OPTIMIZED NEURAL NETWORK FOR IMAGE CLASSIFICATION USING PRUNING TECHNIQUES

A PROJECT REPORT

for

ITE2010 – ARTIFICIAL INTELLIGENCE

by

17BIT0001 - VARUN MUPPALLA

17BIT0233 – ASHISH KUMAR

SUBMITTED TO:

PROF. DR. GERALDINE BESSIE AMALI D.

Abstract

Image classification is playing a big part in most industries today. It is going to be one of the fundamental applications of artificial intelligence in the upcoming decades. The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm known as a convolutional neural network. In this project we use artificial neural networks as classifiers to classify theimages into 10 sets of images.The dataset taken is that of fashion MNIST dataset. For these purposes a total of 60000 images have been used for training the neural network and the rest of 10000 are used for testing. The major problem as tackled by this project is that of neural network time issues and hyperparameter optimization. Hence, herein we use in this project we implement a convulutional neural network using built in python libraries to classify the fashion MNIST dataset and later we compare it with a pruned artificial neural networks. We use different techniques of artificial neural network pruning techniques to create and test the data precision and accuracy.The testing is to see whether of not a smaller neural network can actually increase the dataset accuracy in a less memory and time consumption way.

KEYWORDS – Convolutional Neural Network, Image classification

I. INTRODUCTION

We are constantly analyzing the world around us. Without conscious effort, we make predictions about everything we see, and act upon them. When we see something, we label every object based on what we have learned in the past. Similar to how a child learns to recognize objects, we need to show an algorithm millions of pictures before it is able to generalize the input and make predictions for images it has never seen before. Computers 'see' in a different way than we do. Their world consists of only numbers. Every image can be represented as 2-dimensional arrays of numbers, known as pixels. But the fact that they perceive images in a different way doesn't mean we can't train them to recognize patterns, like we do. We just have to think of what an image is in a different way. To teach an algorithm how to recognize objects in images, we use a specific type of Artificial Neural Network: a Convolutional Neural Network (CNN). Their name stems from one of the most important operations in the network: convolution. Convolutional Neural Networks are inspired by the brain. Research in the 1950s and 1960s by D.H Hubel and T.N Wiesel on the brain of mammals suggested a new model for how mammals perceive the world visually. They showed that cat and monkey visual cortexes include neurons that exclusively respond to neurons in their direct environment. In this project we demonstrate the process of building a convolutional neural network model and train it against the fashion MNSIT dataset so that we can classify based on the labels it has.Later in this project we create a pruned artificial neural networks for the same classification problem and compare the two.

II. LITERATURE SURVEY

The following table represents the information that I have found from researching similar techniques and related topics.

S. No.	Title of the Paper and Year Published	Techniques Used	Relevance	Future Scope
1)	Image classification using artificial neural networks: An experimental study on Corel database 2011	Artificial neural network, back propagation.	In this paper high-level image classes are inferred from low-level image features like color and shape features with the help of artificial neural network. Back propagation neural network algorithm is used for integrating knowledge as well.	The algorithm has a fairly low accuracy of 83.21%. We can use optimization techniques to make the model more efficient.
2)	Artificial neural networks and other methods of image classification 2007	Artificial Neural Networks (ANN), Support Vector Machines (SVM), Fuzzy measures, Genetic Algorithms (GA), Fuzzy support Vector Machines (FSVM)	Demonstrates different accuracy values for traditional classification methods.	Improve the automatic techniques, and apply CNNs for a higher accuracy result. This paper also fails to consider overfitting between different models.
3)	Convolutional Neural Network (CNN) for Image Detection and Recognition 2006	CNN	Uses CNN to demonstrate classification of a UCI lung cancer dataset.	Try to improve the algorithm through heuristic algorithms.
4)	Design of Artificial Neural Network Architecture for Handwritten Digit Recognition on FPGA 2016	2 layer feed-forward artificial neural network	Optimizes the regular neural network using node pruning and tree pruning	N/A
5)	Computational Complexity Of Neural Networks: A Survey 2011	ANN, CNN, ANN with back propagation	This paper compares the time complexity of various neural networks for image classification	N/A

6)	Pruning Convolutional Neural Networks for Image Instance Retrieval 2010	CNN, node pruning	In this paper CNNs are optimized by pruning them. This comes at a loss of accuracy but results in the neural network to be 3% faster for large datasets over 35000 entries	Different ways of optimization can be tried which lead to lesser loss of accuracy.
7)	Deep Convolutional Neural Networks for Image classification 2014	Deep convolutional neural network, and small text mining	It exploits the use of neural network for performing sentiment analysis which is the aim of my project based on facial expressions	Improving the accuracy of the model by providing a mask for less significant features to aid in error detection in the model.
8)	Improving optimization of convolutional neural networks through parameter fine-tuning 2019	CNN, hyperparameter tuning	This model compares the original CNN built and tests it against an optimized version of the same network by using parameter tuning method and feature reduction	As this is a comparative study not much is to be improved.
9)	Optimization of convolutional neural network parameters for image classification 2017	CNN	Method is proposed to make an improvement on the accuracy of the CNN by two means. One is by increasing the number of layers and another is to reduce the image size of the window.	As this is a comparative study not much is to be improved.
10)	Very deep convolutional neural network based image classification using small training sample size 2015	CNN, Regularisation, Batch normalisation	Uses regularization and batch normalization on CNN to fit small datasets with simple and proper modifications and don't need to re-design specific small networks.	We can think of optimizing the overfitting problem by removing some of the nodes in the network to make it slightly lightweight.

III. PROPOSED METHODOLOGY:

1.Convolutional Neural Networks:

The convolutional neural network (CNN) is a class of deep learning neural networks. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics. The architecture of a CNN is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Architecture:

Convolution Layer:

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

Strides:

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Non Linearity (ReLU)

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = max(0,x)$.
- Why ReLU is important :ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

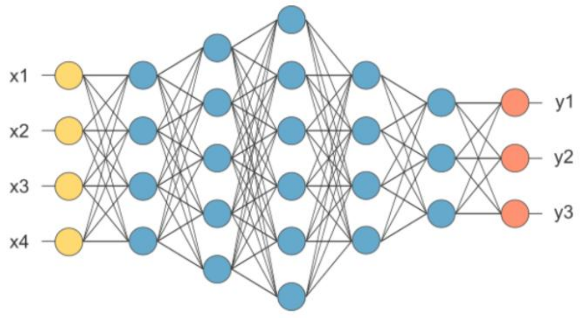
There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

Pooling Layer:

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling is also called sub sampling or down sampling which reduces the dimensionality of each map but retains important information.

Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



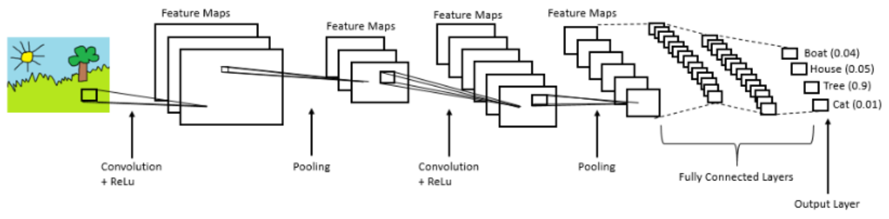
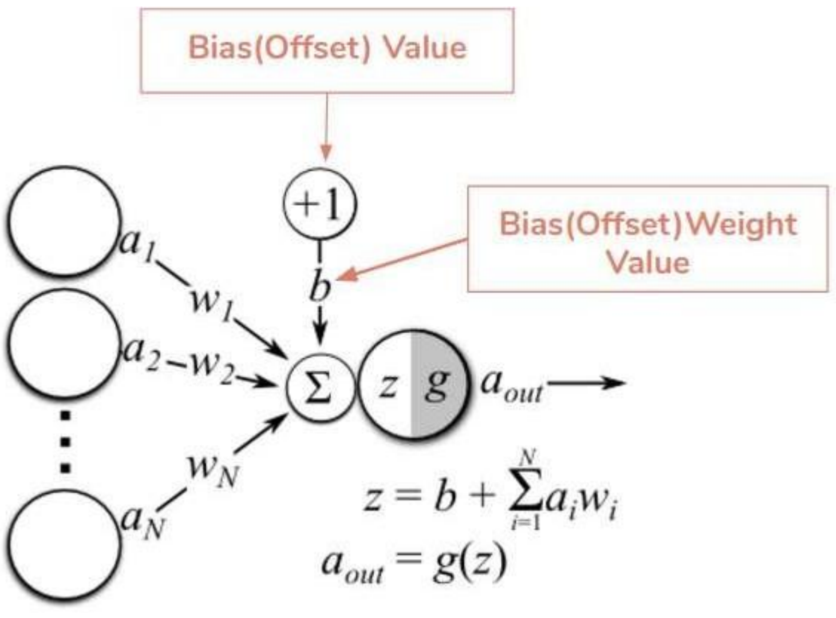


Figure 10 : Complete CNN architecture

ANN (Artificial Neural Networks):

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by,but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task- specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.



<u>Artificial Neural Network Structure</u>

To summarize, the ANN Layer:

Accepts a which receive input, combine the input with their internal state (activation) and an optional threshold using an activation function, and produce output using an outputfunction.

The network consists of connections, each connection providing the output of oneneuronasaninputtoanotherneuron.Eachconnectionisassignedaweight that represents its relative importance. A given neuron can have multiple input and outputconnections.

The propagation function computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A bias term can be added to the result of thepropagation

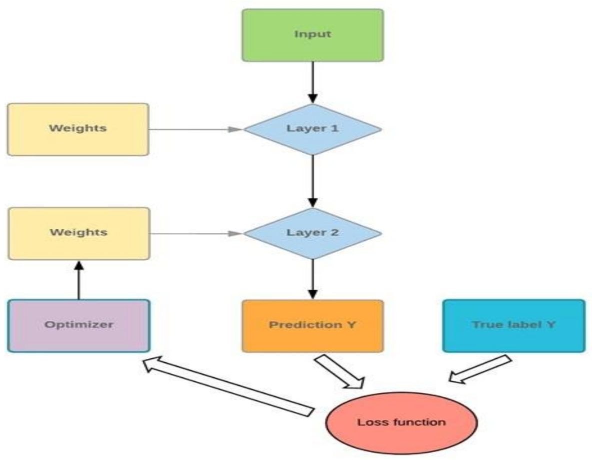
The neurons are typically organized into multiple layers, especially in deep learning. Neurons of one layer connect only to neurons of the immediately preceding and immediately following layers. The layer that receives external data is the input layer.

A hyperparameter is a parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size.

Learning is the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result.

The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A high learning rate shortens the training time,butwithlowerultimateaccuracy,whilealowerlearningratetakeslonger, but with the potential for greateraccuracy.

While it is possible to define a cost function ad hoc, frequently the choice is determinedbythefunctionsdesirableproperties(suchasconvexity)orbecause it



arises from the model (e.g., in a probabilistic model the model's posterior probability can be used as an inversecost).

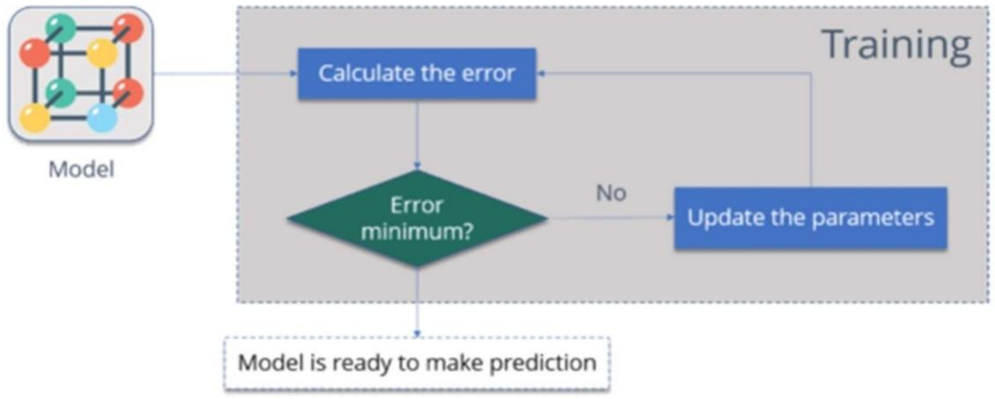
BACKPROPAGATION:-

Backpropagation is a supervised learning algorithm, for training Multi-layer Perceptrons (Artificial Neural Networks).

We need backpropagation while designing a neural network we initialize weightswithsomerandomvalues,butwecannotbesurethattheinitialweights we have selected will be correct or it fits our model thebest.

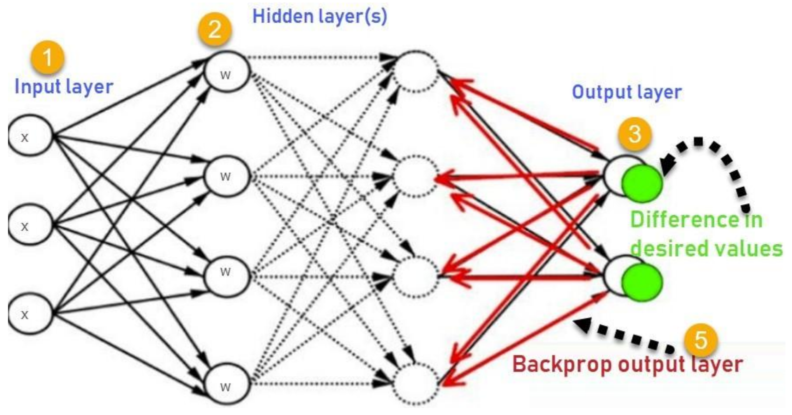
When we calculate our model output with our initial weights if we find our model output to be way different than the actual output, that is , the error value is huge then we need to somehow explain the model to change the parameters. One way to train our model is called Backpropagation.

