Deep Learning HW02

Siffi Singh – 0660819

Program to build a convolutional neural network to perform Image Recognition.

Part 1: Image Recognition on MNIST dataset

a) Showing:

1. Network architecture
2. Effect of stride size and filter size
3. Learning curve
4. Accuracy of Training and Test Sets
5. Distribution of weights and biases

b) Correctly classified and miss-classified images

c) Feature maps of different convolutional layers

d) Effect of L2 regularization term

Part 2: Image Recognition on Preprocessed CIFAR-10 dataset

a) Preprocess CIFAR-10 dataset

b) Showing:

   1. Network architecture
   2. Effect of stride size and filter size
   3. Learning curve
   4. Accuracy of Training and Test Sets
   5. Distribution of weights and biases

c) Correctly classified and miss-classified images

d) Feature maps of different convolutional layers

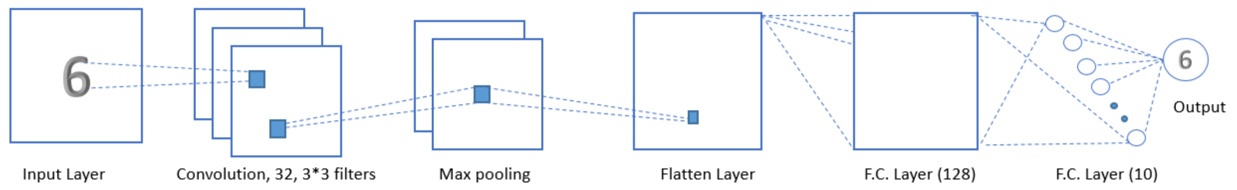
e) Effect of L2 regularization term

Part 1: Image Recognition on MNIST dataset

a) 1. Network architecture

The way we design the network parameters, the choice of layers, its dimensions, every little detail in the formation of deep neural networks play a key role in determining the performance of the model. Sometimes, a model works well for a particular dataset and doesn't works well for other. Since, the whole idea of neural networks is based on data and computing power, we have to get to an exclusive

designing of architecture when it comes to form a convolutional neural network for Image recognition on MNIST dataset. For our network, we have designed a network that has the following structure:
Total examples: 70,000
Total Training examples: 55,000
Validation examples: 5,000
Testing examples: 10,000
No. of layers: 5
Details of layer: 1 convolution, 1 pooling, 1 flattening, 2 fully connected layer
Activation function used in layers: ReLU
Loss function: Cross Entropy + L2 regularization
Evaluation Function: Accuracy



Input Layer: The input layer has 28*28, i.e. 784 entries. The intuition behind choosing just one convolution layer is finalized after trying two and three convolution layers. Since, a convolution layers are a combination of a bunch of convolutional and max pooling layers. Details about the layers chosen are as follows:

1. **Convolution** Layer 1: The first layer has **28, 3*3 filters** with **stride** of **1** and **no padding** applied to it.
2. **Max Pooling** Layer: Then, using a max pooing layer to reduce the parameter counts and computational complexity.
3. **Flattening Layer**: Then, a flattening layer is used to flatten the matrix values in order to pass it to the fully connected layer.
4. **Fully connected layer** with **128 units**: Taking the output from the flattened layer, it is passed on to a fully connected layer. We use a **ReLU** activation function for this layer.
5. **Fully connected layer** with **10 units**: The units chosen in this layer makes sense as the number of classes are 10, and we have to classify our data into 10 classes. The function used here is the **Softmax** function that gives us the probability for each data to belong to one of the ten classes, and the class with maximum probability is chosen.

