Deep Hierarchical Reconstruction for Open-Set Recognition

by

Kalyan Ainala

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Pavan Turaga, Chair
Hemanth Kumar Demakethepalli Venkateswara
Bahman Moraffah

ARIZONA STATE UNIVERSITY

August 2021

ABSTRACT

The field of Computer Vision has seen great accomplishments in the last decade due to the advancements in Deep Learning. With the advent of Convolutional Neural Networks, the task of image classification has achieved unimaginable success when perceived through the traditional Computer Vision lens. With that being said, the state-of-the-art results in the image classification task were produced under a closed set assumption i.e. the input samples and the target datasets have a knowledge of class labels in the testing phase. When any real-world scenario is considered, the model encounters unknown instances in the data. The task of identifying these unknown instances is called Open-Set Classification. This dissertation talks about the detection of the unknown classes and the classification of the known classes. The problem is approached by using a neural network architecture called Deep Hierarchical Reconstruction Nets (DHRNets). It is dealt by leveraging the reconstruction part of the DHRNets to identify the known class labels from the data. Experiments were also conducted on Convolutional Neural Networks (CNN) on the basis of softmax probability, Autoencoders on the basis of reconstruction loss and Mahalanobis distance on CNNs to approach this problem.

ACKNOWLEDGMENTS

I would like to thank Professor Hemanth Kumar Demakethepalli Venkateswara, for providing with such a great opportunity to work on my dissertation under his guidance. I am grateful for his patience, encouragement, and valuable suggestions that helped me finish my thesis.

Special thanks to Prof. Pavan Turaga for putting up with my slow progress and supporting my thesis extension. I am grateful to Prof. Bahman Moraffah for showing interest to serve as a panel member.

I am thankful to my friend Balavenkat Gottimukkala who assisted me with his knowledge by providing help with technical issues and also to Daniel for assisting me with my queries.

This thesis would not have been possible without the Graphics Processing Unit (GPU) support from Agave cluster, a high-performance computing resource managed by the Research Computing team. Thanks to the help desk staff for making it easy to use.

Finally, I am grateful to my brother, parents, friends for their financial and mental support especially in the year 2020.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

While striding towards the objective of image classification, let us consider the following two scenarios:

1. **Closed set classifier**

   Let's say that a classifier is trained on a dataset containing images of cats and dogs. During the testing phase, we verify our model's performance using the following samples and record the model's predictions:

   When the input sample is an image of a dog, the model predicts that it's a dog with 90% confidence. When the input sample is an image of a cat, the model predicts that it's a cat with 80% confidence. When the input sample is an image of a truck, the model predicts that it's a cat with 60% confidence and 40% dog .

2. **Open set classifier**

   Similar to the previous scenario, the classification model is trained on a dataset containing images of cats and dogs. But this time, the model has some additional qualities. It gives the following outputs for the respective inputs.

   When the input sample is an image of a dog, the model predicts that it's a dog with 90% confidence. When the input sample is an image of a cat, the model predicts that it's a cat with 80% confidence. When the input sample is an image of a truck, the model classifies it as an unknown instance.

The above examples clearly demonstrate the difference between performing the Image Classification in a closed set and open-set scenario. Speaking of image classification and the latest advancements in Computer Vision, it is paramount to discuss about the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The challenge which happens annually evaluates the algorithms for Image Classification as well as Object Detection. Often times the challenge is referred to as the ImageNet challenge as it deals with the dataset called ImageNet. This particular dataset ImageNet is credited well in the field of Computer vision as it contains about 14 million images and almost 21 thousand classes.

When we look at the milestones of this challenge, Alexnet, VGGNet, GoogLeNet, and ResNet have achieved the lowest error rates between 2012 to 2017 ImageNet challenges. From a traditional Computer Vision standpoint, these results are considered as the greatest achievements in the history of Image classification tasks. Though we talk high about these algorithms, Can these state-of-the-art methods be directly used in the real world? It's a big NO.

As they were developed in a closed set assumption, using those Deep learning models in any Real-world scenario like self-driving cars, medical diagnoses like breast cancer classification would drastically deteriorate the model robustness. This is where algorithms developed for an open set environment come into the picture.

The term Open set Recognition is also referred to as Open set Learning. It deals with two important roles.

1. Detect known and unknown samples from a pile of data.

2. Classification of the known data.

1.2 Objective

The primary aim of this thesis is to detect the known and unknown samples from a pile of data. We have applied various deep learning techniques to approach this task. Convolutional Neural Networks (CNN), Autoencoders, Deep Hierarchical Reconstruction Net were used to find the unknown samples.

1.3 Thesis Outline

This thesis is divided into seven parts,

Chapter 1 gives the introduction to the open set problem, and why is it important to solve it.

Chapter 2 is the literature survey for the task of open set learning. It discusses about various approaches applied by the previous researches to deal with the open set problem using CNN, Autoencoders and Generative Adversarial Network (GAN).

Chapter 3 introduces the concept of CNNs and how it functions. It goes into the details of various architectures that were used in the task of Image Classification.

Chapter 4 covers the concept of Autoencoders and the types of autoencoders.

Chapter 5 talks about a distance metric called Mahalanobis distance which can be used with CNNs to find the unknown samples.

Chapter 6 introduces the Deep Hierarchical Reconstruction Net's and how they benefit the task of open set learning.

Chapter 7 includes all the experiments conducted on CNNs, Autoencoders, Mahalanobis metric on CNNs, and DHRNets.

Chapter 2

LITERATURE REVIEW

The task of Image classification has achieved tremendous success in the closed set environment, while the research was very limited in the open-set scenario. The initial research traces itself to traditional machine learning where Support vector machines(SVM), Nearest Neighbour methods were used to tackle the open-set problem. Toward Open-set recognition, Scheirer *et al.* (2012) is one of the early papers on open-set which solely works on SVM. It finds the open-set samples by taking advantage of an extra hyper line. Another paper on SVM is Jain *et al.* (2014) which keeps the outlier at bay by using Extreme value theory and Pi-SVM. In the nearest neighbor method, we have Júnior *et al.* (2017) which works on the measurement of similarity scores between any two analogous classes. Bendale and Boult (2015) defines the problem as open space risk and approach the problem with a distance-based method called Nearest Non-Outlier (NNO) algorithm.

2.1 Convolutional Neural Networks

The primitive work on open-set using CNNs was discussed in Hendrycks and Gimpel (2016) which depends on the softmax probability to detect the unknown samples. It works on the assumption that the out-of-distribution samples get a lower probability compared to the known samples. But later on, results showed that there is a good chance of unknown samples getting a higher probability. ODIN (Out-of-DIstribution detector for Neural networks) was proposed by Liang *et al.* (2017) which expands the previous work by Hendrycks and Gimpel (2016). This work introduced two concepts. Firstly, temperature scaling increased the gap between the softmax probability scores for known and unknown samples. Secondly, adding little perturbations to the input sample in order to push the maximum

predicted softmax score. These two methods gave better results than the prior work i.e Hendrycks and Gimpel (2016) which is considered as a baseline method. The other works that we are going to discuss have introduced alternatives to the softmax part of the network.

In the next work, Bendale and Boult (2016) adapted Meta-Recognition concepts to the second to last layer of the neural network. They have introduced a new layer called OpenMax a substitute to softmax operator to find the probability of an input image belonging to an unknown class.

Shu *et al.* (2017) works on solving the open-set problem in text classification by introducing a model called the Deep Open Classifier (DOC) model, which replaces the softmax layer with a 1-vs-rest layer containing sigmoid functions.

2.2 Autoencoders and Generative Adversarial Networks (GAN)

OpenMax was implemented in GAN's as Generative OpenMax to create unknown samples while training. This research deals with hand-written characters only. Similarly, this concept was used in the Neal *et al.* (2018), Counterfactual image generation. This involves image augmentation and deals with creating unknown or out-of-distribution images. Oza and Patel (2019) proposed a class conditional autoencoder (C2AE) which takes advantage of the reconstructed image that is generated by the decoder. In order to find the threshold to detect the known or unknown classes, the reconstruction errors were modeled using the Extreme Value Theory.

Chapter 3

CONVOLUTIONAL NEURAL NETWORKS

The idea of coming with a good algorithm in order to make machines perceive visual data similar to humankind existed since the 1950's. It was only since 2012 that the field of Computer Vision has seen tremendous growth which helped to narrow the gap between human vision and machine vision. This rapid advancement in Computer Vision was only possible thanks to Convolutional Neural Networks. They are also known as CNNs or ConvNet's. CNNs are inspired by the hypothesis of the researchers D. H. Hubel and T. N. Wiesel. They proposed that the way mammals see the visual world is because of the neural architecture in the brain. This inspired Computer Scientists to come up with a similar idea that resulted in the development of CNNs.

Definition :

The Convolution part in the name "Convolutional neural Network" refers to the mathematical operation of the same name. CNNs use this operation in order to perform matrix multiplication.

A CNN consists of three important layers which are stacked together. They are Convolutional layers, Pooling layers and Fully-connected layers.

3.1 Convolutional Neural Network (CNN)

Convolutional Neural networks are an extension of neural networks comprising of neurons with learnable weights and biases. CNNs are designed to be used explicitly with images. This allows us to encode certain properties into the architecture that aid in learning tasks relevant to image data and its distribution. CNNs take advantage of the image as an

input and help constrain the architecture and vastly reduce the total number of network parameters. CNNs are made up of a sequence of layers that transform a volume of activations into another using differentiable functions Goodfellow *et al.* (2016). Figure **??** shows the basic architecture of a Convolutional Neural network (CNN) with its primary layers. The CNN layers and its components have been discussed in the following sections.