To



summarize the steps for you:

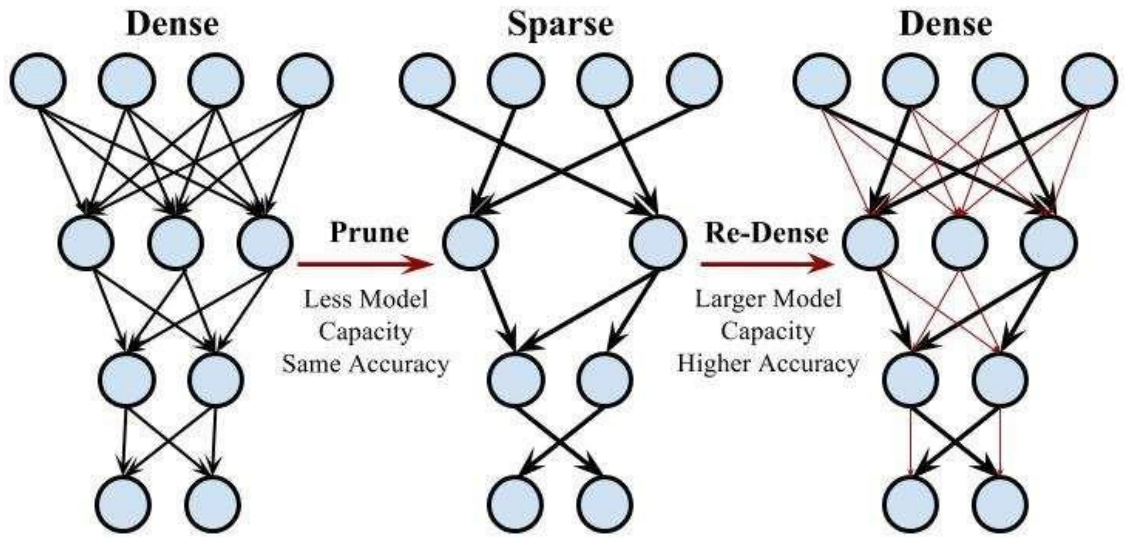
- **Calculate the error** – How far is our model output fromthe actual output.
- **Minimum Error** – We check whether the error is minimized ornot.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error we repeat the process until the error becomesminimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, we canfeedsomeinputstoourmodelanditwillproducetherequiredoutput.



WEIGHT PRUNING:-

Set individual weights in the weight matrix to zero. This corresponds to deleting connections as in the figure above.

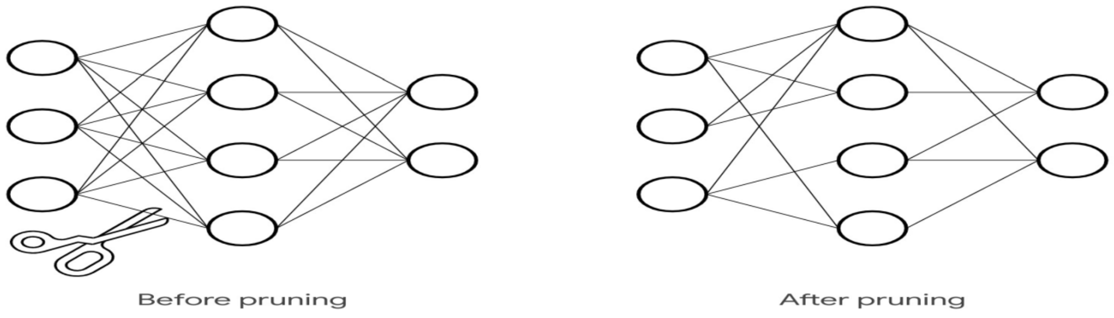
Here, to achieve sparsity of k% we rank the individual weights in weight matrix W according to their magnitude, and then set to zero the smallest k%.



NEURON PRUNING:-

Set entire columns to zero in the weight matrix to zero, in effect deleting the corresponding output neuron.Here to achieve sparsity of k% we rank the columns of a weight matrix according to their L2-norm and delete the smallest k%.

‘



IV. TOOLS AND SOFTWARES USED:

OS: Windows OS

Frontend: Python

Backend: Tensorflow, Jupyter Notebook

Libraries to be Used: NumPy, Pandas, Matplotlib, cv2 etc.

Software to be Used: Google colabcloud,spyder.

V. IMPLEMENTATION/CODE:

CNN

```
importtensorflow as tf
importnumpy as np
importmatplotlib.pyplot as plt

import time
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()



print("x_train shape:", x_train.shape, "y_train shape:", y_train.shape)
print("x_test shape:", x_test.shape, "y_test shape:", y_test.shape)



x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)



x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]


fromkeras.models import Sequential
fromkeras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D


model = Sequential()

model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu',
input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))

model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))


model.summary()
```

```
start1=time.time()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

fromkeras.callbacks import ModelCheckpoint


checkpointer = ModelCheckpoint(filepath='model_best_weights.hdf5', verbose = 1,
save_best_only=True)
history = model.fit(x_train,
y_train,
batch_size=32,
epochs=1,
validation_data=(x_valid, y_valid),
callbacks=[checkpointer])




model.load_weights('model_best_weights.hdf5')


score = model.evaluate(x_test, y_test, verbose=0)
end1=time.time()
print("Time taken=",end1-start1)

print('\n', 'Test accuracy:', score[1])
```

CODE FOR PRUNING

```
importtensorflow as tf
fromtensorflow import keras
import math
# Helper libraries
importnumpy as np
importmatplotlib.pyplot as plt
import time
fashion_mnist = keras.datasets.fashion_mnist#loading fashion MNIST dataset
fromnumpy import array
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_images = train_images / 255.0
test_images = test_images / 255.0



model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(1000,activation=tf.nn.relu),
keras.layers.Dense(1000,activation=tf.nn.relu),
keras.layers.Dense(500,activation=tf.nn.relu),
keras.layers.Dense(200,activation=tf.nn.relu),
keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
print('Accuracy Percentage=', test_acc)
predictions = model.predict(test_images)
np.argmax(predictions[0])
test_labels[0]


arr=model.get_weights()#this will store pruned weight matrix by individual weights
arr1=model.get_weights()#this will store pruned weight matrix neuron wise


defweightprune(percent):#function for weight pruning

layer=list()#to store layer where weight is present
row=list()
collumn=list()
weight_list=list()
forctr in range(0,7,2):
fori in range(0,len(arr[ctr])):
for j in range(0,len(arr[ctr][0])):
weight_list.append(abs(arr[ctr][i][j]))
layer.append(ctr)
row.append(i)
collumn.append(j)
   weight_list,layer,row,collumn=zip(*sorted(zip(weight_list,layer,row,collumn)))#sorting
all the weights of neural network with corresponding layer,collumn and row
weight_list=list(weight_list)
layer=list(layer)
row=list(row)
collumn=list(collumn)
fori in range(0,int((percent/100)*(len(weight_list)))):
```