The designing is inspired by the implementation of pre-defined function but for the purpose of this homework, we have considered writing all function implemented

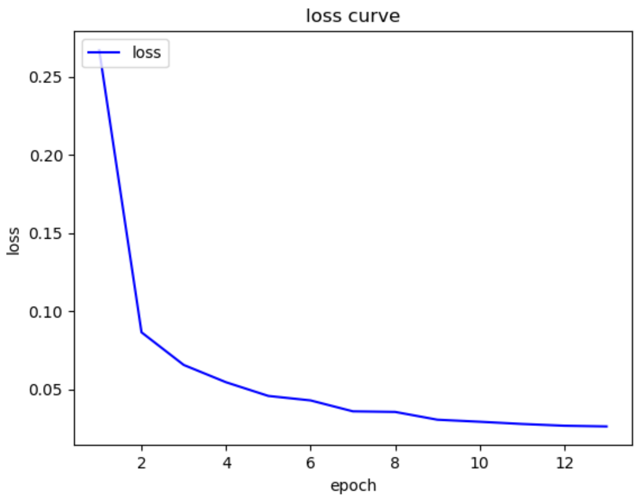
by ourselves, including the forward propagation, backward propagation, the function for convolution 2D. The drawback of self implementation is that it is extremely slow and does not run as smooth as the pre-built function.
The result has been compared with the other pre-defined methods and it seems to be reasonable. MNIST dataset is a greyscale dataset, containing values from 0-255. Hence, using the above defined model, I was able to achieve an accuracy of 99.13%. The effect of the issues and solutions determined during the implementation has been explained in the other sections. The most important role was played by the dimensions of the matrix as it goes into the network, with each layer, it was critical to note that the previous layer is in congruence of the requirements of the next layer. For example, using a flattening layer before passing to the Fully connected layer. The intuition behind the chosen parameters is just by referring to other networks and hit-n-trial to see what works best for our dataset.

a) 2. Effect of stride size and filter size

Choosing the value of stride and filter size is a key point of convolution and max pooling layer. If we choose the stride too big, according to the intuition, we might lose out capturing the important features, at the same time, if we keep it too low, we might not be able to capture the inert relationship of the pixels well. We need to find the stride value that also suits the padding. Such as keeping a stride 1 with no padding gives better result than keeping a stride 1 with zero padding. The combined effect of padding and stride gives an overall good feature extraction.
Another point to note is the value filter size. We have used a 3*3 filter size for convolutional layer. In simulation, I tried to insert two convolution layers, one with 16, 3*3 and the other with 16, 5*5, and the resultant accuracy dropped by 2.1%. Keeping that into consideration, we changed the convolution layer to one by keeping the filter size 32, 5*5. The accuracy compared with the current obtained accuracy was different by 0.7%, which is not much, but the best accuracy was noted by using 28, 3*3 filters.

a) 3. Learning Curve

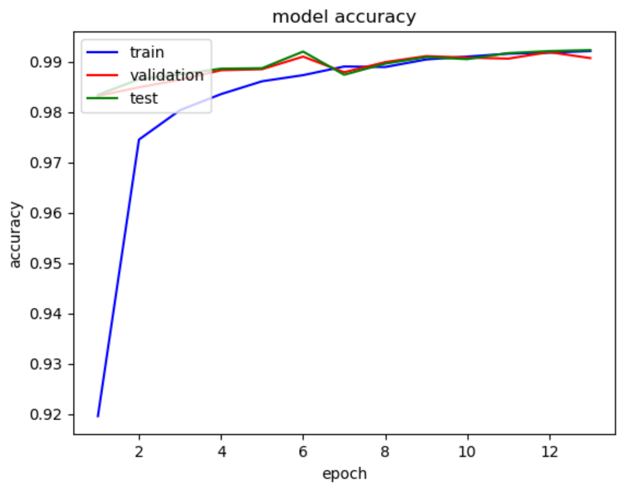
The learning curve for the designed convolutional neural network for Image recognition here is a constant learning rate for all iterations. We plotted the graph after verifying multiple values of **learning rate ranging from** as small as 0.00001 to 0.001 and the result on which the best result was seen for the chosen network architecture was selected. The following graph was formed by using a learning rate of 0.001.