3.1.1 Convolutional Layer

The main aim of these layers is to produce a feature map. This is achieved by using learnable kernels or windows or filters. The process involves a filter which is passed on over a subset of an input image and in order to produce a feature map. This is repeated for all the subsets of the image.

In the figure 3.1, It shows a matrix multiplication between a 3×3 sized kernel and same sized subset of an input image matrix. The output of the mathematical operation is then summed and placed on the destination pixel on the feature map. Similarly, this operation is performed on the remaining 3×3 subsets of the image to complete the feature map.

The prime agenda of this operation is to extract the high level features from the input image like edges, gradient orientation, color etc. In order to do this we use multiple convolutional layers. All these layers are responsible for extracting the low level features of the input.

While executing the convolution process we can see how the kernal gives less weightage to the pixels at the corner by passing over the pixels lying inside the image matrix more than the pixels that are lying at the border of the image. To tackle this we introduce Padding, where we add an extra row and column of zero's to the sides of the input matrix. By assigning the pixel value as zero we do not provide any extra information but it helps to solve the problem by giving importance to the corner pixels.

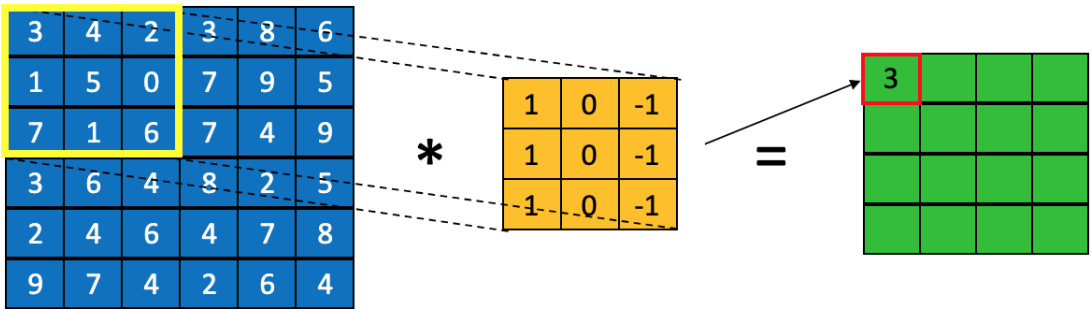

Figure 3.1: Convolution Operation

While performing convolution, we have always moved our filter on the input by a pixel. Striding is a concept in this process which considers the length of the step or stride value as a hyperparameter. By increasing the stride we perform fewer calculations and also decrease the spatial dimension of our feature map. The dimension of the output matrix considering the hyperparameters padding and striding can be calculated below where p is padding, s is striding parameter, f is the filter size and n_{in} is the image size.

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - f}{s} + 1 \right\rfloor$$

We pass the resulting feature map through a non-linear threshold which is applied on all the pixel values. A ReLU function is a popular activation function which replaces a negative value by '0' and keeps the value the same if it is otherwise. This whole operation completes the convolution part of the network. Before moving to the next layer, It is important to know that the filter and the image should have equal number of channels. When we are dealing with multiple filters on a single image we perform the convolution separately for every filter and finally stack the results to get the output.

The dimensions of the received tensor (as our 3D matrix can be called) meet the following equation, in which: n is image size, f is filter size, n_c is number of channels in the image, p is padding, s is stride, n_f is the number of filters.

$$[n, n, n_c] * [f, f, n_c] = \left[\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, n_f \right]$$

3.1.2 Pooling Layer

The role of the pooling layer is to decrease the dimensionality of the feature representation. By doing that we cut down the parameters eventually leading to lower computation. These layers perform similar operations as the filters by sliding over the feature map on but they have no learnable weights. Max Pooling is a common pooling operation that finds the maximum value of the subset of the feature map which leads to a reduction in the feature map. We make sure the subsets not to overlap by keeping a stride that is equal to the size of the subsets that are being pooled. When we perform max pooling we mostly use 2×2 kernels with a stride or step size of 2. This downsizes the output matrix to one-fourth of the original size but the volume remains the same. In addition to dimensionality reduction, max pooling also performs de-noising by discarding the noisy activations. The figure 3.2 shows the max pooling operation.

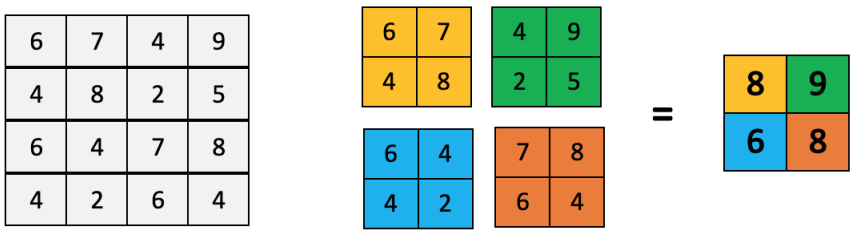

Figure 3.2: Max Pooling Operation

Average Pooling is another pooling technique that returns the average of the subsets of the image instead of maximum values. The figure 3.3 shows the Average pooling operation.

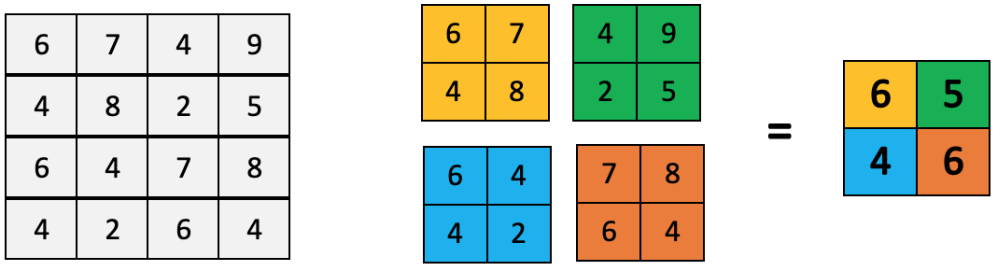

Figure 3.3: Average Pooling Operation

3.1.3 Activation Functions

The basic aim of an activation function is to learn complex patterns in the given data. The activation function is also known as a Transfer Function. It is used to introduce non-linearity to the neural network. This non-linear output is then transferred to the next neuron.

As we train the neural networks through gradient descent, we expect the layers to be differentiable. In order to satisfy this requirement all activation functions are differentiable functions.

Binary Step

This is a threshold based activation function. If the input of the activation function is higher than the threshold then the node is activated. It is deactivated in all the other cases. This particular activation function very well works for a binary classifier. The limitation is that it doesn't work if there are multiple classes.

The first thing that comes to our mind when we have an activation function would be a threshold based classifier i.e. whether or not the neuron should be activated based on the value from the linear transformation.

This function gives a zero gradient and has no affect on updating the weights during backpropagation.

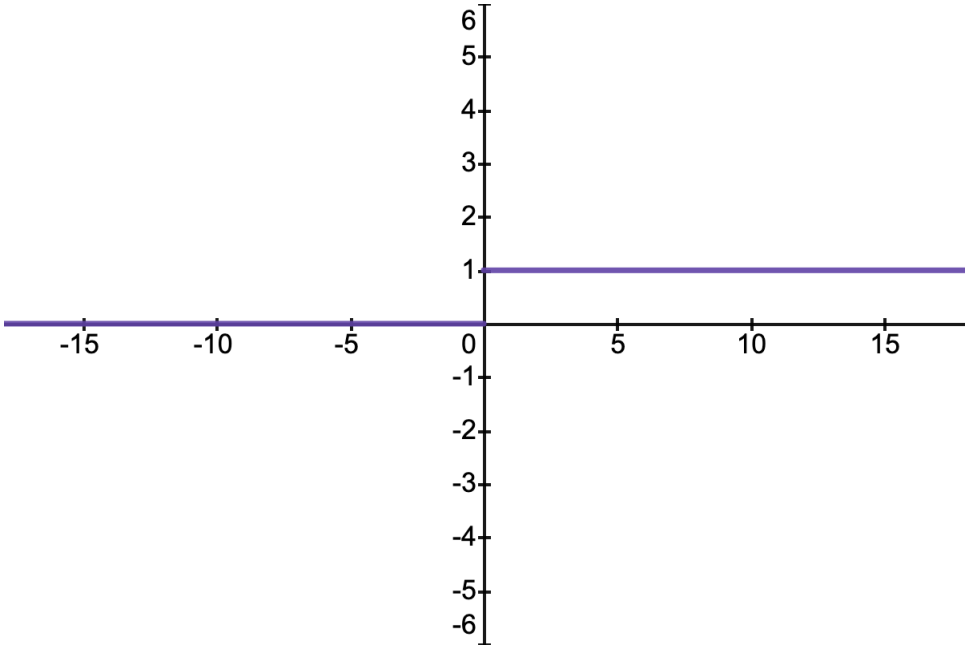

Figure 3.4: Binary Step Function

Linear Function

The activation or the output function is directly proportional to the input. A linear function has better functionality compared to Binary step as it contains the x component. The gradient of the function is not zero.

The gradient of the linear function though not zero, is a constant and is independent of the input x. During Backpropagation, the updating factors for weights and biases would be constant and doesn't help the network to learn.

