Automatic Facial Expression Recognition in Videos and still Images

MS COMPUTER SCIENCE
Hamza Khalid
MSCS-S19-019
2019-2021

DEPARTMENT OF COMPUTER SCIENCE
INFORMATION TECHNOLOGY THE SUPERIOR COLLEGE
LAHORE, PAKISTAN

Automatic Facial Expression Recognition in Videos and still Images

A thesis submitted in partial fulfillment of the requirements for the
Degree of Master of Science in
Computer Science.

Hamza Khalid

Dr. Arfan Jaffar

DECLARATION

This thesis is a summary of my original research work. Every attempt is taken to clearly indicate where others' contributions are involved, with adequate reference to the literature and acknowledgement of collaborative research and discussions. I further certify that this work is the result of my own investigations, except where sources are included, and that it is free of plagiarism.

Signature: …….
Hamza Khalid
Date: …….

The undersigned hereby certify that they have read and recommend the thesis entitled "**Automatic Facial Expression Recognition in Videos and still Images**" by **Hamza Khalid** for the degree of Master of Science in Computer Science.

_____
Dr. Arfan Jaffar, Thesis Supervisor

_____
Thesis Committee Member 1

_____
Thesis Committee Member 2

_____
Dr. Arfan Jaffar, Head of Department

Acknowledgements

First of all, In the name of Allah, who is most Merciful and most Beneficent.

I'd also like to express my gratitude to our supervisor, **Dr. Arffan Jaffer & Sir Fawad**, for his advice and encouragement to work hard and clever. While discussing the scope, implementation, and writing of this dissertation, I found him to be really helpful. His constructive criticism of my work has undoubtedly improved the structure of my thesis.

I thank God Almighty for providing me with all resources of every type so that I can put them to good use for the sake of humanity. May He continue to provide me with all of the resources and wisdom I need to continue assisting humanity.

Thank you
Hamza Khalid

DEDICATION

This thesis is dedicated to my dear family, teachers, and friends. Who always help me and in my hard times. Especially thanks to my parent who are stand with me and pray for me whenever I'm in problem.

Table of Contents

TABLE OF FIGURE

Abstract

In the realm of computer vision, facial expression recognition systems have sparked a lot of attention. For feature extraction, many suggested facial expression recognition (FER) systems use hand-crafted features; however, such methods often don't work very well for challenging scenarios. This thesis report builds upon the recent success of Convolution Neural Networks (CNNs) to classify images of human faces into six basic emotions (Happy, Fear, Sadness, Anger, Surprise, Neutral). Each layer of ConvNet extracts the features of objects like edges and corners, SVM and Softmax are used for the classification of different classes when I use both together we can easily recognize the objects. I will understand and analyse by performing different classification methods; Support Vector Machine (SVM), Softmax and Two Neural Network to examine the performance of facial expression recognition systems. I will also address classification of emotion expressed by the primary human subject in short video clips extracted from feature length movies.

1 Chapter

Introduction

Computer vision systems capable of recognizing human emotion have attracted much research interest in the recent past due to those potential applications such as customer attentive marketing, health monitoring and emotionally intelligent robotic interfaces [1]. In the light of the important role that facial expression plays in communicating emotion in humans, there has been substantial research interest in computer vision systems to recognize human emotion. For this purpose, many research papers published after mid 20's, explains the basic emotion of the human face and how we can recognize these features.

Some facial expressions are universally understood. Ekman et al. defined six universal facial expressions in a 1971 paper titled "Constants Across Cultures in the Face and Emotion": anger, disgust, fear, happiness, sadness, and surprise [2]. They are shown in Fig. 1 and are employed by computer vision researchers to identify modern face expressions. These emotions, as well as a seventh, neutral emotion, are presented for classification in research challenges such as Emotion Recognition in the Wild (EmotiW), Karolinska Directed Emotional Faces, and Kaggle's Facial Expression Recognition Challenge.In this project we classify images of human faces into six discrete emotion categories. Many established facial expression recognition (FER) systems use standard machine learning and hand crafted scenario features, which do not have significant performance in challenging unseen data [3]. Noting the success of CNNs in other domains such as speed recognition, object recognition, and object detection [4], our objective is to experiment primarily with basic CNN architectures to achieve acceptable results on existing data [5].

Understanding human emotion is an important area of research because the ability to recognise one's emotion can lead to a variety of opportunities and applications, ranging from friendlier human-PC associations to better focusing on promotional efforts to reaching a climax with enhanced correspondence among people, all of which can be achieved by improving each of us emotional Knowledge ("EQ"). While there are several approaches to study the recognition of human emotions, ranging from facial expression to body position to voice speed and tone, we will focus on just one aspect of this topic in this project: visual recognition of facial expression.

Recent academics are using the same emotions (Happy, Surprise, Sad, Fear, Angry and Neutral) to recognise face expression in computer vision or competitions, such as Kaggle's Facial

Expression Acknowledgement Challenge and Karolinska Directed Emotional Faces. The Kaggle's Facial Expression Recognition Test and the Karolinska Directed Emotional Faces (KDEF) datasets will be used in our investigation. Because of their size, the unstructured character of faces (in terms of facial orientation, ethnicity, age, and sex of the subjects), and the generally consistent distribution of the information across the six basic human emotions, we noticed these datasets being employed.

Figure 1: A sample of images from the data set, labelled with their corresponding emotions [2].

1.1 Objectives

1. We will train our system for following six basic facial expressions like sad, happy, surprise, Neutral, Fear and anger.
2. Performance analysis of various ML (Machine Learning) methods.
3. Our system will be capable of recognizing facial expression in still images and in video sequences, which would not be seen during training.

1.2 Overview

- **Chapter 1**: This chapter includes the introduction of the project, the importance of facial expression recognition (FER) and significance of facial expression.
- **Chapter 2**: This chapter includes the literature review of the project based upon the previous work relevant to our project.
- **Chapter 3**: This chapter includes, what are Support Vector Machine, Softmax and Two Layer Network? Why I used these different techniques (SVM, Softmax, Two Layer Network) in our project. Why I go toward Softmax and then Two Layer Network after SVM.
- **Chapter 4**: This chapter includes how I implement these techniques using CIFAR 10 Dataset and Karolinska Dataset. I also see which technique gave me better results among these techniques.
- **Chapter 5:** This chapter includes the table of the accuracies which were calculated during the used of different techniques (SVM, Softmax and Two Layer Network). In this chapter I also include the table of best learning rate and regularization strength of the best accuracies.
- **Chapter 6:** This chapter concludes my thesis and this section gives me clear view why Two Layer Network is better than Softmax and SVM. This chapter also includes the future work and what will be done in future to have better accuracies on this dataset.

2 Chapter

Literature Review

This chapter quickly reviews the relevant background research on facial expressions recognition, in which we focus during our project. This chapter also includes the different classification methods like a Support Vector Machine (SVM), Softmax, Neural Networks (NN), Deep Learning (DL) and Convolution Neural Network (ConvNets) and how these methods help us in our project. We also described why we used Two Neural Networks (NN) instead of Support Vector Machine (SVM) and Softmax in our project.

2.1 Psychological Framework

Significant research has been done in the field of human emotion perception [6, 7, 8], and the Facial Action Coding System (FACS) [9] has emerged as the primary psychological framework for characterising facial movement. FACS is a system that uses Action Units to classify human facial movements based on how they look on the face (AU). Each AU is one of 46 atomic components that make up observable facial movement or deformity; an expression is often made up of many AUs.

2.2 Area of Focus

The focus area of the project is human emotion with recognition in images and videos. Many projects try to recognize and coordinate appearances [10], yet most projects don't utilize Convolutional Neural Network (CNN) to extract features from still pictures. In this project we utilize the Support Vector Machine (SVM), Softmax and Convolution Neural Network.

2.3 Support Vector Machine (SVM)

We used a linear classifier trained using multi-class (more than one class, like our project, which has 6 classes, so our project problem is likewise multi-class) support vector machine loss as our baseline, which has the following score function. (Eq. 1):

$$f(x_i, W, b) = Wx_i + b. \qquad (1)$$

W is a C×D weight matrix, and b is a C×1 bias vector, where x_i is an image's pixel data flattened to a D×1 vector, W is a C×D weight matrix, and b is a C×1 bias vector. The function returns a C×1 vector of class scores, with C denoting the number of classes. We may interpret the linear classifier as how much an image fits the "template" for a class because the score for a class is the weighted sum of an image's pixel value. We utilise a loss function to quantify the class scores once they've been computed. Where the i-th loss uses the formula, how well the classifier performs. (Eq. 2):

$$L_i = \sum_{j \neq y_i} \quad max\ (0, s_j - s_{y_i} + \Delta) \qquad (2)$$

Where x_i is the correct class, and y_i is the correct class for x_i. When the score for class j is not at least lower than the score for the correct class y_i, the SVM loss will be non-zero. The value 1 is a regularly used and adopted value for Δ. We add an L2 regularisation term to the loss function to prevent the weights from taking on arbitrarily large values (Eq. 3):

$$L = \frac{1}{N} \sum_{i=1}^{N} \quad L_i + \curlywedge \sum_{j=1}^{C} \quad \sum_{k=1}^{D} \quad W_j^2, k \qquad (3)$$

Where W_j ,**k** is the weight matrix's (j,k) entry, and **λ** is a cross-validation-derived hyper-parameter. The purpose of training is to reduce the amount of data lost. Each weight and bias element is initialised with a Gaussian with a mean of zero and a small standard deviation. The derivative of the loss with respect to W and b is calculated at each iteration, and the parameters are updated using stochastic gradient descent.[11].

2.4 Softmax Classifier

It turns out that SVM is one of the two typically observed classifiers (Softmax, SVM) [12]. The other popular choice is the Softmax classifier, which has an alternate loss function. Unlike the SVM, which treats the output **f (**x_i**, W)** as (uncalibrated and possibly difficult to translate) scores for each class, the Softmax classifier produces a slightly more intuitive output (standardised class probabilities) and also has a probabilistic interpretation, which we will discuss shortly. The

function mapping **f(x_i;W) = W*x_i**remains unchanged in the Softmax classifier; however, we interpret these scores as unnormalized log probabilities for each class and restore the hinge loss with a **cross-entropy loss** of the type [13]:

$$L_i = -log\ log\ (\frac{e^{f_{yi}}}{\sum_j \blacksquare\blacksquare e^{f_j}}) \text{ or equivalently } L_{i=} - f_{yi} + log\ log\ \sum_j \blacksquare\blacksquare e^{f_j} \quad (1)$$

Where the documentation f_j refers to the j-th component of the vector of class scores f. Eventually, the total loss for the dataset is equal to the mean of Li's entire training illustration plus a regularization term R. (W). The Softmax function is defined as $f_j(z) = \frac{e^{z_j}}{\sum_k \quad e^{z_k}}$. It confines a vector of real-valued scores (in z) to a vector of values between zero and one that sum to one. If you're seeing it for the first time, the entire cross-entropy loss that includes the Softmax function may appear unappealing, but it's usually simple to motivate.

The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as:

$$H(p,q) = -\sum_j \quad p(x) log\ q(x) \quad (2)$$

The Softmax classifier has thus limited the cross-entropy between the estimated class probabilities (**q** = $\frac{e^{f_{yi}}}{\sum_j \blacksquare\blacksquare e^{f_j}}$ as observed above in eq. 1) and the "true" distribution, which in this understanding is where all likelihood masses is in the correct class (i.e. p=[0,… 1,… ,0]) contains a single 1 at the y_i-th position).

Looking at the expression, we see that:

$$P(y_i | x_i; W) = \frac{e^{f_{yi}}}{\sum_j \blacksquare\blacksquare e^{f_j}} \quad (3)$$

Can be expressed as the (standardised) probability assigned to the right label y_i given the picture x_i, with W as the parameter. Remember that the Softmax classifier solves the scores inside the yield (output) vector f as unnormalized log probabilities to see this. The (unnormalized) probabilities are obtained by exponentiating these values, and the normalisation is achieved by

dividing them so that the probabilities are all equal to one. This viewpoint has the advantage of allowing us to translate the regularisation term R (W) in the entire loss function as starting from a Gaussian prior to the weight matrix W. These understandings are provided to assist your instincts., The comprehensive explanation of this derivation, on the other hand, is beyond the scope of this thesis.

2.5 SVM vs Softmax

An illustration of the differences between the Softmax and SVM classifiers can be found here:

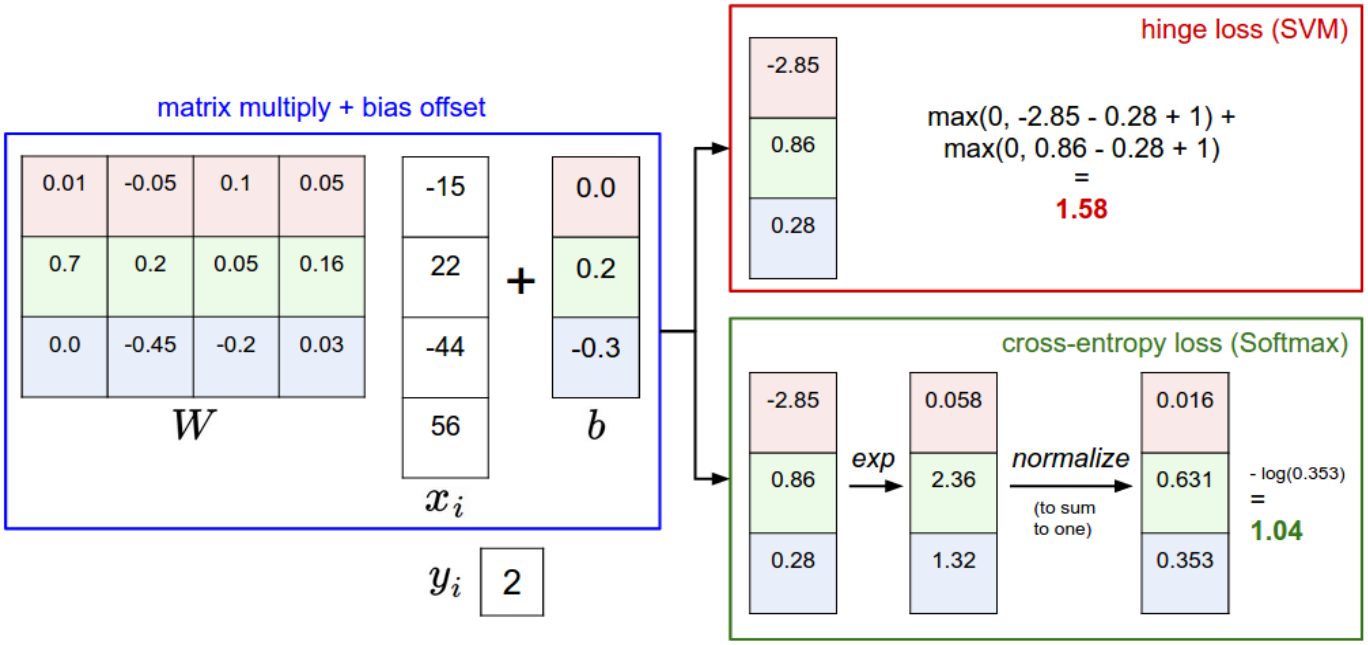


Figure 2: Differentiate between Softmax and Support Vector Machine [13].

I used SVM and Softmax for the classification of objects. Fig. 2 clearly explains that the function of Softmax has less loss as compared to SVM. That's why I moved toward Softmax as our classifier instead of SVM.

2.6 Neural Networks (NN)

A Neural Network (NN) is a mathematical model that is inspired by the way natural sensory systems, such as the brain, process information and data [14]. As shown in Fig. 3 the cell body of the brain gets inputs from Dendrites and gives an output at Axon. In neural networks the same procedure applies, the inputs are going into cell body and cell body give output as shown in Fig. 3.

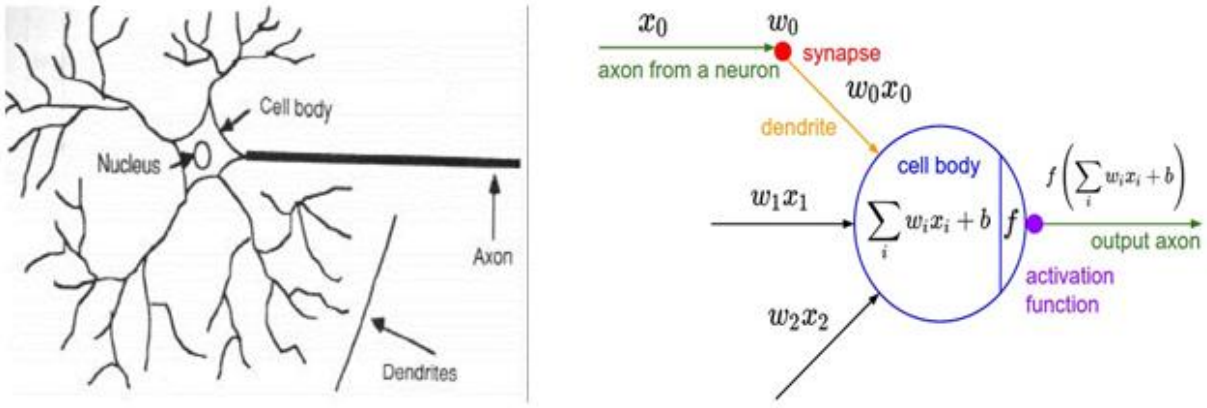


Figure 3: The Neuron Model and The Mathematical Model of Neuron [15].

2.6.1 Activation Functions

The activation function (also known as non-linearity) is a mathematical operation that takes a binary value and executes a specified mathematical operation on it. In practice, you can employ a variety of activation functions. e.g.

2.6.1.1.1 Sigmoid:

The Sigmoid function is commonly used since it has a range of 0 to 1. When we need to anticipate the likelihood of an outcome, we apply this model. [16].

2.6.1.1.2

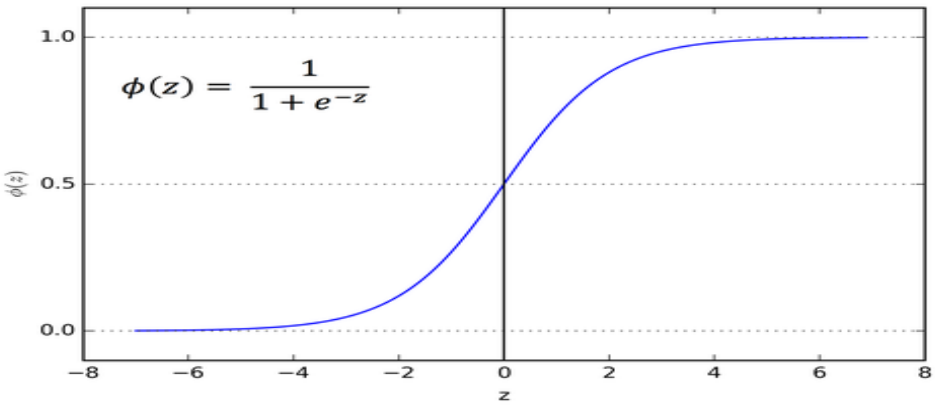


Figure 4: Sigmoid Function [16].

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

ReLU:

This is the one of the most common activation functions used in Convolution Neural Network and Deep Learning. As you can see in the Fig. 5 ReLU is half rectified and its range from (0 to infinity) [16].

The main problem of ReLU is that it discards the negative values become zero which means that ReLU ignoring the negative edges (inside edges) of the object due to which training of the object cannot be done properly.