We also observed that using a learning rate too small, resulted in a less accuracy and rare convergence. The value of lr = 0.001 is set along with using a cross entropy loss function with L2 regularization function. The graph is plotted against the no. of **epochs** and the **loss** value, no wonder there is a sudden drop in the loss value seen from 0 to 2nd epoch. The plot is for a total of **13 epochs** with a corresponding loss value of 0.0181.

a) 4. Accuracy of Training and Test Sets

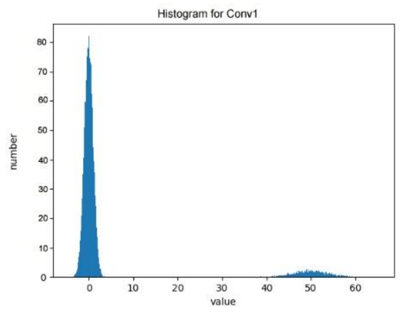
The evaluation metric used for this convolutional layer is Accuracy. To plot the training, validation and testing accuracy, I had to fit the model to all three datasets. The following graph was obtained for training, validating and testing datasets of MNIST by using the above proposed convolutional neural network.

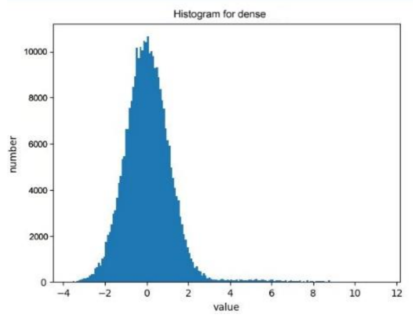


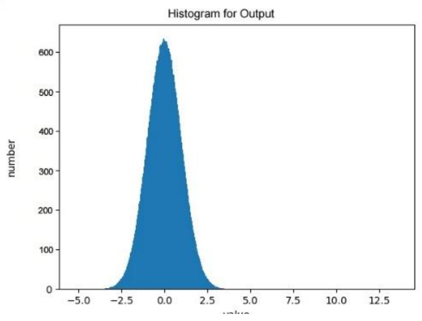
The graph is plotted between no. of epoch and training, validation and test accuracy. The reason of the sudden sharpness of graph at certain points is due to the plot of no. of epoch. It shows that the accuracy plot for training, is close to validation and testing dataset. It is also reflective of the fact, that the model is not overfitting the data.

a) 5. Distribution of weights and biases

The distribution of weights for the given convolutional layer is given below. There were several challenges to visualizing the weights. To plot the weights, we flattened the higher dimension matrix to a 1 dimensional vector and created a histogram of its values. Since our convolutional model has one convolution layer, so the first histogram corresponds to that. The second histogram is after the first fully connected or dense layer, and the third one is the distribution of weights at the output layer with L2 regularization.







b) Correctly classified and miss-classified images

The analysis of correctly classified and miss-classified images is important as it helps us figure out the **corner cases** or the **hard cases** in the dataset. In our dataset, we are dealing with a simple dataset of images, but it gives us a good idea of what cases is our model not working on.

It also helps us to divide the dataset into two categories, the **high confidence** data and the **low confidence** data. The high confidence data corresponds to the data elements that are easier to classify and are far away from the decision boundary separating our data. The low confidence data on the other hand are the data cases that are close to the decision boundary and are difficult to perform classification. In our dataset, we present some examples.

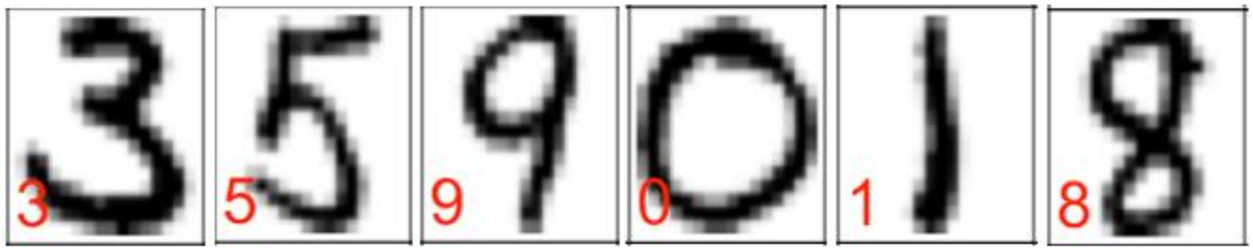


Figure 1: Correctly classified images in MNIST

Below are the incorrectly classified images, along with blue marked values that the convolutional neural network model found classified, against the correct marked.

