EE569 Digital Image Processing

HOMEWORK # 5

Aagam Shah

USC ID-8791018480

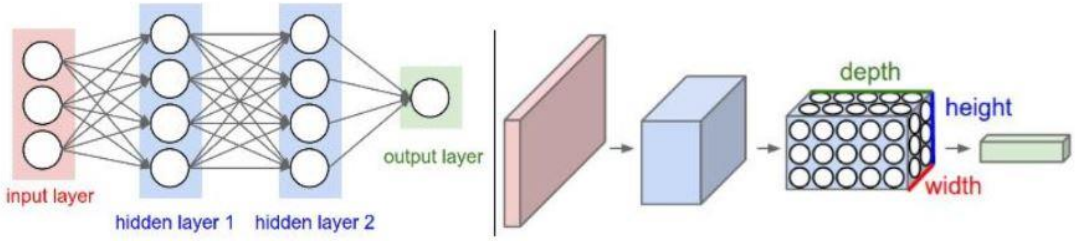
USC Email – aagamman@usc.edu

Submission Date – May 3, 2020

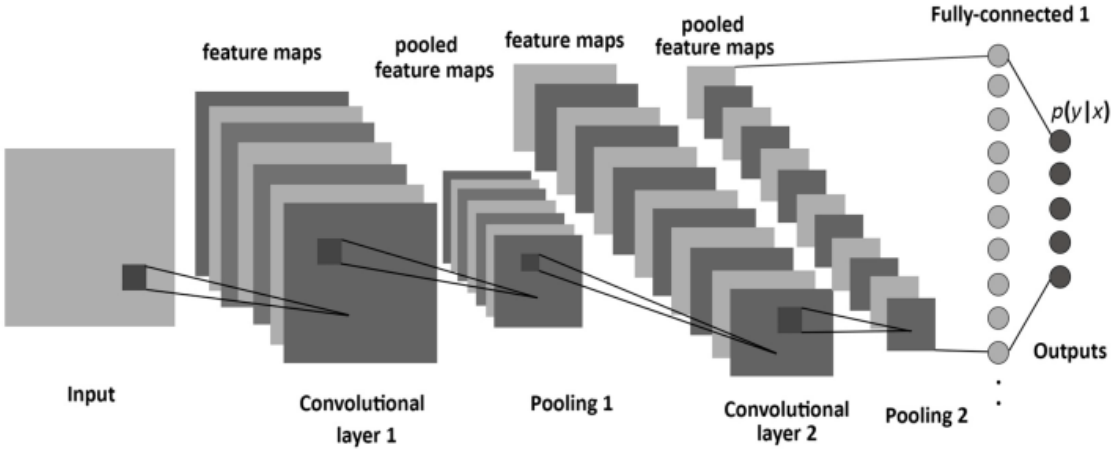
Problem 2: EE569 Competition --- CIFAR10 Classification

1. MOTIVATION

A Convolutional Neural Network (CNN) is of Deep Neural Networks, which is a category of Deep Learning. Basically, the terminology the Convolutional Neural Network is nothing but the mathematical operation which is convolution which is used in place of the matrix multiplication of the layer of the Neural Network.



Convolutional Neural Network technically is a self-learning procedure which basically performs the extraction of features from the input data and then it will perform the classification automatically but at the same time it also gives less classification time, but it also makes the network computationally intensive. As being computationally intensive for larger datasets we use CNN here for the smaller dataset for the CIFAR 10 classification.



Previously, we used simple LeNet -5 architecture to train the simple convolutional neural network for the CIFAR – 10 datasets but the accuracy yielded was too less, approximately around 70% which needs to be improved. LeNet – 5 architecture is basically the feedforward convolutional neural network where the artificial neurons react to the nearby neurons and then progressively following in similar fashion perform well on the large-scale image processing operations.

2. LOGIC & DISCUSSION:

As, we know that the base and shallow CNN model yielded less accuracy, so basically in order to achieve high accuracy we aim to build a deep and dense neural network in order to achieve high accuracy along with keeping the model size or the total number of model parameters as low as possible, i.e., doing a tradeoff between the two. Here I aim to improve the base CNN model for the CIFAR 10 dataset in order to obtain the high classification accuracy for the training as well as the test dataset. In order to achieve so I have tried to modify the base CNN model with various changes as described below. The idea was basically taken from the paper "*All Convolutional Net*".