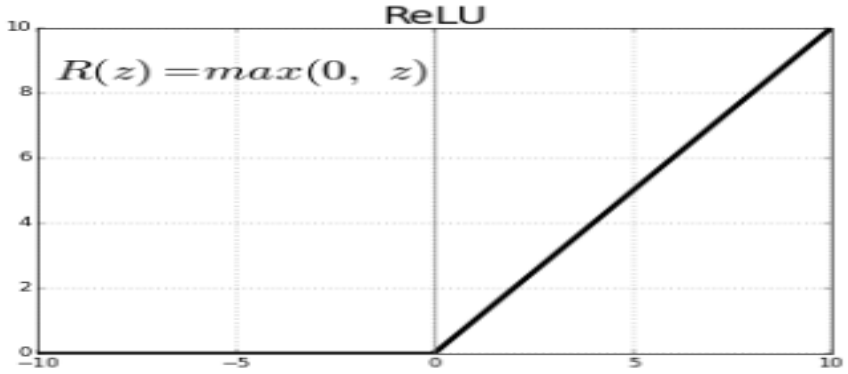


Figure 5: ReLU Function [16].

$$f(x) = max(0, x)$$

2.7 Deep Neural Network

Deep neural networks are different from neural networks due to their depth. Multiple hidden layers are used for data manipulation. In simple words, if any neural network has more than one hidden layer in between the input layer and output layer that is called Deep Neural Networks (DNN). A simplest Deep Neural Networks (DNN) is shown in Fig. 6.

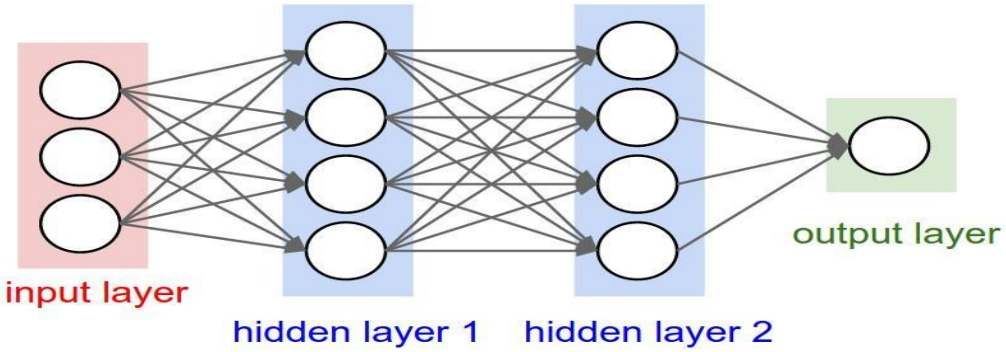


Figure 6: Deep Neural Networks Model [12].

2.8 Convolution Neural Network (ConvNets)

Convolutional Neural Networks are similar to the traditional Neural Networks discussed before in this section in that they are made up of neurons with learnable weights and biases. Every neuron receives a small number of inputs, does a dot product, and then follows it with non-linearity. The entire system still communicates a single differentiable scoring function, which goes from raw picture pixels on one side to class scores on the other. Regardless, they contain a loss function on the last (fully-connected) layer (e.g. SVM/Softmax), and all of the guidelines we provided for conventional Neural Networks still apply[12].

So, what is different now? ConvNet architectures are built with the explicit assumption that the source of information (inputs) is images, allowing us to incorporate specific properties into the architecture. As a result, the forward function becomes more capable of execution, and the number of parameters in the networks is drastically reduced [1].

2.8.1 Layers

Each layer of a ConvNet changes one volume of activation to another through a differentiable function [12]. A basic ConvNet is a grouping (sequence) of layers, and each layer of a ConvNet

changes one volume of activation to another through a differentiable function. Convolutional Layer, Pooling Layer, and Fully-Connected Layer are the three types of layers we use to create ConvNet architecture (exactly as seen in regular Neural Networks). These layers will be stacked to form a complete ConvNet architecture. The following are the main phases in creating Convolution Neural Networks (ConvNets) [2]:

- INPUT [32x32x3] will save the raw pixel values of the images, in this case an image having a width of 32 pixels, a height of 32 pixels, and three colour channels (R, G, and B).
- The CONV layer will process the yield (output) of neurons connected with a nearby region in the input, with each neuron computing a dot product between their weights and a small region of the input volume with which they are linked. If we choose to use 12 filters, this may result in a volume of [32x32x12].
- The RELU layer will use an elementwise actuation function, such as, the max (0, x) max (0, x) thresholding at zero. The volume's size remains unchanged as a result of this ([32x32x12]).
- The POOL layer will do a down sampling operation along the spatial measurements (width and height), resulting in volume, for examle[16x16x12].
- The FC (completely connected) layer will register the class scores, resulting in a volume of size [1x1x10], where each of the ten values corresponds to a class score, such as among CIFAR-10's ten categories. Similarly, in traditional Neural Networks, every neuron in this layer will be associated with each of the numbers in the preceding volume, as the name implies.

2.9 Conclusion

In this chapter, first I explored Support Vector Machine (SVM), Softmax on random data and manipulated our parameters repeatedly to get an accurate score and saturated loss. After those same classification methods Support Vector Machine (SVM), Softmax and Convolution Neural Networks (ConvNets) apply on a random dataset (Cifar-10) to get an accurate score. Then same classification methods apply on Karolinska Dataset to get an accurate score.

3 Chapter

Methodology

3.1 Methodology

This chapter briefly explains Support Vector Machine (SVM), Softmax and Two Layer Network. This section discusses how these techniques will be used in the thesis and also explains why I choose Softmax, Two Layer Network after SVM.

3.1.1 Support Vector Machine

The loss function can be defined in a number of different ways. The Multiclass Support Vector Machine (SVM) loss is a commonly used loss that will be developed in this thesis. The SVM loss is set up so that the correct class for each image must have a score that is greater than the incorrect classes by a given margin **Δ** [3].

Review the given pixels of the image $\boldsymbol{x_i}$, as well as the label $\boldsymbol{y_i}$, which shows the right class index, for the $\boldsymbol{i-th}$ case. The scoring function uses the pixels to solve the vector $\boldsymbol{f(xi, W)}$ of class scores, which will be replaced by s in the future (short for scores). The $\boldsymbol{j-th}$ component, for example, is the $\boldsymbol{sj = f(xi, W)j}$ score for the $\boldsymbol{j-th}$ class. The loss of the $\boldsymbol{i-th}$ Multiclass SVM formalized as given below:

$$\boldsymbol{Li = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta)} \qquad (1)$$

3.1.2 Softmax

For the Softmax classifier, this thesis builds a fully vectorized loss function. Then, for its analytical gradient, use the fully vectorized expression. The numerical gradient is then used to verify that our analytical gradient is correct [2]. There should be very little difference between the analytical and numerical gradients. The validation set is then used to fine-tune the learning

rate and regularisation strength in order to enhance training and validation accuracy. Then use SGD to reduce the loss. Finally, imagine the final weights you've learned.

The Softmax equation we utilise is:

$$\boldsymbol{L_i = -\log\left(\frac{e^{f_{yi}}}{\sum_j e^{f_j}}\right)} \text{ or equivalently } \boldsymbol{L_{i=} - f_{yi} + \log \sum_j e^{f_j}} \quad (2)$$

3.1.3 Two Layer Network

To achieve classification, use a neural network with fully linked layers. In this architecture, all input images (raw pixels) flow through fully connected layers initially, then through rectifier neural networks (ReLU), fully connected layers again, and finally through Softmax classifier to obtain probabilities for each class [3] [4]. Then, for a two-layer fully connected neural network, compute the loss and gradients. Perform the forward pass and compute the class scores for the input in machine learning, then pass through the ReLU activation function and apply Softmax classifiers to extract neural features for each class. The backward pass is then computed. We compute the derivatives of the weights and biases during this process, which will give us the right class score.

4 Chapter

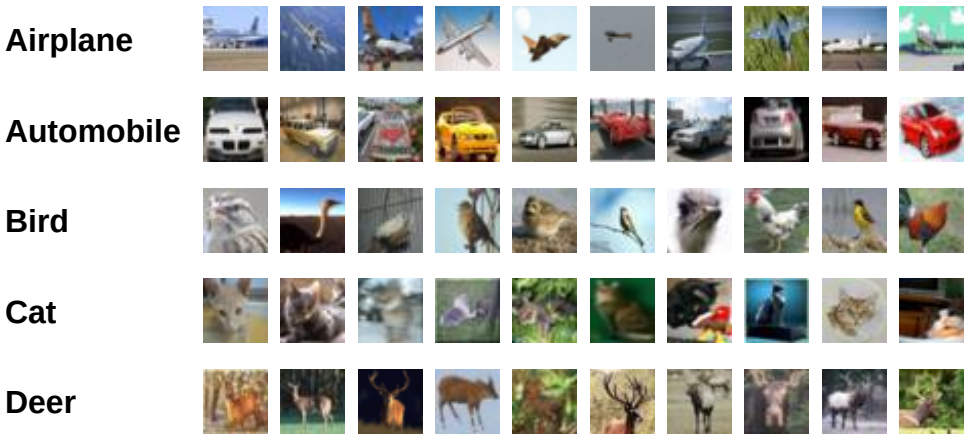
Implementation

This section includes how I implemented different techniques (Softmax, SVM, and Two Layer Network) on CIFAR 10 dataset to get results. My thesis also applies the same techniques on the Karolinska dataset to recognize human emotions. My thesis used Support Vector Machine (SVM), Softmax and two layer network.I compared the results and examine which gave better results.

4.1 CIFAR 10 Dataset

The CIFAR-10 collection contains 60000 32x32 colour images divided into ten categories (Airplane, Automobile, Bird, Cat, Dog, Deer, Frog, Horse, Ship, and Truck), each with 6000 images. [5] There are 50000 training images and ten thousand test images.

Each of the 10000 images in the dataset is divided into five training batches and one test batch. The test batch contains 1000 images from each class that were chosen at random. The rest of the images are in random order in the training batches, but some training batches may include more images from one class than another. The training batches contain exactly 5000 images from each class between them.I performed these techniques on CIFAR-10 Dataset because it is very close to my thesis research. The CIFAR-10 project predict the object in the images while my thesis research predict the emotions of human faces. That's why first I explained how CIFAR-10 Dataset works.



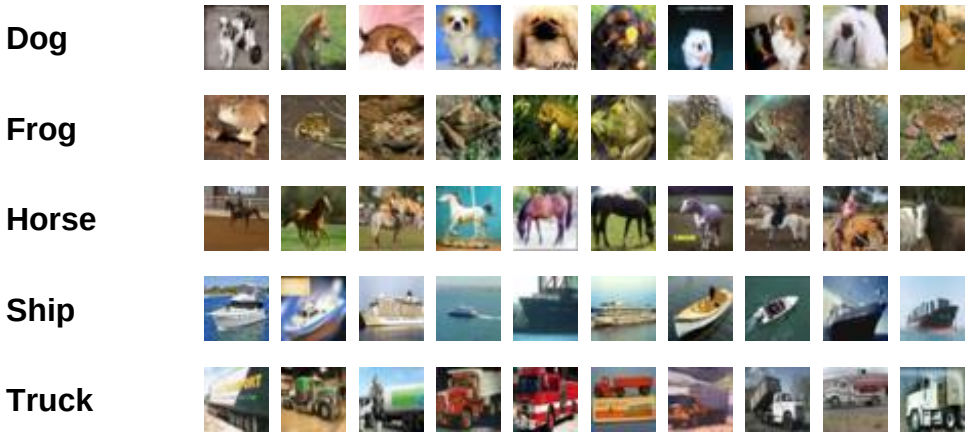


Figure 1: The CIFAR 10 Dataset [16].

4.1.1 Support Vector Machine

The results of CIFAR 10 Dataset using Support Vector Machine (SVM) technique are:

First of all visualize a few examples of training images from each class of CIFAR 10 dataset using SVM.

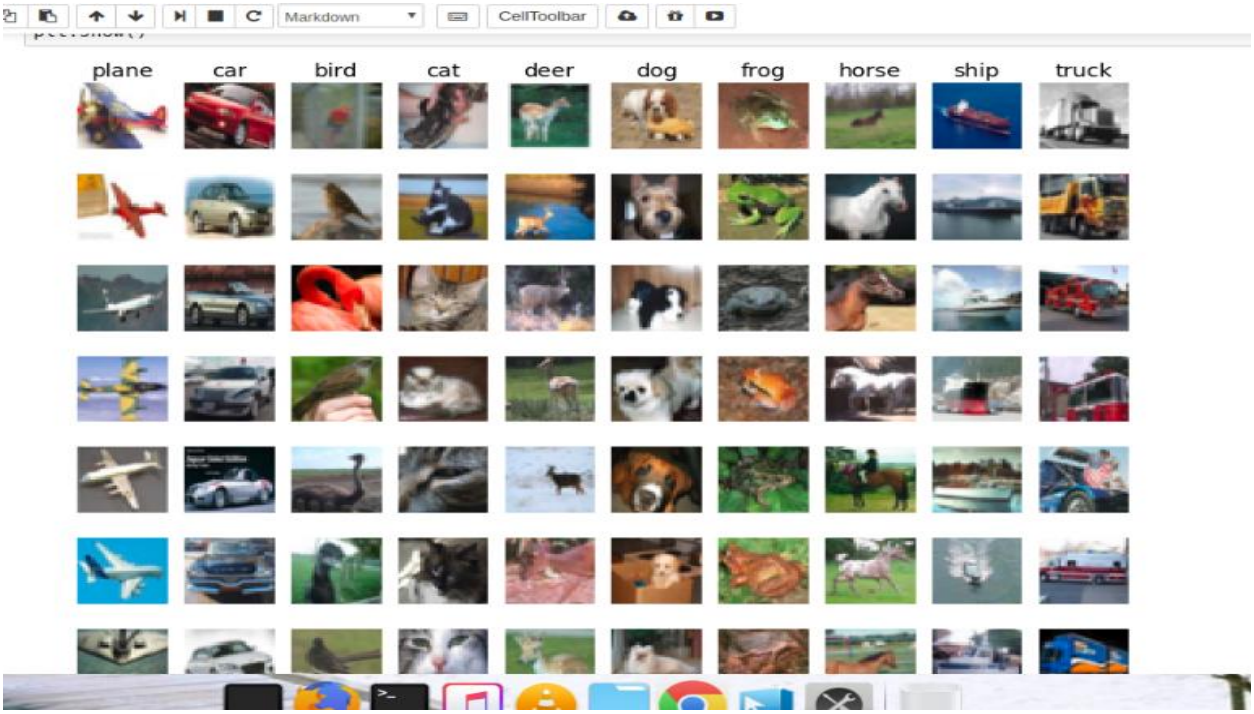


Figure 2: Visualize CIFAR 10 Dataset using SVM.

Then split data into training, validation and testing sets as shown in Fig. 9. I had also created a small development set as a subset of the training data. From 50000 images, 49000 images used for training, 1000 images used for validation, 1000 images used for testing and 500 images used for development. After that I had done pre-processing and reshape image data into rows as shown in Fig. 9.

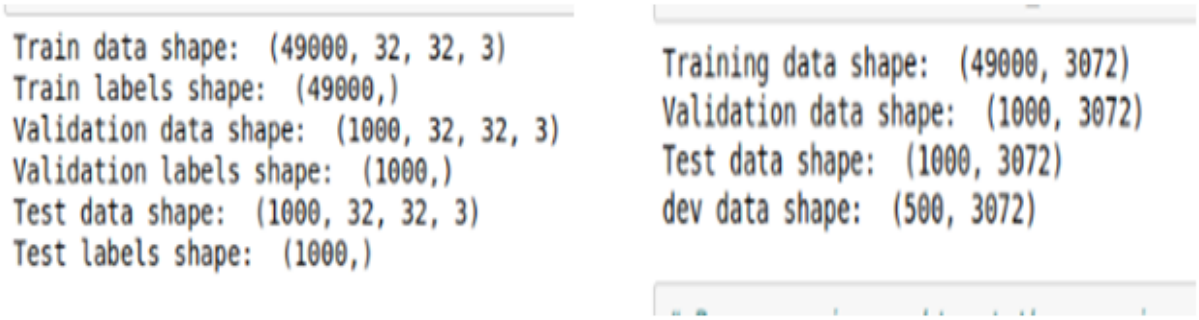


Figure 3: Split Data and Pre-process Data into rows.

Then subtract the mean image and compute image mean based on the training data. Only print a few values as shown in Fig. 10.

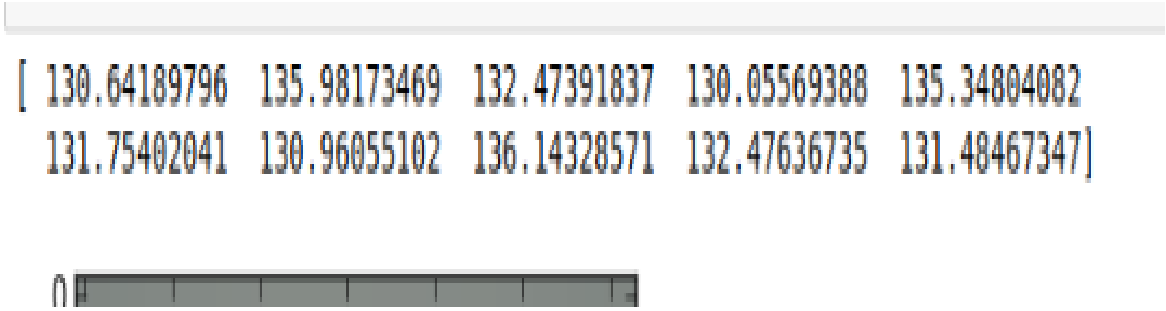


Figure 4: Random 10 values from Training Data.

The naive implementation (in non-optimization way) of the loss without using Gradient is **9.545684**.

Then computed the Gradient for Support Vector Machine (SVM) cost function and compared the result with and without using Gradient in SVM. First compare the numeric estimate to the gradient that had computed. The numerical number and analytic number approximately should match each other.

```
numerical: -12.299436 analytic: -12.287224, relative error: 4.966761e-04
numerical: 1.354178 analytic: 1.354178, relative error: 2.745930e-10
numerical: 16.517718 analytic: 16.517718, relative error: 1.228103e-11
numerical: -21.221938 analytic: -21.240247, relative error: 4.311648e-04
numerical: 15.310430 analytic: 15.310430, relative error: 4.539642e-11
numerical: 15.278771 analytic: 15.278771, relative error: 2.985502e-11
numerical: -8.841190 analytic: -8.841190, relative error: 3.555038e-11
numerical: -1.629165 analytic: -1.629165, relative error: 2.017678e-10
numerical: 0.967198 analytic: 0.967198, relative error: 4.667693e-11
numerical: -26.982726 analytic: -26.982726, relative error: 8.550405e-12
numerical: 0.026381 analytic: 0.026381, relative error: 5.325220e-09
numerical: -0.247837 analytic: -0.247837, relative error: 1.096465e-09
numerical: 36.126077 analytic: 36.126077, relative error: 1.623992e-11
numerical: -46.790424 analytic: -46.790424, relative error: 6.011237e-13
numerical: -0.331808 analytic: -0.331808, relative error: 7.417470e-10
numerical: -1.766356 analytic: -1.769633, relative error: 9.268180e-04
numerical: 5.861484 analytic: 5.861484, relative error: 1.272327e-11
numerical: 2.106834 analytic: 2.106834, relative error: 7.986596e-11
numerical: 27.983016 analytic: 27.983016, relative error: 1.408090e-11
numerical: 6.122041 analytic: 6.122041, relative error: 3.845909e-11
```

Figure 5: Comparison of Analytic and Numeric computed Gradient.