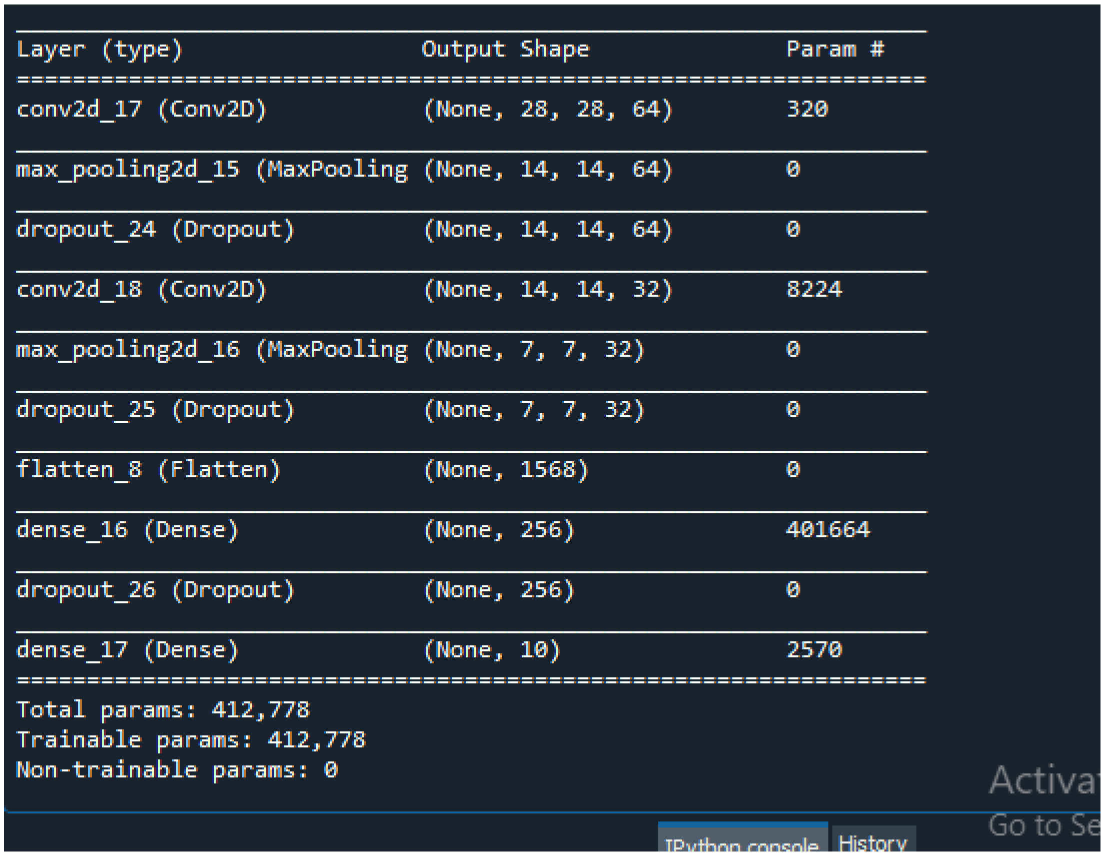
```
arr[layer[i]][row[i]][collumn[i]]=0
returnnp.asarray(arr)




defneuronprune(percent):#function for neuron pruning
sum_list=[[0]*1000,[0]*1000,[0]*500,[0]*200]
collumn_list=[[0]*1000,[0]*1000,[0]*500,[0]*200]
square_sum=0
forctr in range(0,7,2):
for j in range(0,len(arr1[ctr][0])):
fori in range(0,len(arr1[ctr])):
square_sum=square_sum+arr1[ctr][i][j]*arr1[ctr][i][j]
sum_list[int(ctr/2)][j]=(math.sqrt(square_sum))
square_sum=0
# print(sum_list)
fori in range(0,4):

collumn_list[i]=np.argsort(sum_list[i])



fori in range(0,len(collumn_list)):
for j in range(0,int((percent/100)*len(collumn_list[i]))):
layer=2*i
collumn=collumn_list[i][j]
for k in range(0,len(arr1[layer])):

arr1[layer][k][collumn]=0#deleting layer wise bottom k% neurons
returnnp.asarray(arr1)
```

```
number=int(input("Number of pruning percentages"))
k=[0]*number

print("Enter percentages(k) for pruning")
fori in range(0,len(k)):
k[i]=int(input())
fori in range(0,len(k)):
start=time.time()
percentage=k[i]
model.set_weights(neuronprune(percentage))
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy for ',percentage,'% pruning using neuron pruning=', test_acc)
end=time.time()
print("Time taken=",end-start)

  start1=time.time()
model.set_weights(weightprune(percentage))
  #print ("Final Matrix",model.get_weights())
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy for ',percentage,'% pruning using weight pruning=', test_acc)
  end1=time.time()
print("Time taken=",end1-start1)
```