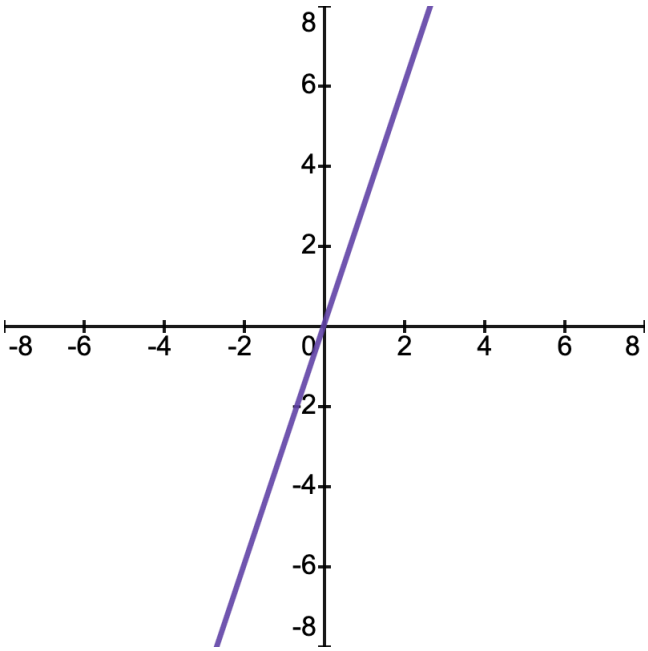

Figure 3.5: Linear Function

3.1.4 Non-Linear Activation Functions

Sigmoid

The Sigmoid Function is a widely used non-linear activation function. It is also known as Logistic function. The output ranges from 0 to 1. It uses a probabilistic approach to do this. The limitation is that it has almost no affect on the prediction for very large or very small inputs. This is the vanishing gradient problem which results in almost no learning by the

neural network.

.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

The figure, 3.6 shows the Sigmoid graph which is a smooth S-shaped function.

Additionally, the sigmoid function is not symmetric around zero. So output of all the neurons will be of the same sign. This can be addressed by scaling the sigmoid function which is exactly what happens in the tanh function.

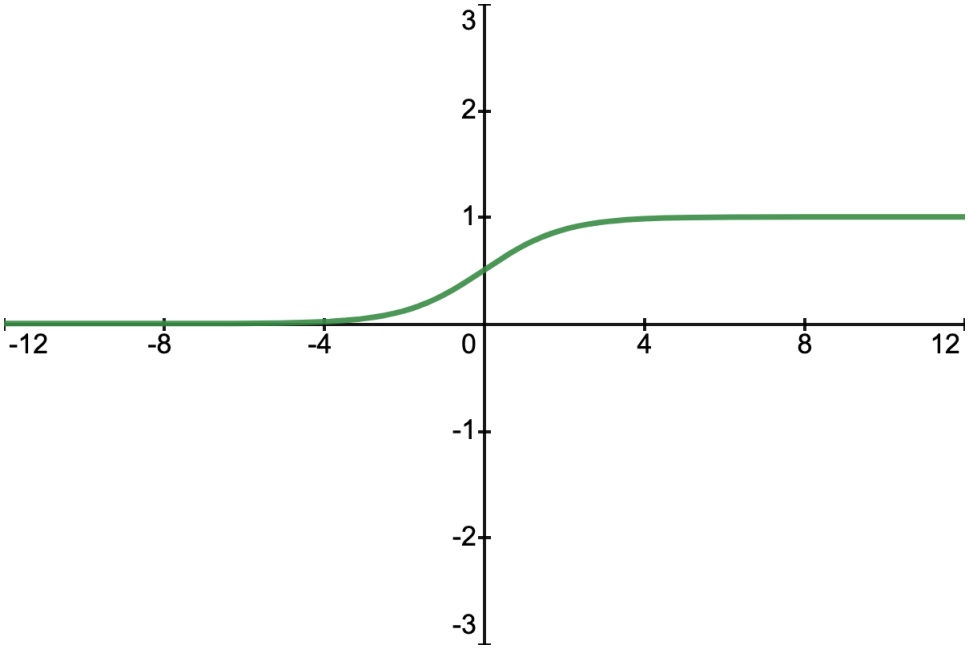

Figure 3.6: Sigmoid Function

Tanh

The Tanh function is almost similar to Sigmoid, except that the Tanh is symmetric around zero. Tanh function ranges from -1 to 1. The symmetry around zero helps in modeling inputs that are very large (positive) or very small (negative). The figure 3.7 shows the Tanh graph.

$$\tanh(x) = \frac{e^{-x} - e^{-x}}{e^{-x} + e^{-x}} \tag{3.2}$$

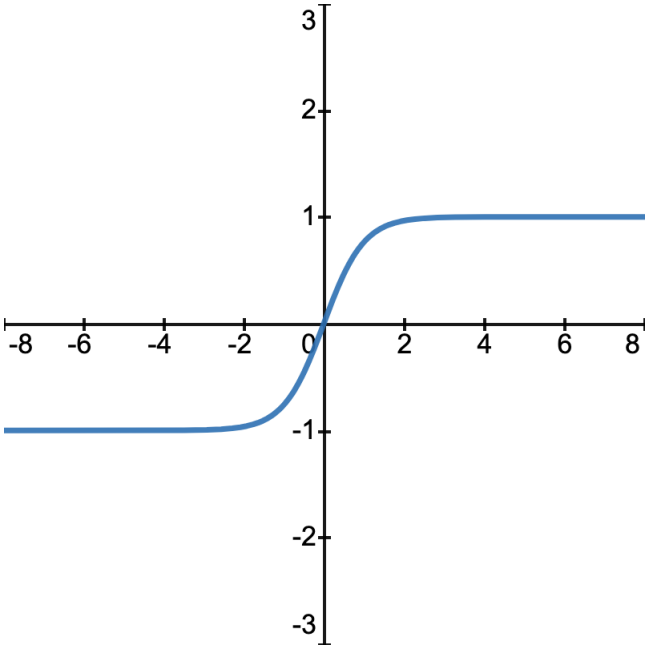

Figure 3.7: Tanh Function

ReLU

ReLU or Rectified Linear Unit is the most popular non-linear function in the Deep Learning domain. This function makes the neurons to be deactivated when the input is less than zero. It ranges from 0 to infinity for positive inputs and gives a 0 for all the negative values.

When we observe the negative side of the graph, the gradient is always zero. The weights and biases for the neurons will not be updated during the backpropogation. This is called the dying ReLU problem.

$$f_{ReLU}(x) = max(0, x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3.3}$$

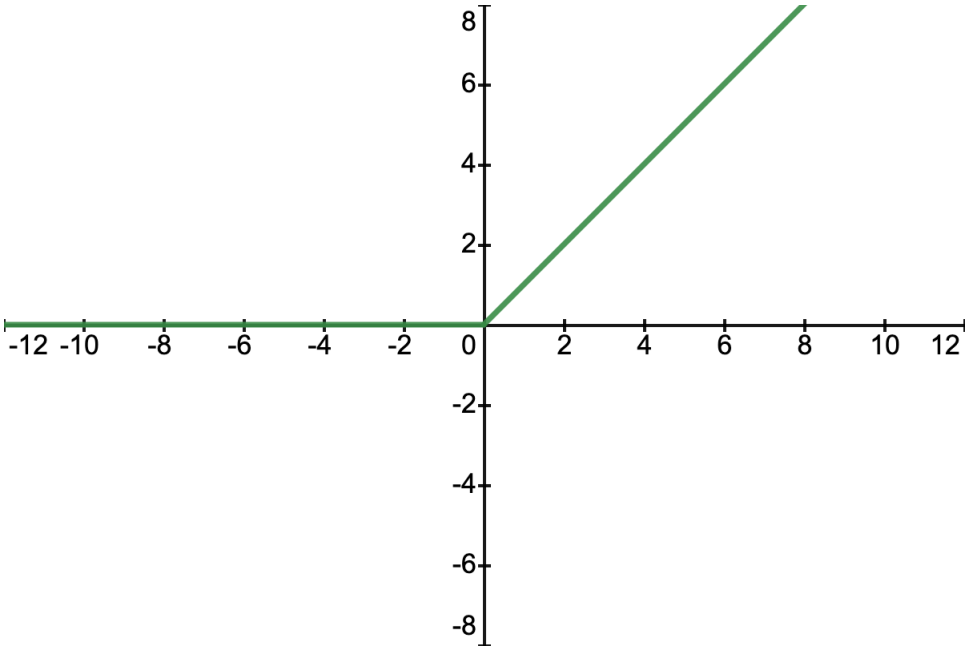

Figure 3.8: ReLU Function

Leaky ReLU

Leaky ReLU function can be considered as the advanced version of the ReLU function. Leaky ReLU solves the dying ReLU problem by coming up with a linear component of x in the negative values of the input. The figure 3.9 shows the Leaky ReLU graph.

$$f_{Leaky}(x) = max(0.1 * x, x) = \begin{cases} 0.1x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3.4}$$