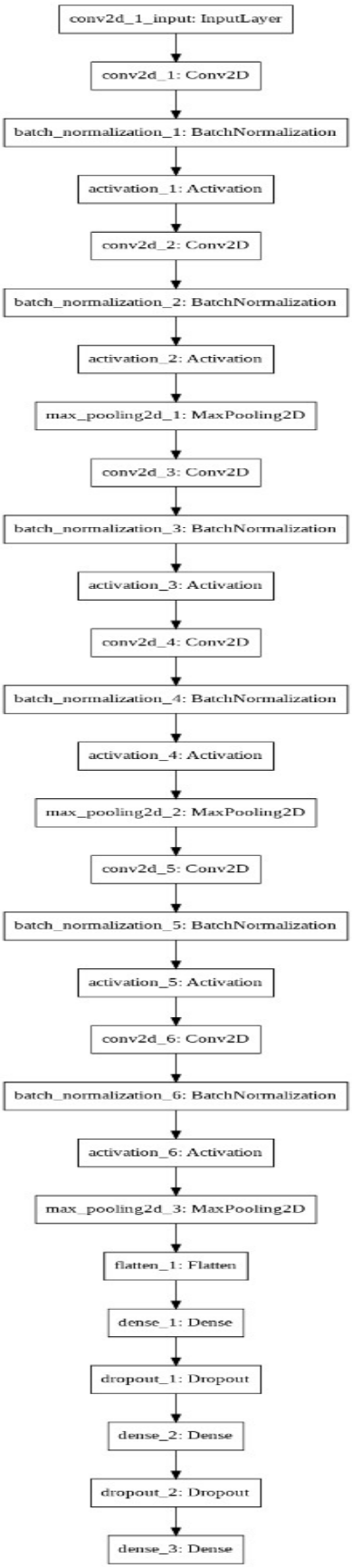
The architecture can be described as follows:

The modified Convolutional Neural Network architecture is very similar to the baseline architecture of the LeNet 5 architecture. The modified Convolutional Neural Network also consists of the input layer and an output layer along with multiple hidden layers in between the input and the output layer. These hidden layers are nothing but the Convolutional Layers, Max Pooling layers and the Normalization layers. The function and the significance of each layers have been described in detail in the previous question number 1 of Homework 5.

So, based upon the understanding of each layers i.e., the Convolutional layers, the Max pooling layers and the Normalization layers, I apply those concepts here and have tried to construct the modified LeNet 5 architecture with the limited hardware resources available in order to achieve as high accuracy as possible along with keeping in mind the model size or the model parameters to be as low as possible, thereby keeping a tradeoff between the two constraints.

Here I have in total used 6 Convolutional Layers with 2 at a time in one chunk. Each Convolutional Layer is followed by the max pooling layer and then the Batch normalization layer whose significance and the role of each layer and its contribution towards the optimum result has been explained in detail in the following section. Each chunk of Convolutional layer, max pooling layer and the Batch Normalization Layer is also followed by the Dropout layer in order to perform dimensionality reduction which is necessary in order to deal with the deeper layers of the Convolutional Neural Network. The number of filters for each Convolutional layer are also varied and experimented with different values and the significance of choosing a particular value for this approach is also explained in the following section with a reasonable explanation. The Flowchart for the architecture can be summarized as given below:

EE569 Digital Image Processing



EE569 Digital Image Processing

The various parameter setting done in order to reach the result below:

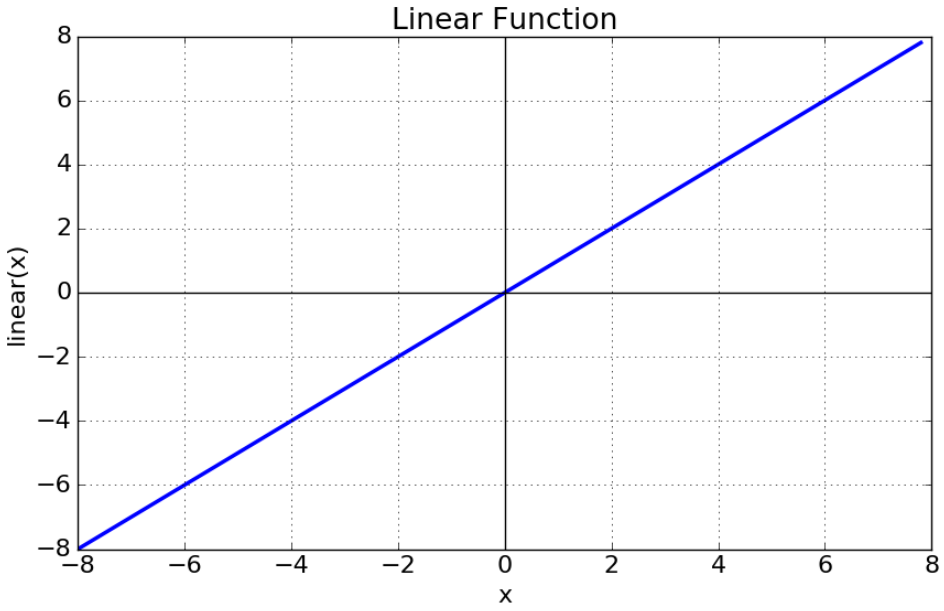
i. At first, I **increased the depth of the neural network** by the addition of few more convolutional layers along with the Max pooling layers. The benefit of adding more layers is basically to learn the features at multiple layers of abstraction. To explain it in detail, the first layer of the convolutional neural network will get trained itself in order to be able to recognize the basic features from an input image like edges, etc. Then the succeeding layer will use the features of the previous layers such as edges in order to get trained. Similarly, the next layer will use the features of the previous layers in order to detect the high-level features progressively such as legs, tails, wheels, wings, etc. Following the same analogy, the final layer which is the most high-level layer will recognize the objects such as dog, cat, ship, airplane, truck, etc. using the high-level features obtained from the preceding layers. So, basically to detect the low-level features we use less number of layers and then progressively we keep on increasing the number of layers as we go for detecting high level features. So, therefore initially I have used two Convolutional layers with 32 filters in each to detect the low-level features followed by the max pooling layers and the dropout layer for the dimensionality reduction and discarding the unwanted nodes respectively. Then later on to detect the mediocre level features I have used two convolutional layers with 64 filters in each and then last but not the least for the detection of high-level features I use more number of filters i.e., the two convolutional layers with 128 filters in each layer. I have also performed the padding at each layer in order to prevent the loss of information from the boundary of the images.
   Also, the value of the **dropout layer** is kept less initially in order to keep the prominent features to train the initial layers optimally and then dropout is increased gradually to discard the features with least importance in the high level layers to make it computationally less intensive and decrease the model size or the model parameters.

ii. Next, I choose the **kernel size as 3x3** for Convolutional layer to detect the prominent features locally more well as compared to the 5x5 kernel size which in turn increases the computation and detects less local features and then I used maximum pooling layer operation for dimensionality reduction with kernel size 2x2 to down sample the input and keeping only valuable information and throwing away the unwanted information. The **stride parameter** used is also 1 with overlapping sets in order to prevent the loss of information while sliding the kernel from left to right and top to bottom.

iii. Then I added the **weight decay factor** in order to perform the regularization which is infact necessary to carry out the learning of the weights to the master the features to the generalization of features as the number of weight parameters increases gradually.

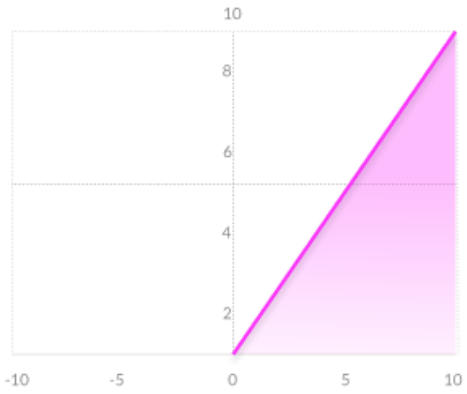
iv. Next, I tried with 4 different **Activation Functions** i.e., ReLU, LeakyReLU, elu and Linear, out of which I found out the best performance was given by the elu Activation Function. Activation function is basically the mathematical operator which will determine the neural network's output. It is basically used to add the non-linearity to the neural network which in turn determines which neuron should be kept or fired depending on the input of the neuron which is relevant to the prediction of the model.

- The **Linear Activation function** is proportional to the input.