Then computed the vectorized loss and compared it with naive loss that computed earlier. Difference zero will prove that the results are correct.

```
Naive loss: 9.545684e+00 computed in 0.082503s
Vectorized loss: 9.545684e+00 computed in 0.067814s
difference: -0.000000
```

Figure 6: Comparison of Vectorized and Naive Loss.

Then I had efficient expressions for the loss, the vectorized loss and my naive loss matched. Therefore, ready to do Stochastic Gradient Descent (SGD) to minimize the loss. After doing SGD, minimum loss will be:

```
iteration 0 / 1500: loss 784.557444
iteration 100 / 1500: loss 286.365773
iteration 200 / 1500: loss 108.610242
iteration 300 / 1500: loss 42.428520
iteration 400 / 1500: loss 18.517332
iteration 500 / 1500: loss 9.753863
iteration 600 / 1500: loss 7.536585
iteration 700 / 1500: loss 5.834442
iteration 800 / 1500: loss 5.986458
iteration 900 / 1500: loss 5.812746
iteration 1000 / 1500: loss 4.793104
iteration 1100 / 1500: loss 5.453767
iteration 1200 / 1500: loss 5.202023
iteration 1300 / 1500: loss 5.320397
iteration 1400 / 1500: loss 5.035379
That took 4.177443s
```

Figure 7: Minimum Loss after SGD using SVM.

Then with the help of graph, I had examined how our loss decreased with respect to increase in the iterations.

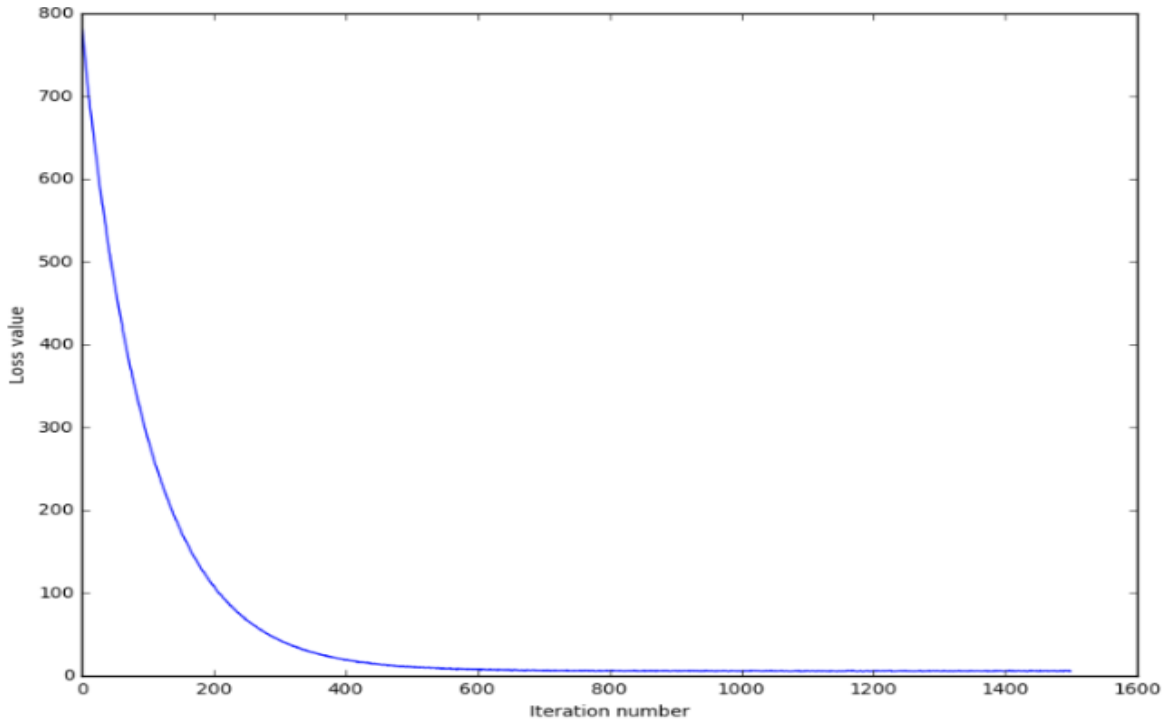


Figure 8: : Loss with respect to Iterations in SVM.

Then predicted and evaluated the performance on both training and validation set using linear Support Vector Machine (SVM).

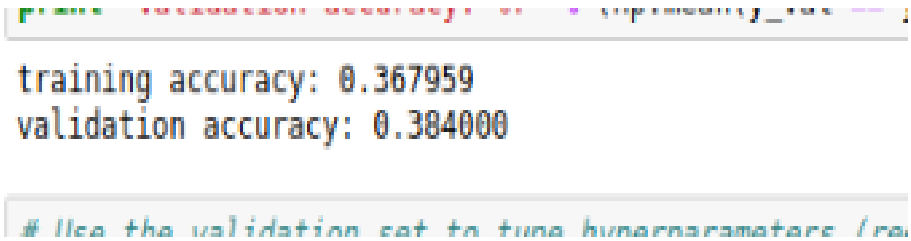


Figure 9: Training and Validation Accuracy.

Then I used validation set to tune hyper parameters (regularization strength and learning rate). Regularization is used to avoid the overfitting. Overfitting means if the algorithm does not fit the training data well, I add new data points.

```
lr 1.000000e-08 reg 1.000000e+04 train accuracy: 0.378551 val accuracy: 0.388000
lr 1.000000e-08 reg 2.000000e+04 train accuracy: 0.387102 val accuracy: 0.392000
lr 1.000000e-08 reg 3.000000e+04 train accuracy: 0.385796 val accuracy: 0.386000
lr 1.000000e-08 reg 4.000000e+04 train accuracy: 0.382163 val accuracy: 0.390000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.376388 val accuracy: 0.389000
lr 1.000000e-08 reg 6.000000e+04 train accuracy: 0.371776 val accuracy: 0.382000
lr 1.000000e-08 reg 7.000000e+04 train accuracy: 0.369980 val accuracy: 0.378000
lr 1.000000e-08 reg 8.000000e+04 train accuracy: 0.367878 val accuracy: 0.382000
lr 1.000000e-08 reg 1.000000e+05 train accuracy: 0.363265 val accuracy: 0.384000
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.393041 val accuracy: 0.402000
lr 1.000000e-07 reg 2.000000e+04 train accuracy: 0.384653 val accuracy: 0.384000
lr 1.000000e-07 reg 3.000000e+04 train accuracy: 0.379224 val accuracy: 0.388000
lr 1.000000e-07 reg 4.000000e+04 train accuracy: 0.370122 val accuracy: 0.382000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.372918 val accuracy: 0.381000
lr 1.000000e-07 reg 6.000000e+04 train accuracy: 0.367857 val accuracy: 0.376000
lr 1.000000e-07 reg 7.000000e+04 train accuracy: 0.366531 val accuracy: 0.368000
lr 1.000000e-07 reg 8.000000e+04 train accuracy: 0.358592 val accuracy: 0.373000
lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.352347 val accuracy: 0.371000
lr 2.000000e-07 reg 1.000000e+04 train accuracy: 0.389959 val accuracy: 0.404000
lr 2.000000e-07 reg 2.000000e+04 train accuracy: 0.370347 val accuracy: 0.400000
lr 2.000000e-07 reg 3.000000e+04 train accuracy: 0.373061 val accuracy: 0.380000
lr 2.000000e-07 reg 4.000000e+04 train accuracy: 0.355429 val accuracy: 0.365000
lr 2.000000e-07 reg 5.000000e+04 train accuracy: 0.360592 val accuracy: 0.372000
lr 2.000000e-07 reg 6.000000e+04 train accuracy: 0.357939 val accuracy: 0.379000
lr 2.000000e-07 reg 7.000000e+04 train accuracy: 0.366143 val accuracy: 0.368000
lr 2.000000e-07 reg 8.000000e+04 train accuracy: 0.338653 val accuracy: 0.356000
lr 2.000000e-07 reg 1.000000e+05 train accuracy: 0.340980 val accuracy: 0.359000
best validation accuracy achieved during cross-validation: 0.404000
```

Figure 10: Validation Accuracy during Cross-Validation in SVM.

At last, visualized learned best weights for each class as shown in (Fig. 17). Learned weights depend upon the choice of your learning rate and regularization strength. This may not be nice to look at.

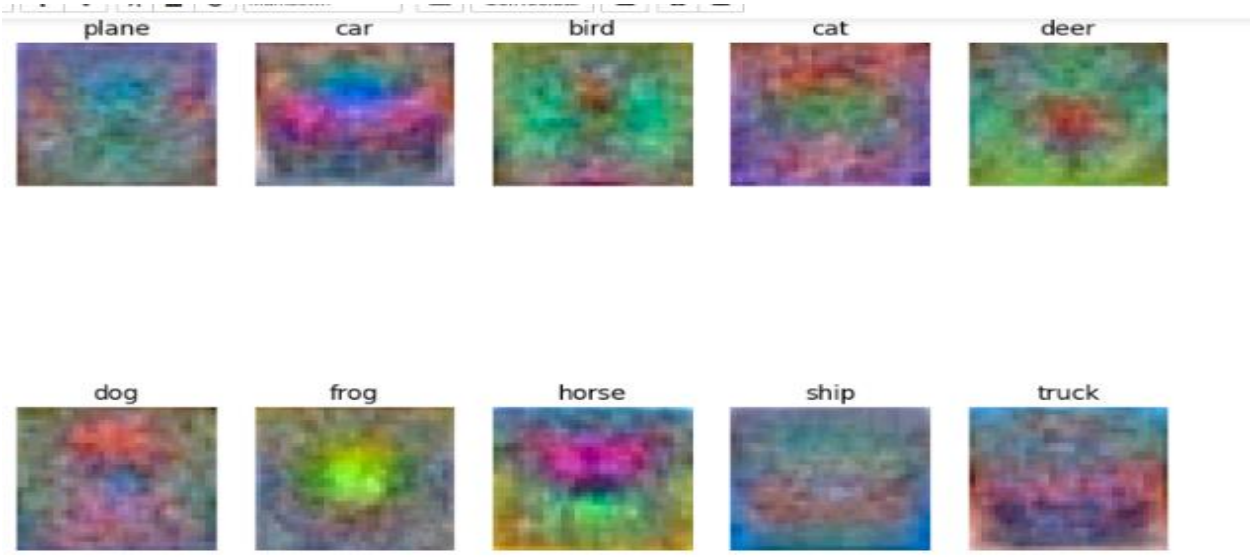


Figure 11: Visualization of SVM technique on CIFAR 10 Dataset.

4.1.2 Softmax

After Support Vector Machine (SVM), Softmax technique has to apply on same CIFAR 10 dataset and compares the results with SVM.

Then split data into training, validation and testing sets as shown in Fig. 18. I had also created a small development set as a subset of the training data. From 50000 images, 49000 images used for training, 1000 images used for validation, 1000 images used for testing and 500 images used for development. After that we had done pre-processing and reshape image data into rows as shown in Fig. 18.

```
Train data shape:  (49000, 32, 32, 3)
Train labels shape:  (49000,)
Validation data shape:  (1000, 32, 32, 3)
Validation labels shape:  (1000,)
Test data shape:  (1000, 32, 32, 3)
Test labels shape:  (1000,)
```

```
Training data shape:  (49000, 3072)
Validation data shape:  (1000, 3072)
Test data shape:  (1000, 3072)
dev data shape:  (500, 3072)
```

Figure 12: Split Data and Pre-process Data into rows.

Then implement the naive Softmax loss function and sanity check. Rough Sanity check of **– log (1/number of total classes) = – log (1/10) = – log (0.1) = 2.302585** and loss should be close to this check.

```
loss: 2.322626
sanity check: 2.302585
```

Figure 13: Comparison of Loss and sanity Check.

Then I implemented the Softmax naive loss and calculated the numerical and analytical gradient. Numerical gradient should be very close to analytical gradient.

```
numerical: -1.382913 analytic: -1.382913, relative error: 7.807133e-08
numerical: -0.024374 analytic: -0.024374, relative error: 1.511043e-06
numerical: -0.441034 analytic: -0.441034, relative error: 1.652191e-07
numerical: -3.874115 analytic: -3.874115, relative error: 2.442677e-08
numerical: 0.714279 analytic: 0.714279, relative error: 2.734362e-08
numerical: -0.155600 analytic: -0.155600, relative error: 6.440278e-07
numerical: 0.410697 analytic: 0.410696, relative error: 6.382315e-08
numerical: 2.546535 analytic: 2.546535, relative error: 3.188101e-08
numerical: 1.859369 analytic: 1.859369, relative error: 4.754036e-08
numerical: -0.525134 analytic: -0.525134, relative error: 8.101723e-08
numerical: 4.262350 analytic: 4.262350, relative error: 3.039005e-08
numerical: -2.387426 analytic: -2.387426, relative error: 5.443991e-09
numerical: 0.380257 analytic: 0.380257, relative error: 2.611075e-08
numerical: 1.159809 analytic: 1.159809, relative error: 7.543713e-10
numerical: 0.720450 analytic: 0.720450, relative error: 7.822430e-08
numerical: -1.642496 analytic: -1.642496, relative error: 2.194444e-08
numerical: -2.357591 analytic: -2.357591, relative error: 5.039976e-09
numerical: 1.637918 analytic: 1.637918, relative error: 3.663940e-08
numerical: -3.595502 analytic: -3.595502, relative error: 4.022752e-09
numerical: -2.641395 analytic: -2.641395, relative error: 5.721639e-09
```

Figure 14: Comparison of Analytic and Numeric computed Gradient.

Then computed the vectorized loss and compared it with our naive loss I computed earlier. Difference zero of loss and the gradient will prove that our results are correct.

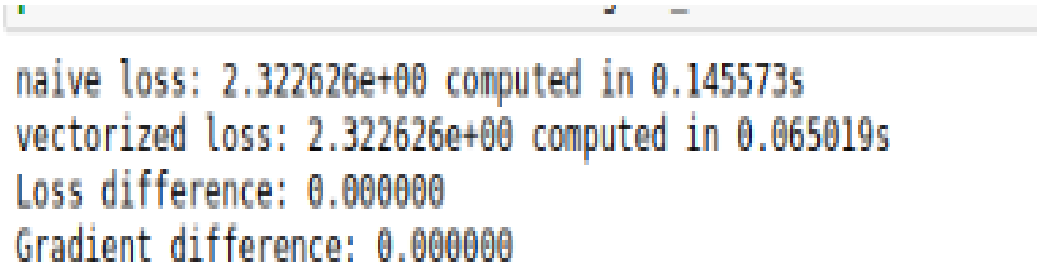


Figure 15: Comparison of Vectorized and Naive Loss.

The loss, the vectorized loss, and my naive loss matches all have efficient expressions. As a result, be prepared to use Stochastic Gradient Descent (SGD) to reduce the loss. The minimum loss after implementing SGD is:

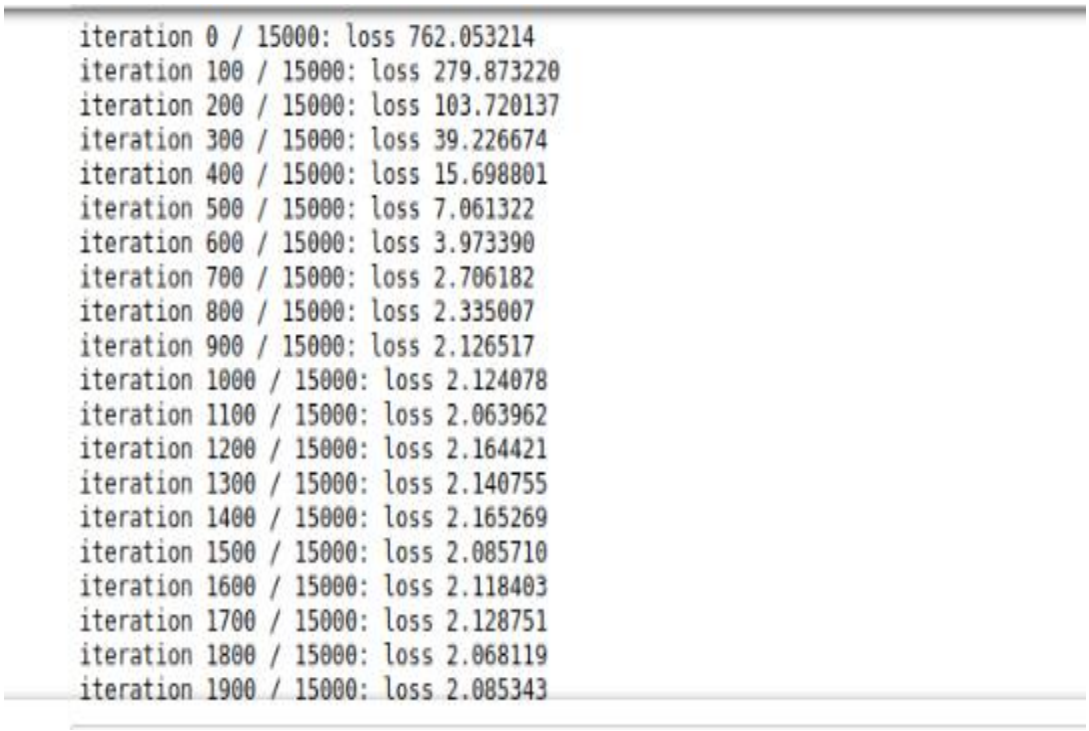


Figure 16: Minimum Loss after SGD using Softmax.

With the help of graph, I had examined how our loss decreased with respect to increase in the iterations.

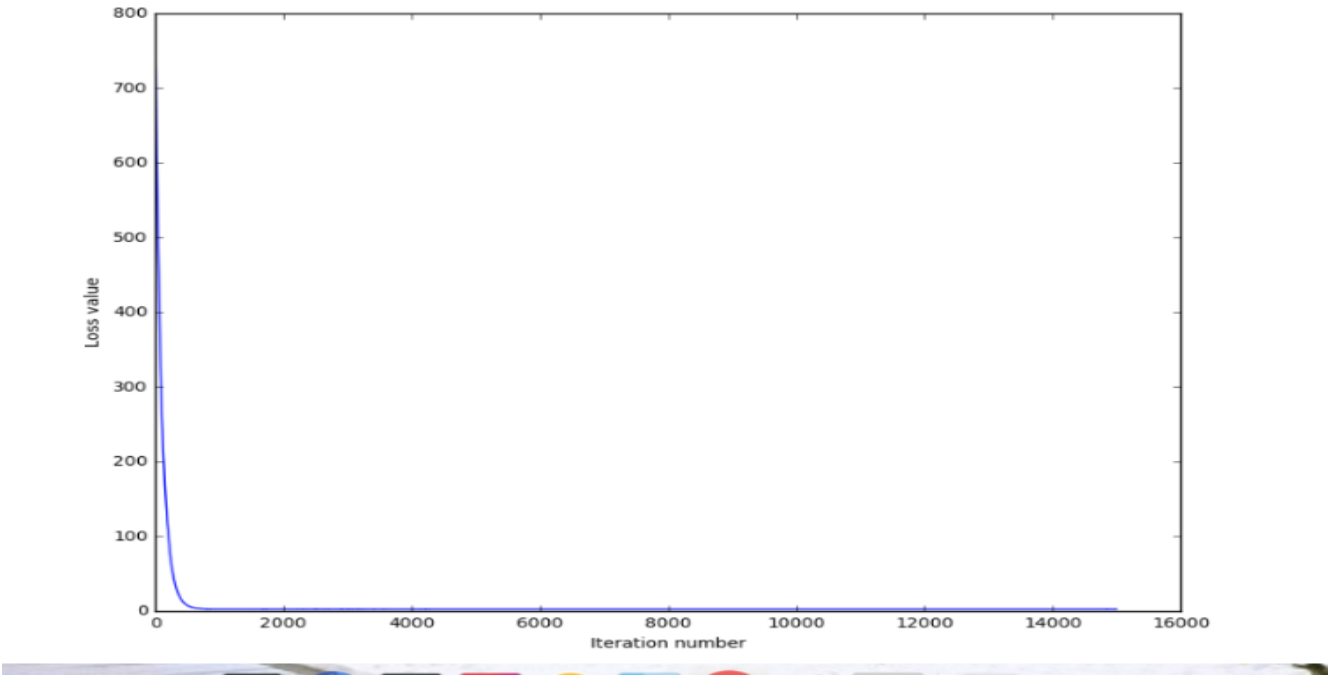


Figure 17: Loss with respect to Iterations in Softmax.