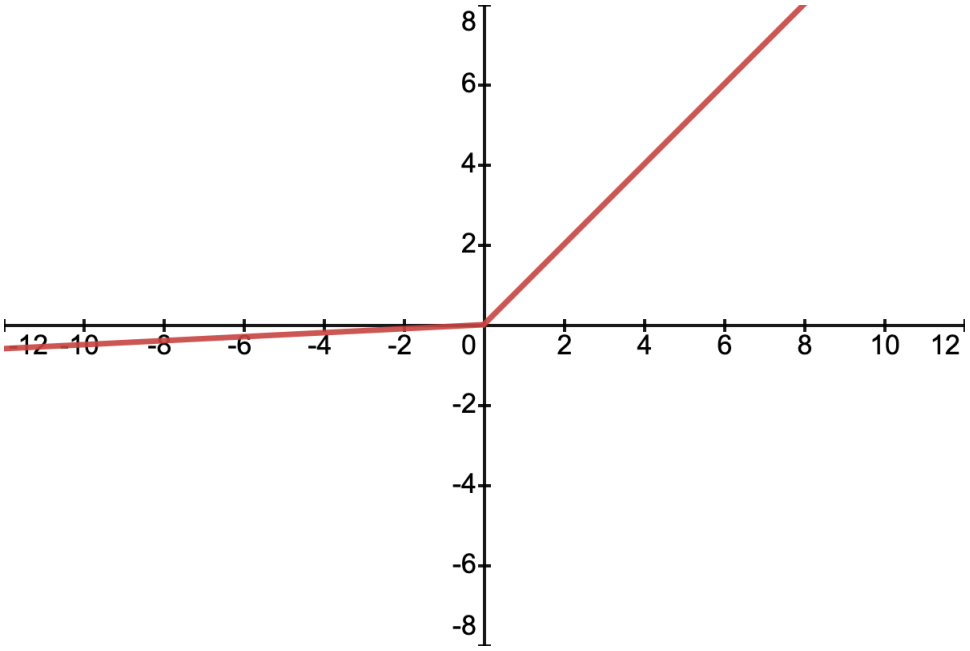

Figure 3.9: Leaky ReLU Function

Parameterised ReLU

Parameterised ReLU is another form of ReLU which solves the problem of dying ReLU problem. As the title "Parameterised", this function introduces a new parameter as a slope of the negative part of the function. The Equation 3.5 shows the Parameterised ReLU function.

$$f_{Leaky}(x) = max(a * x, x) = \begin{cases} ax & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3.5}$$

Exponential ReLU

Exponential ReLU is an another form of ReLU which solves the problem of dying ReLU problem. For defining the negative side of the graph, Exponential ReLU modifies the slope by using a log curve instead of a straight line as seen in the Leaky ReLU and parametric ReLU. The Equation 3.6 shows the Parameterised ReLU function.

$$f_{Leaky}(x) = max(a * exp(x) - 1, x) = \begin{cases} a(exp(x) - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3.6}$$

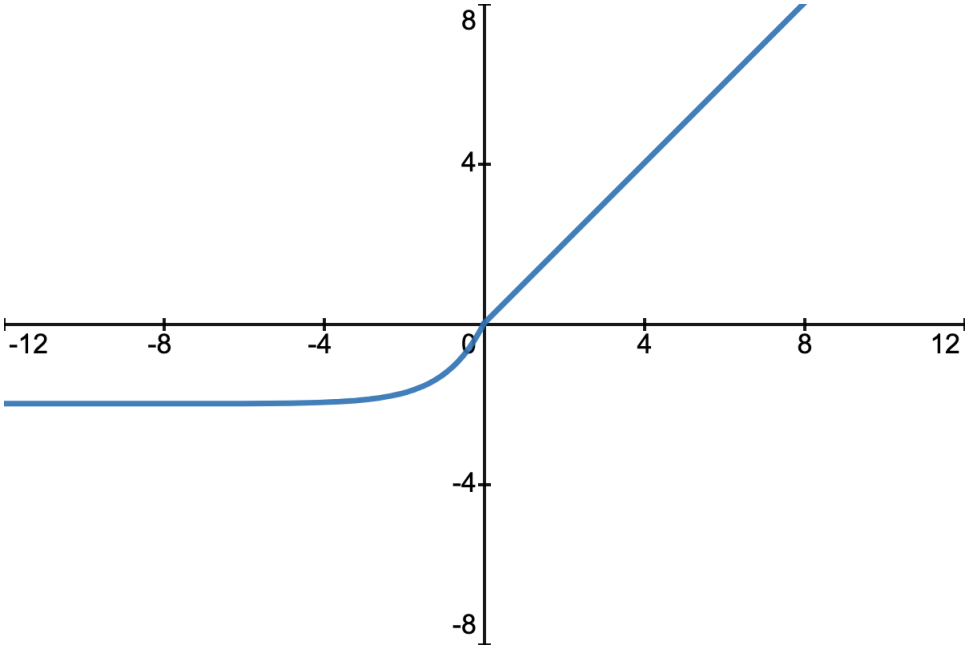

Figure 3.10: Exponential ReLU Function

Swish Function

Swish Function is a new activation function proposed by Google Brain Ramachandran *et al.* (2017). Swish shows better performance than a ReLU activation function especially on deeper networks. The output ranges from negative infinity to infinity.

In this activation function, the value of the function may increase when the input values are decreasing i.e. it is a non-monotonic function.

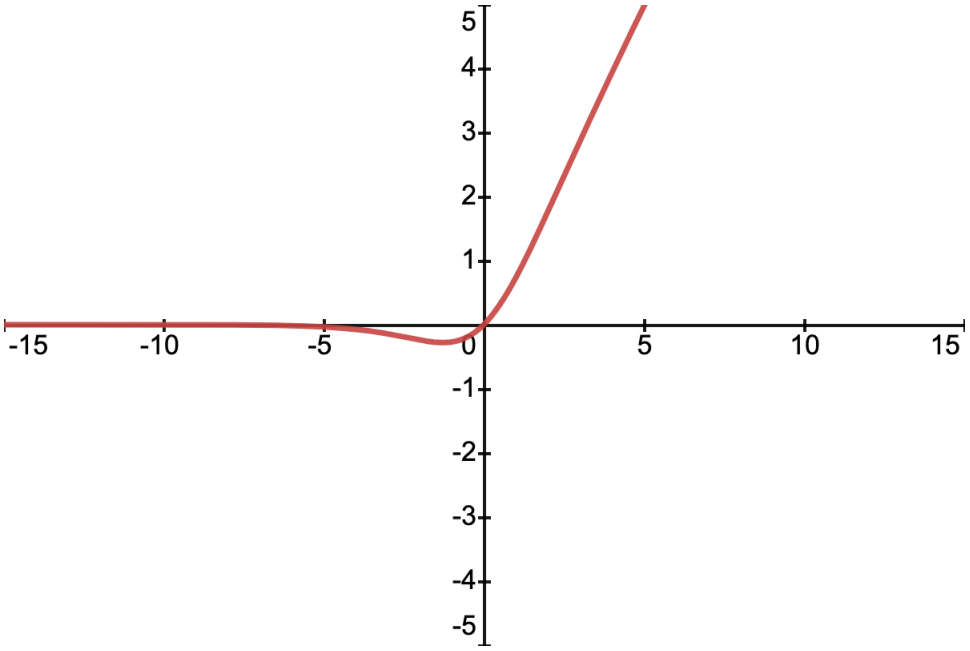

Figure 3.11: Swish Function

3.1.5 Fully-connected Layer

The last part of CNN architecture is the Fully connected layer is the important part for any Image Classification task. The neurons of this layer are directly connected to the neurons in the next layers but not within themselves. This layer consists of few hidden layers and the final part is passed on to the Softmax function to output softmax probabilities. Check figure 3.12 for a fully connected layer.

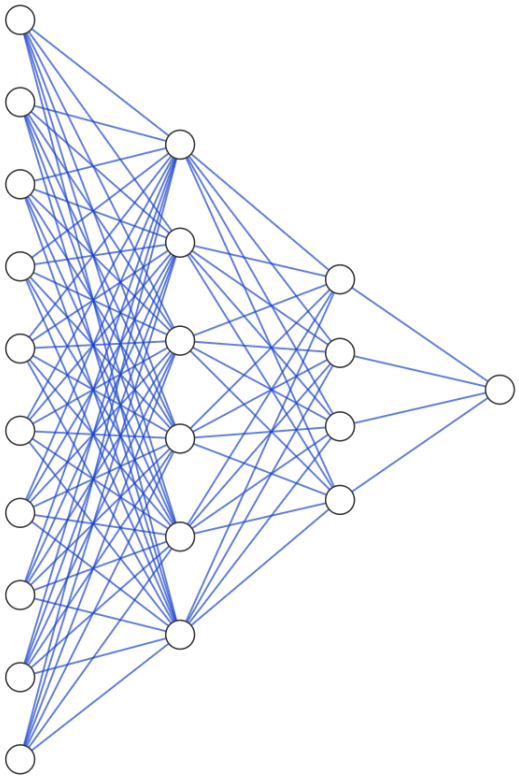

Figure 3.12: Fully Connected Layers

3.1.6 Batch Normalization Layer

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. Batch normalization methods help in the acceleration of training deep neural networks.

3.1.7 Backpropagation

When the CNN is trained on a dataset containing images, for every iteration the error that is generated by the layers will be fed back to CNN in order to adjust the weights of each

layer i.e. this backpropagation or backward propagation of errors fine-tunes the weights of a network from the errors generated from the previous iteration. By running more epochs we allow the network to lessen the errors and make the CNN model achieve generalization. The Backprop leverages the chain rules while calculating the gradient descent. Moreover, It simplifies the model by eliminating the weighted links that have a minuscule effect on the network. In a way, this method helps to show how much a given input has on the output of the network. Because of this, it is highly sensitive to noisy inputs, which is the biggest disadvantage.

3.1.8 Dropout Layer

The concept of Dropout was first introduced in the paper Srivastava *et al.* (2014). It is used to reduce the problem of overfitting. When the model fits overwhelmingly well to the training data, we call it overfitting. Due to this, the model loses the capacity to learn new patterns from new data. In this case, the training data gives a great accuracy while the validation or test accuracy is poor. Adding more data, using data augmentation, using dropout are some of the ways to tackle overfitting. On the other hand, underfitting occurs when the model failed to capture the patterns of the data. We use Dropout within the neural network which ignores or drops some nodes while training. This ignoring of neurons occurs with a probability of $1 - p$.

The figure 3.13, shows how a complete CNN architecture would look like.