VI. OUTPUT AND RESULTS:

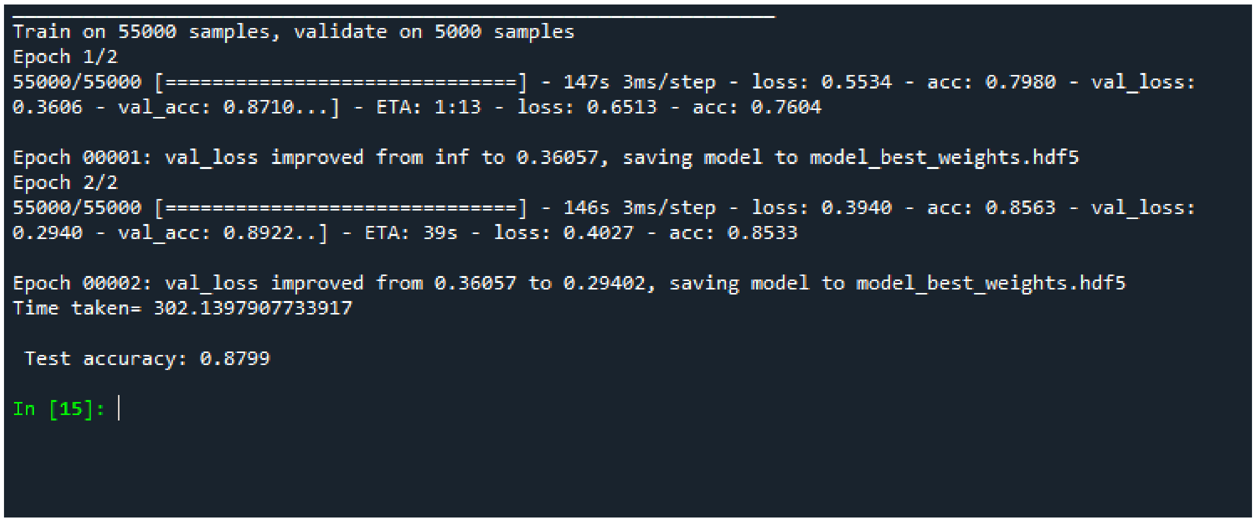
CNN

Model Summary



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
dropout_24 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 32)        8224
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
dropout_25 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               401664
_________________________________________________________________
dropout_26 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                2570
=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
```

Accuracy And Time Taken



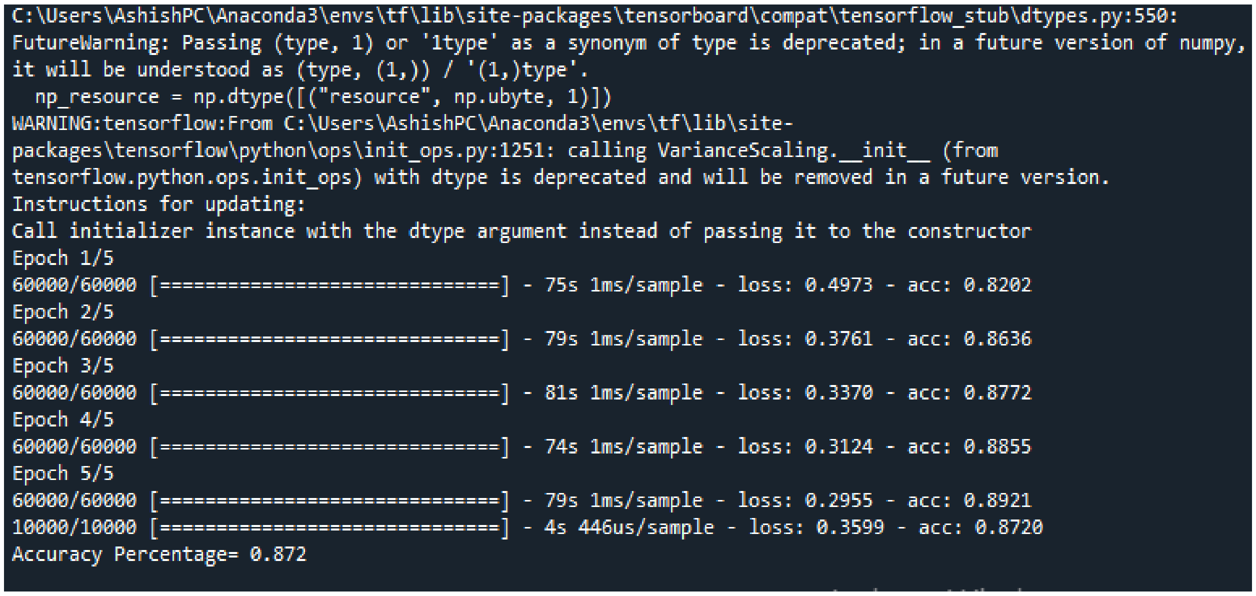
```
Train on 55000 samples, validate on 5000 samples
Epoch 1/2
55000/55000 [==============================] - 147s 3ms/step - loss: 0.5534 - acc: 0.7980 - val_loss:
0.3606 - val_acc: 0.8710...] - ETA: 1:13 - loss: 0.6513 - acc: 0.7604

Epoch 00001: val_loss improved from inf to 0.36057, saving model to model_best_weights.hdf5
Epoch 2/2
55000/55000 [==============================] - 146s 3ms/step - loss: 0.3940 - acc: 0.8563 - val_loss:
0.2940 - val_acc: 0.8922..] - ETA: 39s - loss: 0.4027 - acc: 0.8533

Epoch 00002: val_loss improved from 0.36057 to 0.29402, saving model to model_best_weights.hdf5
Time taken= 302.1397907733917