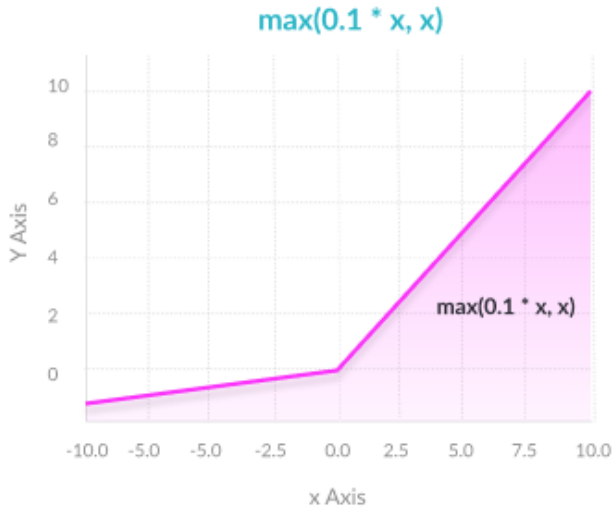


As we can see that the gradient here doesn't become zero but instead it stays constant without being dependent on the input which shows that the biases as well as the weights will get updated while performing the backpropagation procedure, which implies that the error rate would not get improved using this function as the gradient function here remains same for all the iterations.

- The **ReLU (Rectified Linear Unit)** is nothing but a basic type of non-linear activation function widely used among all. The crucial part of using this activation function is that it basically does not activate all the neurons at similar time, which in simple terms means that if the output of the linear transformation comes out to be zero then only the neuron will be deactivated. As, this results in the activation of only certain neurons so the ReLU function is quite computationally efficient as compared to other activation function such as sigmoid or tanh.

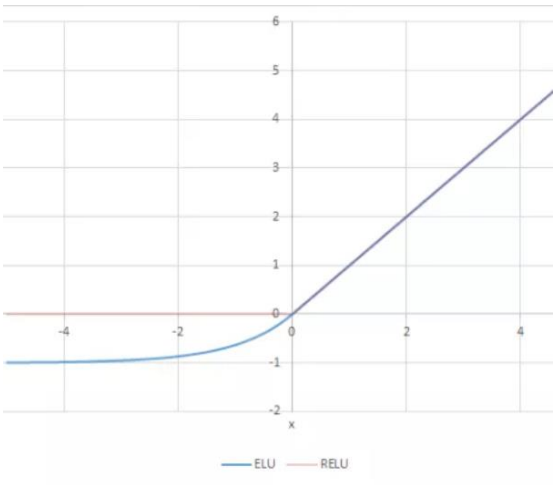


- Next is **Leaky ReLU,** which is noting but an improvised version of the ReLU activation function the only difference about Leaky ReLU is that it is defined for the very small portion of the negative values of x.



With this improvisation in the ReLU function in order to achieve the LeakyReLU function the left gradient region of the graph remains no more zero and hence we do not find anymore the dead neurons in that particular region.

- The other one I tried was the **elu (**Exponential Linear Unit**)** activation function which is nothing but the improvisation of the pervious one, the slope on the negative side of the function is modified which means that instead of just minor portion of the negative slope which was a straight line in case of the LeakyReLU function, is now a logarithmic curve for the negative values. This particular activation function yielded me the highest accuracy without overfitting the data.



- The last but not the least, is the **Softmax** function which is nothing but the combination of the multiple sigmoid functions. This activation function is basically used in the last dense layer for the classification of the multiclass problem. This activation function basically returns the probability value of the datapoint of interest which is belonging to a particular class.

$$\sigma(z)j = \frac{ezj}{\sum_{k=1}^{k} ezk} \quad for\ j = 1....k.$$

To explain it in detail, in the multiclass problem the output layer will consist neurons as much as the number of classes present in the target. So, after applying softmax activation function we will basically get the probabilities of each datapoint which will be belonging to a particular class.

v. Then as we know that the data requirement increases as the network goes more deeper for which we perform the **data augmentation** technique. In data augmentation, particularly in Keras we have an inbuilt function known as Image Data Generator whose role is basically to create batches of specific sizes as given of the tensor image data. The data is basically into loop which means it is continuously given into batches for indefinite times. Basically, in the Image Data Generator the tensor image data is transformed from the actual input image data by performing various operations such as translation, rotation, shearing, reflection, stretching, shifting, normalization, etc.

vi. However, as the network goes more deeper the issue of overfitting becomes the more prominent problem, so in order to prevent this situation to happen I inculcated the dropout layers after every max pooling layer as explained earlier. Next is the **kernel_regularizer** which is nothing but the l2 regularization of the

weights. In simple terms, it allows to incorporate the penalties in the loss function during the optimization step. This basically decays the weights of all the inputs during the gradient descent and the l2 regularization finally decays the weight linearly.