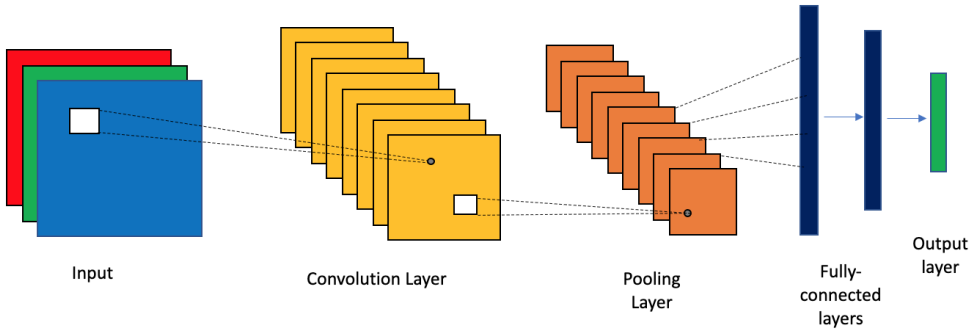

Figure 3.13: Complete CNN Architecture

3.2 State-Of-The-Art (SOTA) Architectures

This section talks about some of the most advanced Deep Learning architectures that have been used for the task of Image Classification.

LeNet-5

The very first paper that deals with application of Convolutional Neural Networks on Image Processing was introduced in 1998 by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner in the LeCun *et al.* (1998). This paper deals with classifying the handwritten gray scale images of digits. The dataset is named as MNIST or Modified National Institute of Standards and Technology Database. It contains images of size 32×32 in grayscale. This paper is considered as the very first successful application of CNNs on images. The initial prototype was used to classify zip codes from the United States Postal Service. The LeNet-5 model achieved 99 % accuracy on the MNIST test data. The figure 3.13 shows the LeNet-5 architecture.

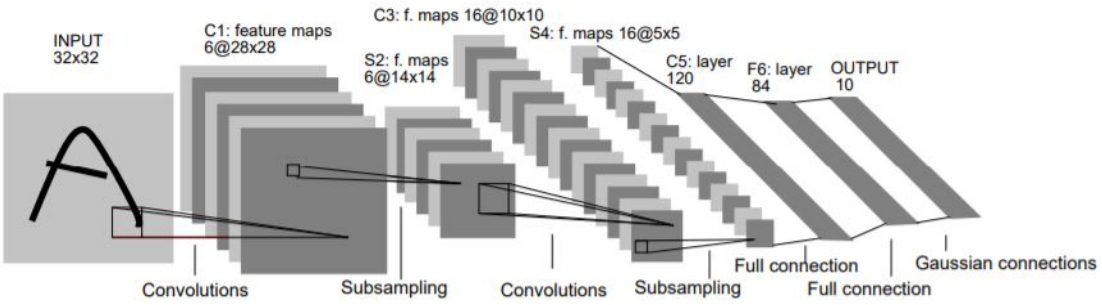

Figure 3.14: Architecture of LeNet Neural Network. From LeCun *et al.* (1998)

Alexnet

It took a decade to come up with a new idea called Alexnet because of the lack of computing power as well as data. AlexNet is the name of a convolutional neural network (CNN), designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor (Wikipedia (2021)). Alexnet competed in the ImageNet Large Scale Visual Recognition Challenge in the year 2012 and achieved an error rate of only 15.3 percent (Krizhevsky *et al.* (2012)). This is a great achievement compared to the 26 % error rate for the runner up. Alexnet, although inspired from LeNet is deeper and bigger. The input here is 224×224 instead of 32×32 in LeNet. A ReLU non-linearity was used in this paper. It contains 5 Conv layers and 3 fully connected layers. The figure 3.15 shows the Alexnet architecture.

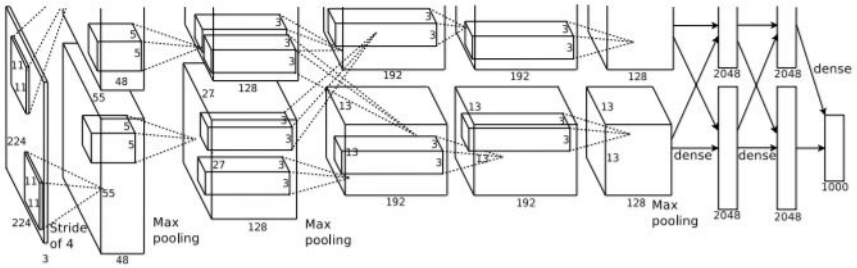

Figure 3.15: Architecture of Alexnet Neural Network. From Krizhevsky *et al.* (2012)

VGGNet

VGGNet was the runner up of ILSVRC 2014. Created by Karen Simonyan and Andrew Zisserman from Visual Geometry Group (VGG) at University of Oxford (Simonyan and Zisserman (2014)). It gave a 92.7 % test accuracy. VGGNet showed how accuracy can be enhanced by increasing the depth of the network. Compared to Alexnet it has 19 layers. VGGNet replaced the bigger filters like 7×7 or 11×11 in Alexnet with 3×3 convolution filters. This modification has reduced computation costs.

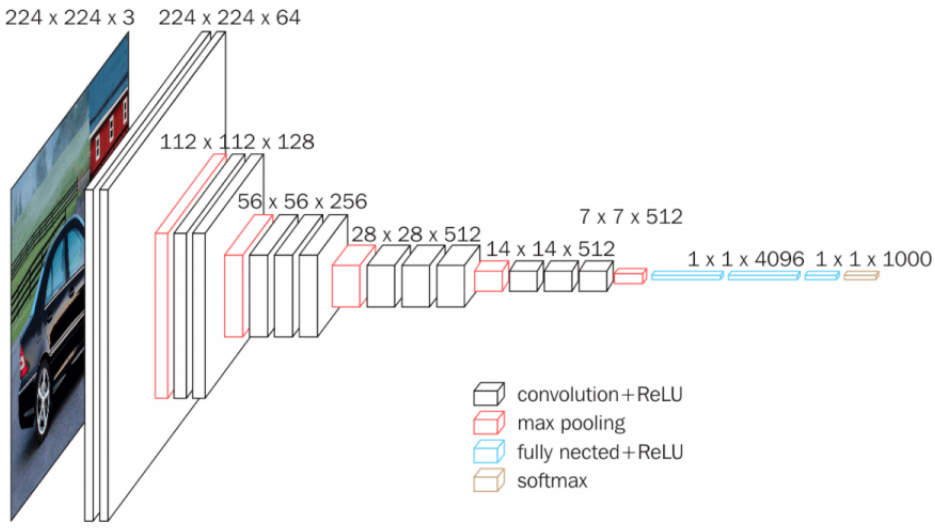

Figure 3.16: Architecture of VGGNet. From Simonyan and Zisserman (2014)

GoogleNet

GoogleNet was the winner of ILSVRC 2014 created by Szegedy *et al.* (2015) from Google. The prime contribution from GoogleNet is that it uses less number of parameters compared to previous networks. Alexnet deals with 60 million parameters whereas GoogleNet has 4 million parameters. A structure called Bottleneck is proposed in this network which introduced 1×1 convolution filters in order to reduce the feature dimensions before passing them through bigger computing operations.

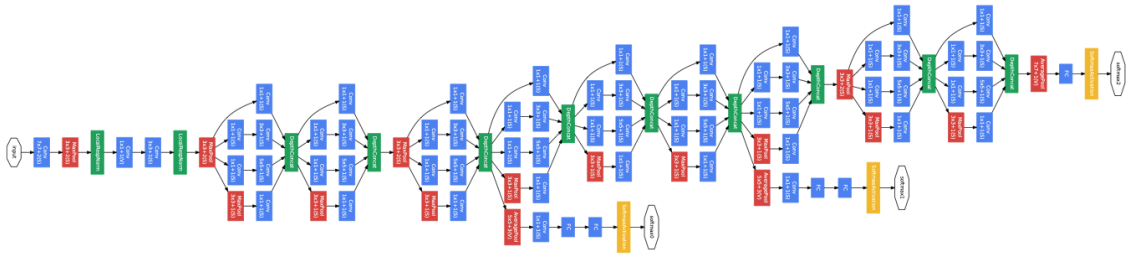

Figure 3.17: Architecture of GoogleNet Neural Network. From Szegedy *et al.* (2015)

ResNet

ResNet or Residual Network was the winner of ILSVRC 2015 created by Kaiming He and others from Microsoft Research Asia (Hayder *et al.* (2016)). The name comes from something called the Residual module. This was introduced to tackle the problem of vanishing gradients. The derivatives become too small when they are back propagated through deeper layers and thus gives the computation a tough challenge to represent such minuscule values.

