based on the study [1], the base model uses only three stages of ResNet50 convolution, followed by Reconstruction network. The fusing and difference techniques are then applied and trained separately on the smaller two thousand training dataset.

The fusing technique (Siamese_Res50_Fuse_Net) uses all the four stages of ResNet50 convolution, followed by Reconstruction network (TransposeConv_Bilinear_Net). The model is trained on approximately two thousand (smaller training dataset) and sixteen thousand (full training dataset) training datasets.

The difference technique (Siamese_Res50_Diff_Net) also uses the four stages of ResNet50 convolution, followed by Reconstruction network. It is also trained on about two thousand and sixteen thousand training datasets.

Finally, the Siamese_Res50_Fuse_Net and Siamese_Res50_Diff_Net models are also trained with the global average pooling layer (GAP). GAP is applied to the last layer (2048 feature maps) of ResNet50. They are also trained on both training datasets.

Since training on the full dataset takes significantly more time, the selection of the models to train on the full dataset is based on the results obtained by training different models on the smaller training dataset. In all these trainings, the fully connected dense layer of ResNet50 is omitted.

Each model is trained for 20 epochs on the smaller training dataset and for 30 epochs on the full training dataset. The technique with the best performance on the full training dataset is trained for an additional 20 epochs.

Adam is used as the learning rate optimizer, using the default hyperparameter settings: β_1=0.9,β_2=0.999. Reduce on Plateau (RoP) is the learning rate scheduler for both smaller and full dataset. The scheduler's patience is set to 3 and 5 when training on both training datasets. The initial learning rate is set to 0.001. For backpropagation, a combination of binary cross entropy and dice loss is chosen as the loss function.

4.4 Spatial Pyramid Pooling

This algorithm will not be presented in the report due to confidentiality issues associated with the company CEVT AB. It is based on CEVT's internal model.

5

Results

In this chapter, the semantic scene change results of both classical and machine learning algorithm using different metrics are presented. The semantic scene change maps produced by both the algorithms are also depicted.

The performance of classical and machine learning algorithms are compared using a fixed set of test data. This test dataset is unknown to the trained machine learning models. The test dataset is ten percent of the whole dataset consisting of approximately two hundred samples (smaller test dataset) for the smaller training dataset and approximately two thousand samples (larger test dataset) for the full training dataset. The metric score to evaluate the performance of the model is rounded off to two decimal values.

5.1 Watershed Algorithm

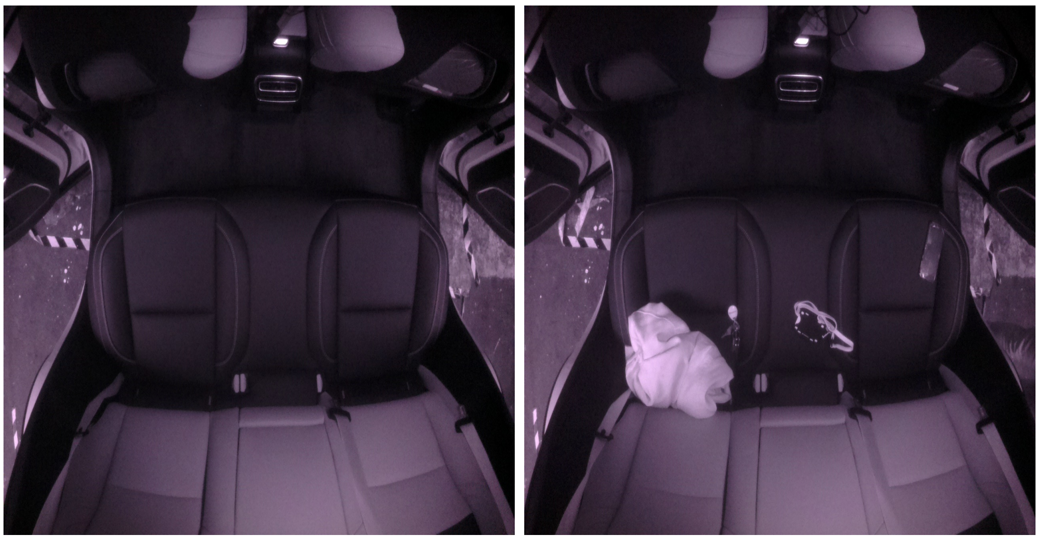


Figure 5.1: (a) Image 1 (b) Image 2

Figure 5.1 represents the two input images for the Watershed algorithm.

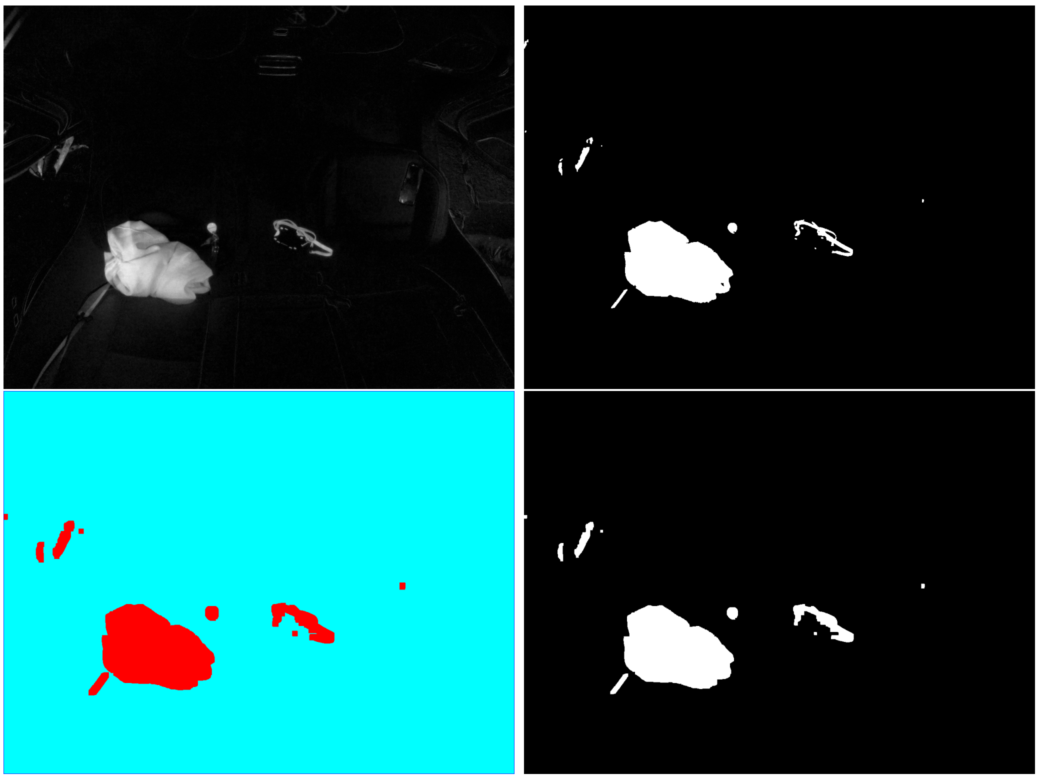


Figure 5.2: IoU : 76.67 (a) Difference image (b) OTSU threshold (c) Watershed (d) Final output

As seen in Figure 5.2, the Watershed algorithm gives fair accuracy with an IoU (Intersection over Union) score of 76.6 for a given image pair. The algorithm segments the larger objects well. In Figure 4.5, the object pixels are comparatively smaller compared to the previous image pair. The algorithm finds it hard to accurately segment the objects as in Figure 4.5, yielding an IoU score of 30.60. The varying level of intensity has left an object undetected. The marker controlled watershed has allowed to control the over-segmentation, i.e. multiple irrelevant regions, only to an extent.

The Watershed algorithm using the samples from the smaller test dataset results in a mIoU (mean Intersection over Union) score of 32.32. The algorithm produced an IoU score ranging from 50.00 to 60.00 for approximately ten percent of the test samples and an IoU score of less than 50.00 for the rest of the test samples. The mIoU score for the larger test dataset is 33.08. The algorithm produced an IoU score ranging from 50.00 to 60.00 for approximately nine percent of the test samples and an IoU score of less than 50.00 for the rest of the test samples.

5.2 ResNet50-Siamese

The ResNet50 based models are evaluated using pixel accuracy, mean intersection over union and dice metrics.

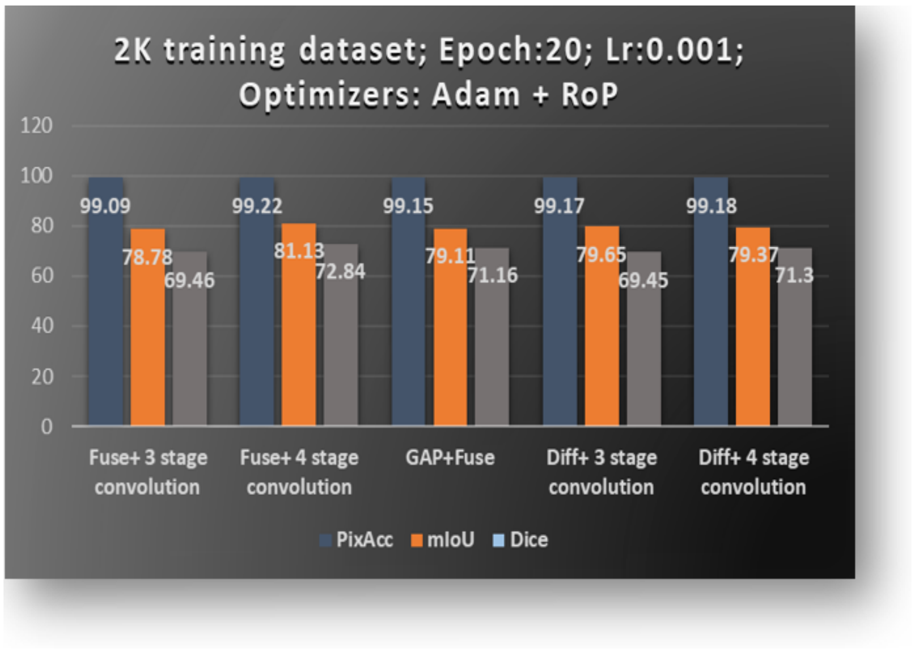


Figure 5.3: ResNet50-Siamese: Inference results on two thousand training dataset

Figure 5.3, shows the inference results obtained by training the models on approximately two thousand custom dataset.

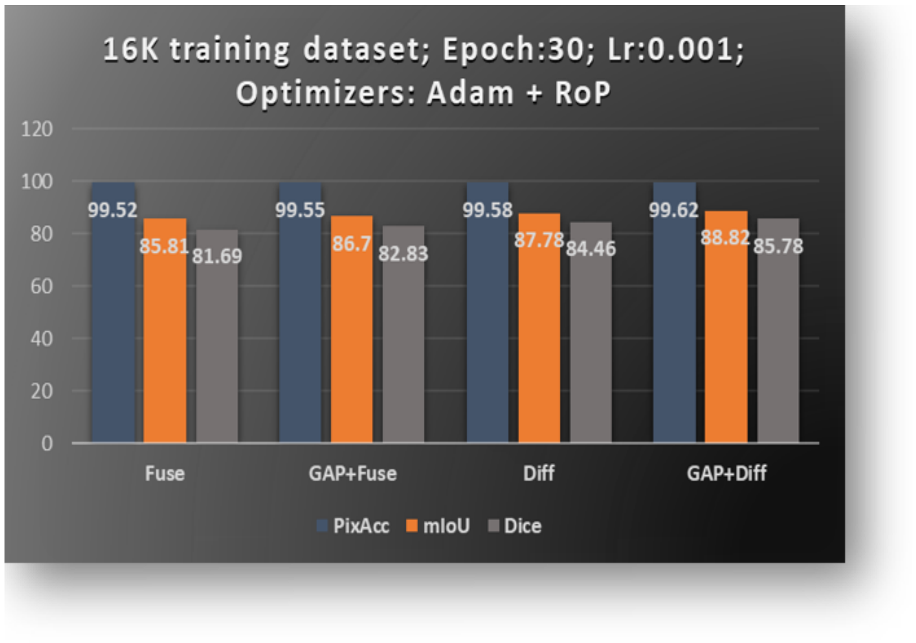


Figure 5.4: ResNet50-Siamese: Inference results on sixteen thousand training dataset

Figure 5.4 shows the inference results obtained by training the models on the full custom dataset, around sixteen thousand.