vii. I also implemented the **Batch Normalization** in order to take care of the internal covariate shift issue. Basically, this technique is used to normalize the output from the activation function of the previous layer in each batch, which means that it transforms in such a way that the mean of the activation stays near to 0 and the standard deviation of the activation stays close to 1. Batch Normalization basically helps to achieve high accuracy with less number of training steps thereby increasing the computation speed of the training process.

viii. The last parameter I varied is the various types of **optimizers**. Basically, I tried with 3 different types of optimizers i.e., the RMSprop, SGD and the Adam. Out of the Adam optimizer performed really well and yielded the optimum results among all 3. Adam is basically is an gradient based optimization algorithm of the stochastic objective functions. It has the perks of 2 SGD, RMSprop and AdaGrad at the same time. While the SGD is the gradient descent variant, it basically performs the computations on small dataset or some random examples instead of performing the computations on the whole dataset, which leads to the decrease in accuracy with some information loss and also with low learning rate. While RMSprop is also one of the variant if the gradient descent algorithm with momentum. Basically, it restricts the vertical oscillations thereby allowing larger steps only in the horizontal directions and faster convergence.

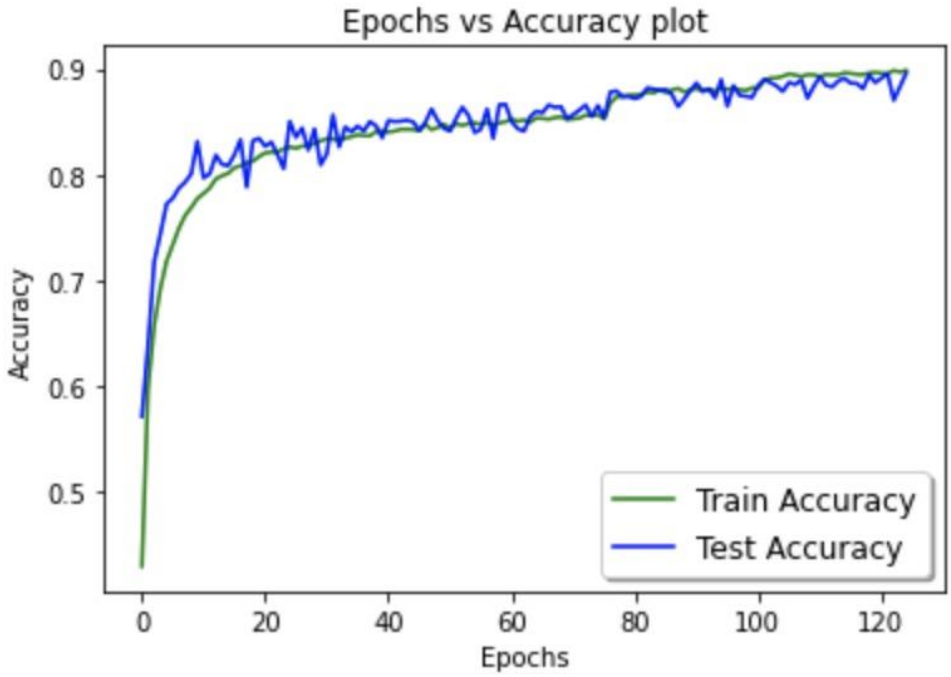
ix. Also, in order to avoid overfitting issue, I have implemented the **Early Stopping Technique**, which is basically to prevent the too many executions of the iterations in the training procedure which means that once the accuracy stops changing much the iterating process is stopped.

3. EXPERIMENTAL RESULTS

Classification accuracy:

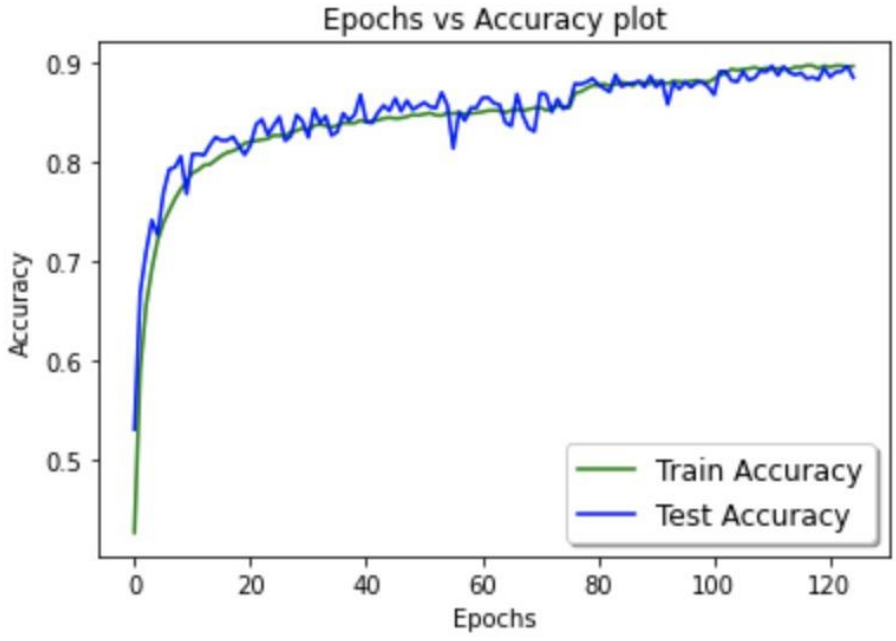
The best train accuracy achieved is **90.530%.**
The best test accuracy achieved is **89.210%.**
The training time is approximately ~ **1.5 hrs.**
The inference time (prediction time) is approximately ~ **1 minute**
Number of epochs = 200
Kernel Size = 3x3
Stride parameter = 1
Activation Function = Elu
Optimizer = Adam
Model Size = Approx. 300k parameters

For Full Supervision:

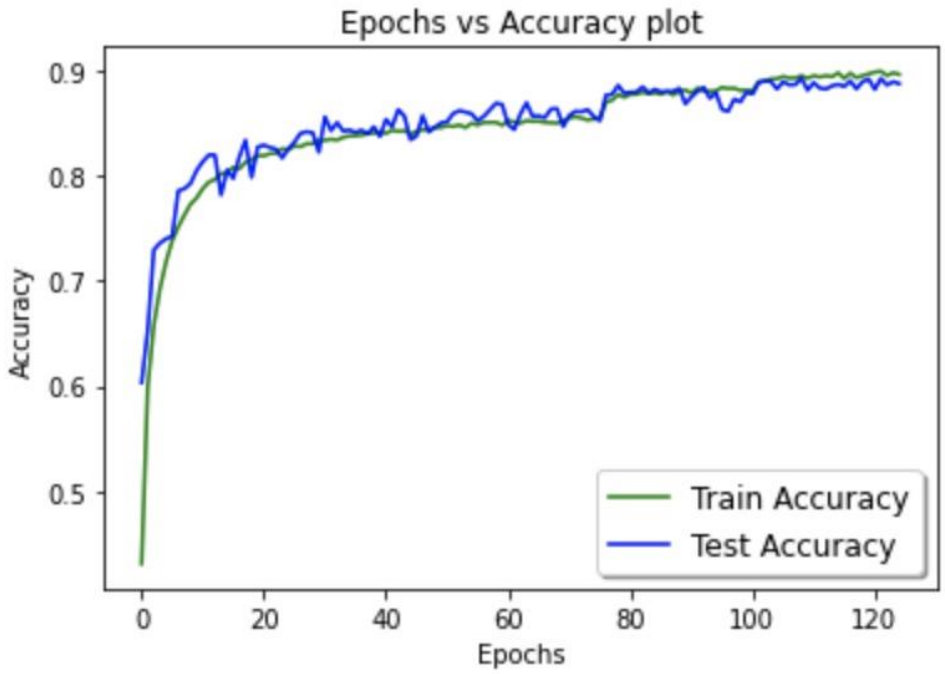


For Weak Supervision:

a) For (1/2): Testing accuracy = **88.9%**



b) For (1/4): Testing accuracy = **87.8%**



Model Size:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
```

```
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
_________________________________________________________________
```

Variations tested:

The other 2 best variations tested with almost same training time as well as the inference time are as follows for Full Supervision:

I. **Activation Function = ‘elu’**
   Optimizer = ‘RMSprop’
   Training Accuracy = 89.8%
   Testing Accuracy = 88.070%

II. **Activation Function = ‘LeakyReLU’**
    Optimizer = ‘Adam’
    Training Accuracy = 89.07%
    Testing Accuracy = 88.730%

EE569 Digital Image Processing

System Specifications:

The reported training time and the inference time as well as other parameters are based on the experiments performed using Google Colab GPU.

References:

1. Discussion slides
2. "*Striving for Simplicity: The All Convolutional Net*" technical research paper.
3. Applied Machine Learning Blog
4. Machine Learning Mastery Blog