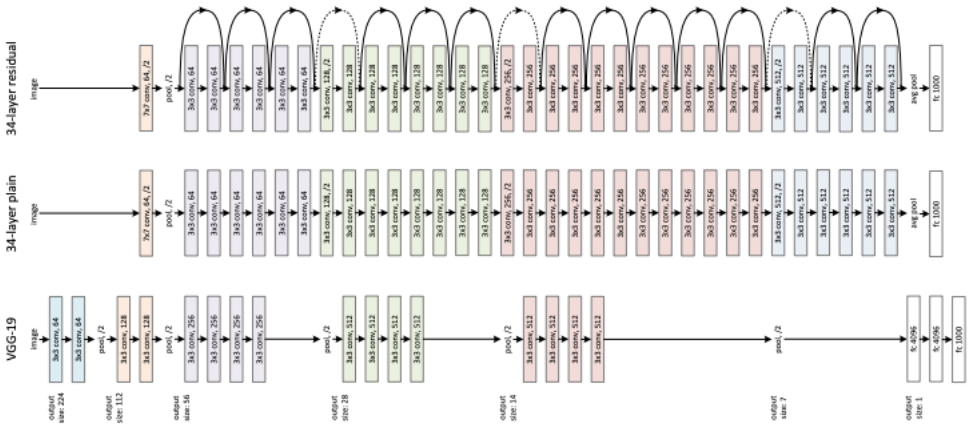

Figure 3.18: Architecture of ResNet Neural Network. From Hayder *et al.* (2016)

Chapter 4

AUTOENCODERS

Under unsupervised learning, we have a specific type of neural network called Autoencoders. They are used to deal with the task of representation learning, where we design the neural network architecture to mimic its input to its output. It transforms the input into a reduced representation i.e. dimensionality reduction. This representation is called code or embedding. In the next step, this code or embedding or latent-space representation is transformed into the original input. The embedding provides the non-linear relationship of input to output. The bottleneck in the neural network architecture does the job of forcing the original input into a compressed representation. The success of this task depends upon how well the input features are related. If the input features are nowhere dependent on each other, then the task of representing the input as well as reconstruction would be problematic. There are two important parts in an Autoencoder architecture,

1. The encoder, which reduces the input into an embedding representation
2. The decoder, which maps the embedding into a reconstructed representation of the original input.

This idea about autoencoders existed in the field of neural networks for decades. The very first application dates back to the 1980's which is dimensionality reduction.

4.1 The benefits of Autoencoders

4.1.1 Data Compression

Autoencoders force the input into a compressed representation. It is implemented in data transmission problems. Let's say the server uploads embedding of the input image and then we download it into our devices by decoding the embedding to get the output. This is used for media files like audio, image, and video.

4.1.2 Unsupervised learning

They work in an unsupervised setting, i.e. we can train an Autoencoder without the need for labels.

4.1.3 Lossy output

The input and the reconstructed output of the autoencoder are never 100 % the same. The decoder output misses some information of the input.

The four important parts of an Autoencoder are:

1. **Encoder:** The model performs dimensionality reduction and then compresses the input into an encoded representation called code or embedding.

2. **Bottleneck:** The layer that contains the Compressed form of the input. This compressed form of data is the least possible dimension of the original input.

3. **Decoder:** The model reconstructs the data from the embedding.

4. **Reconstruction Loss:** measures how close the output of the decoder is to the original input.

1. **Vanilla Autoencoder:** It is the simplest or basic form of an Autoencoder. It is well known as simple or Vanilla Autoencoder. It contains only one hidden layer. The figure 4.1 shows the Vanilla Autoencoder architecture.

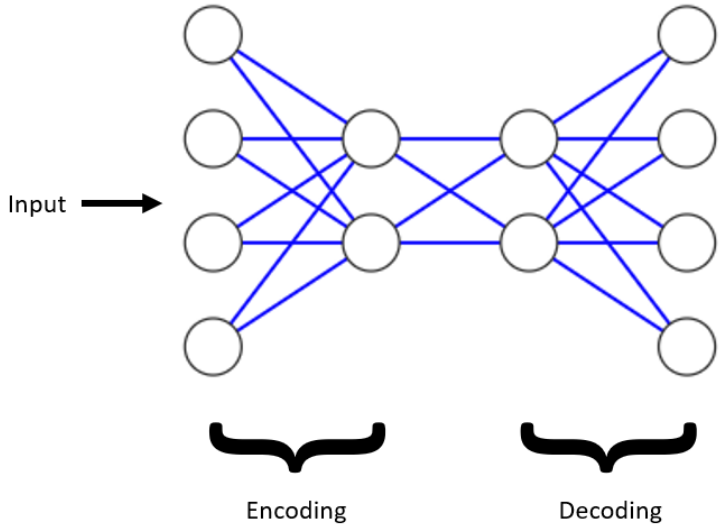

Figure 4.1: Vanilla Autoencoder

2. **Deep Autoencoder:** This is an extension of Vanilla autoencoders. It contains more hidden layers. The encoder and decoder consist of similar deep neural networks. The figure 4.2 shows the Deep Autoencoder architecture.

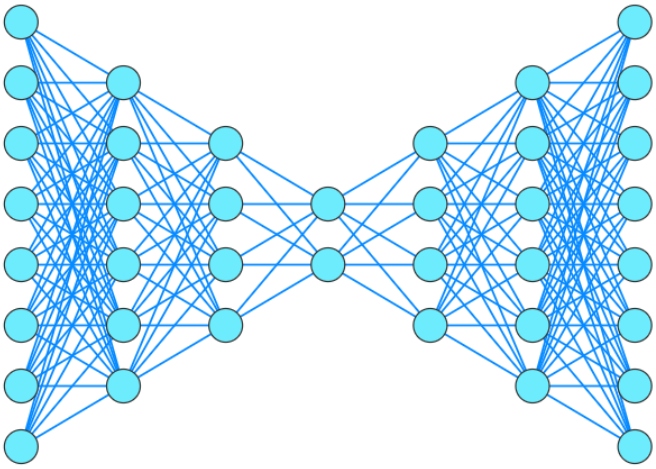

Figure 4.2: Deep Autoencoder

3. **Denoising Autoencoder:** Here we add noise to the original input to create a noisy version of the original image. When this corrupted version of the original input is presented to the autoencoder it is forced to learn features from the input but not to mimic the input by memorizing it. The figure 4.3 shows the Deep Autoencoder architecture.

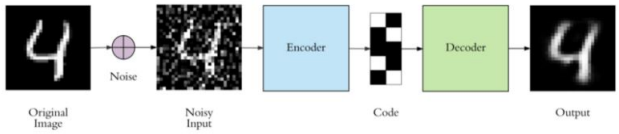

Figure 4.3: Denoising Autoencoder

4. **Convolutional Autoencoder:** As the name says, It is truly a variant of Convolutional Neural Networks. When convolution filters were used in an unsupervised learning scenario we call it Convolutional Autoencoders. They keep the spatial information of the image by using convolution layers, pooling layers and fully connected layers.

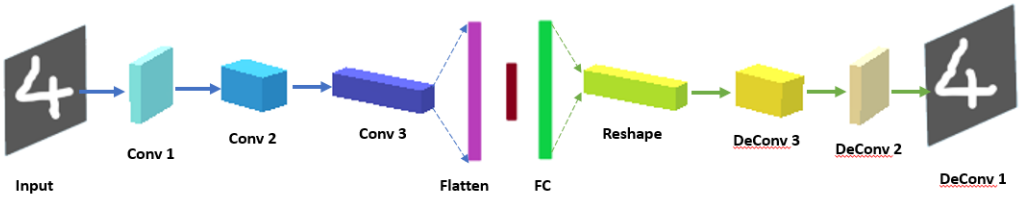

Figure 4.4: Convolutional Autoencoder

5. **Variational Autoencoder:** This is the most popular autoencoder. The encoder outputs mean and standard deviation. These two vectors describe the latent state attributes. In order to reconstruct they are then combined to pass on to the decoder. The figure 4.5 shows the Deep Autoencoder architecture.

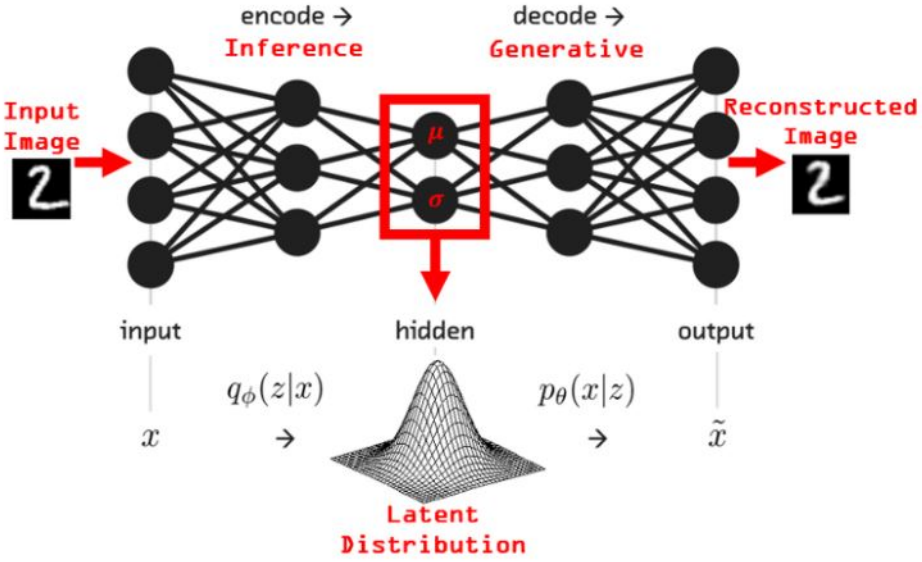

Figure 4.5: Variational Autoencoder

Chapter 5

MAHALANOBIS SCORE FOR OPEN SET DETECTION

Detecting the unknown or out-of-distribution samples from the given input samples is the prime objective of Open set Recognition. We can find the out-of-distribution samples by checking how far they exist from a training distribution. Mahalanobis distance is the distance between a point and a distribution. It was first introduced in the year 1936 by P.C. Mahalanobis. It is the most commonly used distance metric in statistics. Before diving into how Mahalanobis distance can be used to detect unknown samples, let's discuss about the most basic distance metric i.e. Euclidean distance and learn how Mahalanobis distance suits well for this problem.

Euclidean distance:

In mathematics, the distance between two points in Euclidean space is the length of a line segment between the two points. It can be calculated from the Cartesian coordinates of the points using the Pythagorean theorem, therefore commonly being known the Pythagorean distance. It is the straight line distance between two points.