The following observations are made for the models trained on two thousand and sixteen thousand samples for 20 and 30 epochs respectively:

The models using four stages of convolutions clearly outperforms the model which used only three stages of convolution. The model using the difference technique (Diff) shows better metric score than the model using the fusing technique (Fuse) with respect to pixel accuracy (PixAcc), mean Intersection over Union (mIoU) and dice. The addition of global average pooling layer, further improves the score for the model trained with smaller dataset.

The model using the difference technique yields a mIoU score of 79.37 and 87.78 on the test samples from the smaller and the larger test dataset. All the test samples yields an IoU score greater than 78.00.

Furthermore, the model based on the difference technique is trained further for 20 more epochs to compare the performance of the model using a global average pooling layer. The model using the difference technique yields a mIoU score of 91.11 and the addition of global average pooling layer yields a mIoU score of 90.39. The model has not yet converged, which implies that it can be trained further for more epochs.

The memory footprint of the model, based on ResNet and Siamese architecture, to train using images of [480x640] resolution is around 6GB. For the graphs on the training loss (Loss/train) and validation loss (Loss/val) for the best four models, refer Appendix A.2.

Figure 5.5 shows two different images (Image 1 and Image 2) captured at different time, respectively.

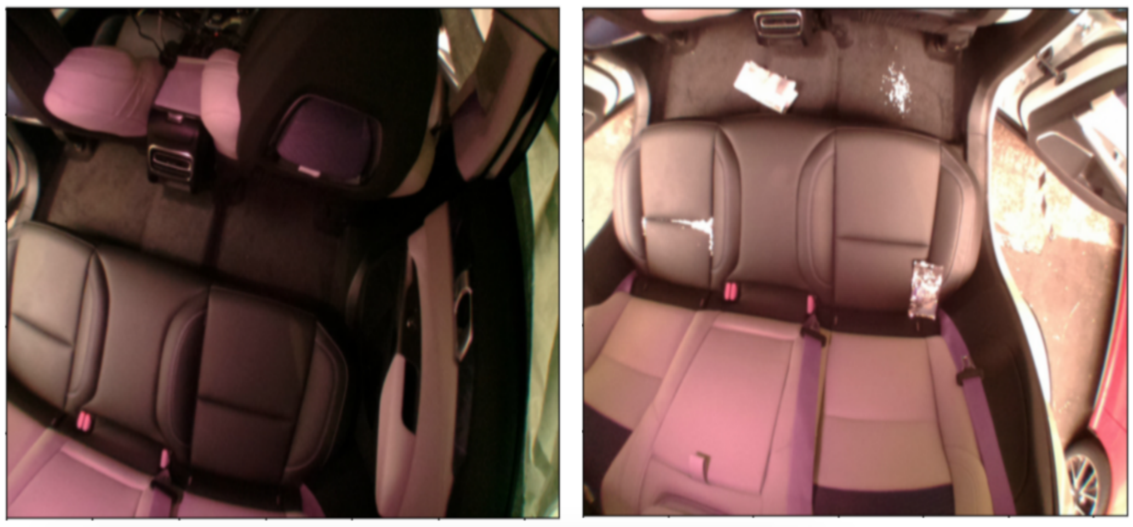


Figure 5.5: (a) Image 1 and (b) Image 2

Figure 5.6, shows the inference results of the four best models. The following images are the prediction results after passing the two images through the four models.

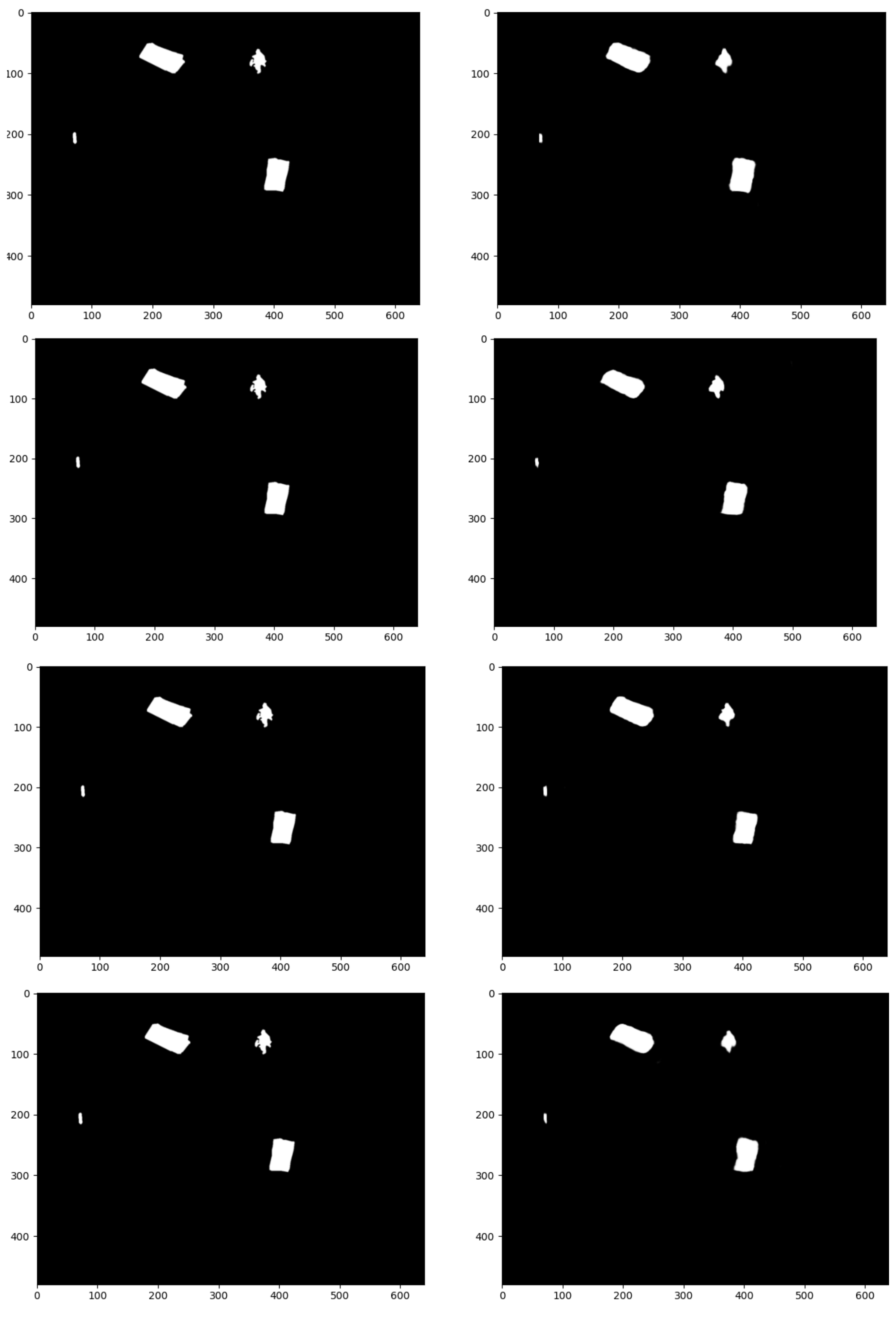


Figure 5.6: From(a)to(d) Left pane: Ground-truth labels; Right pane: Predicted results, (a)Fuse, (b)Diff, (c)Fuse+GAP, (d)Diff+GAP

The models and their respective scores for the given image pair are illustrated in Figure 5.7.

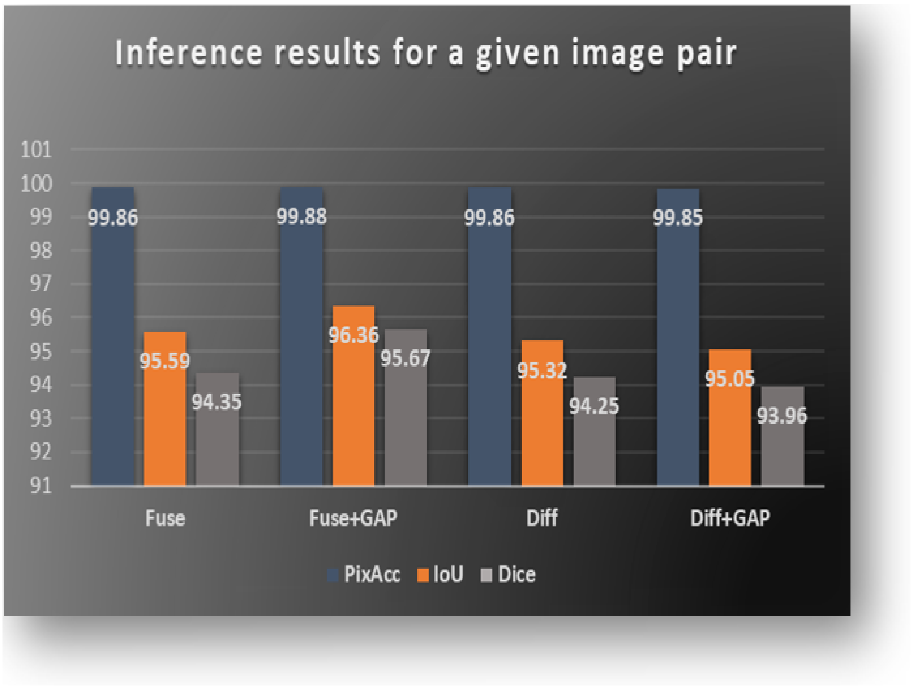


Figure 5.7: ResNet50-Siamese: Inference results for the given image pair

All the models yield good results. Even though the fuse technique with global average pooling layer scored better for the given input image pair, we have seen in Figure 5.4 that the difference technique with global average pooling layer scored better on the whole test dataset.

5.3 Spatial Pyramid Pooling

The model is evaluated using pixel accuracy, mean intersection over union and accuracy.

Figure 5.8 shows the inference results obtained by training the models on the two thousand images custom dataset. The inference is run on 205 test dataset.

Figure 5.9 shows the inference results obtained by training the models on the sixteen thousand images custom dataset. The inference is run on 2056 test dataset. It leaves a memory footprint of 10 GB.