Then I used validation set to tune hyper parameters (regularization strength and learning rate). Regularization is used to avoid the overfitting.

```
lr 1.000000e-10 reg 1.000000e-03 train accuracy: 0.095837 val accuracy: 0.091000
lr 1.000000e-10 reg 1.000000e-02 train accuracy: 0.103224 val accuracy: 0.096000
lr 1.000000e-10 reg 1.000000e-01 train accuracy: 0.081612 val accuracy: 0.080000
lr 1.000000e-10 reg 1.000000e+00 train accuracy: 0.121653 val accuracy: 0.126000
lr 1.000000e-10 reg 1.000000e+01 train accuracy: 0.086857 val accuracy: 0.080000
lr 1.000000e-10 reg 1.000000e+02 train accuracy: 0.099510 val accuracy: 0.091000
lr 1.000000e-10 reg 1.000000e+03 train accuracy: 0.108347 val accuracy: 0.105000
lr 1.000000e-10 reg 1.000000e+04 train accuracy: 0.098694 val accuracy: 0.074000
lr 1.000000e-10 reg 1.000000e+05 train accuracy: 0.102122 val accuracy: 0.081000
lr 1.000000e-10 reg 1.000000e+06 train accuracy: 0.086571 val accuracy: 0.083000
lr 1.668101e-08 reg 1.000000e-03 train accuracy: 0.120694 val accuracy: 0.117000
lr 1.668101e-08 reg 1.000000e-02 train accuracy: 0.076082 val accuracy: 0.084000
lr 1.668101e-08 reg 1.000000e-01 train accuracy: 0.109796 val accuracy: 0.105000
lr 1.668101e-08 reg 1.000000e+00 train accuracy: 0.084551 val accuracy: 0.083000
lr 1.668101e-08 reg 1.000000e+01 train accuracy: 0.095816 val accuracy: 0.101000
lr 1.668101e-08 reg 1.000000e+02 train accuracy: 0.118612 val accuracy: 0.107000
lr 1.668101e-08 reg 1.000000e+03 train accuracy: 0.108347 val accuracy: 0.121000
lr 1.668101e-08 reg 1.000000e+04 train accuracy: 0.100347 val accuracy: 0.087000
lr 1.668101e-08 reg 1.000000e+05 train accuracy: 0.100245 val accuracy: 0.092000
lr 1.668101e-08 reg 1.000000e+06 train accuracy: 0.124469 val accuracy: 0.110000
lr 2.782559e-06 reg 1.000000e-03 train accuracy: 0.267653 val accuracy: 0.262000
lr 2.782559e-06 reg 1.000000e-02 train accuracy: 0.273898 val accuracy: 0.278000
lr 2.782559e-06 reg 1.000000e-01 train accuracy: 0.268184 val accuracy: 0.305000
lr 2.782559e-06 reg 1.000000e+00 train accuracy: 0.279592 val accuracy: 0.284000
lr 2.782559e-06 reg 1.000000e+01 train accuracy: 0.267449 val accuracy: 0.271000
lr 2.782559e-06 reg 1.000000e+02 train accuracy: 0.275347 val accuracy: 0.266000
lr 2.782559e-06 reg 1.000000e+03 train accuracy: 0.285796 val accuracy: 0.286000
lr 2.782559e-06 reg 1.000000e+04 train accuracy: 0.343408 val accuracy: 0.366000
lr 2.782559e-06 reg 1.000000e+05 train accuracy: 0.265551 val accuracy: 0.281000
lr 2.782559e-06 reg 1.000000e+06 train accuracy: 0.079388 val accuracy: 0.068000
lr 4.641589e-04 reg 1.000000e-03 train accuracy: 0.219143 val accuracy: 0.227000
lr 4.641589e-04 reg 1.000000e-02 train accuracy: 0.240184 val accuracy: 0.244000
lr 4.641589e-04 reg 1.000000e-01 train accuracy: 0.228143 val accuracy: 0.228000
lr 4.641589e-04 reg 1.000000e+00 train accuracy: 0.260184 val accuracy: 0.261000
lr 4.641589e-04 reg 1.000000e+01 train accuracy: 0.262980 val accuracy: 0.270000
lr 4.641589e-04 reg 1.000000e+02 train accuracy: 0.223592 val accuracy: 0.211000
lr 4.641589e-04 reg 1.000000e+03 train accuracy: 0.151163 val accuracy: 0.172000
```

Figure 18: Validation Accuracy during Cross-Validation in Softmax.

At last, visualized learned best weights for each class as shown in Fig. 25. Learned weights depend upon the choice of your learning rate and regularization strength.

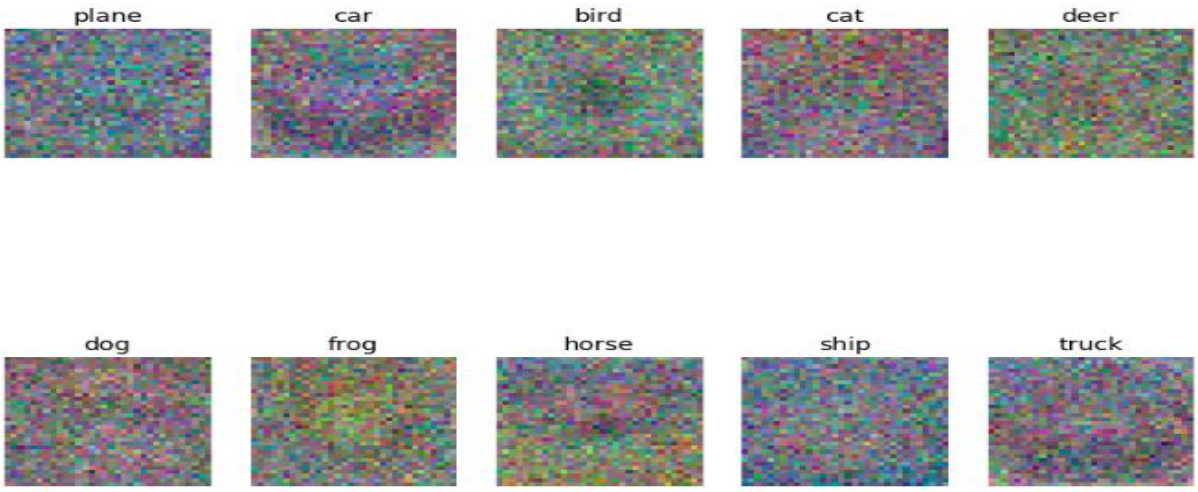


Figure 19: Visualization of Softmax technique on CIFAR 10 Dataset.

4.1.3 Two Layer Network

Then, to conduct classification, create a neural network with fully connected layers and test it on the CIFAR 10 dataset.

The identical loss function that I had constructed for the SVM and Softmax was employed in this strategy. It takes the data, weights it, and calculates the class scores, loss, and parameter gradients.

First, implement the forward pass, which computes the scores for all inputs using the weights and biases. The disparities between the correct and incorrect scores are really minor.

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
3.68027209255e-08
```

Figure 20: Comparison of Your scores and correct scores.

Then I had implemented the second part which computed the data and regularization loss. This difference is very small.

```
Difference between your loss and correct loss:
1.79412040779e-13
```

Figure 21: Result of Your loss and Correct loss after computing the data and regularization loss.

The rest of the function was then implemented. The gradient of the loss was determined with respect to the variables W1, b1, W2, and b2. To do so, I used a numeric gradient checker to examine your backward pass implementation. If an implementation is accurate, the difference between numeric and analytic gradients for W1, W2, b1, and b2 should be smaller than 1e-8.

```
W1 max relative error: 3.669858e-09
W2 max relative error: 3.440708e-09
b2 max relative error: 4.447677e-11
b1 max relative error: 2.738421e-09
```

Figure 22: Checking of Backward Pass.

I used Stochastic Gradient Descent (SGD) to train the toy model, as I had done with the SVM and Softmax classifiers. This should be quite similar to the SVM and Softmax classifiers' training procedures. While the network was being trained, the training procedure did prediction on a regular basis to maintain track of accuracy over time. The loss in training should be less than **0.2**.

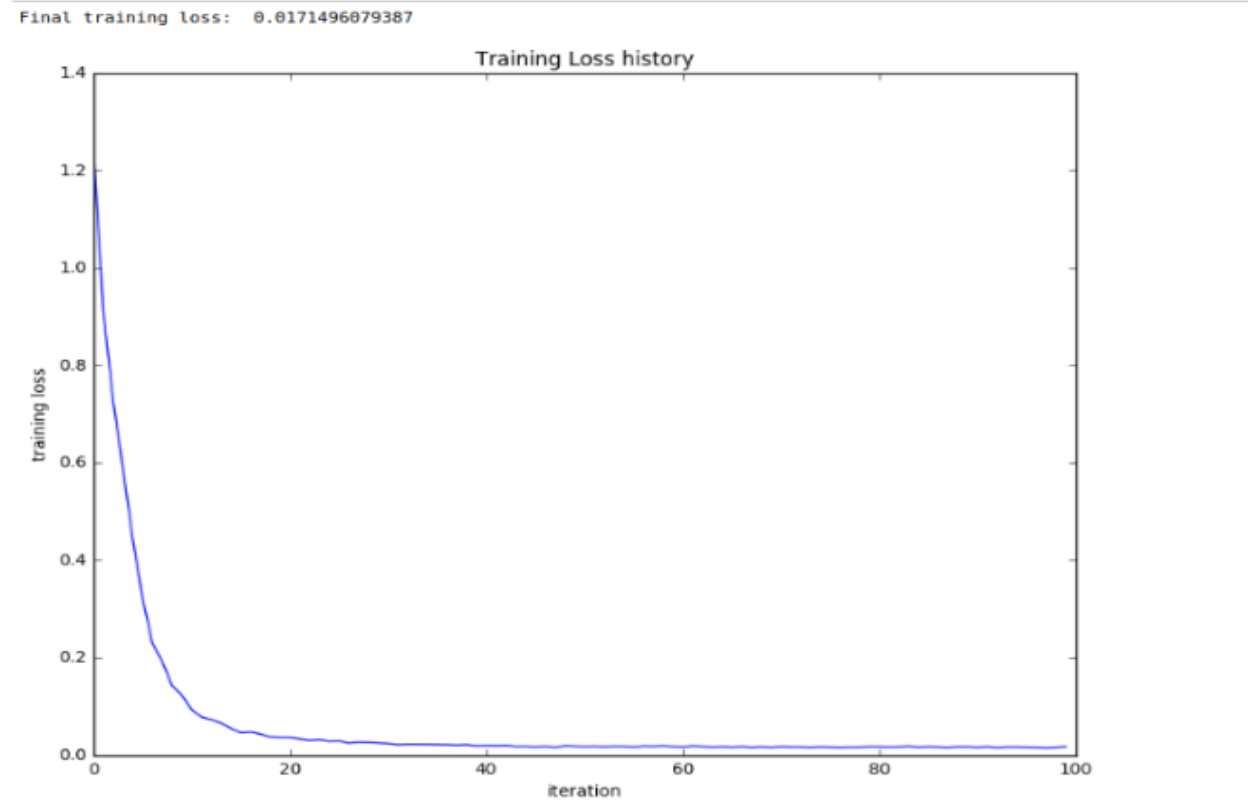


Figure 23: Final training loss and loss history.

After that, I built a two-layer network that passed the CIFAR 10 data's gradient checks. On an actual dataset, I can use it to train a classifier. Then divide the data into three groups: training, validation, and testing. From a total of 50000 images, 49000 were used for training, 1000 for validation, and 1000 for testing. After that, I pre-processed the image data and reshaped it into rows as shown in Fig. 30.

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

Figure 24: Split Data.

Use SGD with momentum to train a network. In addition, as optimization progressed, the learning rate was changed with an exponential learning rate; after each epoch, the learning rate was reduced by multiplying it by a decay rate.

```
iteration 0 / 1000: loss 2.302954
iteration 100 / 1000: loss 2.302550
iteration 200 / 1000: loss 2.297648
iteration 300 / 1000: loss 2.259602
iteration 400 / 1000: loss 2.204170
iteration 500 / 1000: loss 2.118565
iteration 600 / 1000: loss 2.051535
iteration 700 / 1000: loss 1.988466
iteration 800 / 1000: loss 2.006591
iteration 900 / 1000: loss 1.951473
Validation accuracy:  0.287
```

Figure 25: Validation Accuracy.

On the validation set, the validation accuracy of roughly 29% with the default values is not good. Then, using a different strategy, visualise the weights that were learned in the neural network's first layer. When the first layer weights of most neural networks trained on visual data are displayed, they usually show some obvious structure.

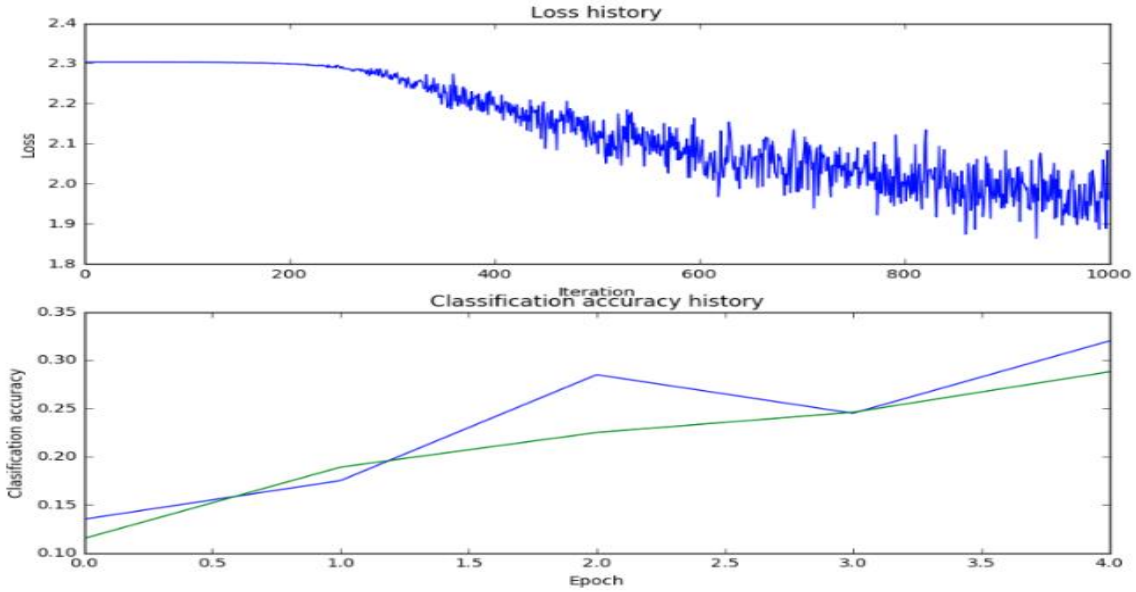


Figure 26: Plot of loss function and validation accuracies of the network.

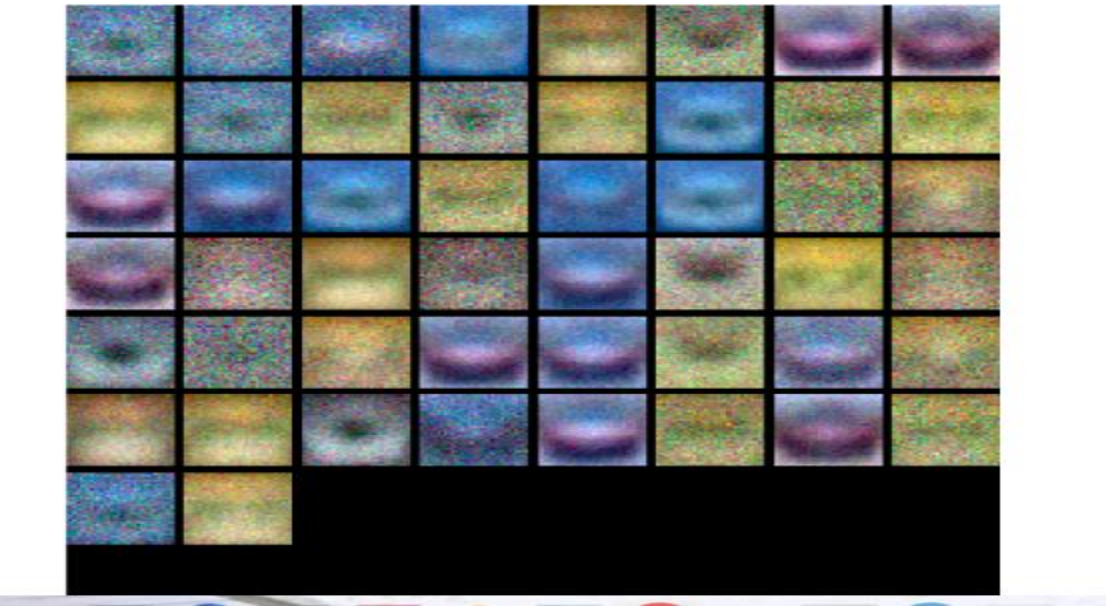


Figure 27: Visualization the weights of the network.

The validation set can then be used to fine-tune hyper parameters (hidden layer size, number of training epochs, regularisation strength and learning rate). On the validation set, you should aim for a classification accuracy of more than **48 percent**.