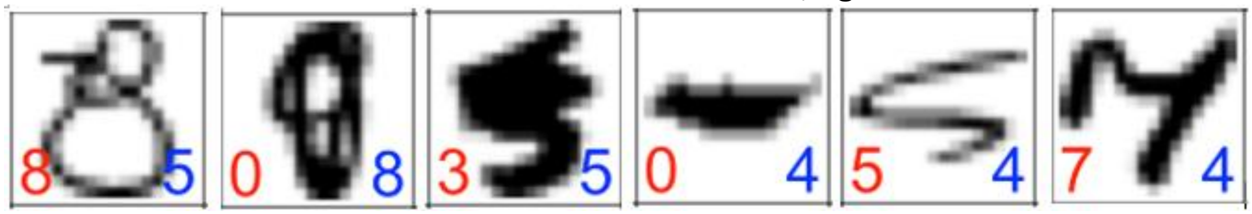


Figure 2: Miss-classified images in MNIST dataset.

c) Feature maps of different convolutional layers

Feature maps gives us a visual insight into how our model is learning. How it is getting the details, from **low-level features** to **high-level features** with increasing **depth** of the network. This also reflects the key features that helps the neural network form a generalized pattern based on which it classifies the test and validation set. Observing the feature maps has told us how it goes from understandable edges to not so sense making lines which is apparently useful for our network and helps it in figuring a set of rules or patterns for image classification. Following are the **four representations of 8** as it traverses the network.

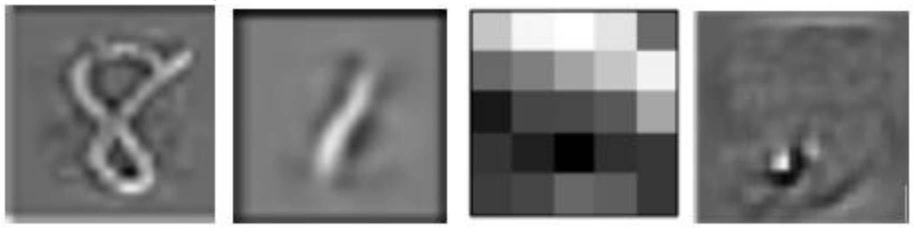


Figure 3: Feature maps of 8

d) Effect of L2 regularization term

By adding a regularization term to our loss function (cross entropy), not only have we prevented the possibility of overfitting but also have seen a slight rise in the accuracy values. The original accuracy of 99.19% after adding a regularization term has shown an increase of 0.2%. The fine tuning of the parameter was needed, so we changed the value of the parameter from 0.1 to 1.0 and saw the best result was achieved at the value of 1. The regularization parameter is given a value of 1 and the accuracy after adding the regularization term is 99.39% on the MNIST Image dataset.

Part 2: Image Recognition on Preprocessed CIFAR-10

a) Preprocess CIFAR-10 dataset

Pre-processing of images is important for instance, that we need to convert the image to a form that's according to the requirement of the network or that is suitable to be processed in the later stages. For our case of CIFAR dataset, before preprocessing it is important knowing the dataset in order to find the necessary preprocessing operations.