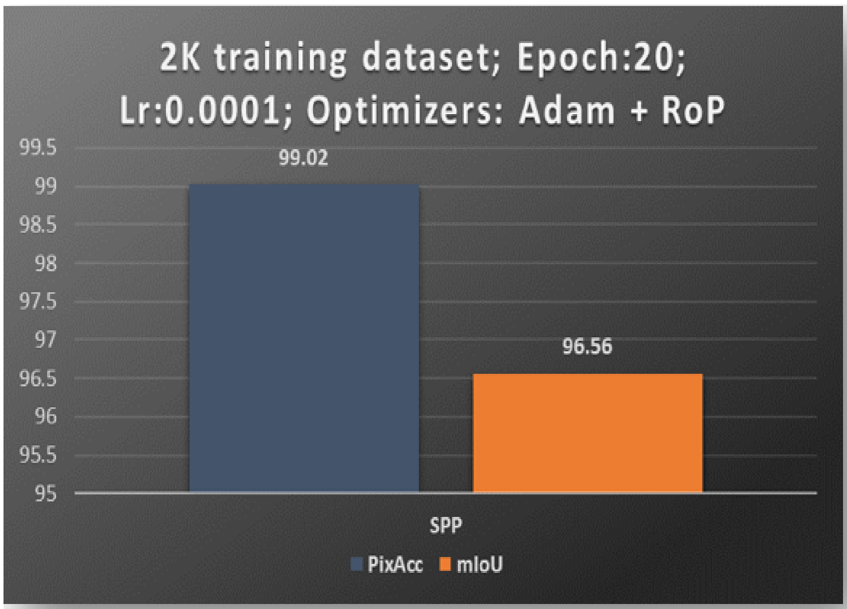


Figure 5.8: SPP: Inference result on two thousand images dataset

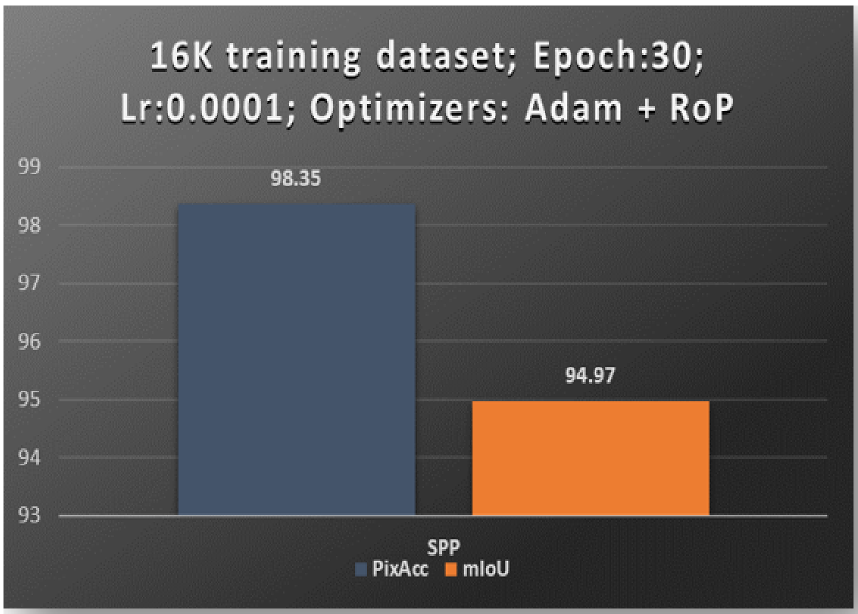


Figure 5.9: SPP: Inference result on sixteen thousand images dataset

In Figure 5.10, it can be observed how the accuracy almost reaches 100 in the initial training itself. The accuracy metric shows how accurately the model predicts change.

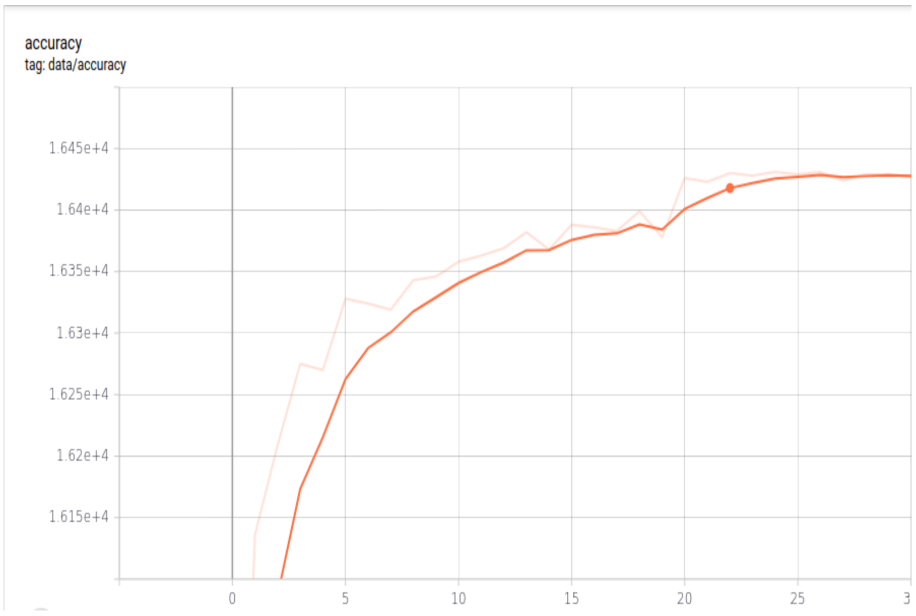


Figure 5.10: SPP: Prediction accuracy graph (sixteen thousand images dataset)

Losses like the reconstruction loss, auto encoder loss as well as the BCE and KL loss are worth seeing as the training progresses and can be found in Appendix 2

5.4 Classical vs. Machine Learning

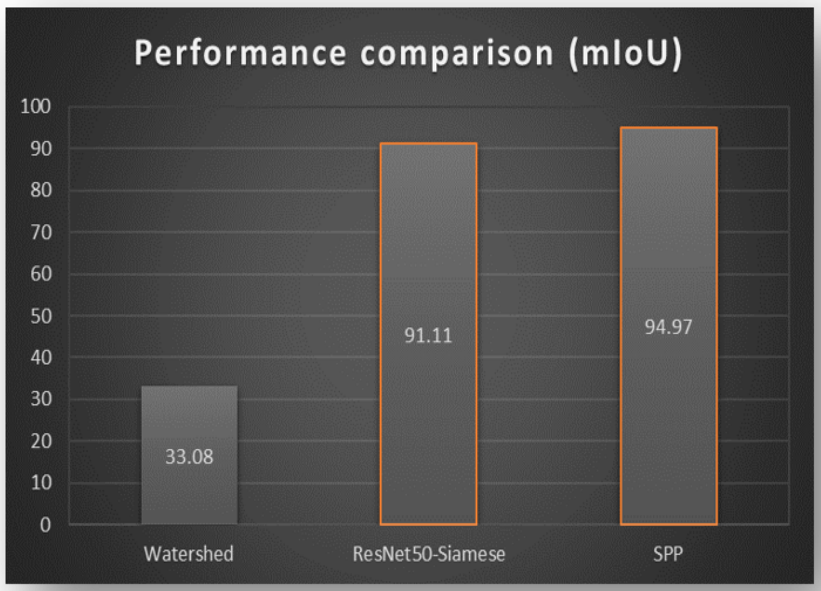


Figure 5.11: Comparison of classical and machine learning models

We compare the performance between the classical and machine learning models based on the results shown in the previous sections. The performance is evaluated using mIoU scores and on the test dataset from the sixteen thousand sample. As seen in Figure 5.11, it is evident that the machine learning models vastly outperform the classical one. Among the model based on ResNet50 architecture, the difference technique yields better mIoU score.

6

Discussion and Conclusion

This chapter discusses the selection of the various techniques used in the study. Future experiments are also briefly discussed. A summary of the study is also presented in Section 6.2.

6.1 Discussion

6.1.1 Watershed Algorithm

The classical algorithm, such as the Watershed algorithm, can be used for image segmentation, but the images must be preprocessed efficiently. The algorithm can work accurately for one image. A diverse image set containing objects of varying size, shape and illumination, make it difficult for the watershed algorithm to segment different images with the same accuracy. We can observe that the algorithm is sometimes prone to over-segmentation problem depending on the image. The problem of over-segmentation is fairly well controlled by using markers. The experiment shows that the classical algorithm is not so efficient for image segmentation for a large diversified image set. A fair segmentation is only possible if the object of interest can be accurately extracted from the image. Therefore, it is necessary to extract the objects more accurately and adequately before applying the classical algorithm.

6.1.2 Machine Learning Algorithm

6.1.2.1 Choice of Residual Neural Network

The deep residual network ensures that all layers generate optimal feature maps by making the identity mapping optimal. The idea is to train the network to learn the residual function, such that it approaches to a zero value. Identity skip connection (shortcut connection) mitigates the vanishing gradient by passing gradients to the initial layers. It also mitigates the problem of accuracy degradation. ResNet50 incorporates batch normalization to mitigate the problem of internal covariate shift and improve the stability of the network. The bottleneck residual block [8] is integrated into the ResNet architecture to increase the performance of the deeper layers and reduce the computational cost. Therefore, ResNet50 is used as a feature extractor since it is a deeper architecture and computationally feasible. ResNet50 has proven to be an ideal architecture to generate extensive feature maps from an image compared to other ResNet layers. Further, a more deep layered convolutional network can be experimented with in future.

It can be observed that the inclusion of all four stages of ResNet50 outperforms the model that only used three stages of ResNet50. This may be because the inclusion of the lowest stage of ResNet50 network helps in detecting the low-level features. The addition of the global average pooling layer has significantly increased the score for both the fusing technique ($Siamese\_Res50\_Fuse\_Net$) and the difference technique ($Siamese\_Res50\_Diff\_Net$). This is due to the fact that the global average pooling increases the context information and also helps in mitigating overfitting problems by reducing the total number of parameters in the network. However, applying global average pooling to more layers did not improve the accuracy score. This may be due to the fact that pooling operation is already applied in the pre-trained convolutional network using strides.

6.1.2.2 Choice of Transfer Learning

Training the entire network is not a viable option because it requires more time to train and adds computational overhead. Since ResNet has a deeper network architecture, the dataset to be trained must be large enough to avoid overfitting. Therefore, feature extraction using pre-trained ResNet50 is an ideal choice and has shown to be one of the ideal architectures for scene change detection.

6.1.2.3 Choice of Image Spatial Dimension

The resolution of the image can improve the accuracy of the model. Since loading the original image with a resolution of 960x1280 hampers the data loading process and memory requirements, it was not practical to use this resolution. Moreover, it is important to preserve the aspect ratio. Therefore, the spatial dimension of the original image is reduced by half, which in turn reduced the memory requirement by about four times.

6.1.2.4 Choice of Loss function

The ground-truth labels are binary images, where the background pixels outnumber the foreground pixels. This often leads to class imbalance. So, an appropriate choice of loss function is required so that correct weights can be updated to reduce the loss in the next validation. Therefore, a combination of BCE loss and dice loss was chosen as the loss function. This combination of loss functions has proved to be an ideal choice in the binary image segmentation task. With larger numbers, BCE loss calculation results in an arithmetic overflow and is numerically less stable. Therefore, a combination of BCE loss and a sigmoid layer is usually chosen to mitigate the overflow problem. We can experiment with training the network using this combination instead of using BCE loss alone in the future.

6.1.2.5 Choice of Optimizer and Learning Rate Scheduler

The pre-trained ResNet50 model trained on the Imagenet dataset uses Stochastic Gradient Descent (SGD) as the optimization algorithm. Adam is the improved version of SGD. Therefore, Adam was chosen. Based on training the network on the smaller dataset, we observe that 0.001 is an ideal initial learning rate for ResNet50.

At a higher learning rate, the parameters vary disorderly and the network cannot settle at local minima. And at a lower learning rate, the network might settle into false minima. The reason for choosing 0.001 as the learning rate is because it is the default value for training ResNet50 using the Imagenet dataset. This is an ideal value that is neither too low nor too high.

The learning rate is updated with a patience of 5 for training the entire dataset, based on monitoring the validation loss. The rate at which the learning rate decays is ideal. Therefore, we reduce the computational wastage, aids the training process, and validate to reach the best position. The exponential decay of the first and second moments is controlled by the hyperparameters, β coefficients. The moment starts at β_1=0.9 and lets it decay over the epochs to β_2=0.999. It is the default value.

Due to time constraint, it was not possible to train the network by setting different hyperparameters. Meanwhile, an attempt to choose the best initial values based on various tests in this study is made. We can further train the network by tuning the hyperparameters of Adam in the future such as varying the β coefficients and learning rate. Adam with weight-decoupled decay is another optimizer to experiment in the future.

Moreover, the best model can be trained for more epochs. The trend of loss functions indicates that the model can score better in predicting the semantic change maps in case of ResNet.

6.2 Conclusion

We have evaluated both classical and machine learning algorithms in scene change detection. The machine learning algorithm requires more computational resources compared to the classical algorithm, depending on the architecture of the training model. However, its ability to infer semantic changes from unknown datasets makes it invaluable. We investigated the feature extraction capability of ResNet50 and built a Siamese network architecture for semantic scene change detection. We also investigated the object localization capability of convolutional neural network using a global average pooling layer. The difference technique yields the best mIoU score among the ResNet50 based models. The addition of global average pooling layer has significantly increased the score for all the three metrics. Finally, we also investigated the capabilities of SPP and has shown good prediction accuracy. Based on our research and the results we inferred from the training, we believe that the machine learning models can yield more score if trained for more number of epochs, i.e., until the network converge. Thus, it can be concluded that the machine learning algorithm significantly outperforms the classical algorithm. In a nutshell, this study suggests that the machine learning models are more suitable for the development of scene change detection functions in taxis and passenger vehicles.