In a two dimensional space, let's say we have two points $(x1, y1)$ and $(x2, y2)$ then the euclidean distance between them would be,

$$d(p, q) = \sqrt{(p_1 - q_1)\,2 + (p_2 - q_2)\,2} \tag{5.1}$$

When we extend to n dimensional space the formula would be,

$$d(p, q) = \sqrt{(p_1 - q_1)\,2 + (p_2 - q_2)\,2 + \cdots + (p_n - q_n)\,2} \tag{5.2}$$

This measurement works perfect until two conditions are met.

1. The dimensions are not correlated to each other.

2. The weightage of each dimension is same.

Mahalanobis distance:

Mahalanobis distance finds the distance between a point and a distribution but not between two points and works perfectly for multivariate data. It can be considered as an equivalent to Euclidean distance in multivariate space. It finds how many standard deviations away is a point from a distribution. Unlike Euclidean, the Mahalanobis distance takes co-variance matrix into consideration. This particular distance metric can detect outliers by finding the distribution patterns of data samples.

Mahalanobis distance can be calculated as,

$$D^2 = (x - m)^T \cdot C^{-1} \cdot (x - m) \tag{5.3}$$

where,

x is the vector of the observation.

m is the vector of mean values of independent variables.

C^{-1} is the inverse co-variance matrix of independent variables.

D^2 is the square of Mahalanobis distance.

$(x–m)$ is the distance of the point from the mean. We then divide this by the co-variance matrix or multiply it with the inverse of the co-variance matrix. When we observe it, the above operation is a multivariate equivalent of the regular standardization, $z = (x–mu)/sigma$. That is, z = (x vector) – (mean vector) / (co-variance matrix). which explains that Mahalanobis distance is a multivariate equivalent of euclidean distance. When there is strong correlation between variables the co-variance would be high. By dividing

the co-variance we reduce the distance. In the same way. When they are not correlated, then the co-variance would be low so dividing the co-variance doesn't reduce the distance. Thus, it tackles the problem of correlation and scale of the variables.

Outlier or Anomaly detection:

If we find some data far away from where majority of data is clustered, then we can consider that data as outliers. Detecting outliers based on the softmax classifier lead to errors as the classifier is notorious to produce high confidence to outliers too. We take advantage of the distance based generative classifier and measure the probability of the data sample on the feature space of deep neural networks. Our assumption is that the pre-trained features can be used to fit the Gaussian distribution for every class. We calculate a confidence value to every data sample based on the nearest class-conditional distribution. We find the class conditional Gaussian distributions for not just the final layer but also intermediate layers under the assumption of Gaussian discriminant analysis. Furthermore, we calculate the confidence score for the features based on their respective layers.

Here, our approach is based on a simple theoretical connection between Gaussian discriminant analysis and the softmax classifier: the posterior distribution defined by the generative classifier under Gaussian discriminant analysis with tied co-variance assumption is equivalent to the softmax classifier. Therefore, the pre-trained features of the softmax neural classifier $f(x)$ might also follow the class-conditional Gaussian distribution. (Lee *et al.* (2018))

To find the parameters of the classifier from the pre-trained softmax neural classifier, we compute the empirical class mean and co-variance of training samples $(x_1, y_1), \ldots, (x_N, y_N)$

Using the above induced class-conditional Gaussian distributions, we define the confidence score $M(x)$ using the Mahalanobis distance between test sample x and the closest

class-conditional Gaussian distribution, i.e.,

$$\widehat{\mu}_c = \frac{1}{N_c} \sum_{i:y_i=c} f\left(\mathbf{x}_i\right), \quad \widehat{\mathbf{\Sigma}} = \frac{1}{N} \sum_{c} \sum_{i:y_i=c} \left(f\left(\mathbf{x}_i\right) - \widehat{\mu}_c\right) \left(f\left(\mathbf{x}_i\right) - \widehat{\mu}_c\right)^\top \tag{5.4}$$

here, N_c is the total number of training samples with label c.

By following the above class-conditional Gaussian distributions, we further define the confidence score $M(x)$ using the Mahalanobis distance between test sample x and the closest class-conditional Gaussian distribution, i.e.,

$$M(\mathbf{x}) = \max_c - \left(f(\mathbf{x}) - \widehat{\mu}_c\right)^\top \widehat{\mathbf{\Sigma}}^{-1} \left(f(\mathbf{x}) - \widehat{\mu}_c\right) \tag{5.5}$$

5.0.1 Calibration techniques

Input pre-processing

In order to make the in and out-of-distribution samples more separable, we add a small noise to the samples. For every sample x we create a pre-processed sample x_b by adding small perturbations as follows:

To make the in and out-of-distribution samples more separable, we consider adding a small controlled noise to a test sample. Specifically, for each test sample x, we calculate the pre-processed sample xb by adding the small perturbations as follows:

$$\widehat{\mathbf{x}} = \mathbf{x} + \varepsilon \operatorname{sign}\left(\nabla_{\mathbf{x}} M(\mathbf{x})\right) = \mathbf{x} - \varepsilon \operatorname{sign}\left(\nabla_{\mathbf{x}} \left(f(\mathbf{x}) - \widehat{\mu}_{\widehat{c}}\right)^\top \widehat{\mathbf{\Sigma}}^{-1} \left(f(\mathbf{x}) - \widehat{\mu}_{\widehat{c}}\right)\right) \tag{5.6}$$

here, ε is a magnitude of noise and b_c is the index of the closest class. We then measure the Mahalanobis distance confidence score based on this new sample.

Feature ensemble

In order to improve the performance, we also measure the confidence scores from not just the final features but also from the other low-level or intermediate layers in the neural network.

We first extract the Mahalanobis confidence scores from all the layers, and then integrate them by weighted averaging: $\sum_\ell \alpha_\ell M_\ell(\mathbf{x})$, where $M_\ell(\cdot)$ and α_ℓ is the confidence score at the ℓ-th layer and its weight, respectively.

Chapter 6

DHRNET

Deep Hierarchical Reconstruction Nets or DHRNets were designed to deal with two important tasks. Firstly, the task of detecting the out-of-distribution samples and then find the classification. In deep neural networks, when an input is passed into deeper and deeper layers of the network the higher layers might lose the details of the input. To tackle this problem of loss of information, we save the latent representations at intermediate layers to be used to reconstruct the input. Loss of information about the input sample would be the biggest drawback when we are reconstructing the input. We produce the latent representations like $z1, z2, z3, \ldots$ from intermediate layers like $x1, x2, x3$. These latent representations play the most important role in reconstructing the input. The DHRNet architecture looks similar to an autoencoder. The left-hand side of the network plays the role of an encoder and the right-hand side mimics a decoder. Here the $z1, z2, z3, \ldots$ gets concatenated to their respective layers on the decoder side. This additional information passed on to the decoder provides the details of the input lost on the encoder side of the network.

Comparison of DHRNet with other networks:

Deep learning classification models only contain 'y' which can be used for softmax prediction. An autoencoder contains the reconstruction part in the decoder and can reconstruct the input image. The drawback is that it compresses the input so much that the image's output dimensionality becomes way too small, which bottlenecks the process of learning all the features of the input. LadderNet's can be considered as a better version of autoencoders for dealing with open-set problems. Here the information from intermediate layers from the

encoder side is passed on to the decoder side in order to have better reconstruction. This is the inspiration behind DHRNet. In the case of DHRNet, the latent representations from the encoder side are compressed and then decompressed in order to support reconstruction.

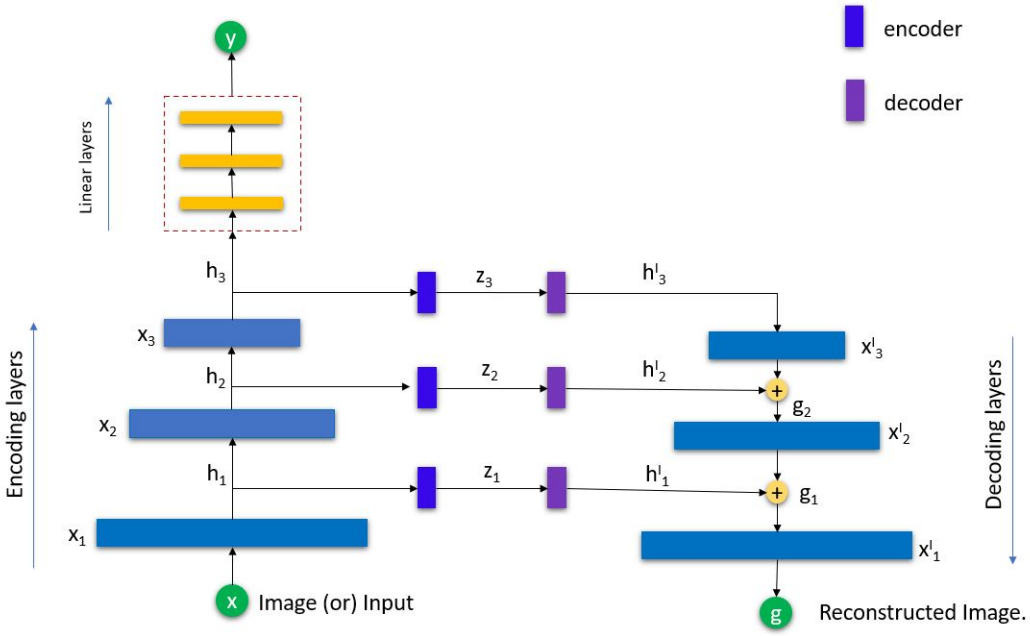

Figure 6.1: Architecture of DHRNet.