```
lr 1.000000e-03 reg 4.000000e-01 train accuracy: 0.516408 val accuracy: 0.480000
lr 1.000000e-03 reg 5.000000e-01 train accuracy: 0.530061 val accuracy: 0.479000
lr 1.000000e-03 reg 6.000000e-01 train accuracy: 0.520592 val accuracy: 0.497000
lr 1.000000e-02 reg 4.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-02 reg 5.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-02 reg 6.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.497000
```

Figure 28: Validation Accuracy during Cross-Validation in Two Layer Network.

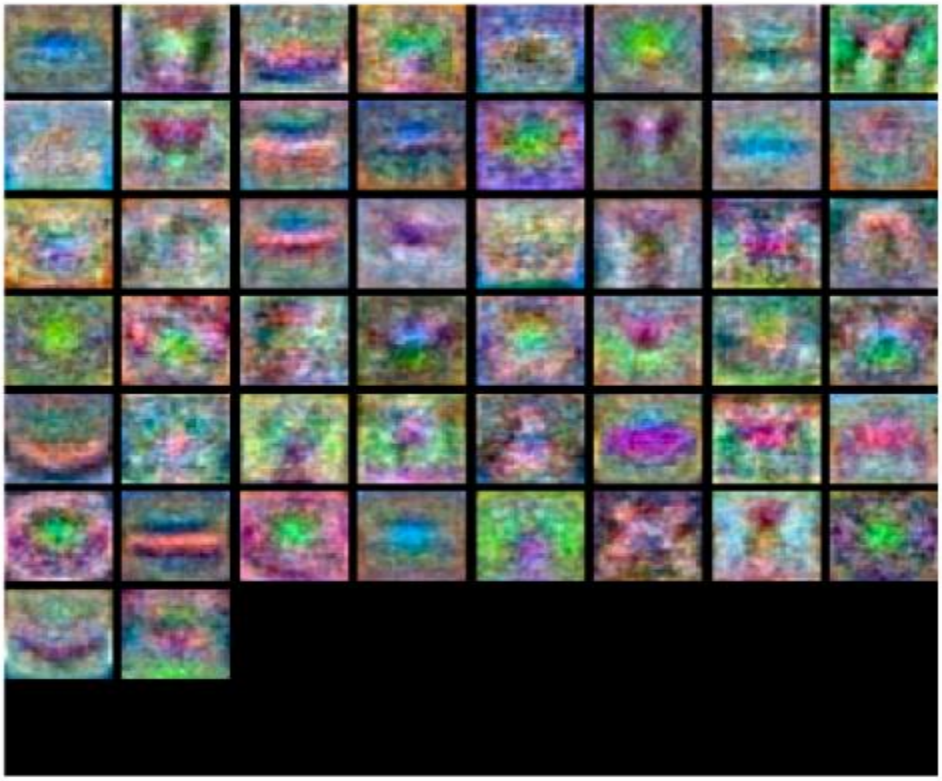


Figure 29: Visualization the weights of the best network.

Then I had shown the loss function and train accuracies graph of the best network and compare it with the previous network graph.

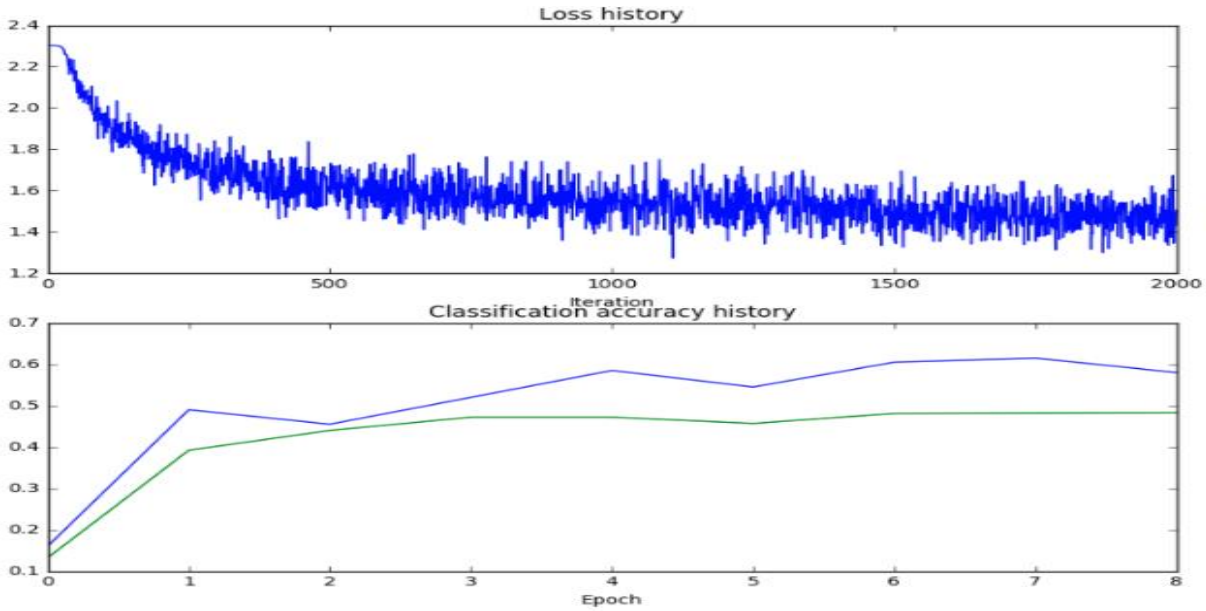


Figure 30: Plot of loss function and validation accuracies of the best network.

After experiments, this should evaluate your final trained network on the test set. The test accuracy of our final trained network is **49.6 %.**

4.2 Karolinska Dataset

Machine learning was one of the learning approaches investigated in the project (SVM, Softmax, and Two Layer Network). To do so, we'll need a dataset of human face expressions to display our findings. These conditions are almost perfectly met by Karolinska Directed Emotional Faces (KDEF). For KDEF photos to have the same colour and format as those from the CIFAR 10 dataset, I'll need to do some data pre-processing. Some of the key features of KDEF are provided below [4]:

- 4830 images.
- Image Size: 562x762 (32-bit RGB).
- 69 people, each with one of the seven different emotional expressions, taken twice from five different angles for each expression.
- Ethnicity, race, sex, and gender are all represented.
- The seven major emotions are spread evenly.

4.2.1 Data Pre-processing

As shown in Fig. 37, we scaled the original 562 × 762 images to 127 x 94 images for KDEF. The system's memory use is reduced as a result of image resizing. The time it took to train, validate, and test the images was reduced by reducing their size. Then took mean image of the whole dataset and subtract from each image of the dataset to normalize our dataset to zero centered (compact data around zero axis in such way mean our data become around to zero). If I don’t do pre-processing of data I cannot get right results because our testing data scattered on the graph on such axis where I had not train our system, because of that my system will not make right predication. At the end flatten each image into a single row.

Figure 31: Original image from KDEF and resize.

4.2.2 Support Vector Machine

First of all, we had visualized a few examples of training images from each class of the Karolinska dataset.

Figure 32: Visualization of Karolinska Dataset using SVM.

As shown in Fig. 39, the Karolinska Dataset divides data into training, validation, and testing sets. As a subset of the training data, I had also produced a tiny development set. 4130 images were used for training, 700 images for validation, 700 images for testing, and 175 images for development from a total of 4830 images. Following that, I pre-processed and altered picture data as shown in Fig. 39.

```
Train data shape:  (4130, 94, 127, 3)
Train labels shape:  (4130,)
Validation data shape:  (700, 94, 127, 3)
Validation labels shape:  (700,)
Test data shape:  (700, 94, 127, 3)
Test labels shape:  (700,)
```

```
Training data shape:  (4130, 35814)
Validation data shape:  (700, 35814)
Test data shape:  (700, 35814)
dev data shape:  (175, 35814)
```

Figure 33: Split Data and Pre-process Data into rows.

Then subtracted the mean image and computed image mean based on the training data. Only print random values as shown in Fig. 40.

```
[ 153.75447942  137.20242131  110.28232446  153.77893462  137.22615012
  110.30677966  153.82130751  137.26731235  110.34358354  153.91961259]
```

Figure 34: Random 10 values from Training Data.

The naive implementation of the loss without using Gradient is **6.017706.**

Then computed the Gradient for Support Vector Machine (SVM) cost function and compared the result with and without using Gradient in SVM. The same method is applied there for gradient as apply in the CIFAR-10 Dataset. First compared the numeric estimated to the gradient that I had computed. The numerical number and analytic number should approximately match each other.

```
numerical: -6.110943 analytic: -6.110943, relative error: 1.288677e-12
numerical: -1.577956 analytic: -1.577956, relative error: 2.801561e-11
numerical: -5.445972 analytic: -5.445972, relative error: 1.354903e-11
numerical: 1.681088 analytic: 1.681088, relative error: 1.285771e-11
numerical: -0.919628 analytic: -0.919628, relative error: 1.986070e-10
numerical: -1.707965 analytic: -1.707965, relative error: 2.934285e-12
numerical: 13.371943 analytic: 13.371943, relative error: 1.592909e-12
numerical: -0.184631 analytic: -0.184631, relative error: 1.967746e-10
numerical: 4.926135 analytic: 4.926135, relative error: 3.614441e-11
numerical: 8.652306 analytic: 8.652306, relative error: 4.577673e-12
numerical: -3.069185 analytic: -3.069185, relative error: 1.863189e-11
numerical: 25.473035 analytic: 25.473035, relative error: 5.703121e-12
numerical: -4.021056 analytic: -4.021056, relative error: 1.597042e-11
numerical: 3.875397 analytic: 3.875397, relative error: 7.185258e-12
numerical: 4.477381 analytic: 4.477381, relative error: 1.252043e-11
numerical: -0.957381 analytic: -0.957381, relative error: 6.756064e-11
numerical: -11.948906 analytic: -11.948906, relative error: 7.927068e-12
numerical: 3.865664 analytic: 3.865664, relative error: 1.001459e-11
numerical: -0.300992 analytic: -0.300992, relative error: 4.739521e-10
numerical: -0.261586 analytic: -0.261586, relative error: 2.050931e-10
```

Figure 35: Comparison of Analytic and Numeric computed Gradient of SVM in Karolinska Dataset.

Then computed the vectorized loss and compared it with my naive loss I computed earlier. Difference zero will prove that our results are correct.

```
Naive loss: 6.017706e+00 computed in 0.159310s
Vectorized loss: 6.017706e+00 computed in 0.034621s
difference: -0.000000
```

Figure 36: Comparison of Vectorized and Naive Loss.

Then, I had efficient expressions for the loss, the vectorized loss and my naive loss matched. Therefore, ready to do Stochastic Gradient Descent (SGD) to minimize the loss. After I had SGD, minimum loss will be:

```
iteration 0 / 9500: loss 6292.165606
iteration 100 / 9500: loss 2304.842339
iteration 200 / 9500: loss 847.288208
iteration 300 / 9500: loss 312.749732
iteration 400 / 9500: loss 116.075517
iteration 500 / 9500: loss 44.986358
iteration 600 / 9500: loss 18.144336
iteration 700 / 9500: loss 9.095490
iteration 800 / 9500: loss 5.401862
iteration 900 / 9500: loss 4.121265
iteration 1000 / 9500: loss 3.612846
iteration 1100 / 9500: loss 3.271982
iteration 1200 / 9500: loss 3.270613
iteration 1300 / 9500: loss 3.633522
iteration 1400 / 9500: loss 3.690108
iteration 1500 / 9500: loss 3.235800
iteration 1600 / 9500: loss 3.438510
iteration 1700 / 9500: loss 3.502993
iteration 1800 / 9500: loss 3.097304
iteration 1900 / 9500: loss 3.349605
iteration 2000 / 9500: loss 3.245668
iteration 2100 / 9500: loss 3.225355
iteration 2200 / 9500: loss 3.335975
iteration 2300 / 9500: loss 3.389907
iteration 2400 / 9500: loss 3.689403
iteration 2500 / 9500: loss 2.977156
iteration 2600 / 9500: loss 3.170630
iteration 2700 / 9500: loss 3.237508
iteration 2800 / 9500: loss 3.320230
iteration 2900 / 9500: loss 3.262945
iteration 3000 / 9500: loss 3.151384
iteration 3100 / 9500: loss 3.477557
iteration 3200 / 9500: loss 3.320709
iteration 3300 / 9500: loss 3.086989
iteration 3400 / 9500: loss 3.370823
iteration 3500 / 9500: loss 3.218083
iteration 3600 / 9500: loss 3.547677
iteration 3700 / 9500: loss 3.335660
iteration 3800 / 9500: loss 3.342358
iteration 3900 / 9500: loss 3.197915
iteration 4000 / 9500: loss 3.174433
iteration 4800 / 9500: loss 3.352132
iteration 4900 / 9500: loss 3.150328
iteration 5000 / 9500: loss 3.370334
iteration 5100 / 9500: loss 3.517245
iteration 5200 / 9500: loss 3.204289
iteration 5300 / 9500: loss 3.285019
iteration 5400 / 9500: loss 3.168728
iteration 5500 / 9500: loss 3.255751
iteration 5600 / 9500: loss 3.655289
iteration 5700 / 9500: loss 3.204211
iteration 5800 / 9500: loss 3.388051
iteration 5900 / 9500: loss 3.045504
iteration 6000 / 9500: loss 3.280222
iteration 6100 / 9500: loss 3.104841
iteration 6200 / 9500: loss 3.308157
iteration 6300 / 9500: loss 3.114628
iteration 6400 / 9500: loss 3.164370
iteration 6500 / 9500: loss 3.378360
iteration 6600 / 9500: loss 3.572884
iteration 6700 / 9500: loss 3.462225
iteration 6800 / 9500: loss 3.294936
iteration 6900 / 9500: loss 3.503194
iteration 7000 / 9500: loss 3.383082
iteration 7100 / 9500: loss 3.474910
iteration 7200 / 9500: loss 3.343828
iteration 7300 / 9500: loss 3.359551
iteration 7400 / 9500: loss 3.426276
iteration 7500 / 9500: loss 3.077965
iteration 7600 / 9500: loss 3.739136
iteration 7700 / 9500: loss 3.722494
iteration 7800 / 9500: loss 3.290685
iteration 7900 / 9500: loss 3.420646
iteration 8000 / 9500: loss 3.108934
iteration 8100 / 9500: loss 3.178706
iteration 8200 / 9500: loss 3.492587
iteration 8300 / 9500: loss 3.542477
iteration 8400 / 9500: loss 3.339350
iteration 8500 / 9500: loss 3.130757
iteration 8600 / 9500: loss 3.251851
iteration 8700 / 9500: loss 3.399047
iteration 8800 / 9500: loss 3.358115
iteration 8900 / 9500: loss 3.260508
iteration 9000 / 9500: loss 3.462579
iteration 5400 / 9500: loss 3.168728
iteration 5500 / 9500: loss 3.255751
iteration 5600 / 9500: loss 3.655289
iteration 5700 / 9500: loss 3.204211
iteration 5800 / 9500: loss 3.388051
iteration 5900 / 9500: loss 3.045504
iteration 6000 / 9500: loss 3.280222
iteration 6100 / 9500: loss 3.104841
iteration 6200 / 9500: loss 3.308157
iteration 6300 / 9500: loss 3.114628
iteration 6400 / 9500: loss 3.164370
iteration 6500 / 9500: loss 3.378360
iteration 6600 / 9500: loss 3.572884
iteration 6700 / 9500: loss 3.462225
iteration 6800 / 9500: loss 3.294936
iteration 6900 / 9500: loss 3.503194
iteration 7000 / 9500: loss 3.383082
iteration 7100 / 9500: loss 3.474910
iteration 7200 / 9500: loss 3.343828
iteration 7300 / 9500: loss 3.359551
iteration 7400 / 9500: loss 3.426276
iteration 7500 / 9500: loss 3.077965
iteration 7600 / 9500: loss 3.739136
iteration 7700 / 9500: loss 3.722494
iteration 7800 / 9500: loss 3.290685
iteration 7900 / 9500: loss 3.420646
iteration 8000 / 9500: loss 3.108934
iteration 8100 / 9500: loss 3.178706
iteration 8200 / 9500: loss 3.492587
iteration 8300 / 9500: loss 3.542477
iteration 8400 / 9500: loss 3.339350
iteration 8500 / 9500: loss 3.130757
iteration 8600 / 9500: loss 3.251851
iteration 8700 / 9500: loss 3.399047
iteration 8800 / 9500: loss 3.358115
iteration 8900 / 9500: loss 3.260508
iteration 9000 / 9500: loss 3.462579
iteration 9100 / 9500: loss 3.539852
iteration 9200 / 9500: loss 3.347484
iteration 9300 / 9500: loss 3.417820
iteration 9400 / 9500: loss 3.374161
That took 320.990743s
```

Figure 37: Minimum Loss after SGD using SVM in Karolinska Dataset.

The graph had shown that how our loss decreased with respect to increase in the iterations.

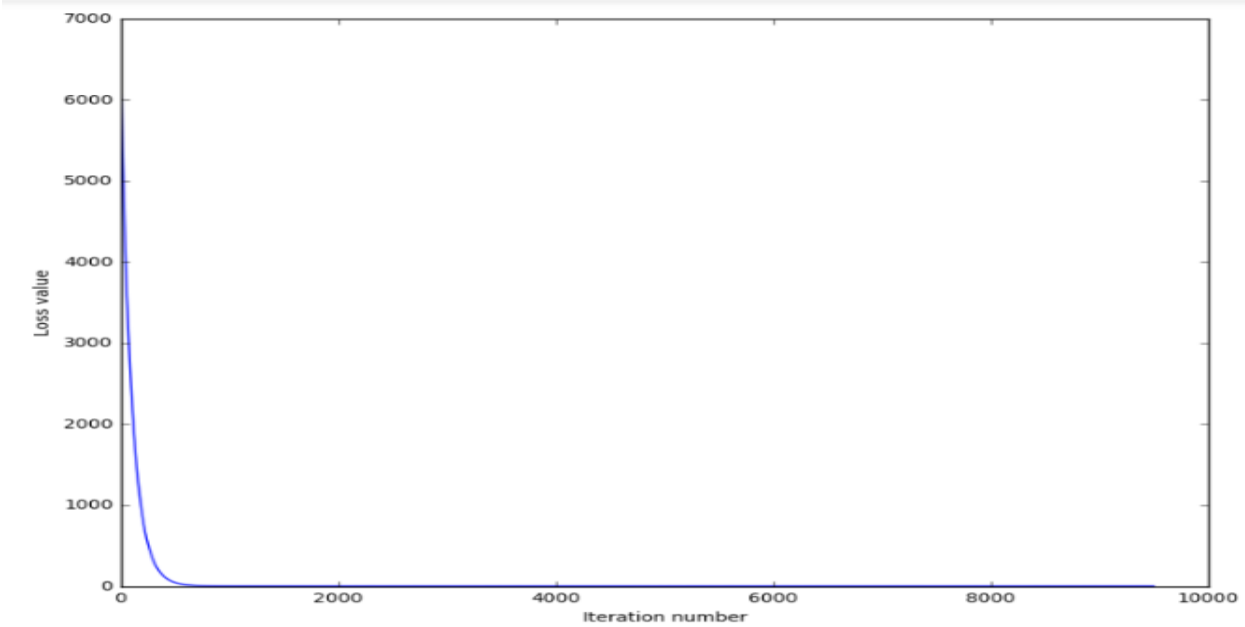


Figure 38: Loss with respect to Iterations in SVM of Karolinska Dataset.

Then predicted and evaluated the performance on both training and validation set using linear Support Vector Machine (SVM). In fig.45 the both accuracies values are approximately near to each other which shows that we implemented correct SVM function.

```
training accuracy: 0.605569
validation accuracy: 0.608571
```

Figure 39: Training and Validation Accuracy of Karolinska Dataset.

Then used validation set to tune hyper parameters (regularization strength and learning rate). Regularization is used to avoid the overfitting. I used those hyper parameters (regularization strength and learning rate) which gave us maximum training and validation accuracy.