We think there is a scope for future research by evaluating the architecture for scene change detection using a deeper network, for example, ResNet with more than 50 layers. We can also implement multi-class object detection where the model can predict the detected objects. Finally, a notification system can also be built to notify the passenger of any misplaced belongings.

Bibliography

[1] K. Sakurada, T. Okatani, and K. Deguchi, “Detecting changes in 3d structure of a scene from multi-view images captured by a vehicle-mounted camera,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 137–144.

[2] R. C. Daudt, B. Le Saux, and A. Boulch, “Fully convolutional siamese networks for change detection,” in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 4063–4067.

[3] Z. Kourtzi and N. Kanwisher, “Cortical regions involved in perceiving object shape,” *Journal of Neuroscience*, vol. 20, no. 9, pp. 3310–3318, 2000.

[4] K. Das, B. Giesbrecht, and M. P. Eckstein, “Predicting variations of perceptual performance across individuals from neural activity using pattern classifiers,” *Neuroimage*, vol. 51, no. 4, pp. 1425–1437, 2010.

[5] C. Spampinato, S. Palazzo, I. Kavasidis, D. Giordano, N. Souly, and M. Shah, “Deep learning human mind for automated visual classification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6809–6817.

[6] J. Laserson, “From neural networks to deep learning: zeroing in on the human brain,” *XRDS: Crossroads, The ACM Magazine for Students*, vol. 18, no. 1, pp. 29–34, 2011.

[7] J. Wu, “Introduction to convolutional neural networks,” *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, p. 23, 2017.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classifica-

tion with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[10] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-vggnet models for scene recognition," *arXiv preprint arXiv:1508.01667*, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] A. Bleau and L. J. Leon, "Watershed-based segmentation and region merging," *Computer Vision and Image Understanding*, vol. 77, no. 3, pp. 317–370, 2000.

[13] S. Beucher *et al.*, "The watershed transformation applied to image segmentation," *Scanning microscopy-supplement-*, pp. 299–299, 1992.

[14] A. Bieniek and A. Moga, "An efficient watershed algorithm based on connected components," *Pattern recognition*, vol. 33, no. 6, pp. 907–916, 2000.

[15] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski, "Medical image segmentation using k-means clustering and improved watershed algorithm," in *2006 IEEE southwest symposium on image analysis and interpretation.* IEEE, 2006, pp. 61–65.

[16] J. A. Anderson, *An introduction to neural networks.* MIT press, 1995.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[18] G. B. Arfken and H. J. Weber, "Mathematical methods for physicists," 1999.

[19] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET).* Ieee, 2017, pp. 1–6.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision.* Springer, 2016, pp. 630–645.

[21] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[22] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1. Citeseer, 2013, p. 3.

[23] Y. Pathak, K. Arya, and S. Tiwari, “Feature selection for image steganalysis using levy flight-based grey wolf optimization,” *Multimedia Tools and Applications*, vol. 78, no. 2, pp. 1473–1494, 2019.

[24] J. M. Duarte, J. B. d. Santos, and L. C. Melo, “Comparison of similarity coefficients based on rapd markers in the common bean,” *Genetics and Molecular Biology*, vol. 22, no. 3, pp. 427–432, 1999.

[25] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations,” in *Deep learning in medical image analysis and multimodal learning for clinical decision support.* Springer, 2017, pp. 240–248.

[26] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[27] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[28] F. Isensee, P. Kickingereder, W. Wick, M. Bendszus, and K. H. Maier-Hein, “Brain tumor segmentation and radiomics survival prediction: Contribution to the brats 2017 challenge,” in *International MICCAI Brainlesion Workshop.* Springer, 2017, pp. 287–297.

[29] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, “Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning,” *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[30] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” in *Advances in neural information processing systems*, 2018, pp. 2483–2493.

[31] K. Sakurada and T. Okatani, “Change detection from a street image pair using cnn features and superpixel segmentation.” in *BMVC*, 2015, pp. 61–1.

[32] X. Zhao, H. Li, R. Wang, C. Zheng, and S. Shi, “Street-view change detection via siamese encoder-decoder structured convolutional neural networks,” *VISIGRAPP*, vol. 2, p. 2, 2019.

[33] A. Varghese, J. Gubbi, A. Ramaswamy, and P. Balamuralidhar, “Changenet:

A deep learning architecture for visual change detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0.

[34] K. L. de Jong and A. S. Bosman, "Unsupervised change detection in satellite images using convolutional neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[35] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[36] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[37] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[38] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5353–5360.

[39] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[40] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *arXiv preprint arXiv:1506.04579*, 2015.

[41] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[42] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[43] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[44] P. Baldi, "Gradient descent learning algorithm overview: A general dynamical systems perspective," *IEEE Transactions on neural networks*, vol. 6, no. 1, pp. 182–195, 1995.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal repre-

sentations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[46] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[47] ——, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[48] ——, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[49] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[52] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1836–1841.

[53] F. Nie, H. Zhanxuan, and X. Li, "An investigation for loss functions widely used in machine learning," *Communications in Information and Systems*, vol. 18, pp. 37–52, 01 2018.

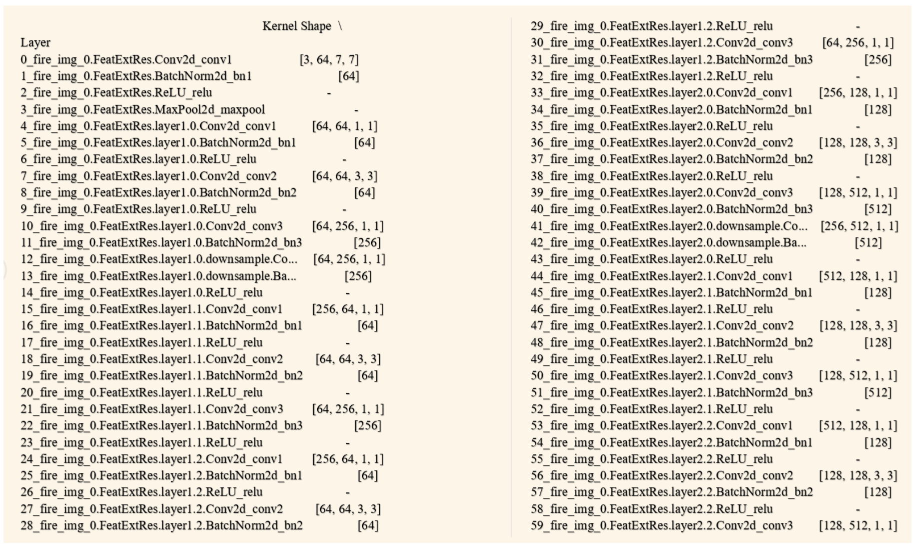
[54] J. Yousefi, "Image binarization using otsu thresholding algorithm," *University of Guelph, Ontario, Canada*, 2011.

A

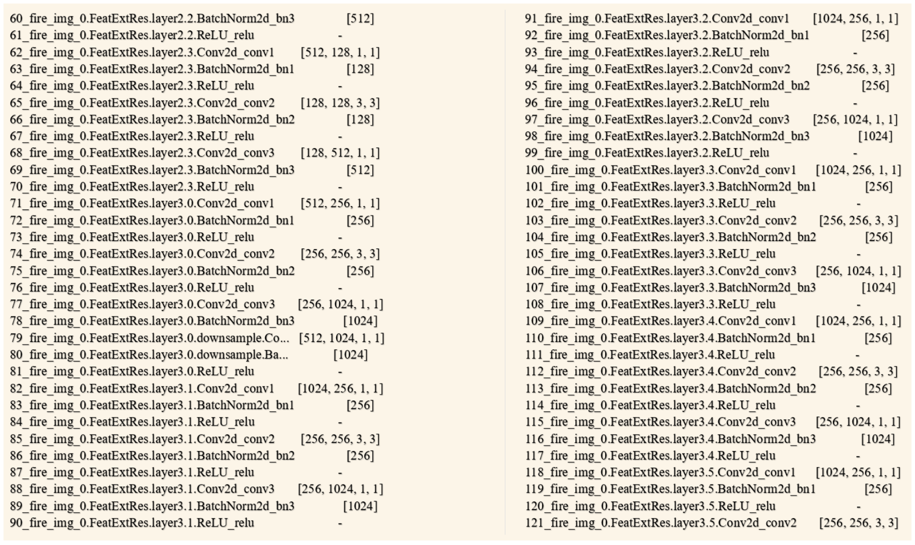
Appendix 1

A.1 Kernel and Output Shape

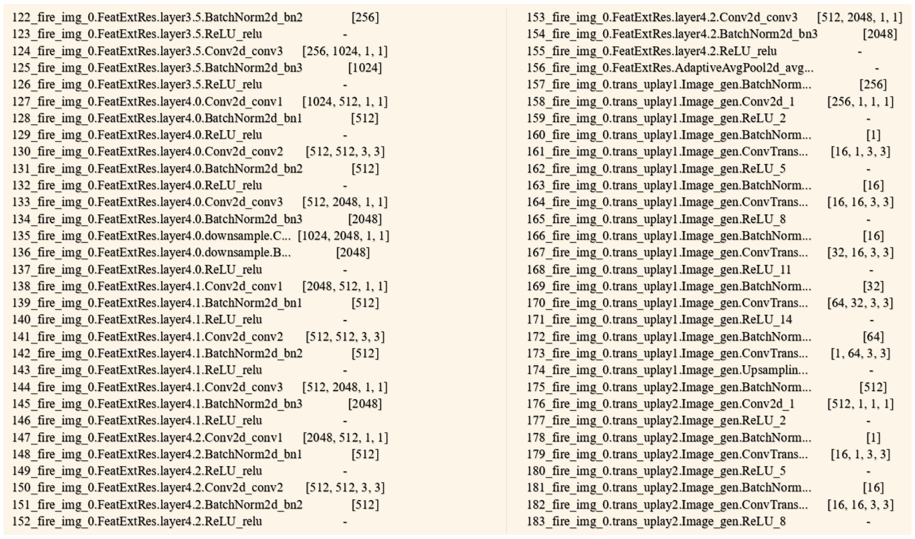
Figure A.1 below, depicts the kernel shape and output shape across each layers of the ResNet50-Siamese architecture.