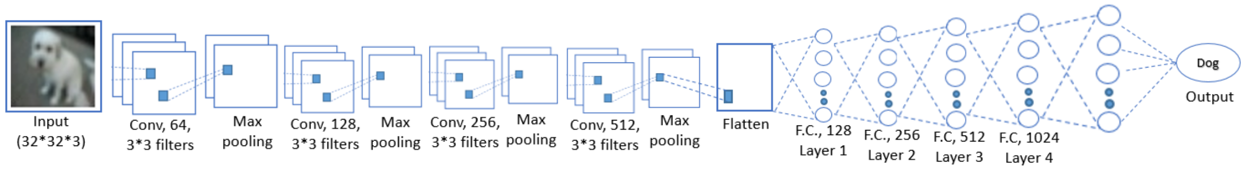
Here, we have used two ways of preprocessing the data. Sometimes, preprocessing is critical for the system, and has the potential to improve network's performance making it robust and scalable. CIFAR-10 dataset is a colored dataset containing images of 10 classes and we do the following two operations:

1) **Min-max normalization**: We apply the normalize function under the intuition that having the data in a smaller range will help us in preserving the relations in a lower dimensional space. The min-max function can be mathematically formulated as: y = (x-min) / (max-min)) but we can simply use a numpy function and it returns normalized numpy array by taking the input data x. By doing so, we transform our original data to a range of 0 to 1. The idea of using a normalization at a preprocessing step is that it will avoid the problem of exploding or vanishing gradient values when we perform back propagation for optimization. Therefore, keeping all the values in the range of 0 to 1, will help us tackle the problem occurring at a later stage. In our simulation, we also used Sigmoid and ReLU function to normalize the data, but the best results were achieved by using min-max normalization.

2) **One-hot encoding**: To express the probabilities in the network implementation, we need to have a vector with the same number of elements as the number of classes. Hence, we one-hot encode the 10 classes into 10 vector of size 10 each.

b) 1. Network architecture

The process of designing a network architecture for this particular dataset, was a bit different. Considering the preprocessing step of normalization and one hot encoding, we have successfully transformed the original input data to a process able form. A summary of the network details is given below:
Total examples: 60,000
Total Training examples: 45,000
Validation examples: 5,000
Testing examples: 10,000
Preprocessing: Yes
No. of layers: 14
Details of layer: 4 convolutions, 4 pooling, 1 flattening, 5 fully connected layer
Activation function used in layers: ReLU
Loss function: Cross Entropy + L2 regularization
Evaluation Function: Accuracy



Input Layer: The input layer has 32*32 sized **colored** images of 10 classes, {'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'}, belonging to these categories. The given dataset has 60,000 images, out of which 6000 images per class. Here, since the image is colored, we have to take care of the RGB channels too. Those RGB channels are one reason why we need to an additional step of pre-processing here. After the preprocessing step, the original data is going to be transformed to the range of 0 to 1 and be represented as a one hot vector. The other details of the intuition of formation of the model is similar to the first part of image classification using MNIST dataset. Just that, the model is certainly driven by the data, and we need to make changes into the format of the data entering and leaving the system based on the model and dataset chosen. Details about the rest of the layers chosen are as follows:

1. **Convolution Layer 1**: The first layer has **64** different filters of size **3*3**, with stride of 1 and zero padding applied, this makes the size of the outputs to be same as the size of the inputs.
2. **Max Pooling Layer 1**: Then, using a max pooing layer to reduce the parameter counts and computational complexity. We use the **ReLU** activation function and the

batch normalization. The batch normalization brings the values in a certain range which makes it convenient for computation in future steps.