```
lr 1.000000e-10 reg 1.000000e+04 train accuracy: 0.148184 val accuracy: 0.151429
lr 1.000000e-10 reg 1.000000e+05 train accuracy: 0.157385 val accuracy: 0.165714
lr 1.000000e-10 reg 1.000000e+06 train accuracy: 0.156174 val accuracy: 0.147143
lr 1.668101e-08 reg 1.000000e-03 train accuracy: 0.384746 val accuracy: 0.360000
lr 1.668101e-08 reg 1.000000e-02 train accuracy: 0.404843 val accuracy: 0.384286
lr 1.668101e-08 reg 1.000000e-01 train accuracy: 0.377482 val accuracy: 0.372857
lr 1.668101e-08 reg 1.000000e+00 train accuracy: 0.402179 val accuracy: 0.388571
lr 1.668101e-08 reg 1.000000e+01 train accuracy: 0.384988 val accuracy: 0.370000
lr 1.668101e-08 reg 1.000000e+02 train accuracy: 0.393462 val accuracy: 0.387143
lr 1.668101e-08 reg 1.000000e+03 train accuracy: 0.404358 val accuracy: 0.385714
lr 1.668101e-08 reg 1.000000e+04 train accuracy: 0.499516 val accuracy: 0.481429
lr 1.668101e-08 reg 1.000000e+05 train accuracy: 0.568765 val accuracy: 0.550000
lr 1.668101e-08 reg 1.000000e+06 train accuracy: 0.296368 val accuracy: 0.292857
lr 2.782559e-06 reg 1.000000e-03 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e-02 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e-01 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+00 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+01 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+02 train accuracy: 0.473850 val accuracy: 0.454286
lr 2.782559e-06 reg 1.000000e+03 train accuracy: 0.516949 val accuracy: 0.497143
lr 2.782559e-06 reg 1.000000e+04 train accuracy: 0.203148 val accuracy: 0.204286
lr 2.782559e-06 reg 1.000000e+05 train accuracy: 0.156416 val accuracy: 0.158571
lr 2.782559e-06 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e-03 train accuracy: 0.999516 val accuracy: 0.998571
lr 4.641589e-04 reg 1.000000e-02 train accuracy: 0.999516 val accuracy: 0.998571
lr 4.641589e-04 reg 1.000000e-01 train accuracy: 0.986683 val accuracy: 0.987143
lr 4.641589e-04 reg 1.000000e+00 train accuracy: 0.990073 val accuracy: 0.990000
lr 4.641589e-04 reg 1.000000e+01 train accuracy: 0.322034 val accuracy: 0.338571
lr 4.641589e-04 reg 1.000000e+02 train accuracy: 0.224939 val accuracy: 0.224286
lr 4.641589e-04 reg 1.000000e+03 train accuracy: 0.165617 val accuracy: 0.157143
lr 4.641589e-04 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e-03 train accuracy: 0.997579 val accuracy: 0.997143
lr 7.742637e-02 reg 1.000000e-02 train accuracy: 0.568039 val accuracy: 0.537143
lr 7.742637e-02 reg 1.000000e-01 train accuracy: 0.410654 val accuracy: 0.415714
lr 7.742637e-02 reg 1.000000e+00 train accuracy: 0.207748 val accuracy: 0.191429
lr 7.742637e-02 reg 1.000000e+01 train accuracy: 0.143826 val accuracy: 0.131429
lr 7.742637e-02 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.291550e+01 reg 1.000000e-03 train accuracy: 0.344310 val accuracy: 0.348571
lr 1.291550e+01 reg 1.000000e-02 train accuracy: 0.184988 val accuracy: 0.174286
lr 1.291550e+01 reg 1.000000e-01 train accuracy: 0.150605 val accuracy: 0.150000
lr 1.291550e+01 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
best validation accuracy achieved during cross-validation: 0.998571
```

Figure 40: Validation Accuracy during Cross-Validation in SVM of Karolinska Dataset.

At last, I had visualized learned weights for each class as shown in Fig. 47. Learned weights depend upon the choice of learning rate and regularization strength.

Figure 41: Visualization of SVM technique on Karolinska Dataset.

4.2.3 Softmax

After Support Vector Machine (SVM), Softmax technique applies on the same Karolinska dataset and compares the results with SVM.

The Karolinska Dataset divides data into three categories: training, validation, and testing. As a subset of the training data, I had prepared a tiny development set. This was done to make my code run faster. 4130 images were used for training, 700 images for validation, 700 images for testing, and 175 images for development from a total of 4830 images. Following that, I pre-processed and altered picture data as shown in Fig. 48.

```
Train data shape:  (4130, 35815)
Train labels shape:  (4130,)
Validation data shape:  (700, 35815)
Validation labels shape:  (700,)
Test data shape:  (700, 35815)
Test labels shape:  (700,)
dev data shape:  (175, 35815)
dev labels shape:  (175,)
```

Figure 42: Split Data and Pre-process Data into rows.

Then, I had implemented the naive Softmax loss function and sanity check. Rough Sanity check of **– log (1/number of total classes) = – log (1/7)** = **– log (0.14285714) = 1.945918** and my loss should be close to this check. In Fig.49 our loss is closer to sanity check which means that I had applied correct loss function for Softmax.

```
loss: 2.122109
sanity check: 1.945910
```

Figure 43: Comparison of Loss and sanity Check of Karolinska Dataset.

Then, I had implemented the Softmax naive loss and calculated the numerical and analytical gradient. If my relative error is very less in all iterations then I can say that Numerical gradient should be close to analytical gradient.

```
numerical: 0.620927 analytic: 0.620927, relative error: 1.258055e-09
numerical: -0.048267 analytic: -0.048267, relative error: 1.449130e-07
numerical: 1.241880 analytic: 1.241880, relative error: 8.525115e-10
numerical: -0.302282 analytic: -0.302282, relative error: 1.853960e-07
numerical: 2.083773 analytic: 2.083773, relative error: 2.164315e-08
numerical: -0.955655 analytic: -0.955655, relative error: 1.170622e-08
numerical: 1.513634 analytic: 1.513633, relative error: 3.207783e-08
numerical: 0.140612 analytic: 0.140612, relative error: 7.545438e-08
numerical: 3.080255 analytic: 3.080255, relative error: 2.773715e-09
numerical: 0.167166 analytic: 0.167166, relative error: 1.324921e-07
numerical: 0.582707 analytic: 0.582707, relative error: 1.201838e-08
numerical: 0.095146 analytic: 0.095146, relative error: 4.697559e-08
numerical: 1.464303 analytic: 1.464303, relative error: 3.311888e-09
numerical: -1.032396 analytic: -1.032396, relative error: 5.918841e-09
numerical: -0.118530 analytic: -0.118530, relative error: 5.025324e-07
numerical: -0.493958 analytic: -0.493958, relative error: 7.289834e-09
numerical: -0.445085 analytic: -0.445085, relative error: 1.556506e-08
numerical: -2.118966 analytic: -2.118965, relative error: 3.950046e-08
numerical: 0.474164 analytic: 0.474164, relative error: 2.017466e-09
numerical: 0.872288 analytic: 0.872288, relative error: 8.090999e-10
```

Figure 44: Comparison of Analytic and Numeric computed Gradient of Softmax in Karolinska Dataset.

After that, I had computed the vectorized loss and compared it with my naive loss which I had computed earlier. This is the same method we used in CIFAR-10 Dataset in which difference zero of loss and the gradient will prove that my results are correct.

```
naive loss: 2.122109e+00 computed in 0.253300s
vectorized loss: 2.122109e+00 computed in 0.465030s
Loss difference: 0.000000
Gradient difference: 0.000000
```

Figure 45: Comparison of Vectorized and Naive Loss in Softmax of Karolinska Dataset.

I had efficient expressions for the loss, the vectorized loss and my naive loss matched. Therefore, ready to do Stochastic Gradient Descent (SGD) to minimize the loss. I performed as much iterations after that my loss will be saturated. After I had done SGD, minimum loss will be:

```
iteration 5400 / 9500: loss 1.795277
iteration 5500 / 9500: loss 1.830582
iteration 5600 / 9500: loss 1.769051
iteration 5700 / 9500: loss 1.783666
iteration 5800 / 9500: loss 1.747904
iteration 5900 / 9500: loss 1.756584
iteration 6000 / 9500: loss 1.787555
iteration 6100 / 9500: loss 1.755951
iteration 6200 / 9500: loss 1.755623
iteration 6300 / 9500: loss 1.794324
iteration 6400 / 9500: loss 1.779367
iteration 6500 / 9500: loss 1.801034
iteration 6600 / 9500: loss 1.839065
iteration 6700 / 9500: loss 1.773126
iteration 6800 / 9500: loss 1.787296
iteration 6900 / 9500: loss 1.765460
iteration 7000 / 9500: loss 1.788879
iteration 7100 / 9500: loss 1.807777
iteration 7200 / 9500: loss 1.792769
iteration 7300 / 9500: loss 1.756393
iteration 7400 / 9500: loss 1.764760
iteration 7500 / 9500: loss 1.819452
iteration 7600 / 9500: loss 1.777768
iteration 7700 / 9500: loss 1.782407
iteration 7800 / 9500: loss 1.775469
iteration 7900 / 9500: loss 1.757910
iteration 8000 / 9500: loss 1.776173
iteration 8100 / 9500: loss 1.769468
iteration 8200 / 9500: loss 1.766141
iteration 8300 / 9500: loss 1.753260
iteration 8400 / 9500: loss 1.797773
iteration 8500 / 9500: loss 1.809471
iteration 8600 / 9500: loss 1.785632
iteration 8700 / 9500: loss 1.777457
iteration 8800 / 9500: loss 1.793501
iteration 8900 / 9500: loss 1.712635
iteration 9000 / 9500: loss 1.746524
iteration 9100 / 9500: loss 1.777685
iteration 9200 / 9500: loss 1.806237
iteration 9300 / 9500: loss 1.771428
iteration 9400 / 9500: loss 1.800759
That took 537.197367s
```

Figure 46: Minimum Loss after SGD using Softmax in Karolinska Dataset.

The graph had shown that my loss decreased with respect to increase in the iterations.

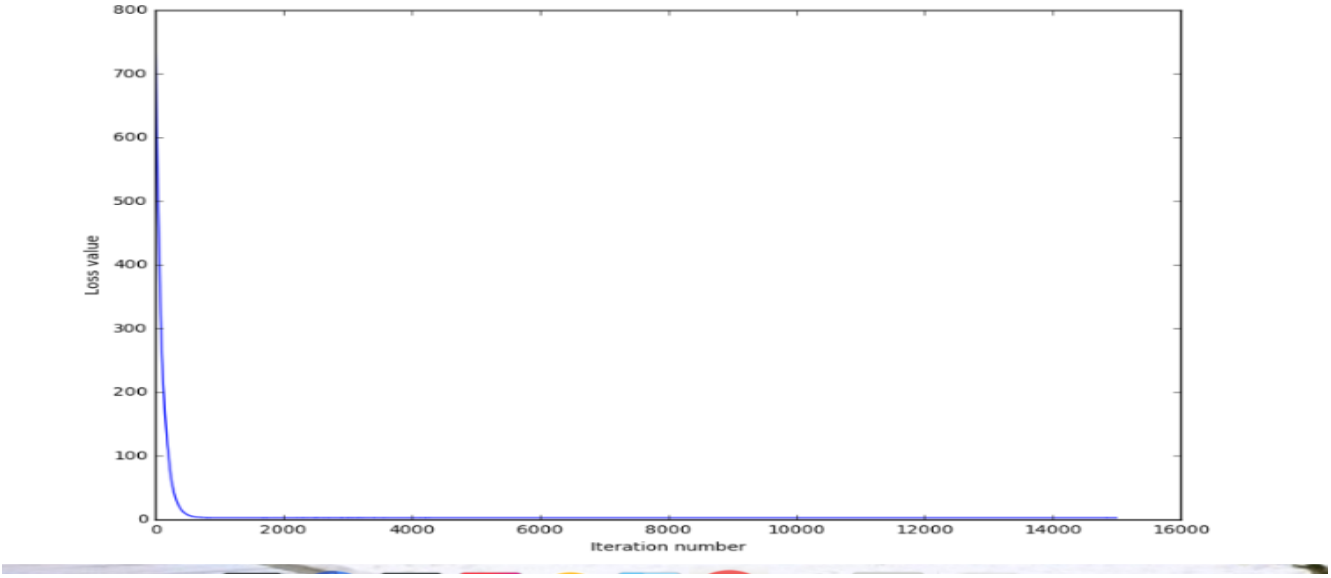


Figure 47: Loss with respect to Iterations in Softmax of Karolinska Dataset.

Then I had used validation set to tune hyper parameters (regularization strength and learning rate).

```
lr 1.000000e-10 reg 1.000000e-03 train accuracy: 0.148668 val accuracy: 0.138571
lr 1.000000e-10 reg 1.000000e-02 train accuracy: 0.142131 val accuracy: 0.141429
lr 1.000000e-10 reg 1.000000e-01 train accuracy: 0.151332 val accuracy: 0.158571
lr 1.000000e-10 reg 1.000000e+00 train accuracy: 0.143341 val accuracy: 0.154286
lr 1.000000e-10 reg 1.000000e+01 train accuracy: 0.139709 val accuracy: 0.140000
lr 1.000000e-10 reg 1.000000e+02 train accuracy: 0.138983 val accuracy: 0.138571
lr 1.000000e-10 reg 1.000000e+03 train accuracy: 0.146489 val accuracy: 0.147143
lr 1.000000e-10 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.131429
lr 1.000000e-10 reg 1.000000e+05 train accuracy: 0.156174 val accuracy: 0.161429
lr 1.000000e-10 reg 1.000000e+06 train accuracy: 0.170944 val accuracy: 0.164286
lr 1.668101e-08 reg 1.000000e-03 train accuracy: 0.268523 val accuracy: 0.234286
lr 1.668101e-08 reg 1.000000e-02 train accuracy: 0.261501 val accuracy: 0.272857
lr 1.668101e-08 reg 1.000000e-01 train accuracy: 0.249395 val accuracy: 0.217143
lr 1.668101e-08 reg 1.000000e+00 train accuracy: 0.263438 val accuracy: 0.234286
lr 1.668101e-08 reg 1.000000e+01 train accuracy: 0.265860 val accuracy: 0.231429
lr 1.668101e-08 reg 1.000000e+02 train accuracy: 0.255932 val accuracy: 0.260000
lr 1.668101e-08 reg 1.000000e+03 train accuracy: 0.268281 val accuracy: 0.252857
lr 1.668101e-08 reg 1.000000e+04 train accuracy: 0.313801 val accuracy: 0.325714
lr 1.668101e-08 reg 1.000000e+05 train accuracy: 0.365133 val accuracy: 0.365714
lr 1.668101e-08 reg 1.000000e+06 train accuracy: 0.265375 val accuracy: 0.272857
lr 2.782559e-06 reg 1.000000e-03 train accuracy: 0.999274 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e-02 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e-01 train accuracy: 0.999274 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+00 train accuracy: 0.999516 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+01 train accuracy: 0.999274 val accuracy: 0.998571
lr 2.782559e-06 reg 1.000000e+02 train accuracy: 0.817191 val accuracy: 0.821429
lr 2.782559e-06 reg 1.000000e+03 train accuracy: 0.411864 val accuracy: 0.411429
lr 2.782559e-06 reg 1.000000e+04 train accuracy: 0.235351 val accuracy: 0.237143
lr 2.782559e-06 reg 1.000000e+05 train accuracy: 0.159322 val accuracy: 0.151429
lr 2.782559e-06 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e-03 train accuracy: 0.998063 val accuracy: 0.997143
lr 4.641589e-04 reg 1.000000e-02 train accuracy: 0.999516 val accuracy: 0.998571
lr 4.641589e-04 reg 1.000000e-01 train accuracy: 0.998789 val accuracy: 0.998571
lr 4.641589e-04 reg 1.000000e+00 train accuracy: 0.624213 val accuracy: 0.604286
lr 4.641589e-04 reg 1.000000e+01 train accuracy: 0.328329 val accuracy: 0.290000
lr 4.641589e-04 reg 1.000000e+02 train accuracy: 0.155206 val accuracy: 0.157143
lr 4.641589e-04 reg 1.000000e+03 train accuracy: 0.153511 val accuracy: 0.158571
lr 4.641589e-04 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 4.641589e-04 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e-03 train accuracy: 0.996126 val accuracy: 0.997143
lr 7.742637e-02 reg 1.000000e-02 train accuracy: 0.782082 val accuracy: 0.760000
lr 7.742637e-02 reg 1.000000e-01 train accuracy: 0.248668 val accuracy: 0.258571
lr 7.742637e-02 reg 1.000000e+00 train accuracy: 0.191525 val accuracy: 0.160000
lr 7.742637e-02 reg 1.000000e+01 train accuracy: 0.150363 val accuracy: 0.145714
lr 7.742637e-02 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 7.742637e-02 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.291550e+01 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.291550e+01 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.291550e+01 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.291550e+01 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 2.154435e+03 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 3.593814e+05 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
best validation accuracy achieved during cross-validation: 0.998571
```

Figure 48: Validation Accuracy during Cross-Validation in Softmax of Karolinska Dataset.

At last, I had visualized learned weights for each class as shown in Fig. 55. Learned weights depend upon the choice of the learning rate and regularization strength which have maximum training and validation accuracy.

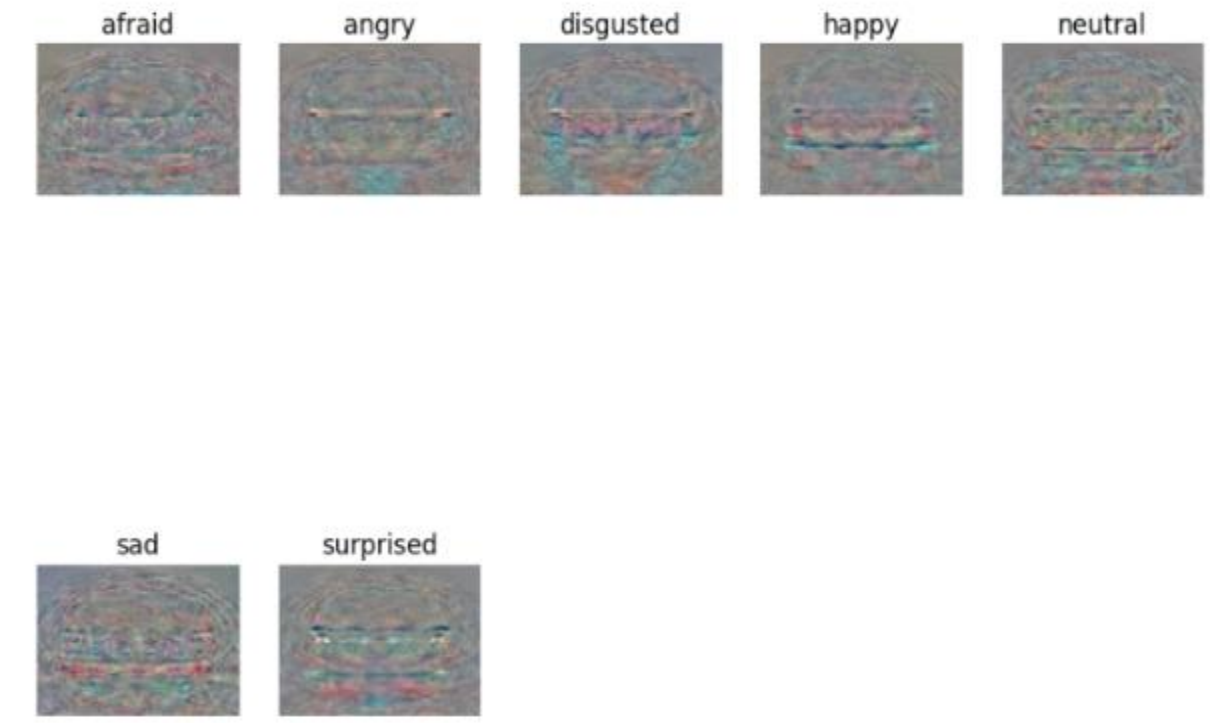


Figure 49: Visualization of Softmax technique on Karolinska Dataset.

4.2.4 Two Layer Network

Then, to do classification, create a neural network with fully connected layers and test it on the Karolinska dataset.

We used the same loss function that I had previously built for the SVM and Softmax in this design. It takes the data, weights it, and calculates the class scores, loss, and parameter gradients. First, I had to implement the forward pass, which computed the scores for all inputs using the weights and biases. There is a very minor difference between the correct and incorrect scores.

Then, on the Karolinska data, I developed a two-layer network that passes gradient testing. On an actual dataset, I can use it to train a classifier. Then divide the data into three sets: training, validation, and testing. 4130 images were used for training, 700 images for validation, and 700

images for testing from a total of 4830 images. After that I do pre-processing and reshape image data into rows as shown in Fig. 56.