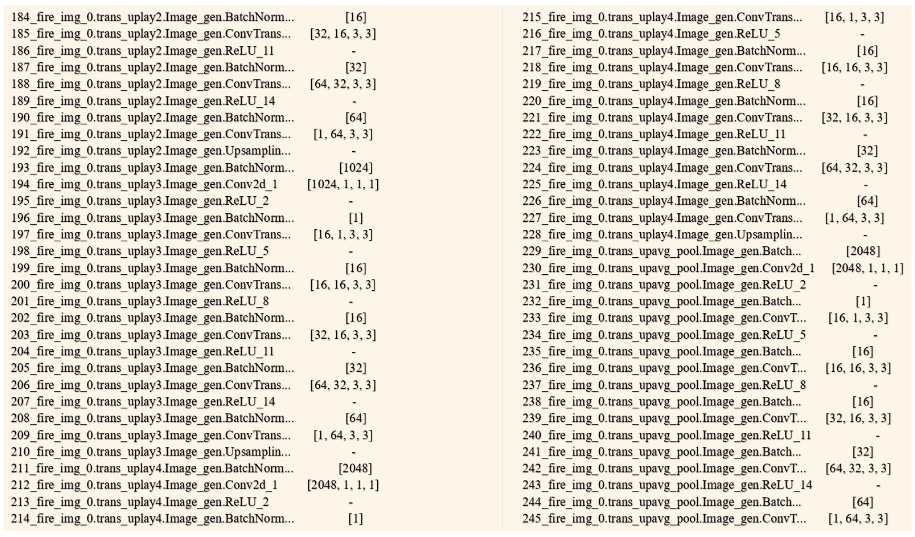
```
                                                  Kernel Shape  \
Layer
0_fire_img_0.FeatExtRes.Conv2d_conv1                      [3, 64, 7, 7]
1_fire_img_0.FeatExtRes.BatchNorm2d_bn1                            [64]
2_fire_img_0.FeatExtRes.ReLU_relu                                     -
3_fire_img_0.FeatExtRes.MaxPool2d_maxpool                             -
4_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv1            [64, 64, 1, 1]
5_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn1                   [64]
6_fire_img_0.FeatExtRes.layer1.0.ReLU_relu                            -
7_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv2            [64, 64, 3, 3]
8_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn2                   [64]
9_fire_img_0.FeatExtRes.layer1.0.ReLU_relu                            -
10_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv3          [64, 256, 1, 1]
11_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn3                 [256]
12_fire_img_0.FeatExtRes.layer1.0.downsample.Co...      [64, 256, 1, 1]
13_fire_img_0.FeatExtRes.layer1.0.downsample.Ba...                [256]
14_fire_img_0.FeatExtRes.layer1.0.ReLU_relu                           -
15_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv1          [256, 64, 1, 1]
16_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn1                  [64]
17_fire_img_0.FeatExtRes.layer1.1.ReLU_relu                           -
18_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv2           [64, 64, 3, 3]
19_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn2                  [64]
20_fire_img_0.FeatExtRes.layer1.1.ReLU_relu                           -
21_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv3          [64, 256, 1, 1]
22_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn3                 [256]
23_fire_img_0.FeatExtRes.layer1.1.ReLU_relu                           -
24_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv1          [256, 64, 1, 1]
25_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn1                  [64]
26_fire_img_0.FeatExtRes.layer1.2.ReLU_relu                           -
27_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv2           [64, 64, 3, 3]
28_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn2                  [64]
29_fire_img_0.FeatExtRes.layer1.2.ReLU_relu                           -
30_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv3          [64, 256, 1, 1]
31_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn3                 [256]
32_fire_img_0.FeatExtRes.layer1.2.ReLU_relu                           -
33_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv1         [256, 128, 1, 1]
34_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn1                 [128]
35_fire_img_0.FeatExtRes.layer2.0.ReLU_relu                           -
36_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv2         [128, 128, 3, 3]
37_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn2                 [128]
38_fire_img_0.FeatExtRes.layer2.0.ReLU_relu                           -
39_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv3         [128, 512, 1, 1]
40_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn3                 [512]
41_fire_img_0.FeatExtRes.layer2.0.downsample.Co...     [256, 512, 1, 1]
42_fire_img_0.FeatExtRes.layer2.0.downsample.Ba...                [512]
43_fire_img_0.FeatExtRes.layer2.0.ReLU_relu                           -
44_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv1         [512, 128, 1, 1]
45_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn1                 [128]
46_fire_img_0.FeatExtRes.layer2.1.ReLU_relu                           -
47_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv2         [128, 128, 3, 3]
48_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn2                 [128]
49_fire_img_0.FeatExtRes.layer2.1.ReLU_relu                           -
50_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv3         [128, 512, 1, 1]
51_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn3                 [512]
52_fire_img_0.FeatExtRes.layer2.1.ReLU_relu                           -
53_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv1         [512, 128, 1, 1]
54_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn1                 [128]
55_fire_img_0.FeatExtRes.layer2.2.ReLU_relu                           -
56_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv2         [128, 128, 3, 3]
57_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn2                 [128]
58_fire_img_0.FeatExtRes.layer2.2.ReLU_relu                           -
59_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv3         [128, 512, 1, 1]
```



```
60_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn3              [512]
61_fire_img_0.FeatExtRes.layer2.2.ReLU_relu                      -
62_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv1         [512, 128, 1, 1]
63_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn1              [128]
64_fire_img_0.FeatExtRes.layer2.3.ReLU_relu                      -
65_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv2         [128, 128, 3, 3]
66_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn2              [128]
67_fire_img_0.FeatExtRes.layer2.3.ReLU_relu                      -
68_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv3         [128, 512, 1, 1]
69_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn3              [512]
70_fire_img_0.FeatExtRes.layer2.3.ReLU_relu                      -
71_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv1         [512, 256, 1, 1]
72_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn1              [256]
73_fire_img_0.FeatExtRes.layer3.0.ReLU_relu                      -
74_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv2         [256, 256, 3, 3]
75_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn2              [256]
76_fire_img_0.FeatExtRes.layer3.0.ReLU_relu                      -
77_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv3        [256, 1024, 1, 1]
78_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn3              [1024]
79_fire_img_0.FeatExtRes.layer3.0.downsample.Co...    [512, 1024, 1, 1]
80_fire_img_0.FeatExtRes.layer3.0.downsample.Ba...           [1024]
81_fire_img_0.FeatExtRes.layer3.0.ReLU_relu                      -
82_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv1        [1024, 256, 1, 1]
83_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn1              [256]
84_fire_img_0.FeatExtRes.layer3.1.ReLU_relu                      -
85_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv2         [256, 256, 3, 3]
86_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn2              [256]
87_fire_img_0.FeatExtRes.layer3.1.ReLU_relu                      -
88_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv3        [256, 1024, 1, 1]
89_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn3              [1024]
90_fire_img_0.FeatExtRes.layer3.1.ReLU_relu                      -
91_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv1        [1024, 256, 1, 1]
92_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn1              [256]
93_fire_img_0.FeatExtRes.layer3.2.ReLU_relu                      -
94_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv2         [256, 256, 3, 3]
95_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn2              [256]
96_fire_img_0.FeatExtRes.layer3.2.ReLU_relu                      -
97_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv3        [256, 1024, 1, 1]
98_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn3              [1024]
99_fire_img_0.FeatExtRes.layer3.2.ReLU_relu                      -
100_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv1       [1024, 256, 1, 1]
101_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn1             [256]
102_fire_img_0.FeatExtRes.layer3.3.ReLU_relu                     -
103_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv2        [256, 256, 3, 3]
104_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn2             [256]
105_fire_img_0.FeatExtRes.layer3.3.ReLU_relu                     -
106_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv3       [256, 1024, 1, 1]
107_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn3             [1024]
108_fire_img_0.FeatExtRes.layer3.3.ReLU_relu                     -
109_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv1       [1024, 256, 1, 1]
110_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn1             [256]
111_fire_img_0.FeatExtRes.layer3.4.ReLU_relu                     -
112_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv2        [256, 256, 3, 3]
113_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn2             [256]
114_fire_img_0.FeatExtRes.layer3.4.ReLU_relu                     -
115_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv3       [256, 1024, 1, 1]
116_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn3             [1024]
117_fire_img_0.FeatExtRes.layer3.4.ReLU_relu                     -
118_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv1       [1024, 256, 1, 1]
119_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn1             [256]
120_fire_img_0.FeatExtRes.layer3.5.ReLU_relu                     -
121_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv2        [256, 256, 3, 3]
```



```
122_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn2             [256]
123_fire_img_0.FeatExtRes.layer3.5.ReLU_relu                     -
124_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv3       [256, 1024, 1, 1]
125_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn3             [1024]
126_fire_img_0.FeatExtRes.layer3.5.ReLU_relu                     -
127_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv1       [1024, 512, 1, 1]
128_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn1             [512]
129_fire_img_0.FeatExtRes.layer4.0.ReLU_relu                     -
130_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv2        [512, 512, 3, 3]
131_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn2             [512]
132_fire_img_0.FeatExtRes.layer4.0.ReLU_relu                     -
133_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv3       [512, 2048, 1, 1]
134_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn3             [2048]
135_fire_img_0.FeatExtRes.layer4.0.downsample.C...  [1024, 2048, 1, 1]
136_fire_img_0.FeatExtRes.layer4.0.downsample.B...          [2048]
137_fire_img_0.FeatExtRes.layer4.0.ReLU_relu                     -
138_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv1       [2048, 512, 1, 1]
139_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn1             [512]
140_fire_img_0.FeatExtRes.layer4.1.ReLU_relu                     -
141_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv2        [512, 512, 3, 3]
142_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn2             [512]
143_fire_img_0.FeatExtRes.layer4.1.ReLU_relu                     -
144_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv3       [512, 2048, 1, 1]
145_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn3             [2048]
146_fire_img_0.FeatExtRes.layer4.1.ReLU_relu                     -
147_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv1       [2048, 512, 1, 1]
148_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn1             [512]
149_fire_img_0.FeatExtRes.layer4.2.ReLU_relu                     -
150_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv2        [512, 512, 3, 3]
151_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn2             [512]
152_fire_img_0.FeatExtRes.layer4.2.ReLU_relu                     -
153_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv3       [512, 2048, 1, 1]
154_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn3             [2048]
155_fire_img_0.FeatExtRes.layer4.2.ReLU_relu                     -
156_fire_img_0.FeatExtRes.AdaptiveAvgPool2d_avg...               -
157_fire_img_0.trans_uplay1.Image_gen.BatchNorm...             [256]
158_fire_img_0.trans_uplay1.Image_gen.Conv2d_1          [256, 1, 1, 1]
159_fire_img_0.trans_uplay1.Image_gen.ReLU_2                     -
160_fire_img_0.trans_uplay1.Image_gen.BatchNorm...              [1]
161_fire_img_0.trans_uplay1.Image_gen.ConvTrans...       [16, 1, 3, 3]
162_fire_img_0.trans_uplay1.Image_gen.ReLU_5                     -
163_fire_img_0.trans_uplay1.Image_gen.BatchNorm...             [16]
164_fire_img_0.trans_uplay1.Image_gen.ConvTrans...      [16, 16, 3, 3]
165_fire_img_0.trans_uplay1.Image_gen.ReLU_8                     -
166_fire_img_0.trans_uplay1.Image_gen.BatchNorm...             [16]
167_fire_img_0.trans_uplay1.Image_gen.ConvTrans...      [32, 16, 3, 3]
168_fire_img_0.trans_uplay1.Image_gen.ReLU_11                    -
169_fire_img_0.trans_uplay1.Image_gen.BatchNorm...             [32]
170_fire_img_0.trans_uplay1.Image_gen.ConvTrans...      [64, 32, 3, 3]
171_fire_img_0.trans_uplay1.Image_gen.ReLU_14                    -
172_fire_img_0.trans_uplay1.Image_gen.BatchNorm...             [64]
173_fire_img_0.trans_uplay1.Image_gen.ConvTrans...       [1, 64, 3, 3]
174_fire_img_0.trans_uplay1.Image_gen.Upsamplin...               -
175_fire_img_0.trans_uplay2.Image_gen.BatchNorm...             [512]
176_fire_img_0.trans_uplay2.Image_gen.Conv2d_1          [512, 1, 1, 1]
177_fire_img_0.trans_uplay2.Image_gen.ReLU_2                     -
178_fire_img_0.trans_uplay2.Image_gen.BatchNorm...              [1]
179_fire_img_0.trans_uplay2.Image_gen.ConvTrans...       [16, 1, 3, 3]
180_fire_img_0.trans_uplay2.Image_gen.ReLU_5                     -
181_fire_img_0.trans_uplay2.Image_gen.BatchNorm...             [16]
182_fire_img_0.trans_uplay2.Image_gen.ConvTrans...      [16, 16, 3, 3]
183_fire_img_0.trans_uplay2.Image_gen.ReLU_8                     -
```



```
184_fire_img_0.trans_uplay2.Image_gen.BatchNorm...             [16]
185_fire_img_0.trans_uplay2.Image_gen.ConvTrans...      [32, 16, 3, 3]
186_fire_img_0.trans_uplay2.Image_gen.ReLU_11                    -
187_fire_img_0.trans_uplay2.Image_gen.BatchNorm...             [32]
188_fire_img_0.trans_uplay2.Image_gen.ConvTrans...      [64, 32, 3, 3]
189_fire_img_0.trans_uplay2.Image_gen.ReLU_14                    -
190_fire_img_0.trans_uplay2.Image_gen.BatchNorm...             [64]
191_fire_img_0.trans_uplay2.Image_gen.ConvTrans...       [1, 64, 3, 3]
192_fire_img_0.trans_uplay2.Image_gen.Upsamplin...               -
193_fire_img_0.trans_uplay3.Image_gen.BatchNorm...            [1024]
194_fire_img_0.trans_uplay3.Image_gen.Conv2d_1         [1024, 1, 1, 1]
195_fire_img_0.trans_uplay3.Image_gen.ReLU_2                     -
196_fire_img_0.trans_uplay3.Image_gen.BatchNorm...              [1]
197_fire_img_0.trans_uplay3.Image_gen.ConvTrans...       [16, 1, 3, 3]
198_fire_img_0.trans_uplay3.Image_gen.ReLU_5                     -
199_fire_img_0.trans_uplay3.Image_gen.BatchNorm...             [16]
200_fire_img_0.trans_uplay3.Image_gen.ConvTrans...      [16, 16, 3, 3]
201_fire_img_0.trans_uplay3.Image_gen.ReLU_8                     -
202_fire_img_0.trans_uplay3.Image_gen.BatchNorm...             [16]
203_fire_img_0.trans_uplay3.Image_gen.ConvTrans...      [32, 16, 3, 3]
204_fire_img_0.trans_uplay3.Image_gen.ReLU_11                    -
205_fire_img_0.trans_uplay3.Image_gen.BatchNorm...             [32]
206_fire_img_0.trans_uplay3.Image_gen.ConvTrans...      [64, 32, 3, 3]
207_fire_img_0.trans_uplay3.Image_gen.ReLU_14                    -
208_fire_img_0.trans_uplay3.Image_gen.BatchNorm...             [64]
209_fire_img_0.trans_uplay3.Image_gen.ConvTrans...       [1, 64, 3, 3]
210_fire_img_0.trans_uplay3.Image_gen.Upsamplin...               -
211_fire_img_0.trans_uplay4.Image_gen.BatchNorm...            [2048]
212_fire_img_0.trans_uplay4.Image_gen.Conv2d_1         [2048, 1, 1, 1]
213_fire_img_0.trans_uplay4.Image_gen.ReLU_2                     -
214_fire_img_0.trans_uplay4.Image_gen.BatchNorm...              [1]
215_fire_img_0.trans_uplay4.Image_gen.ConvTrans...       [16, 1, 3, 3]
216_fire_img_0.trans_uplay4.Image_gen.ReLU_5                     -
217_fire_img_0.trans_uplay4.Image_gen.BatchNorm...             [16]
218_fire_img_0.trans_uplay4.Image_gen.ConvTrans...      [16, 16, 3, 3]
219_fire_img_0.trans_uplay4.Image_gen.ReLU_8                     -
220_fire_img_0.trans_uplay4.Image_gen.BatchNorm...             [16]
221_fire_img_0.trans_uplay4.Image_gen.ConvTrans...      [32, 16, 3, 3]
222_fire_img_0.trans_uplay4.Image_gen.ReLU_11                    -
223_fire_img_0.trans_uplay4.Image_gen.BatchNorm...             [32]
224_fire_img_0.trans_uplay4.Image_gen.ConvTrans...      [64, 32, 3, 3]
225_fire_img_0.trans_uplay4.Image_gen.ReLU_14                    -
226_fire_img_0.trans_uplay4.Image_gen.BatchNorm...             [64]
227_fire_img_0.trans_uplay4.Image_gen.ConvTrans...       [1, 64, 3, 3]
228_fire_img_0.trans_uplay4.Image_gen.Upsamplin...               -
229_fire_img_0.trans_upavg_pool.Image_gen.Batch...            [2048]
230_fire_img_0.trans_upavg_pool.Image_gen.Conv2d_1     [2048, 1, 1, 1]
231_fire_img_0.trans_upavg_pool.Image_gen.ReLU_2                 -
232_fire_img_0.trans_upavg_pool.Image_gen.Batch...              [1]
233_fire_img_0.trans_upavg_pool.Image_gen.ConvT...       [16, 1, 3, 3]
234_fire_img_0.trans_upavg_pool.Image_gen.ReLU_5                 -
235_fire_img_0.trans_upavg_pool.Image_gen.Batch...             [16]
236_fire_img_0.trans_upavg_pool.Image_gen.ConvT...      [16, 16, 3, 3]
237_fire_img_0.trans_upavg_pool.Image_gen.ReLU_8                 -
238_fire_img_0.trans_upavg_pool.Image_gen.Batch...             [16]
239_fire_img_0.trans_upavg_pool.Image_gen.ConvT...      [32, 16, 3, 3]
240_fire_img_0.trans_upavg_pool.Image_gen.ReLU_11                -
241_fire_img_0.trans_upavg_pool.Image_gen.Batch...             [32]
242_fire_img_0.trans_upavg_pool.Image_gen.ConvT...      [64, 32, 3, 3]
243_fire_img_0.trans_upavg_pool.Image_gen.ReLU_14                -
244_fire_img_0.trans_upavg_pool.Image_gen.Batch...             [64]
245_fire_img_0.trans_upavg_pool.Image_gen.ConvT...       [1, 64, 3, 3]
```

```
246_fire_img_0.trans_upavg_pool.Image_gen.Upsam...                 -

                                                       Output Shape  \
Layer
0_fire_img_0.FeatExtRes.Conv2d_conv1                 [4, 64, 240, 320]
1_fire_img_0.FeatExtRes.BatchNorm2d_bn1              [4, 64, 240, 320]
2_fire_img_0.FeatExtRes.ReLU_relu                    [4, 64, 240, 320]
3_fire_img_0.FeatExtRes.MaxPool2d_maxpool            [4, 64, 120, 160]
4_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv1        [4, 64, 120, 160]
5_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn1     [4, 64, 120, 160]
6_fire_img_0.FeatExtRes.layer1.0.ReLU_relu           [4, 64, 120, 160]
7_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv2        [4, 64, 120, 160]
8_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn2     [4, 64, 120, 160]
9_fire_img_0.FeatExtRes.layer1.0.ReLU_relu           [4, 64, 120, 160]
10_fire_img_0.FeatExtRes.layer1.0.Conv2d_conv3       [4, 256, 120, 160]
11_fire_img_0.FeatExtRes.layer1.0.BatchNorm2d_bn3    [4, 256, 120, 160]
12_fire_img_0.FeatExtRes.layer1.0.downsample.Co...   [4, 256, 120, 160]
13_fire_img_0.FeatExtRes.layer1.0.downsample.Ba...   [4, 256, 120, 160]
14_fire_img_0.FeatExtRes.layer1.0.ReLU_relu          [4, 256, 120, 160]
15_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv1       [4, 64, 120, 160]
16_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn1    [4, 64, 120, 160]
17_fire_img_0.FeatExtRes.layer1.1.ReLU_relu          [4, 64, 120, 160]
18_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv2       [4, 64, 120, 160]
19_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn2    [4, 64, 120, 160]
20_fire_img_0.FeatExtRes.layer1.1.ReLU_relu          [4, 64, 120, 160]
21_fire_img_0.FeatExtRes.layer1.1.Conv2d_conv3       [4, 256, 120, 160]
22_fire_img_0.FeatExtRes.layer1.1.BatchNorm2d_bn3    [4, 256, 120, 160]
23_fire_img_0.FeatExtRes.layer1.1.ReLU_relu          [4, 256, 120, 160]
24_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv1       [4, 64, 120, 160]
25_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn1    [4, 64, 120, 160]
26_fire_img_0.FeatExtRes.layer1.2.ReLU_relu          [4, 64, 120, 160]
```