3. **Convolution Layer 2**: The first layer has **128** different filters of size **3*3**, with stride of 1 and zero padding applied, this makes the size of the outputs to be same as the size of the inputs.
4. **Max Pooling Layer 2**: Then, using a max pooing layer to reduce the parameter counts and computational complexity. We use the **ReLU** activation function and the **batch normalization**.
5. **Convolution Layer 3**: The first layer has **256** different filters of size **3*3**, with stride of 1 and zero padding applied, this makes the size of the outputs to be same as the size of the inputs.
6. **Max Pooling Layer 3**: Then, using a max pooing layer to reduce the parameter counts and computational complexity. We use the **ReLU** activation function and the **batch normalization**.
7. **Convolution Layer 4**: The first layer has **512** different filters of size **3*3**, with stride of 1 and zero padding applied, this makes the size of the outputs to be same as the size of the inputs.
8. **Max Pooling Layer 4**: Then, using a max pooing layer to reduce the parameter counts and computational complexity. We use the **ReLU** activation function and the **batch normalization**.
9. **Flattening Layer**: Now, here is a tricky part, since we have a 3D convolving layer, so before we pass it on to the fully connected layers, we need to flatten it to 1D. The way we do it is by using the numpy.flatten() operation.
10. **Fully connected layer with 128 units**: Taking the output from the flattened layer, it is passed on to a fully connected layer. We use a dropout layer, where we use variable called keep_prob that tells the program whether we want to use a **dropout** layer or not. Dropout layer also helps in fighting the overfitting by removing some of the neurons at the time of training. We also perform **batch normalization** here, which ensures convenience of computations at layers.
11. **Fully connected layer with 256 units**: Taking the output from the last fully connected layer with 128 units layer, it is passed on to a fully connected layer. We perform **dropout** and a **batch normalization** here, just like the previous layer but with different number of units.
12. **Fully connected layer with 512 units**: Taking the output from the last fully connected layer with 256 units layer, it is passed on to a fully connected layer. We perform **dropout** and a **batch normalization** here, just like the previous layer but with different number of units.

13. **Fully connected layer with 1024 units**: Taking the output from the last fully connected layer with 512 units layer, it is passed on to a fully connected layer. We perform **dropout** and a **batch normalization** here, just like the previous layer but with different number of units.

14. **Fully connected layer with 10 units**: The units chosen in this layer makes sense as the number of classes are 10, and we have to classify our data into 10 classes. Practically, no activation function is used here, for all the other previously mentioned fully connected layers, I have used the ReLU function, but for this layer, we have not used any activation function. The intuition is from the increasing number of the convolutional layers, and the increasing units of the fully connected layers, we have extracted the quality function. I tried using the softmax activation function for this layer, but the result by using No activation function is better than using a softmax activation function.

b) 2. Effect of stride size and filter size

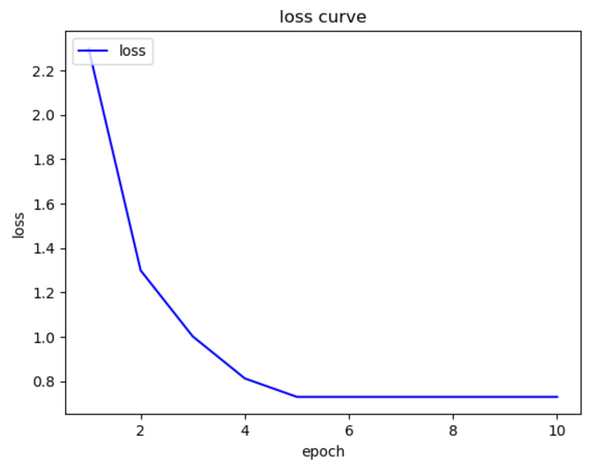
The fact that if we use more filters in the convolution layers, we don't want to lose the important features of the images entered. We want to preserve the referential structure of the images, and hence, we use optimal number of features, which is why for our model we have used the number of filters by power of 2 after every alternative convolution and pooling layer. The filter size is determined by two criterions on which we want to base our model.

One is that low level features are local and that the what is useful at one place might be useful at other places too. If we go by the first rule, and choose a 1*1 kernel, it means that low level features are per pixel, and that we should apply the same operation to all pixels making don't affect neighboring pixels at all. If we go by the other criteria, and consider the other extreme case that the size of filter is equal to the size of the input image, then in that case, it does not make sense to call it a CNN at all, as you are not considering the low level features in an image at all.

As for the stride size too, if we consider smaller strides, it means we will have more overlaps, will means overall large volumes. In the case when we consider the bigger strides, we have less overlaps, and thus lesser memory will be used, and the computation of output will be easier as it will have to operate on smaller inputs. Along with this it also simplifies the model and leads to avoiding overfitting. For our case here, we have considered 3*3 filters of changing size of number of filters by the power of 2, and the stride of 1 with zero padding.