Let's say we have an image x that was input to the left hand side or encoder side of the network. It outputs h_1 after passing through the first convolutional layer x_1. This h_1 becomes the latent representation of our input image. When this representation is passed on to the second convolutional layer x_2, It outputs h_2 which is the second latent representation of our input. Similarly, we get h_3 as the third latent representation through the convolutional layer x_3. We have finished the encoding operation of the network.

All these intermediate latent representations h_1, h_2, h_3 are now passed on to an encoder which gives z_1, z_2, z_3 respectively. These outputs are decoded by a decoder to output h_1^l, h_2^l, h_3^l.

On the decoding side, we pass the h_3^l to the decoding layer x_3^l, a de-convolution layer. The output of the x_3^l is linearly added to h_2^l to result g_2. This g_2 is passed on to the next de-convolution layer x_2^l. Similarly, the output of x_2^l is added to h_1^l to result g_1. The final reconstructed image, g is the result of de-convolution of g_1.

Passing h_1, h_2, h_3 from the encoder to decoder as h_1^l, h_2^l, h_3^l is the most important part in DHRNet. The idea is that there is good amount of information loss that happens on the encoder. Thus, reconstruction of the image becomes difficult. In order to deal with the loss of information, we add the latent representations h_1, h_2, h_3 to the decoder side as h_1^l, h_2^l, h_3^l only after encoding and then decoding them.

Chapter 7

EXPERIMENTS, RESULTS AND ANALYSIS

In this dissertation, I have implemented an approach to deal with Openset Recognition. I have used various neural network architectures to detect out-of-distribution samples. The experiments were implemented on Convolutional Neural Networks, Autoencoders, pre-trained CNN's with Mahalanobis distance metric, Deep Hierarchical Reconstruction Net (DHRNet).

7.1 Datasets

In the Openset scenario, which happens in the real world we always deal with known and unknown samples. In order to create an openset environment, we deal with two kinds of datasets in our experiments. They are, in-distribution or known dataset and out-of-distribution or unknown dataset.

7.1.1 In-Distribution Datasets

1. **Cifar-10:** The CIFAR-10 dataset contains 60,000 RBG images. They are of 32×32 in resolution. The number 10 in the name Cifar-10 signifies 10 classes. Each class contains 6000 images. The classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

2. **Cifar-100:** The CIFAR-100 dataset contains 60,000 RBG images. They are of 32×32 in resolution. The number 100 in the name Cifar-100 signifies 100 classes. Each class contains 600 images.

7.1.2 *Out-of-Distribution Datasets*

1. **TinyImageNet:** This dataset contains 10,000 test images from 200 different classes. These images come from the original 1,000 classes of ImageNet data hence, the name TinyImageNet. Since the images from are of size 224×224, we have resized them to 32×32.

2. **LSUN:** The Large-scale Scene Understanding dataset (LSUN) consists of 10,000 images from 10 different scene categories. All images are resized to 32×32.

7.2 Threshold

In order to have a threshold while dealing with reconstruction losses of all samples in Autoencoders and DHRNets or with Mahalanobis confidence scores, I have used Otsu's method to calculate a threshold. The Otsu method is mostly used in Image processing while dealing with automatic thresholding. It outputs a pixel value to separate the foreground and background in an image.

7.3 Experiments using Convolutional Neural Networks

I have dealt with openset Recognition task on an assumption that the softmax probabilities for in-distribution would be high and out-of-distribution probabilities would be low.

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
DenseNet	Cifar-10	TinyImageNet	34.0	45.4
DenseNet	Cifar-10	LSUN	35.6	45.8

Table 7.1: Experiments Using Convolutional Neural Networks

7.4 Experiments on Autoencoders

I have approached the detection of out-of-distribution samples by looking at how well they are reconstructed by the decoder. I calculated the reconstruction loss for all the images and apply a threshold to filter out the out-of-distribution samples. I have trained the model on $45,000$ samples of cifar-10. I have tested the model on $55,000$ samples, $45,000$ cifar-10 and $10,000$ LSUN/TinyImageNet dataset.

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
Vanilla	Cifar-10	TinyImageNet	23.7	40
Vanilla	Cifar-10	LSUN	24.4	42.8
Deep	Cifar-10	TinyImageNet	30.3	51.3
Deep	Cifar-10	LSUN	31.5	52.9
Convolutional	Cifar-10	TinyImageNet	48.6	64.1
Convolutional	Cifar-10	LSUN	49.5	64

Table 7.2: Experiments Using Autoencoders

7.5 Experiments on CNN's using Mahalanobis distance

In this case, by considering the unknown or out-of-distribution samples as outliers we find the Mahalanobis score for each sample using a pre-trained CNN. I have tested the model on $20,000$ samples, $10,000$ cifar-10 and $10,000$ LSUN/TinyImageNet dataset.

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
DenseNet	Cifar-10	TinyImageNet	98.7	37.1
DenseNet	Cifar-10	LSUN	92.6	41.3

Table 7.3: Experiments on CNNs Using Mahalanobis Distance

7.6 Experiments using DHRNet

I have trained the model on $45,000$ samples of cifar-10. While training, the loss factor was either classification loss or reconstruction loss or both. Calculated the threshold using Otsu. I have tested the model on $55,000$ samples, $45,000$ cifar-10 and $10,000$ LSUN/TinyImageNet dataset.

DHRNet with loss = classification loss

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
DHRNet	Cifar-10	TinyImageNet	67.0	68.4
DHRNet	Cifar-10	LSUN	65.3	69.5

Table 7.4: Experiments on DHRNet Using Only Classification Loss

DHRNet with loss = reconstruction loss

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
DHRNet	Cifar-10	TinyImageNet	71.7	72.1
DHRNet	Cifar-10	LSUN	69.8	71.8

Table 7.5: Experiments on DHRNet Using Only Reconstruction Loss

DHRNet with loss = classification loss and reconstruction loss

Architecture	In-Distribution	Out-of-Distribution	True Positive rate	True Negative rate
DHRNet	Cifar-10	TinyImageNet	72.2	72
DHRNet	Cifar-10	LSUN	70.5	72.8

Table 7.6: Experiments on DHRNet Using Reconstruction Loss, Classification Loss

Chapter 8

CONCLUSION

In this thesis, I have successfully implemented the ways to detect unknown samples using CNNs, Auotencoders, CNNs using Distance metrics, and DHRNet. The experiments show that the DHRNet trained on different loss functions performs better than CNNs or Autoencoders. These experiments also show the importance of using different loss functions like Reconstruction, classification loss on DHRNet, and most importantly the adding the latent representation to the decoder side to solve the problem of lack of information faced in the reconstruction of the image. This process achieved a good performance when compared to the baseline methods.

8.1 Future Directions

There is a lot of scope for Openset detection in Computer Vision, especially in the autonomous car industry. Openset detection can be applied along with Active learning when the cost of labeling is expensive. I believe that there is a lot of room for improvement in the True Negative rate and True Positive rates by training the model on less or more amount of in-distribution samples or data from known samples. Another way of dealing with the openset problem is by considering half of CIFAR-10 classes as in-distribution and another half as out-of-distribution samples. Apply the algorithm on more real-world data.

REFERENCES

Bendale, A. and T. Boult, "Towards open world recognition", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1893–1902 (2015).

Bendale, A. and T. E. Boult, "Towards open set deep networks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1563–1572 (2016).

Goodfellow, I., Y. Bengio, A. Courville and Y. Bengio, *Deep learning*, vol. 1 (MIT press Cambridge, 2016).

Hayder, Z., X. He and M. Salzmann, "Shape-aware instance segmentation", arXiv preprint arXiv:1612.03129 **2**, 5, 7 (2016).

Hendrycks, D. and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks", arXiv preprint arXiv:1610.02136 (2016).

Jain, L. P., W. J. Scheirer and T. E. Boult, "Multi-class open set recognition using probability of inclusion", in "European Conference on Computer Vision", pp. 393–409 (Springer, 2014).

Júnior, P. R. M., R. M. De Souza, R. d. O. Werneck, B. V. Stein, D. V. Pazinato, W. R. de Almeida, O. A. Penatti, R. d. S. Torres and A. Rocha, "Nearest neighbors distance ratio open-set classifier", Machine Learning **106**, 3, 359–386 (2017).

Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", Advances in neural information processing systems **25**, 1097–1105 (2012).

LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE **86**, 11, 2278–2324 (1998).

Lee, K., K. Lee, H. Lee and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks", arXiv preprint arXiv:1807.03888 (2018).

Liang, S., Y. Li and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks", arXiv preprint arXiv:1706.02690 (2017).

Neal, L., M. Olson, X. Fern, W.-K. Wong and F. Li, "Open set learning with counterfactual images", in "Proceedings of the European Conference on Computer Vision (ECCV)", pp. 613–628 (2018).

Oza, P. and V. M. Patel, "C2ae: Class conditioned auto-encoder for open-set recognition", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 2307–2316 (2019).

Ramachandran, P., B. Zoph and Q. V. Le, "Swish: a self-gated activation function", arXiv preprint arXiv:1710.05941 **7**, 1 (2017).

Scheirer, W. J., A. de Rezende Rocha, A. Sapkota and T. E. Boult, "Toward open set recognition", IEEE transactions on pattern analysis and machine intelligence **35**, 7, 1757–1772 (2012).

Shu, L., H. Xu and B. Liu, "Doc: Deep open classification of text documents", arXiv preprint arXiv:1709.08716 (2017).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", The journal of machine learning research **15**, 1, 1929–1958 (2014).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1–9 (2015).

Wikipedia, "Alexnet — Wikipedia, the free encyclopedia", `https://en.wikipedia.org/w/index.php?title=AlexNet&oldid=1013596377`, [Online; accessed 3-April-2021] (2021).