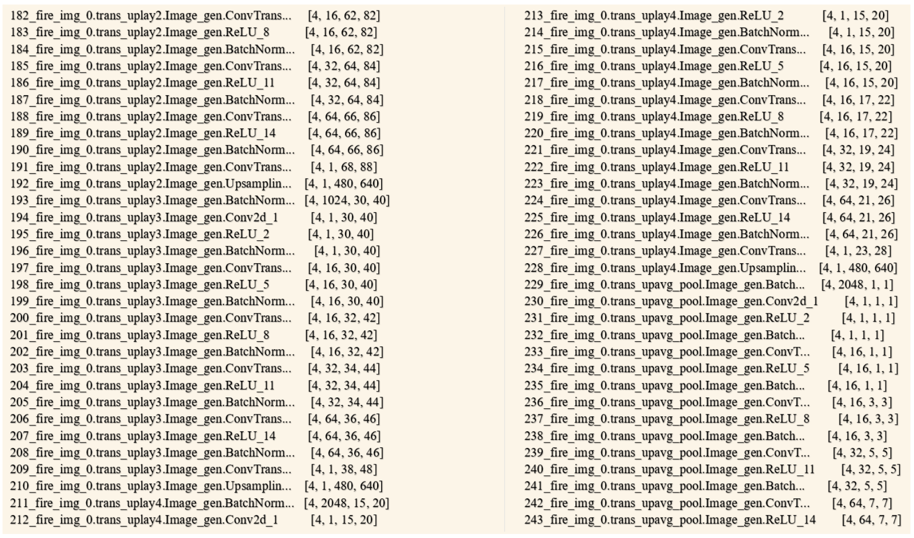
```
27_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv2       [4, 64, 120, 160]
28_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn2    [4, 64, 120, 160]
29_fire_img_0.FeatExtRes.layer1.2.ReLU_relu          [4, 64, 120, 160]
30_fire_img_0.FeatExtRes.layer1.2.Conv2d_conv3       [4, 256, 120, 160]
31_fire_img_0.FeatExtRes.layer1.2.BatchNorm2d_bn3    [4, 256, 120, 160]
32_fire_img_0.FeatExtRes.layer1.2.ReLU_relu          [4, 256, 120, 160]
33_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv1       [4, 128, 120, 160]
34_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn1    [4, 128, 120, 160]
35_fire_img_0.FeatExtRes.layer2.0.ReLU_relu          [4, 128, 120, 160]
36_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv2       [4, 128, 60, 80]
37_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn2    [4, 128, 60, 80]
38_fire_img_0.FeatExtRes.layer2.0.ReLU_relu          [4, 128, 60, 80]
39_fire_img_0.FeatExtRes.layer2.0.Conv2d_conv3       [4, 512, 60, 80]
40_fire_img_0.FeatExtRes.layer2.0.BatchNorm2d_bn3    [4, 512, 60, 80]
41_fire_img_0.FeatExtRes.layer2.0.downsample.Co...   [4, 512, 60, 80]
42_fire_img_0.FeatExtRes.layer2.0.downsample.Ba...   [4, 512, 60, 80]
43_fire_img_0.FeatExtRes.layer2.0.ReLU_relu          [4, 512, 60, 80]
44_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv1       [4, 128, 60, 80]
45_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn1    [4, 128, 60, 80]
46_fire_img_0.FeatExtRes.layer2.1.ReLU_relu          [4, 128, 60, 80]
47_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv2       [4, 128, 60, 80]
48_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn2    [4, 128, 60, 80]
49_fire_img_0.FeatExtRes.layer2.1.ReLU_relu          [4, 128, 60, 80]
50_fire_img_0.FeatExtRes.layer2.1.Conv2d_conv3       [4, 512, 60, 80]
51_fire_img_0.FeatExtRes.layer2.1.BatchNorm2d_bn3    [4, 512, 60, 80]
52_fire_img_0.FeatExtRes.layer2.1.ReLU_relu          [4, 512, 60, 80]
53_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv1       [4, 128, 60, 80]
54_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn1    [4, 128, 60, 80]
55_fire_img_0.FeatExtRes.layer2.2.ReLU_relu          [4, 128, 60, 80]
56_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv2       [4, 128, 60, 80]
57_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn2    [4, 128, 60, 80]
```

```
58_fire_img_0.FeatExtRes.layer2.2.ReLU_relu          [4, 128, 60, 80]
59_fire_img_0.FeatExtRes.layer2.2.Conv2d_conv3       [4, 512, 60, 80]
60_fire_img_0.FeatExtRes.layer2.2.BatchNorm2d_bn3    [4, 512, 60, 80]
61_fire_img_0.FeatExtRes.layer2.2.ReLU_relu          [4, 512, 60, 80]
62_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv1       [4, 128, 60, 80]
63_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn1    [4, 128, 60, 80]
64_fire_img_0.FeatExtRes.layer2.3.ReLU_relu          [4, 128, 60, 80]
65_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv2       [4, 128, 60, 80]
66_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn2    [4, 128, 60, 80]
67_fire_img_0.FeatExtRes.layer2.3.ReLU_relu          [4, 128, 60, 80]
68_fire_img_0.FeatExtRes.layer2.3.Conv2d_conv3       [4, 512, 60, 80]
69_fire_img_0.FeatExtRes.layer2.3.BatchNorm2d_bn3    [4, 512, 60, 80]
70_fire_img_0.FeatExtRes.layer2.3.ReLU_relu          [4, 512, 60, 80]
71_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv1       [4, 256, 60, 80]
72_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn1    [4, 256, 60, 80]
73_fire_img_0.FeatExtRes.layer3.0.ReLU_relu          [4, 256, 60, 80]
74_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv2       [4, 256, 30, 40]
75_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn2    [4, 256, 30, 40]
76_fire_img_0.FeatExtRes.layer3.0.ReLU_relu          [4, 256, 30, 40]
77_fire_img_0.FeatExtRes.layer3.0.Conv2d_conv3       [4, 1024, 30, 40]
78_fire_img_0.FeatExtRes.layer3.0.BatchNorm2d_bn3    [4, 1024, 30, 40]
79_fire_img_0.FeatExtRes.layer3.0.downsample.Co...   [4, 1024, 30, 40]
80_fire_img_0.FeatExtRes.layer3.0.downsample.Ba...   [4, 1024, 30, 40]
81_fire_img_0.FeatExtRes.layer3.0.ReLU_relu          [4, 1024, 30, 40]
82_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv1       [4, 256, 30, 40]
83_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn1    [4, 256, 30, 40]
84_fire_img_0.FeatExtRes.layer3.1.ReLU_relu          [4, 256, 30, 40]
85_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv2       [4, 256, 30, 40]
86_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn2    [4, 256, 30, 40]
87_fire_img_0.FeatExtRes.layer3.1.ReLU_relu          [4, 256, 30, 40]
88_fire_img_0.FeatExtRes.layer3.1.Conv2d_conv3       [4, 1024, 30, 40]
```