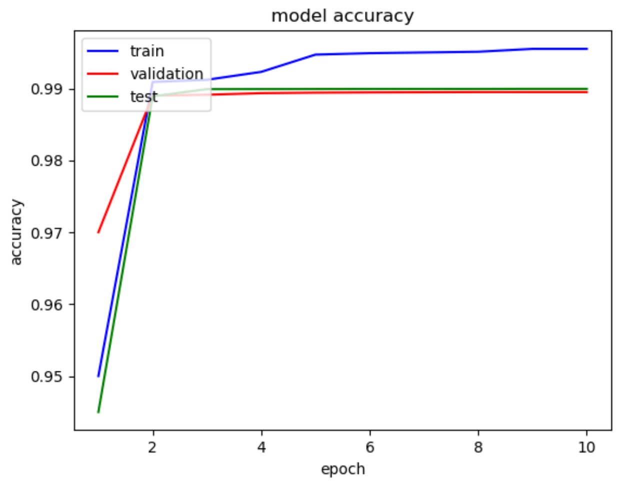
b) 3. Learning curve

The learning curve plotted based on the proposed model for image recognition of CIFAR-10 dataset is given below. The graph shows the loss value for 10 epochs.



b) 4. Accuracy of Training and Test Sets

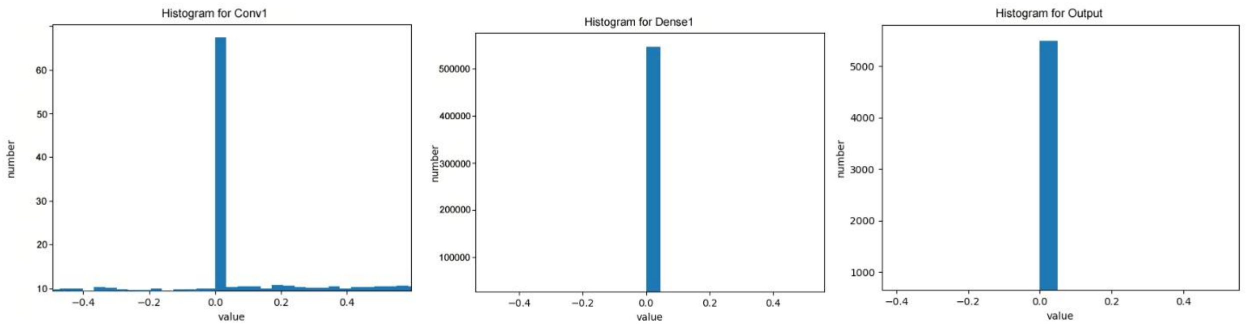
The process of plotting the following curve was the same as in part-1, we fit the model training, validation and testing data to model. In our simulation we tried reducing the training data size to 20,000 and were able to achieve an accuracy of 65.16% for 50 epochs. We decided to run on a smaller dataset, because of the time it takes for computation. The current accuracy is 74.08% with training on 55,000 training data for 10 epochs. Following is the curve:



The sharpness in the graph above is due to plotting the graph with epochs rather than iterations.

b) 5. Distribution of weights and biases

The distribution of weights and biases, shows the following few observations. Data is very sparse and since the input is normalized in the range of 0-1, the values plotted are particularly small. Activation functions uses are all ReLU and I purposely after each activation perform normalization as at every step only a small fraction of neurons is being activated.
Since our model has 4 convolutional layers and 5 dense layers, hence following is histograms after convolutional layer with 512 different filter, then the histogram of weights after dense layer with 1024 units and last one is the histogram of weights after output layer with L2 regularization.



c) Correctly classified and miss-classified images

The correctly identified and miss classified images have helped us to identify the corner cases in the CIFAR-10 dataset. The low confidence data, that is closer to the decision boundary, the cases because of which even in the best case we are losing out on an accuracy of 25.93%. To check those hard cases, we will present some of the cases that are correctly and incorrectly classified in our dataset.