 Test accuracy: 0.8799

In [15]: 
```

Time Taken=302 seconds

Accuracy=0.8799

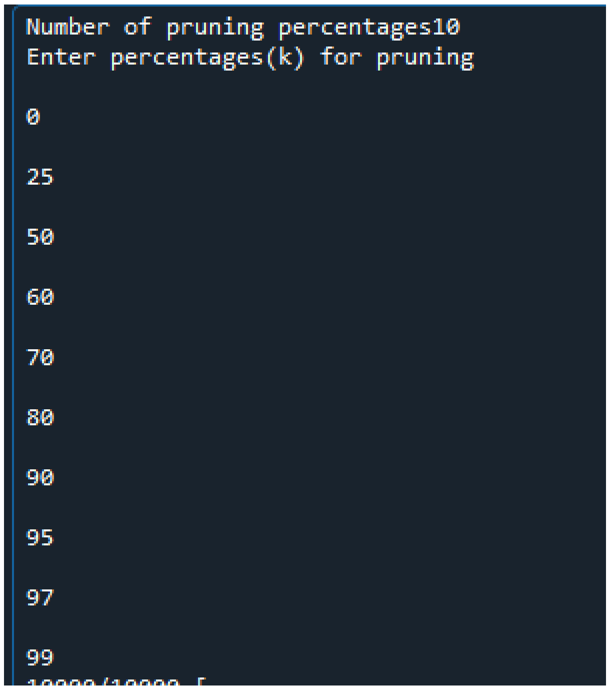
ANN

Training Accuracy of ANN



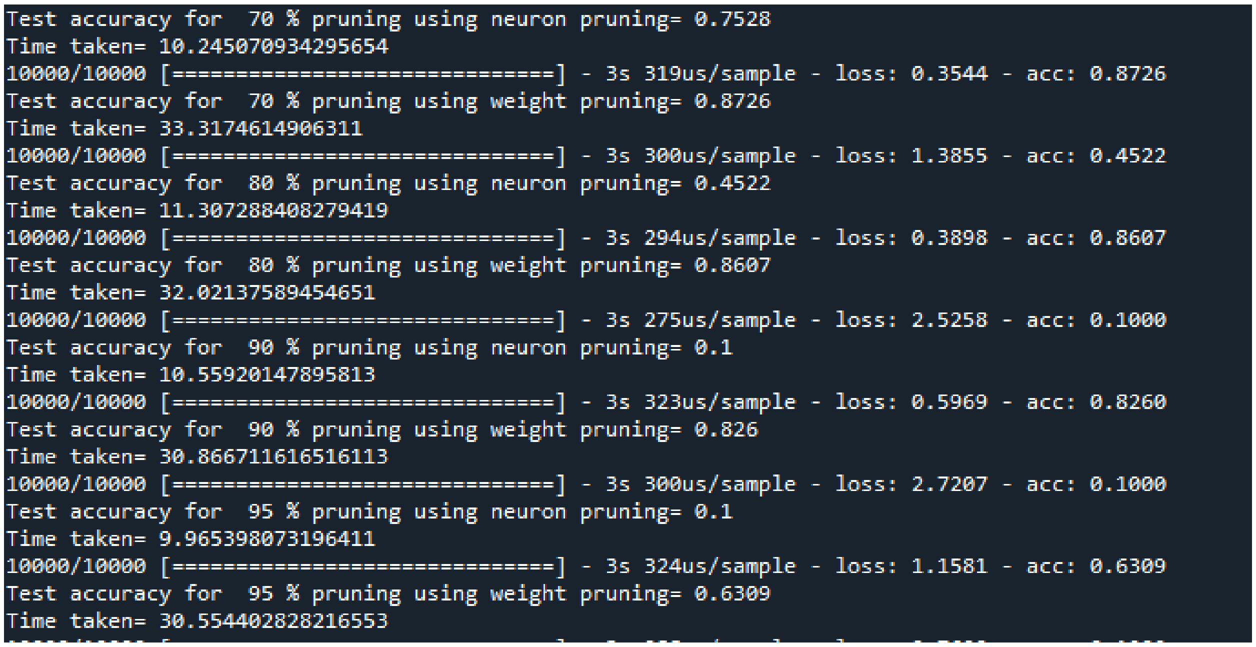
```
C:\Users\AshishPC\Anaconda3\envs\tf\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
WARNING:tensorflow:From C:\Users\AshishPC\Anaconda3\envs\tf\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from
tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Epoch 1/5
60000/60000 [==============================] - 75s 1ms/sample - loss: 0.4973 - acc: 0.8202
Epoch 2/5
60000/60000 [==============================] - 79s 1ms/sample - loss: 0.3761 - acc: 0.8636
Epoch 3/5
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.3370 - acc: 0.8772
Epoch 4/5
60000/60000 [==============================] - 74s 1ms/sample - loss: 0.3124 - acc: 0.8855
Epoch 5/5
60000/60000 [==============================] - 79s 1ms/sample - loss: 0.2955 - acc: 0.8921
10000/10000 [==============================] - 4s 446us/sample - loss: 0.3599 - acc: 0.8720
Accuracy Percentage= 0.872
```

INPUT1

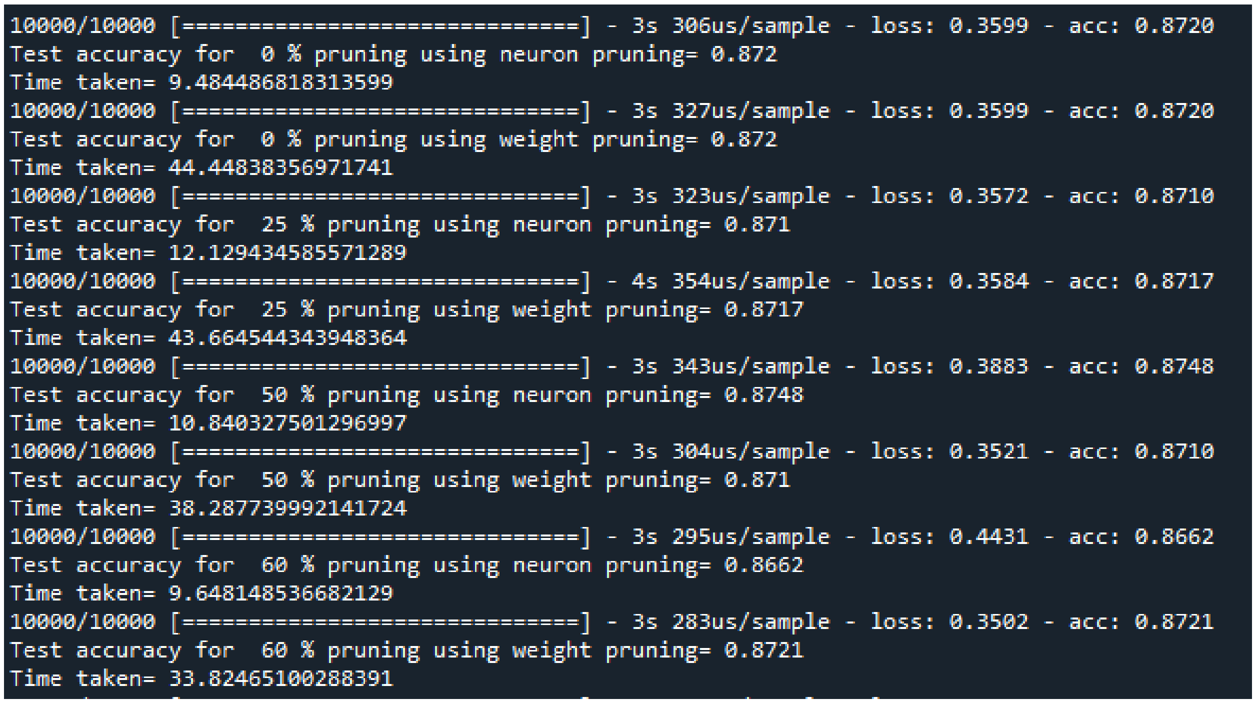


```
Number of pruning percentages10
Enter percentages(k) for pruning

0

25

50

60

70

80

90

95

97

99
10000/10000 [
```

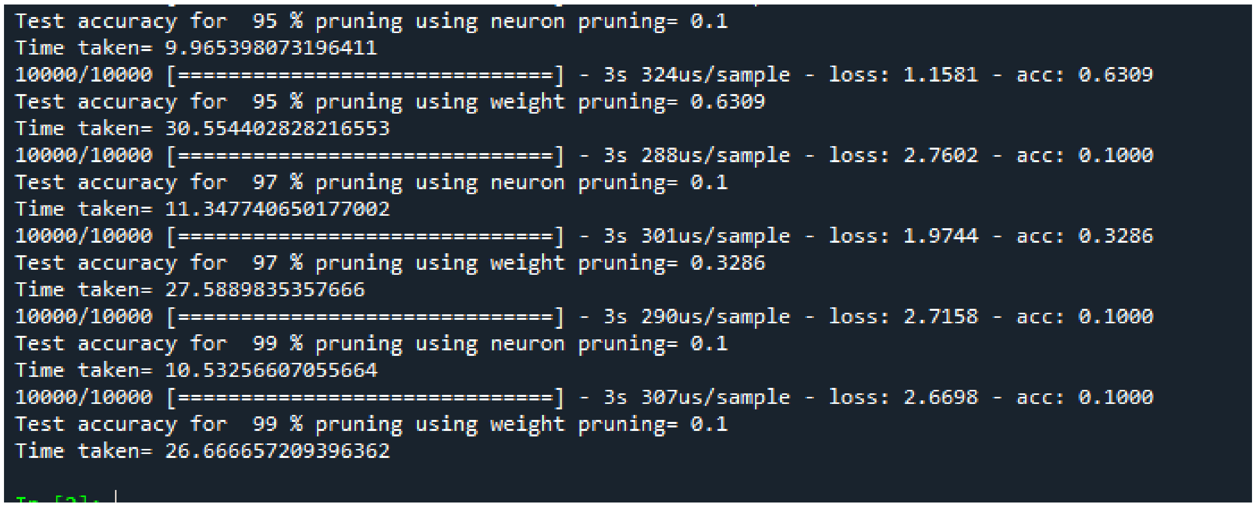
OUTPUT FOR DIFFERENT PRUNING PERCENTAGES