```
Train data shape:  (4130, 35814)
Train labels shape:  (4130,)
Validation data shape:  (700, 35814)
Validation labels shape:  (700,)
Test data shape:  (700, 35814)
Test labels shape:  (700,)
```

Figure 50: Split Data and Preprocess into rows.

I utilised Stochastic Gradient Descent (SGD) with momentum to train a network. In addition, as optimization progressed, I altered the learning rate with an exponential learning rate; after each epoch (iteration), I reduced the learning rate by multiplying it by a decay rate.

```
iteration 0 / 9500: loss 1.950347
iteration 100 / 9500: loss 1.950092
iteration 200 / 9500: loss 1.949299
iteration 300 / 9500: loss 1.948105
iteration 400 / 9500: loss 1.940780
iteration 500 / 9500: loss 1.939585
iteration 600 / 9500: loss 1.926096
iteration 700 / 9500: loss 1.922030
iteration 800 / 9500: loss 1.901787
iteration 900 / 9500: loss 1.928980
iteration 1000 / 9500: loss 1.890440
iteration 1100 / 9500: loss 1.914837
iteration 1200 / 9500: loss 1.920779
iteration 1300 / 9500: loss 1.899218
iteration 1400 / 9500: loss 1.896778
iteration 1500 / 9500: loss 1.876841
iteration 1600 / 9500: loss 1.927019
iteration 1700 / 9500: loss 1.924550
iteration 1800 / 9500: loss 1.889087
iteration 1900 / 9500: loss 1.908356
iteration 2000 / 9500: loss 1.890073
iteration 2100 / 9500: loss 1.880980
iteration 2200 / 9500: loss 1.891276
iteration 2300 / 9500: loss 1.906099
iteration 2400 / 9500: loss 1.880738
iteration 2500 / 9500: loss 1.912755
iteration 2600 / 9500: loss 1.888441
iteration 2700 / 9500: loss 1.906593
iteration 2800 / 9500: loss 1.919672
iteration 2900 / 9500: loss 1.915278
iteration 3000 / 9500: loss 1.890832
iteration 3100 / 9500: loss 1.912159
iteration 3200 / 9500: loss 1.880600
iteration 3300 / 9500: loss 1.918661
iteration 3400 / 9500: loss 1.887528
iteration 3500 / 9500: loss 1.898805
iteration 3600 / 9500: loss 1.877076
iteration 3700 / 9500: loss 1.878550
iteration 3800 / 9500: loss 1.874745
iteration 3900 / 9500: loss 1.886353
iteration 4000 / 9500: loss 1.861366
iteration 4100 / 9500: loss 1.889353
iteration 4200 / 9500: loss 1.904965
iteration 5100 / 9500: loss 1.906158
iteration 5200 / 9500: loss 1.896614
iteration 5300 / 9500: loss 1.899157
iteration 5400 / 9500: loss 1.900704
iteration 5500 / 9500: loss 1.901581
iteration 5600 / 9500: loss 1.904906
iteration 5700 / 9500: loss 1.881579
iteration 5800 / 9500: loss 1.930204
iteration 5900 / 9500: loss 1.868717
iteration 6000 / 9500: loss 1.906651
iteration 6100 / 9500: loss 1.907576
iteration 6200 / 9500: loss 1.913815
iteration 6300 / 9500: loss 1.914465
iteration 6400 / 9500: loss 1.887680
iteration 6500 / 9500: loss 1.903622
iteration 6600 / 9500: loss 1.887399
iteration 6700 / 9500: loss 1.919508
iteration 6800 / 9500: loss 1.902738
iteration 6900 / 9500: loss 1.895420
iteration 7000 / 9500: loss 1.896827
iteration 7100 / 9500: loss 1.899596
iteration 7200 / 9500: loss 1.912510
iteration 7300 / 9500: loss 1.895806
iteration 7400 / 9500: loss 1.894988
iteration 7500 / 9500: loss 1.883766
iteration 7600 / 9500: loss 1.925944
iteration 7700 / 9500: loss 1.920230
iteration 7800 / 9500: loss 1.883441
iteration 7900 / 9500: loss 1.889555
iteration 8000 / 9500: loss 1.920104
iteration 8100 / 9500: loss 1.909568
iteration 8200 / 9500: loss 1.906967
iteration 8300 / 9500: loss 1.896202
iteration 8400 / 9500: loss 1.887605
iteration 8500 / 9500: loss 1.907690
iteration 8600 / 9500: loss 1.914481
iteration 8700 / 9500: loss 1.907277
iteration 8800 / 9500: loss 1.894828
iteration 8900 / 9500: loss 1.900338
iteration 9000 / 9500: loss 1.902070
iteration 9100 / 9500: loss 1.889926
iteration 9200 / 9500: loss 1.929989
iteration 9300 / 9500: loss 1.913568
```

Figure 51: Minimum Loss after SGD using Two Layer Network in Karolinska Dataset.

With the default parameters, the validation accuracy of about **22.46%** is not good on the validation set. Then I used a different method to visualise the weights learned in the neural network's first layer. When the first layer weights of most neural networks trained on visual data are displayed, they usually show some obvious structure.

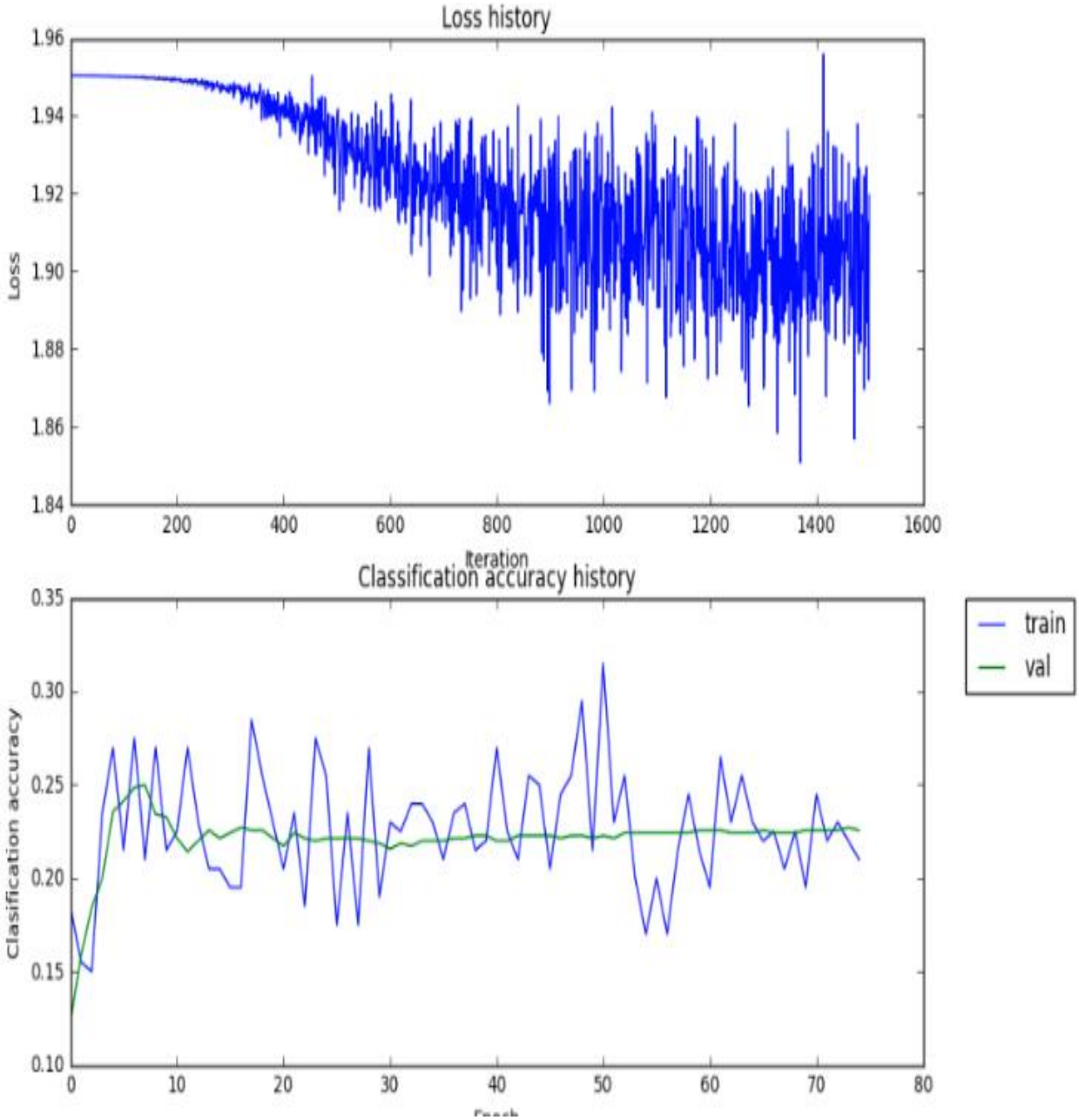


Figure 52: Plot of loss function and validation accuracies of the network.

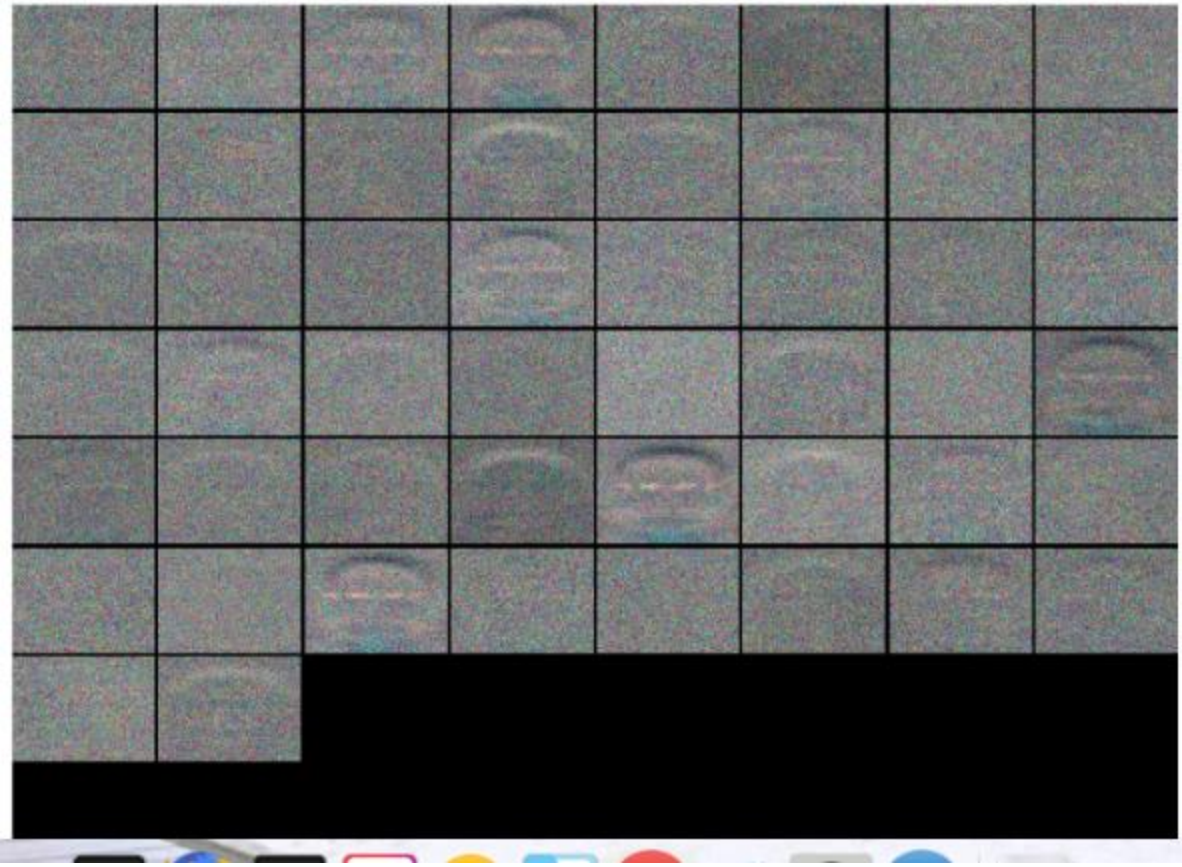


Figure 53: Visualization the weights of the network.

Then I had used the validation set to tune hyper parameters (hidden layer size, number of training epochs, regularization strength and learning rate). This should be aiming to achieve a maximum classification accuracy on the training and validation set.

```
lr 1.000000e-03 reg 4.000000e-01 train accuracy: 0.965133 val accuracy: 0.957143
lr 1.000000e-03 reg 5.000000e-01 train accuracy: 0.960533 val accuracy: 0.937143
lr 1.000000e-03 reg 6.000000e-01 train accuracy: 0.955448 val accuracy: 0.950000
lr 1.000000e-02 reg 4.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e-02 reg 5.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e-02 reg 6.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
best validation accuracy achieved during cross-validation: 0.957143
```

Figure 54: Validation Accuracy during Cross-Validation in Two Layer Network.

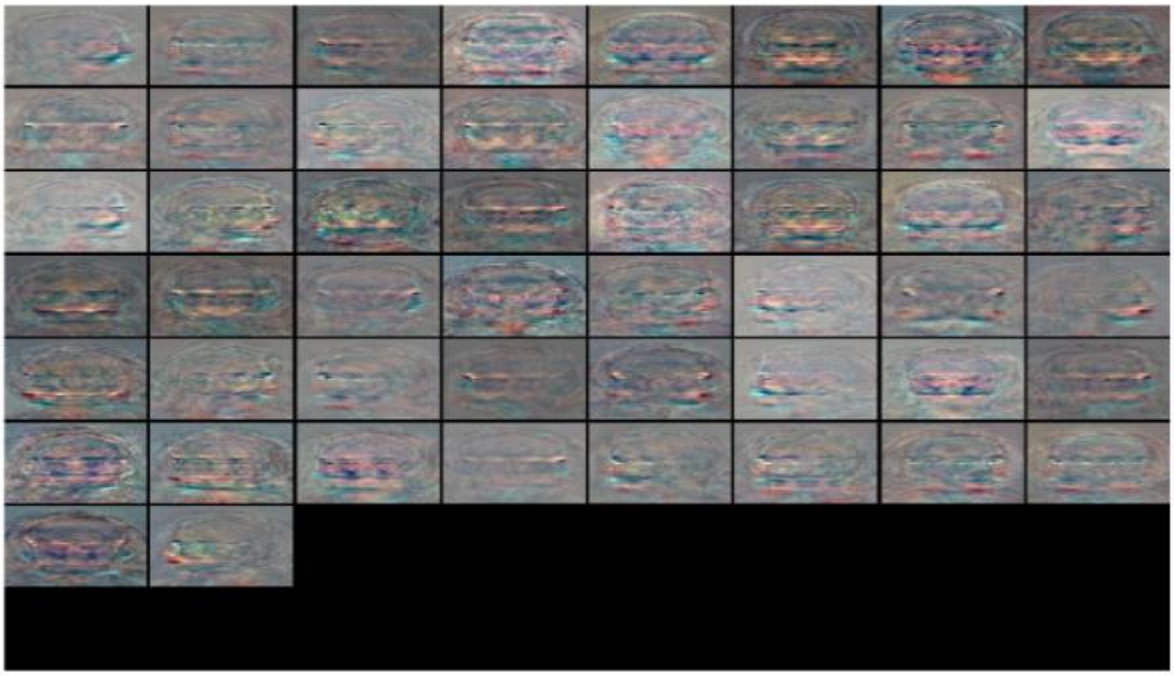


Figure 55: Visualization the weights of the best network.

Then shown the loss function and train accuracies graph of the best network and compared it with the previous network graph.

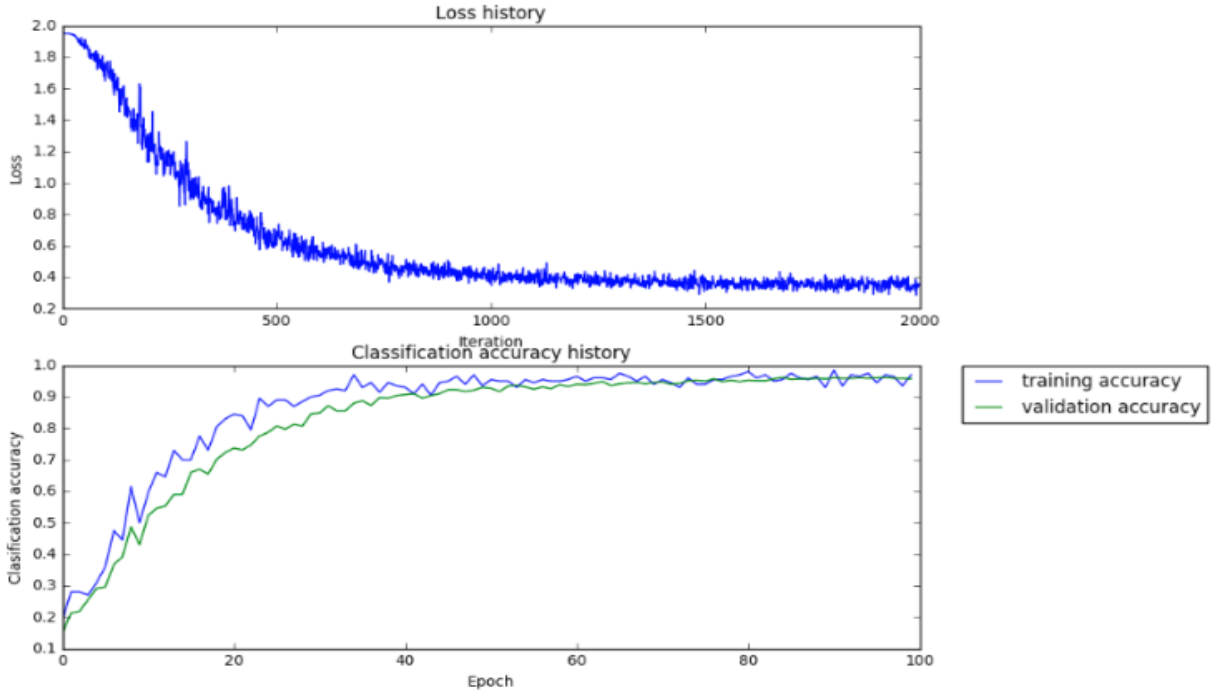


Figure 56: Plot of loss function and validation accuracies of the best network.

After experiments, this would evaluate your final trained network on the test set. The test accuracy of my final trained network is **65.57%.**

4.2.5 VVG 16

VGG-16 is one of the best Convolutional Neural Networks (CNN) designs, with 16 CNV/FC layers and a great homogeneous architecture that only conducts 3x3 convolutions and 2x2 pooling from start to finish, as shown in Fig.63 [5].

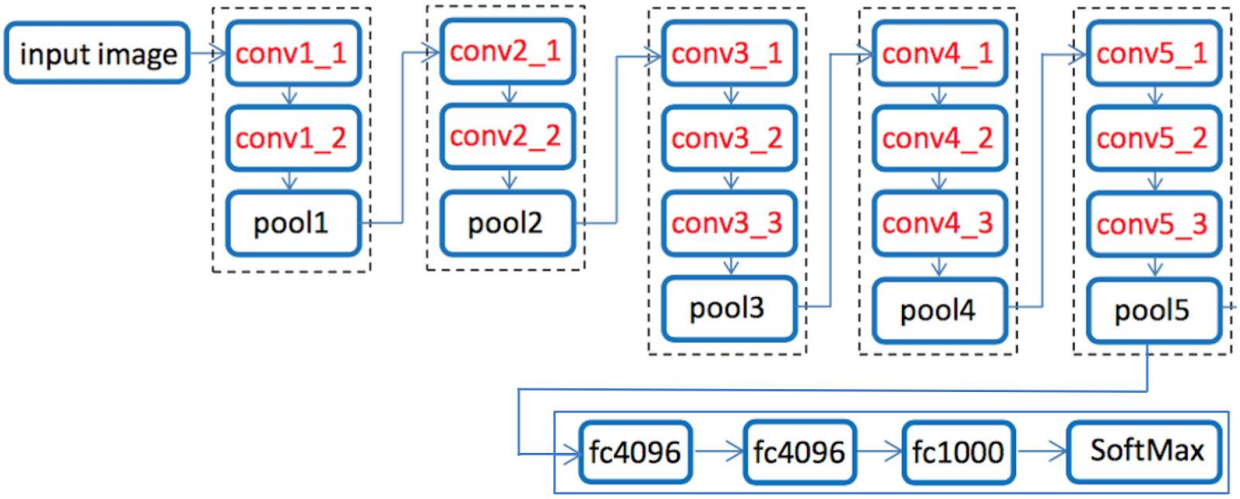


Figure 57: VVG-16 Architecture [5].

Karolinska Dataset divides data into training, validation, and testing sets in VVG-16. 4130 images were used for training, 700 images for validation, and 700 images for testing, out of a total of 4830. Following that, we pre-processed and reshaped picture data as shown in Fig. 64.