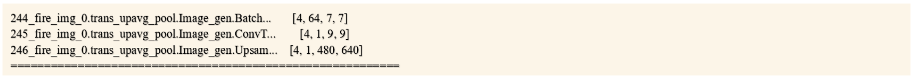
```
89_fire_img_0.FeatExtRes.layer3.1.BatchNorm2d_bn3    [4, 1024, 30, 40]
90_fire_img_0.FeatExtRes.layer3.1.ReLU_relu          [4, 1024, 30, 40]
91_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv1       [4, 256, 30, 40]
92_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn1    [4, 256, 30, 40]
93_fire_img_0.FeatExtRes.layer3.2.ReLU_relu          [4, 256, 30, 40]
94_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv2       [4, 256, 30, 40]
95_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn2    [4, 256, 30, 40]
96_fire_img_0.FeatExtRes.layer3.2.ReLU_relu          [4, 256, 30, 40]
97_fire_img_0.FeatExtRes.layer3.2.Conv2d_conv3       [4, 1024, 30, 40]
98_fire_img_0.FeatExtRes.layer3.2.BatchNorm2d_bn3    [4, 1024, 30, 40]
99_fire_img_0.FeatExtRes.layer3.2.ReLU_relu          [4, 1024, 30, 40]
100_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv1      [4, 256, 30, 40]
101_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn1   [4, 256, 30, 40]
102_fire_img_0.FeatExtRes.layer3.3.ReLU_relu         [4, 256, 30, 40]
103_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv2      [4, 256, 30, 40]
104_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn2   [4, 256, 30, 40]
105_fire_img_0.FeatExtRes.layer3.3.ReLU_relu         [4, 256, 30, 40]
106_fire_img_0.FeatExtRes.layer3.3.Conv2d_conv3      [4, 1024, 30, 40]
107_fire_img_0.FeatExtRes.layer3.3.BatchNorm2d_bn3   [4, 1024, 30, 40]
108_fire_img_0.FeatExtRes.layer3.3.ReLU_relu         [4, 1024, 30, 40]
109_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv1      [4, 256, 30, 40]
110_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn1   [4, 256, 30, 40]
111_fire_img_0.FeatExtRes.layer3.4.ReLU_relu         [4, 256, 30, 40]
112_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv2      [4, 256, 30, 40]
113_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn2   [4, 256, 30, 40]
114_fire_img_0.FeatExtRes.layer3.4.ReLU_relu         [4, 256, 30, 40]
115_fire_img_0.FeatExtRes.layer3.4.Conv2d_conv3      [4, 1024, 30, 40]
116_fire_img_0.FeatExtRes.layer3.4.BatchNorm2d_bn3   [4, 1024, 30, 40]
117_fire_img_0.FeatExtRes.layer3.4.ReLU_relu         [4, 1024, 30, 40]
118_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv1      [4, 256, 30, 40]
119_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn1   [4, 256, 30, 40]
```

```
120_fire_img_0.FeatExtRes.layer3.5.ReLU_relu         [4, 256, 30, 40]
121_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv2      [4, 256, 30, 40]
122_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn2   [4, 256, 30, 40]
123_fire_img_0.FeatExtRes.layer3.5.ReLU_relu         [4, 256, 30, 40]
124_fire_img_0.FeatExtRes.layer3.5.Conv2d_conv3      [4, 1024, 30, 40]
125_fire_img_0.FeatExtRes.layer3.5.BatchNorm2d_bn3   [4, 1024, 30, 40]
126_fire_img_0.FeatExtRes.layer3.5.ReLU_relu         [4, 1024, 30, 40]
127_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv1      [4, 512, 30, 40]
128_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn1   [4, 512, 30, 40]
129_fire_img_0.FeatExtRes.layer4.0.ReLU_relu         [4, 512, 30, 40]
130_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv2      [4, 512, 15, 20]
131_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn2   [4, 512, 15, 20]
132_fire_img_0.FeatExtRes.layer4.0.ReLU_relu         [4, 512, 15, 20]
133_fire_img_0.FeatExtRes.layer4.0.Conv2d_conv3      [4, 2048, 15, 20]
134_fire_img_0.FeatExtRes.layer4.0.BatchNorm2d_bn3   [4, 2048, 15, 20]
135_fire_img_0.FeatExtRes.layer4.0.downsample.C...   [4, 2048, 15, 20]
136_fire_img_0.FeatExtRes.layer4.0.downsample.B...   [4, 2048, 15, 20]
137_fire_img_0.FeatExtRes.layer4.0.ReLU_relu         [4, 2048, 15, 20]
138_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv1      [4, 512, 15, 20]
139_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn1   [4, 512, 15, 20]
140_fire_img_0.FeatExtRes.layer4.1.ReLU_relu         [4, 512, 15, 20]
141_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv2      [4, 512, 15, 20]
142_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn2   [4, 512, 15, 20]
143_fire_img_0.FeatExtRes.layer4.1.ReLU_relu         [4, 512, 15, 20]
144_fire_img_0.FeatExtRes.layer4.1.Conv2d_conv3      [4, 2048, 15, 20]
145_fire_img_0.FeatExtRes.layer4.1.BatchNorm2d_bn3   [4, 2048, 15, 20]
146_fire_img_0.FeatExtRes.layer4.1.ReLU_relu         [4, 2048, 15, 20]
147_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv1      [4, 512, 15, 20]
148_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn1   [4, 512, 15, 20]
149_fire_img_0.FeatExtRes.layer4.2.ReLU_relu         [4, 512, 15, 20]
150_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv2      [4, 512, 15, 20]
```

```
151_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn2   [4, 512, 15, 20]
152_fire_img_0.FeatExtRes.layer4.2.ReLU_relu         [4, 512, 15, 20]
153_fire_img_0.FeatExtRes.layer4.2.Conv2d_conv3      [4, 2048, 15, 20]
154_fire_img_0.FeatExtRes.layer4.2.BatchNorm2d_bn3   [4, 2048, 15, 20]
155_fire_img_0.FeatExtRes.layer4.2.ReLU_relu         [4, 2048, 15, 20]
156_fire_img_0.FeatExtRes.AdaptiveAvgPool2d_avg...   [4, 2048, 1, 1]
157_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 256, 120, 160]
158_fire_img_0.trans_uplay1.Image_gen.Conv2d_1       [4, 1, 120, 160]
159_fire_img_0.trans_uplay1.Image_gen.ReLU_2         [4, 1, 120, 160]
160_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 1, 120, 160]
161_fire_img_0.trans_uplay1.Image_gen.ConvTrans...   [4, 16, 120, 160]
162_fire_img_0.trans_uplay1.Image_gen.ReLU_5         [4, 16, 120, 160]
163_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 16, 120, 160]
164_fire_img_0.trans_uplay1.Image_gen.ConvTrans...   [4, 16, 122, 162]
165_fire_img_0.trans_uplay1.Image_gen.ReLU_8         [4, 16, 122, 162]
166_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 16, 122, 162]
167_fire_img_0.trans_uplay1.Image_gen.ConvTrans...   [4, 32, 124, 164]
168_fire_img_0.trans_uplay1.Image_gen.ReLU_11        [4, 32, 124, 164]
169_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 32, 124, 164]
170_fire_img_0.trans_uplay1.Image_gen.ConvTrans...   [4, 64, 126, 166]
171_fire_img_0.trans_uplay1.Image_gen.ReLU_14        [4, 64, 126, 166]
172_fire_img_0.trans_uplay1.Image_gen.BatchNorm...   [4, 64, 126, 166]
173_fire_img_0.trans_uplay1.Image_gen.ConvTrans...   [4, 1, 128, 168]
174_fire_img_0.trans_uplay1.Image_gen.Upsamplin...   [4, 1, 480, 640]
175_fire_img_0.trans_uplay2.Image_gen.BatchNorm...   [4, 512, 60, 80]
176_fire_img_0.trans_uplay2.Image_gen.Conv2d_1       [4, 1, 60, 80]
177_fire_img_0.trans_uplay2.Image_gen.ReLU_2         [4, 1, 60, 80]
178_fire_img_0.trans_uplay2.Image_gen.BatchNorm...   [4, 1, 60, 80]
179_fire_img_0.trans_uplay2.Image_gen.ConvTrans...   [4, 16, 60, 80]
180_fire_img_0.trans_uplay2.Image_gen.ReLU_5         [4, 16, 60, 80]
181_fire_img_0.trans_uplay2.Image_gen.BatchNorm...   [4, 16, 60, 80]
```



```
182_fire_img_0.trans_uplay2.Image_gen.ConvTrans...     [4, 16, 62, 82]
183_fire_img_0.trans_uplay2.Image_gen.ReLU_8           [4, 16, 62, 82]
184_fire_img_0.trans_uplay2.Image_gen.BatchNorm...     [4, 16, 62, 82]
185_fire_img_0.trans_uplay2.Image_gen.ConvTrans...     [4, 32, 64, 84]
186_fire_img_0.trans_uplay2.Image_gen.ReLU_11          [4, 32, 64, 84]
187_fire_img_0.trans_uplay2.Image_gen.BatchNorm...     [4, 32, 64, 84]
188_fire_img_0.trans_uplay2.Image_gen.ConvTrans...     [4, 64, 66, 86]
189_fire_img_0.trans_uplay2.Image_gen.ReLU_14          [4, 64, 66, 86]
190_fire_img_0.trans_uplay2.Image_gen.BatchNorm...     [4, 64, 66, 86]
191_fire_img_0.trans_uplay2.Image_gen.ConvTrans...     [4, 1, 68, 88]
192_fire_img_0.trans_uplay2.Image_gen.Upsamplin...     [4, 1, 480, 640]
193_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 1024, 30, 40]
194_fire_img_0.trans_uplay3.Image_gen.Conv2d_1         [4, 1, 30, 40]
195_fire_img_0.trans_uplay3.Image_gen.ReLU_2           [4, 1, 30, 40]
196_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 1, 30, 40]
197_fire_img_0.trans_uplay3.Image_gen.ConvTrans...     [4, 16, 30, 40]
198_fire_img_0.trans_uplay3.Image_gen.ReLU_5           [4, 16, 30, 40]
199_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 16, 30, 40]
200_fire_img_0.trans_uplay3.Image_gen.ConvTrans...     [4, 16, 32, 42]
201_fire_img_0.trans_uplay3.Image_gen.ReLU_8           [4, 16, 32, 42]
202_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 16, 32, 42]
203_fire_img_0.trans_uplay3.Image_gen.ConvTrans...     [4, 32, 34, 44]
204_fire_img_0.trans_uplay3.Image_gen.ReLU_11          [4, 32, 34, 44]
205_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 32, 34, 44]
206_fire_img_0.trans_uplay3.Image_gen.ConvTrans...     [4, 64, 36, 46]
207_fire_img_0.trans_uplay3.Image_gen.ReLU_14          [4, 64, 36, 46]
208_fire_img_0.trans_uplay3.Image_gen.BatchNorm...     [4, 64, 36, 46]
209_fire_img_0.trans_uplay3.Image_gen.ConvTrans...     [4, 1, 38, 48]
210_fire_img_0.trans_uplay3.Image_gen.Upsamplin...     [4, 1, 480, 640]
211_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 2048, 15, 20]
212_fire_img_0.trans_uplay4.Image_gen.Conv2d_1         [4, 1, 15, 20]
213_fire_img_0.trans_uplay4.Image_gen.ReLU_2           [4, 1, 15, 20]
214_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 1, 15, 20]
215_fire_img_0.trans_uplay4.Image_gen.ConvTrans...     [4, 16, 15, 20]
216_fire_img_0.trans_uplay4.Image_gen.ReLU_5           [4, 16, 15, 20]
217_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 16, 15, 20]
218_fire_img_0.trans_uplay4.Image_gen.ConvTrans...     [4, 16, 17, 22]
219_fire_img_0.trans_uplay4.Image_gen.ReLU_8           [4, 16, 17, 22]
220_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 16, 17, 22]
221_fire_img_0.trans_uplay4.Image_gen.ConvTrans...     [4, 32, 19, 24]
222_fire_img_0.trans_uplay4.Image_gen.ReLU_11          [4, 32, 19, 24]
223_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 32, 19, 24]
224_fire_img_0.trans_uplay4.Image_gen.ConvTrans...     [4, 64, 21, 26]
225_fire_img_0.trans_uplay4.Image_gen.ReLU_14          [4, 64, 21, 26]
226_fire_img_0.trans_uplay4.Image_gen.BatchNorm...     [4, 64, 21, 26]
227_fire_img_0.trans_uplay4.Image_gen.ConvTrans...     [4, 1, 23, 28]
228_fire_img_0.trans_uplay4.Image_gen.Upsamplin...     [4, 1, 480, 640]
229_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 2048, 1, 1]
230_fire_img_0.trans_upavg_pool.Image_gen.Conv2d_1     [4, 1, 1, 1]
231_fire_img_0.trans_upavg_pool.Image_gen.ReLU_2       [4, 1, 1, 1]
232_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 1, 1, 1]
233_fire_img_0.trans_upavg_pool.Image_gen.ConvT...     [4, 16, 1, 1]
234_fire_img_0.trans_upavg_pool.Image_gen.ReLU_5       [4, 16, 1, 1]
235_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 16, 1, 1]
236_fire_img_0.trans_upavg_pool.Image_gen.ConvT...     [4, 16, 3, 3]
237_fire_img_0.trans_upavg_pool.Image_gen.ReLU_8       [4, 16, 3, 3]
238_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 16, 3, 3]
239_fire_img_0.trans_upavg_pool.Image_gen.ConvT...     [4, 32, 5, 5]
240_fire_img_0.trans_upavg_pool.Image_gen.ReLU_11      [4, 32, 5, 5]
241_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 32, 5, 5]
242_fire_img_0.trans_upavg_pool.Image_gen.ConvT...     [4, 64, 7, 7]
243_fire_img_0.trans_upavg_pool.Image_gen.ReLU_14      [4, 64, 7, 7]
```



```
244_fire_img_0.trans_upavg_pool.Image_gen.Batch...     [4, 64, 7, 7]
245_fire_img_0.trans_upavg_pool.Image_gen.ConvT...     [4, 1, 9, 9]
246_fire_img_0.trans_upavg_pool.Image_gen.Upsam...     [4, 1, 480, 640]
==================================================================
```