```
Test accuracy for  70 % pruning using neuron pruning= 0.7528
Time taken= 10.245070934295654
10000/10000 [==============================] - 3s 319us/sample - loss: 0.3544 - acc: 0.8726
Test accuracy for  70 % pruning using weight pruning= 0.8726
Time taken= 33.3174614906311
10000/10000 [==============================] - 3s 300us/sample - loss: 1.3855 - acc: 0.4522
Test accuracy for  80 % pruning using neuron pruning= 0.4522
Time taken= 11.307288408279419
10000/10000 [==============================] - 3s 294us/sample - loss: 0.3898 - acc: 0.8607
Test accuracy for  80 % pruning using weight pruning= 0.8607
Time taken= 32.02137589454651
10000/10000 [==============================] - 3s 275us/sample - loss: 2.5258 - acc: 0.1000
Test accuracy for  90 % pruning using neuron pruning= 0.1
Time taken= 10.55920147895813
10000/10000 [==============================] - 3s 323us/sample - loss: 0.5969 - acc: 0.8260
Test accuracy for  90 % pruning using weight pruning= 0.826
Time taken= 30.866711616516113
10000/10000 [==============================] - 3s 300us/sample - loss: 2.7207 - acc: 0.1000
Test accuracy for  95 % pruning using neuron pruning= 0.1
Time taken= 9.965398073196411
10000/10000 [==============================] - 3s 324us/sample - loss: 1.1581 - acc: 0.6309
Test accuracy for  95 % pruning using weight pruning= 0.6309
Time taken= 30.554402828216553
```

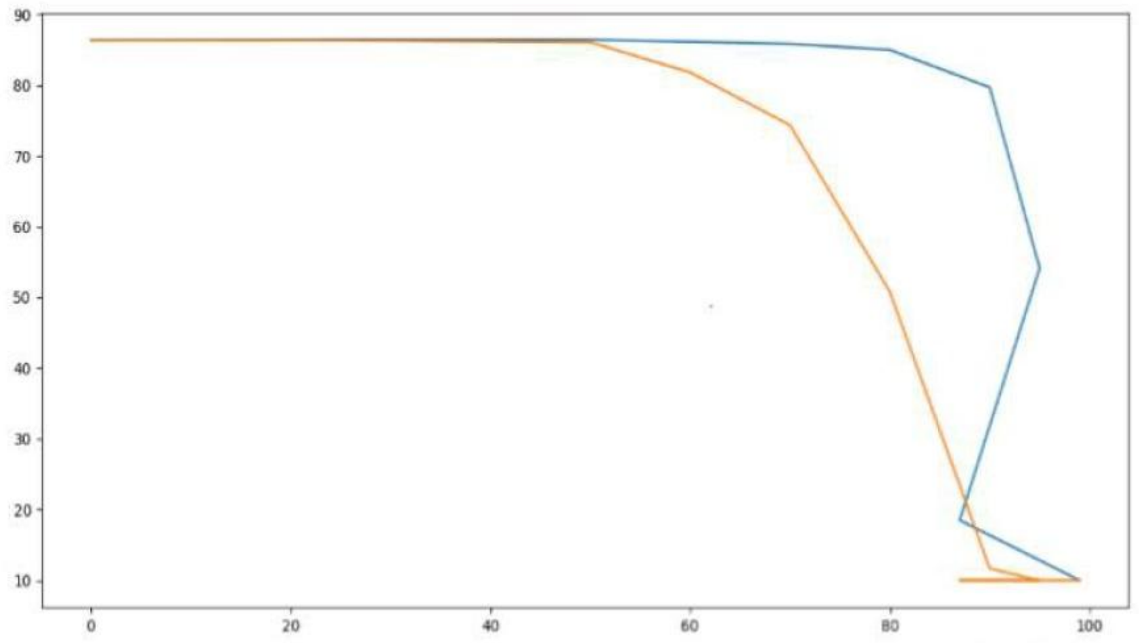


```
10000/10000 [==============================] - 3s 306us/sample - loss: 0.3599 - acc: 0.8720
Test accuracy for  0 % pruning using neuron pruning= 0.872
Time taken= 9.484486818313599
10000/10000 [==============================] - 3s 327us/sample - loss: 0.3599 - acc: 0.8720
Test accuracy for  0 % pruning using weight pruning= 0.872
Time taken= 44.44838356971741
10000/10000 [==============================] - 3s 323us/sample - loss: 0.3572 - acc: 0.8710
Test accuracy for  25 % pruning using neuron pruning= 0.871
Time taken= 12.129434585571289
10000/10000 [==============================] - 4s 354us/sample - loss: 0.3584 - acc: 0.8717
Test accuracy for  25 % pruning using weight pruning= 0.8717
Time taken= 43.664544343948364
10000/10000 [==============================] - 3s 343us/sample - loss: 0.3883 - acc: 0.8748
Test accuracy for  50 % pruning using neuron pruning= 0.8748
Time taken= 10.840327501296997
10000/10000 [==============================] - 3s 304us/sample - loss: 0.3521 - acc: 0.8710
Test accuracy for  50 % pruning using weight pruning= 0.871
Time taken= 38.287739992141724
10000/10000 [==============================] - 3s 295us/sample - loss: 0.4431 - acc: 0.8662
Test accuracy for  60 % pruning using neuron pruning= 0.8662
Time taken= 9.648148536682129
10000/10000 [==============================] - 3s 283us/sample - loss: 0.3502 - acc: 0.8721
Test accuracy for  60 % pruning using weight pruning= 0.8721
Time taken= 33.82465100288391
```



```
Test accuracy for  95 % pruning using neuron pruning= 0.1
Time taken= 9.965398073196411
10000/10000 [==============================] - 3s 324us/sample - loss: 1.1581 - acc: 0.6309
Test accuracy for  95 % pruning using weight pruning= 0.6309
Time taken= 30.554402828216553
10000/10000 [==============================] - 3s 288us/sample - loss: 2.7602 - acc: 0.1000
Test accuracy for  97 % pruning using neuron pruning= 0.1
Time taken= 11.347740650177002
10000/10000 [==============================] - 3s 301us/sample - loss: 1.9744 - acc: 0.3286
Test accuracy for  97 % pruning using weight pruning= 0.3286
Time taken= 27.5889835357666
10000/10000 [==============================] - 3s 290us/sample - loss: 2.7158 - acc: 0.1000
Test accuracy for  99 % pruning using neuron pruning= 0.1
Time taken= 10.53256607055664
10000/10000 [==============================] - 3s 307us/sample - loss: 2.6698 - acc: 0.1000
Test accuracy for  99 % pruning using weight pruning= 0.1
Time taken= 26.666657209396362
```

<u>LineGraphFor The Model WithPercentage Pruning On x-Axis and accuracies on Y-axis</u>

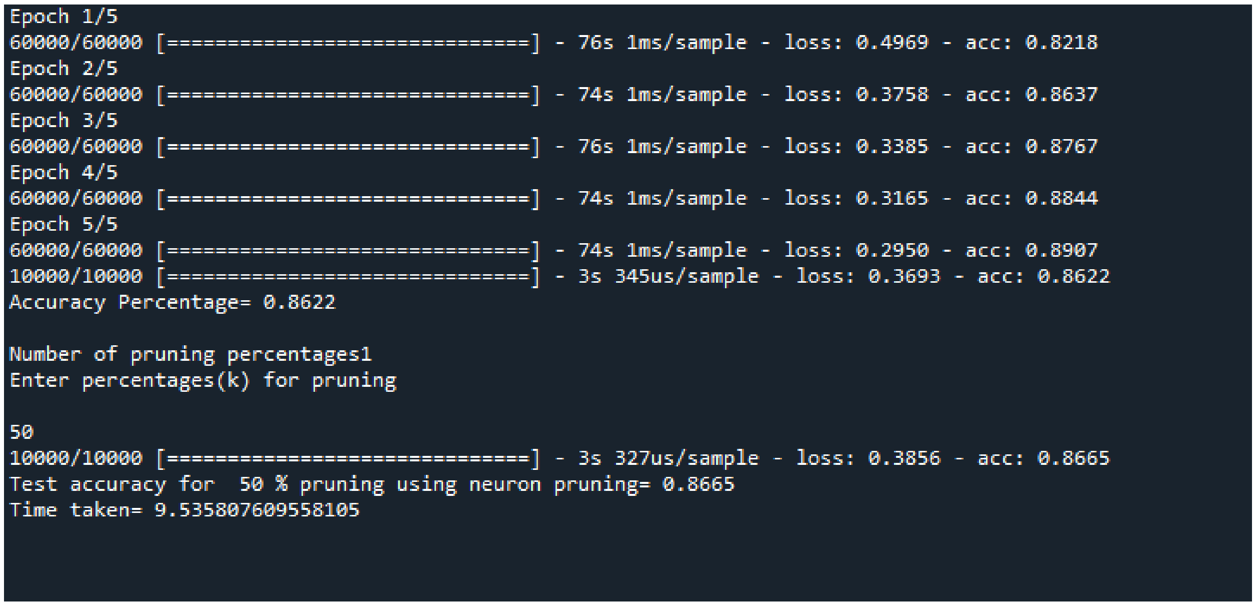


Blue Curve=Percentage vs Weights pruned

Orange Curve=Percentage vs Neurons pruned

INPUT2

From the line graph it can be easily observed that the model performs best when the pruning percentage is around 50% hence we will use a ANN model with 50% of pruning