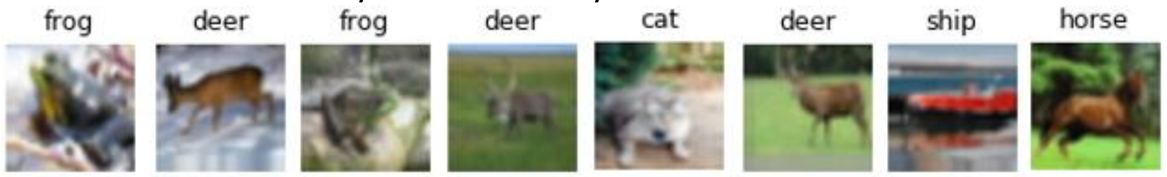


Figure 4: Correctly classified images in CIFAR 10 dataset

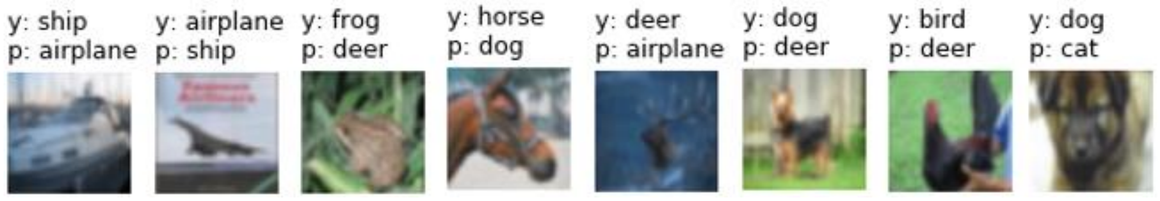


Figure 5: Miss-classified images in CIFAR 10 dataset

d) Feature maps of different convolutional layers

The feature maps of the initial few layers, just show some few pattern on the image, some random effect on the image, which does not make sense at first, but when we reach the last stage, we start to see patterns of different objects on a particular image passed. Shown below are the feature maps after 3rd, 5th, and 7th layer and dense layer with 128, 256, 1024 units.

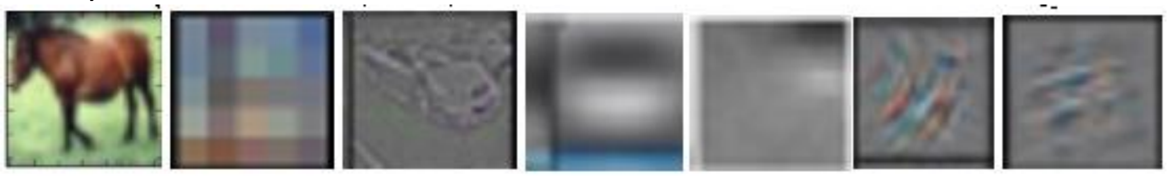


Figure 6: Feature maps of horse class in CIFAR10

e) Effect of L2 regularization term

The use of a regularization term depends a lot on the size of the dataset we are choosing, if the dataset size is small, the data has rarely a chance of overfitting, so using a regularization term is reasonable if dataset size is large. For our case, we are using the value of the hyper parameter as 1.0. This is just to show its validation and the change in result after using the regularization term is very slight. Since our proposed uses several dropout layers, hence we did not observe a significant impact of using L2 normalization. The original accuracy of 74.081% moved to 74.084%.

Inferences from the result

From this homework, I have learnt:

1) Writing the code made me understood the implementation of basic neural network components such as the convolution layer, dropout layer, sigmoid, ReLU from scratch.
2) The importance of the selection of hyper parameters such as learning rate, the impact of which can lead us to having multiple layers or just a few layers to get the same result.

3) Selection of the activation function best suited for the problem at hand. Although there is no proper technique to find out which activation function will work for which case.