Figure A.1: Kernel and Output Shape

A.2 Training and Validation Loss - ResNet50-Siamese

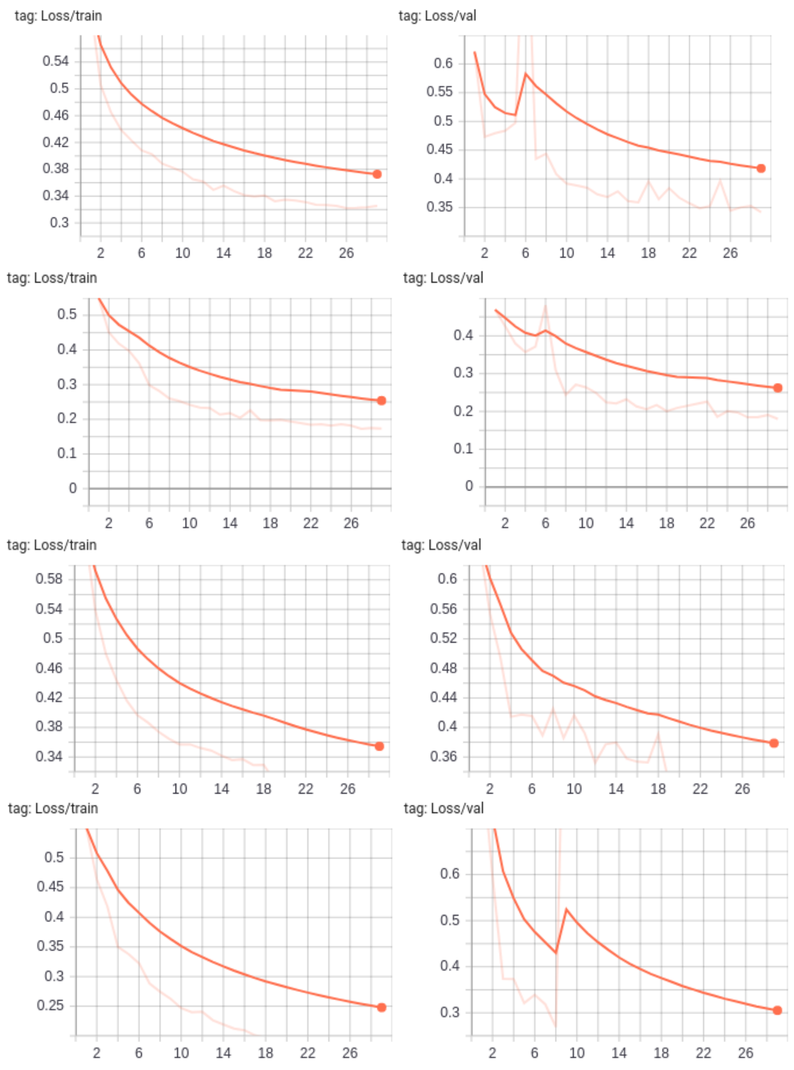


Figure A.2: Training and Validation loss (a) Fuse (b) Diff (c) Fuse + GAP (d) Diff + GAP

The training loss (Loss/train) and validation loss (Loss/val) for the best four models are shown using a graph, as shown in Fig. A.2. The X-axis represents the number of epochs the models are trained, the Y-axis represents the loss values. The full dataset is trained for 30 epochs. Every model run on the full dataset took approximately 2 days and 12 hours to train. The model is not completely converged as it is trained only for 30 epochs.

The fuse technique ($Siamese\_Res50\_Fuse\_Net$) shows a small spike in the validation loss during the initial training epochs. Meanwhile, the difference technique ($Siamese\_Res50\_Diff\_Net$) shows the best trend in the loss function.

Looking at the loss trends, the model is learning and the progressive decrease in the loss value shows that the models will converge if trained for more number of epochs and yield a better score during evaluation.

B

Appendix 2

Fig. B.1 below, show the losses for SPP which incude the Bce losses, Reconstruction losses , Ae losses and Kl losses

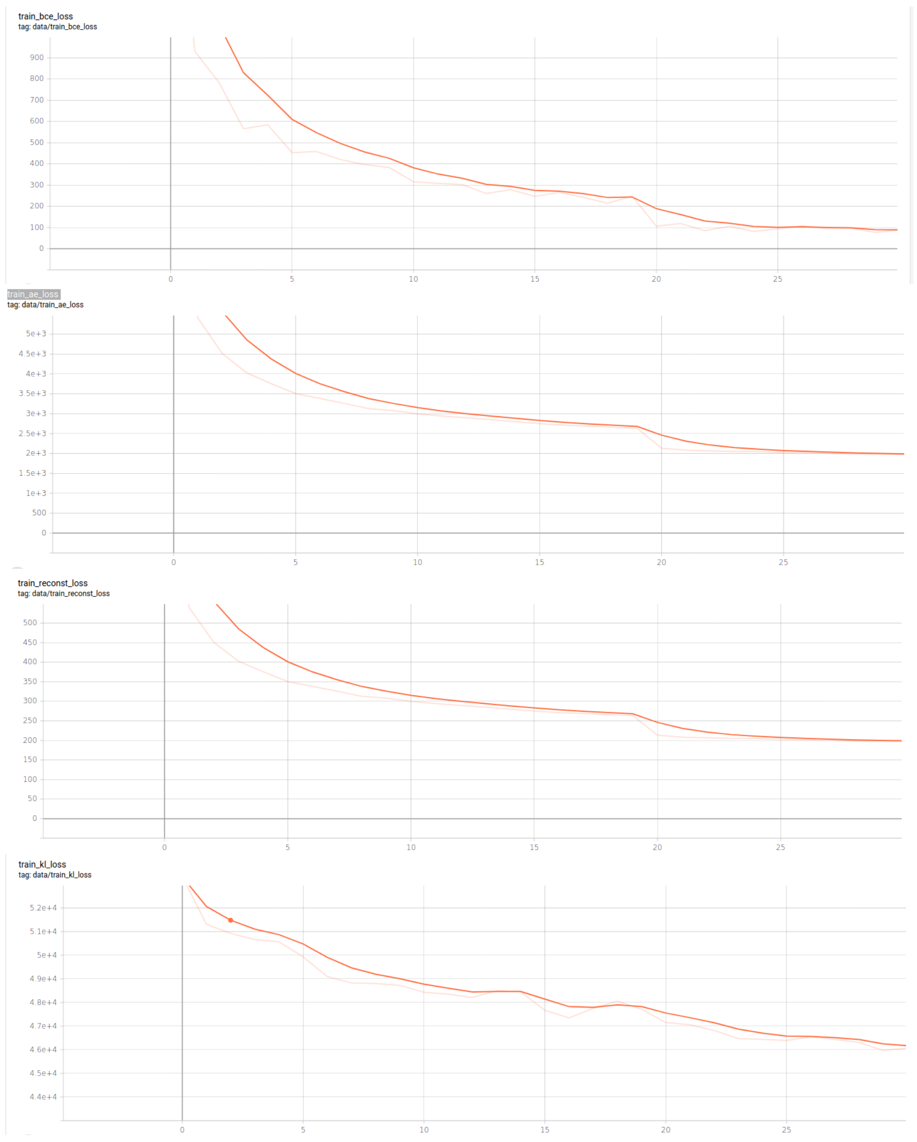


Figure B.1: SPP Losses