```
Epoch 1/5
60000/60000 [==============================] - 76s 1ms/sample - loss: 0.4969 - acc: 0.8218
Epoch 2/5
60000/60000 [==============================] - 74s 1ms/sample - loss: 0.3758 - acc: 0.8637
Epoch 3/5
60000/60000 [==============================] - 76s 1ms/sample - loss: 0.3385 - acc: 0.8767
Epoch 4/5
60000/60000 [==============================] - 74s 1ms/sample - loss: 0.3165 - acc: 0.8844
Epoch 5/5
60000/60000 [==============================] - 74s 1ms/sample - loss: 0.2950 - acc: 0.8907
10000/10000 [==============================] - 3s 345us/sample - loss: 0.3693 - acc: 0.8622
Accuracy Percentage= 0.8622

Number of pruning percentages1
Enter percentages(k) for pruning

50
10000/10000 [==============================] - 3s 327us/sample - loss: 0.3856 - acc: 0.8665
Test accuracy for  50 % pruning using neuron pruning= 0.8665
Time taken= 9.535807609558105
```

Time Taken=9.53 Seconds

Accuracy=0.8665

VII. RESULTS:

- As time taken for a program to execute is directly proportional to the number of computations in the program,CNN is computationally less efficient than pruned ANN for classification of the used image dataset.

- Correlation coefficient between K and W= -0.58636
- Correlation coefficient between K and N= -0.82998

- Implying neural network performance is more sensitive towards increasing sparsity of weight matrix in neuron based pruning.

- Weight pruning with sparsity of 25% and 50% shows even betterperformance than the original neural network model.

- Weight based pruning has a considerable drop in performance when
- 90% of weights are pruned.

- For neuron based pruning this drop is observed at 50% sparsity.

- A pruned neural network may take less time than a CNN to be trained to achieve the same accuracy and hence in some cases may be more efficient than a CNN.

VIII. CONCLUSION AND FUTURE SCOPE:

Pruning is one of the methods for inference to efficiently produce models smaller in size, more memory-efficient, more power-efficient and faster at inference with minimal loss in accuracy.

With the rise of mobile inference and machine learning capabilities, pruning will become more relevant than ever before. Lightweight algorithms are going to be the future, as more and more applications find use with neural networks. The most recent example of this comes in the form of Apple's new products, which use neural networking to ensure a multitude of privacy and security features across products. Owing to the disruptive nature of the technology, it is easy to see its adoption by various companies.

The easy availability of neural networks is also required due to the varied nature of their applications. Their move to mobile is also complemented by the standalone computer in flagship devices. This is the reason pruning is going to be more relevant in the future, as the applications need to get lighter and faster without sacrificing accuracy. Though the weight matrices in both the methods are k% sparse, the performance differs as when the whole neuron is deleted, the corresponding output it sends to the neurons of the next layers is deleted as well, which gives a loss to every neuron. While deleting the corresponding weights, the output of the neuron suffers slightly and is not set to a 0, hence the loss incurred by the neuron to the next layer is not as dominant. This effect is more visible when the sparsity increases and is well supported by the plot. Upto 50% pruning both the pruning techniques show little change in the performance of the neural network. With 25% and 50% sparsity achieved using weight pruning method the model

even makes better predictions. We think the reason might be lack of parameter ignorance. Every parameter while training might not be a valid factor for the outcome or might be affecting the outcome very little. For instance while classifying ripe or unripe mangoes the valid parameters are : Weight,Size,Colour.
Suppose introduction of another parameter “Plantation Name” takes place which signifies the plantation wherein the mango was grown. Since we are training fully connected models the neural network after some training samples does set the weights from the corresponding neuron (Plantation Name) to a lesser value but it won’t be an absolute 0. Hence while making predictions this parameter is holding some weight (though negligible),but once pruning happens we are able to set these lesser magnitude weights to 0 which further increase the accuracy or does not affect it to a greater extent.

IX. REFERENCES:

1) https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
2) https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/#:~:text=In%20summary%2C%20CNNs%20are%20especially,value%20on%20the%20feature%20map.
3) https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
4) https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-networks-image-classification/
5) https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8
6) https://ieeexplore.ieee.org/abstract/document/7486599
7) https://www.researchgate.net/publication/281468750_Image_classification_using_artificial_neural_networks_An_experimental_study_on_Corel_database
8) https://www.researchgate.net/publication/238093001_Artificial_neural_networks_and_other_methods_of_image_classification#:~:text=Different%20advanced%20techniques%20in%20image,being%20developed%20for%20image%20classification.
9) https://ieeexplore.ieee.org/document/8703316
10) https://www.researchgate.net/publication/310448045_Design_of_Artificial_Neural_Network_Architecture_for_Handwritten_Digit_Recognition_on_FPGA
11) https://dl.acm.org/citation.cfm?id=640186.640192
12) https://arxiv.org/abs/1707.05455#:~:text=Pruning%20Convolutional%20Neural%20Networks%20for%20Image%20Instance%20Retrieval,-

Gaurav%20Manek%2C%20Jie&text=In%20this%20work%2C%20we%20focus,edges%20while%20maintaining%20retrieval%20performance.

13) https://www.researchgate.net/publication/317496930_Deep_Convolutional_Neural_Networks_for_Image_Classification_A_Comprehensive_Review
14) https://link.springer.com/article/10.1007/s00521-017-3285-0