```
Train data shape:  (4130, 100352)
Train labels shape:  (4130,)
Validation data shape:  (700, 100352)
Validation labels shape:  (700,)
Test data shape:  (700, 100352)
Test labels shape:  (700,)
dev data shape:  (175, 100352)
dev labels shape:  (175,)
```

Figure 58: Split Data in VVG-16 Technique.

In VVG-16, 16 layers are connected with each other and the format in which they are connected with each other as shown in Fig.65 along with the dimension of the images at each stage.

```
0 conv1_1_W (3, 3, 3, 64)
1 conv1_1_b (64,)
2 conv1_2_W (3, 3, 64, 64)
3 conv1_2_b (64,)
4 conv2_1_W (3, 3, 64, 128)
5 conv2_1_b (128,)
6 conv2_2_W (3, 3, 128, 128)
7 conv2_2_b (128,)
8 conv3_1_W (3, 3, 128, 256)
9 conv3_1_b (256,)
10 conv3_2_W (3, 3, 256, 256)
11 conv3_2_b (256,)
12 conv3_3_W (3, 3, 256, 256)
13 conv3_3_b (256,)
14 conv4_1_W (3, 3, 256, 512)
15 conv4_1_b (512,)
16 conv4_2_W (3, 3, 512, 512)
17 conv4_2_b (512,)
18 conv4_3_W (3, 3, 512, 512)
19 conv4_3_b (512,)
20 conv5_1_W (3, 3, 512, 512)
21 conv5_1_b (512,)
22 conv5_2_W (3, 3, 512, 512)
23 conv5_2_b (512,)
24 conv5_3_W (3, 3, 512, 512)
25 conv5_3_b (512,)
26 fc6_W (25088, 4096)
27 fc6_b (4096,)
28 fc7_W (4096, 4096)
29 fc7_b (4096,)
30 fc8_W (4096, 1000)
31 fc8_b (1000,)
```

Figure 59: VVG-16 Layers Format.

I utilised Stochastic Gradient Descent to train a VVG-16 network (SGD). In addition, as the optimization progressed, I altered the learning rate with an exponential learning rate; after each epoch (iteration), I reduced the learning rate by multiplying it by a decay rate.

```
iteration 0 / 3000: loss 17530.694075
iteration 100 / 3000: loss 6425.739599
iteration 200 / 3000: loss 2355.910103
iteration 300 / 3000: loss 864.205656
iteration 400 / 3000: loss 317.494634
iteration 500 / 3000: loss 116.729237
iteration 600 / 3000: loss 43.304147
iteration 700 / 3000: loss 16.432067
iteration 800 / 3000: loss 6.592022
iteration 900 / 3000: loss 2.949243
iteration 1000 / 3000: loss 1.618017
iteration 1100 / 3000: loss 1.162423
iteration 1200 / 3000: loss 1.021269
iteration 1300 / 3000: loss 0.882904
iteration 1400 / 3000: loss 0.952278
iteration 1500 / 3000: loss 0.869525
iteration 1600 / 3000: loss 0.828286
iteration 1700 / 3000: loss 0.859290
iteration 1800 / 3000: loss 0.881240
iteration 1900 / 3000: loss 0.855745
iteration 2000 / 3000: loss 0.897807
iteration 2100 / 3000: loss 0.872534
iteration 2200 / 3000: loss 0.820385
iteration 2300 / 3000: loss 0.876169
iteration 2400 / 3000: loss 0.860709
iteration 2500 / 3000: loss 0.899245
iteration 2600 / 3000: loss 0.872502
iteration 2700 / 3000: loss 0.846770
iteration 2800 / 3000: loss 0.897488
iteration 2900 / 3000: loss 0.869156
That took 476.243655s
```

Figure 60: Minimum Loss after using SGD in Karolinska Dataset.

The graph had shown that our loss decreased with respect to increase in the iterations.

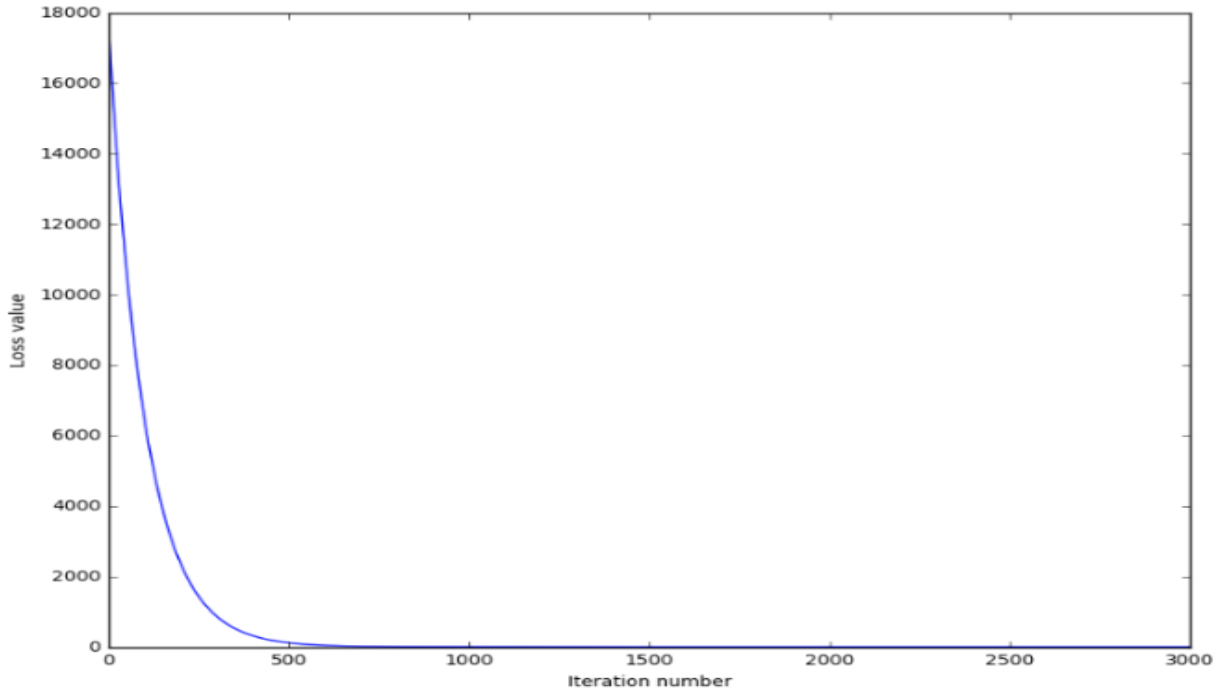


Figure 61: Loss with respect to Iterations in VVG-16 of Karolinska Dataset.

Then I had used the validation set to tune hyper parameters (Conv layer size, Fully Connected (FC) layer size, regularization strength and learning rate). This should be aiming to achieve a maximum classification accuracy on the training and validation set.

```
lr 5.994843e+07 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 5.994843e+07 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e-01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+00 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+01 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+02 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+03 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+04 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+05 train accuracy: 0.142857 val accuracy: 0.142857
lr 1.000000e+10 reg 1.000000e+06 train accuracy: 0.142857 val accuracy: 0.142857
best validation accuracy achieved during cross-validation: 0.998571
```

Figure 62: Validation Accuracy during Cross-Validation in VVG-16 Network.

After experiments, this would evaluate your final trained network on the test set. The test accuracy of my final trained VVG-16 network is **55.5714%.**

5 Chapter

Results

This chapter includes the table of the accuracies that I achieved during our experiments. I'm also comparing the results of different techniques that I had applied to CIFAR 10 dataset and Karolinska Dataset.

5.1 CIFAR 10 Dataset Accuracy

	Training Accuracy	**Validation Accuracy**	**Testing Accuracy**
Support Vector Machine	36.7959 %	38.4000 %	36.4000 %
Softmax	38.8000 %	38.7000 %	36.3000 %
Two Layer Network	52.0592 %	49.7000 %	49.6 %

Table 1: CIFAR 10 Dataset Accuracy

This table shows the best learning rates and regularization strengths of the accuracies that I had calculated during Support Vector Machine (SVM), Softmax and Two Layer Network applied to CIFAR 10 Dataset.

	Regularization Strength	**Learning Rate**
Support Vector Machine	1.0000e+04	2.0000e-07
Softmax	1.0000e+00	2.782559e-06
Two Layer Network	6.0000e-01	1.0000e-03

Table 2: Best Regularization Strength and Learning Rate of CIFAR 10 Dataset

5.2 Karolinska Dataset Accuracy

	Training Accuracy	Validation Accuracy	Testing Accuracy
Support Vector Machine	69.0557 %	66.5714%	44.0000 %
Softmax	94.96 %	94.0000 %	57.8571 %
Two Layer Network	96.5133 %	95.7143 %	65.57%
VVG-16	99.8741%	99.8571%	55.5771%

Table 3: Karolinska Dataset Accuracy

This table shows the best learning rates and regularization strengths of the accuracies that I had calculated during Support Vector Machine (SVM), Softmax and Two Layer Network applied to Karolinska Dataset.

	Regularization Strength	Learning Rate
Support Vector Machine	1.0000e-01	2.78256e-06
Softmax	1.0000e-02	2.78256e-06
Two Layer Network	4.0000e-01	1.0000e-03

Table 4: Best Regularization Strength and Learning Rate of Karolinska Dataset.

5.3 Confusion Matrix

A confusion matrix is a method of describing the performance of a classification model using test data for which true values are known [6].

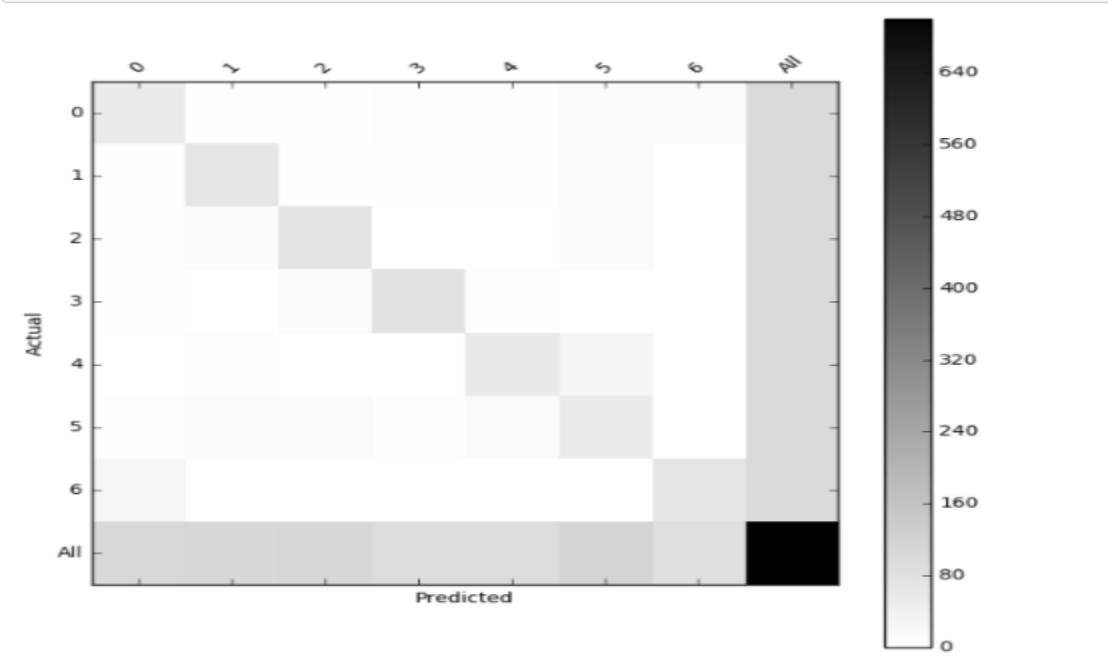


Figure 63: Confusion Matrix.

Confusion Matrix tells us the true number of images our system recognizes. Right side of the above Fig.69 tells us the total number of images. The diagonals of the matrix box tell us the total numbers of true predicated images from the actual images and labels tell us the emotions. Label 0, 1, 2 , 3, 4, 5, 6 represents Afraid, Angry, Disgusted, Happy, Neutral, Sad, Surprised respectively.

6 Chapter

Conclusion & Future Work

I have explored the Support Vector Machine (SVM), Softmax and Two Layer Network architectures for recognizing facial expressions. The results have demonstrated that I was able to achieve acceptable results in comparison to CIFAR 10 Dataset and Karolinska Directed Emotional Faces (KDEF) Dataset.

First of all when I had applied Support Vector Machine (SVM) on CIFAR 10 Dataset as shown in (Table 1), I had examined that we got accuracies of training, validation and testing are very small less than 40%. So, I had needed another technique which had given us better results. Then I had implemented Softmax on same Dataset. I had examined that this technique has better accuracy than SVM but in this case also I get accuracies less than 40% as shown in (Table 1). We need to apply a technique which gave us accuracy near or greater than 50%. So, I had implemented Two Neural Network on same Dataset. After applying this technique we got accuracies greater than 50% as shown in (Table 1).

After that I had applied the same techniques (SVM, Softmax and Two Neural Network) on Karolinska Dataset to extract human expressions. Again first I had implemented Support Vector Machine (SVM) on Karolinska Dataset. After applying that I had examined that the training, validation and testing accuracies have much better results than the result of CIFAR 10 Dataset as shown in (Table 3). But I'm not satisfied with my result as much as I needed. So I had implemented Softmax technique on Karolinska Dataset. I had examined that Softmax gave me much better results than SVM. The training and validation accuracies are greater than 90% and testing accuracy is also greater than 55% as shown in (Table 3). I had needed such technique which had given me better testing accuracy as training and validation accuracies of Softmax. So I had implemented Two Layer Network on the same dataset. After applying this we had gotten the accuracies greater than the Softmax accuracies in all aspects. I get training and validation accuracies greater than 95% and testing accuracy near to 65% as shown in (Table 3). I had better accuracies on Karolinska Dataset than CIFAR 10 Dataset as same techniques had applied to both datasets because in Karolinska Dataset every image has the same background, clothes, distance from camera and light on the face. But in CIFAR 10 Dataset there are different background, clothes, distance from camera and light. So that's the main reason behind different accuracies by

applying the same techniques on datasets. When using a two-layer network with transfer learning, I was able to get **49.6%** accuracy on the CIFAR 10 Dataset and **65.57%** accuracy on the KDEF Dataset.

In order to train the models for this thesis, I used a pre-processed version of the raw image pixels. I'll improve these models even more by combining the outputs of the five neural networks and ResNet into an ensemble model. I'd like to experiment with incorporating other facial and picture aspects to boost model performance even further. I'd like to extend our work to analyse different traits such as confidence, composure, and credibility obtained from human micro expressions, and I'd also like to study deep learning beyond the seven basic human emotions.

References

[1] P.-L. Carrier, A. Courville, I. J. Goodfellow, M. Mirza and a. Y. Bengio, “FER-2013 Face Database,” Montreal, 2013.

[2] B. Graham, Fractional max-pooling, 2015.

[3] K. Andrej, “CS231n Convolution Neural Networks for Visual Recognition,” 11 April 2016. [Online]. Available: http://cs231n.github.io/linear-classify/.

[4] D. F. A. &. Ö. A. Lundqvist, “The Karolinska Directed Emotional Faces,” Stockholm, Sweden, Karolinska Institutet, 1998.

[5] F. Chollet and Keras, “GitHub Repository,” 2015. [Online]. Available: https://github.com/fchollet/keras.

[6] K. Markham, “Data School,” 25 March 2014. [Online]. Available: http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/.

[7] A. Raghuvanshi and V. Choski, “Facial Expression Recognition with Convolution Neural Networks,” Standford Universty , California, 2015.

[8] . P. Ekman and W. V. F. Emotiona, “Facial Action Coding System,” Universty of California, San Fancisco, 1983.

[9] . D. C. Ali Mollahosseini and . M. H. Mahoor. , “Facial Expresstion recognition using deep neural networks,” IEEE winter Conference on Applications of Computer Vision, 2016.

[10] B.-K. K. S.-Y. D, R. Jihyeon and L. S.-Y., “Journal on Multimodal User Interfaces,” *S.-Y. D, Bo-Kyeong Kim; Jihyeon, Roh; S.-Y., Lee;,* no. Hierarchical committee of deep convolutional neural networks, pp. 1-17, 2015.

[11] Z. M. S.-S. S. M. B. Z. L. X. S. B. P. J., M. S. and M. R., “Salient object subitizing,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015.

[12] T. Y., Kanade and C. J., “Recognizing action units for facial expression analysis,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[13] B. M.S. , L. G. , F. M. , . L.-. s. C. , F. I. and M. J. , “Fully automatic facial action recognition in spontaneous behavior.,” in *IEEE Conference on Automatic Facial and Gesture Recognition,* 2006.

[14] P. M. and R. J.M. , “Facial action recognition for facial expression analysis from static face images,” in *IEEE Transactions on Systems, Man and Cybernetics*, 2004.

[15] E. P. and F. W., “A Technique for the Measurement of Facial Movement,” in *Facial Action Coding System*, Consulting Psychologists Press,, 1978.

[16] H. L. Thai, “Applying artificial neural networks for face Recognition,” in *Advances in Artificial Neural Systems*, 2011.

[17] Scikit-learn, “Machine Learning in Python,” vol. 12, no. Pedregosa et al., pp. 2825-2830, 2011.

[18] K. Andraj, “CS231n Convolution Neural Network for Visual Recognition,” 11 April 2016. [Online]. Available: http://cs231n.github.io/convolutional-networks/.

[19] I. Aleksander and H. Morton, “Neural Networks,” 2014. [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

[20] A. Karpathy, “Cs231n Convolution Neural Networks for Visual Recognition,” 2016. [Online]. Available: http://cs231n.github.io/neural-networks-1/#nn.

[21] A. Krizhevsky, “CIFAR Dataset,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[22] A. Karpathy, “CS231n Convolutional Neural Networks for Visual Recognition,” April 2016. [Online]. Available: http://cs231n.github.io/neural-networks-1/.

[23] Wikipedia, “Rectifier (neural networks),” June 2015. [Online]. Available: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[24] S. Sharma, “Activation Functions: Neural Networks,” 2014. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6?token=r2ApfXElMoCFYHeb.

[25] A. Karpathy, “CS231n Convolutional Neural Networks for Visual Recognition,” 2016. [Online]. Available: http://cs231n.github.io/neural-networks-1/.

[26] A. Krizhevsky, “CIFAR Dataset,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[27] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. In ICLR, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.

[29] J. Minker. Logic-Based Artificial Intelligence. Springer Science & Business Media, 2000.

[30] M. D. Zeiler and R. Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. arXiv preprint arXiv:1301.3557, 2013.