DIFFERENT CONVOLUTIONAL NETWORK ARCHITECHTURES ON IMAGENET32

1. Build the best network using Keras –

Since the dataset is huge and requires very high computational capacity, we could only run a few epochs in the range of 5-25. The best network we could build on this dataset was with 6 convolutional layers, 3 Max pooling layers and 3 fully connected layers. The network configuration and outcome is given below –

Epochs – 10

Activation – Relu

Optimizer – Adamax

Batch Size – 128

CNN Layers – 64, 64, 256, 256, 128, 128

FC layers – 256, 512, 200

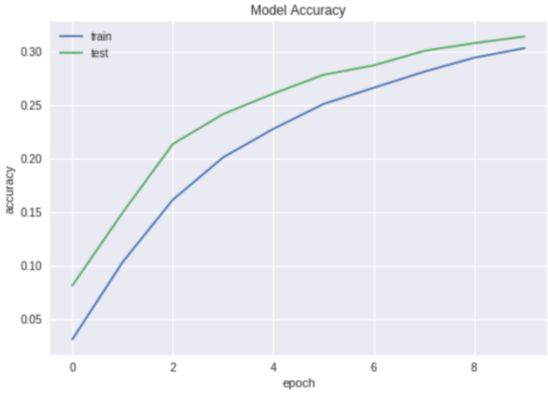
Dropout – 0.35

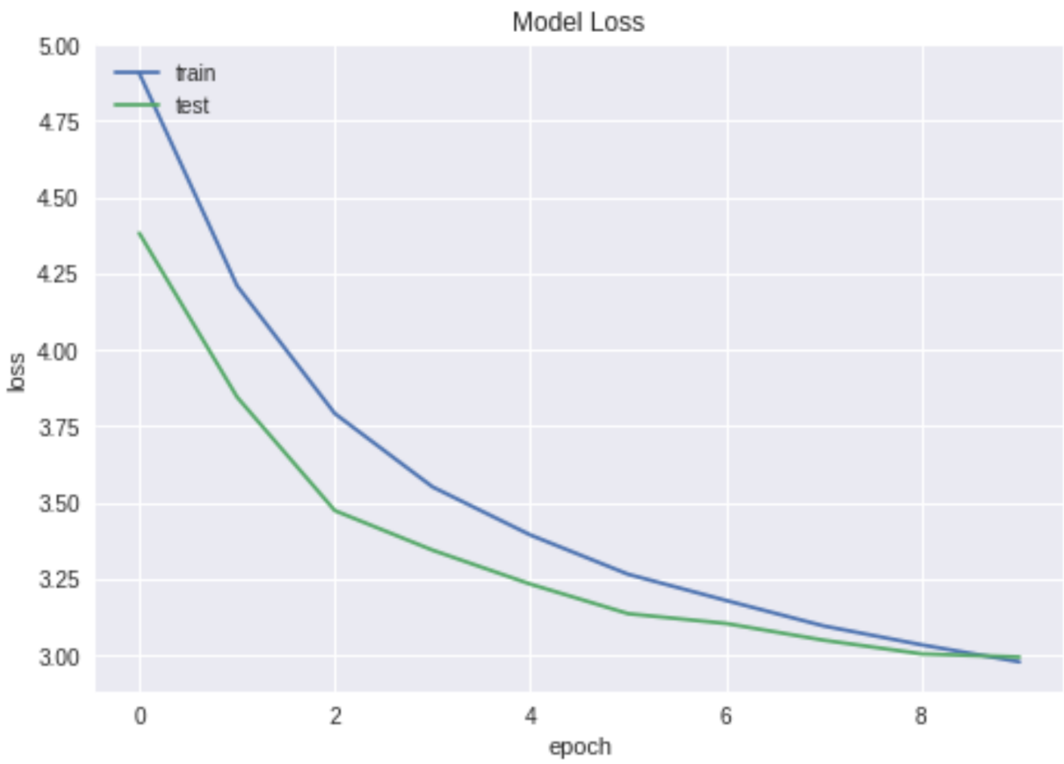
The Testing accuracy is 31.38% and training accuracy is 30.29

```python
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same',
                 input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.24))


model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.35))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.24))
model.add(Dense(num_classes))
```

```
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        "/content/drive/My Drive/Tar_File/ImageNet_Data/Train",
        target_size=(32, 32),
        batch_size=batch_size,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        '/content/drive/My Drive/Tar_File/ImageNet_Data/Validation',
        target_size=(32, 32),
        batch_size=batch_size,
        class_mode='categorical')

train_history = model.fit_generator(
        train_generator,
        steps_per_epoch=2000,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=200)

# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
2000/2000 [==============================] - 1055s 528ms/step - loss: 3.0352 - acc: 0.2940 - val_loss: 3.0057 - val_acc: 0.3076
Epoch 10/10
2000/2000 [==============================] - 1085s 542ms/step - loss: 2.9794 - acc: 0.3029 - val_loss: 2.9954 - val_acc: 0.3138
Saved trained model at /content/drive/My Drive/Tar_File/saved_models/keras_MaxNet_trained_model.h5
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





The accuracy is converging and will increase further if we train it for more epochs using GPU.

Prediction on Test Image –

```
[ ]  # load a single image
     new_image = load_image("/content/drive/My Drive/Tar_File/ImageNet_Data/Test/test_1004.png")

     # check prediction
     pred = model.predict(new_image)
```

Looks like a woman in a white dress wearing glasses and robe

```
[ ]  df.sort_values(['probability'], ascending=0).head(10)
```

	folder	class	probability
112	n03617480	kimono	0.114352
60	n02669723	academic gown or academic robe or judge's robe	0.060282
123	n03814639	neck brace	0.060033
163	n04456115	torch	0.056395
68	n02802426	basketball	0.055654
77	n02883205	bow tie or bow-tie or bowtie	0.039199
131	n03970156	plunger or plumber's helper	0.039052
62	n02730930	apron	0.036937
169	n04532106	vestment	0.036476
138	n04023962	punching bag or punch bag or punching ball o...	0.030814

2. Applied Transfer Learning using Densenet121 pretrained model on CIFAR10 –

The network configuration and outcome is given below –

Epochs – 5

Activation – Swish , Relu

Optimizer – Adamax

Batch Size – 16

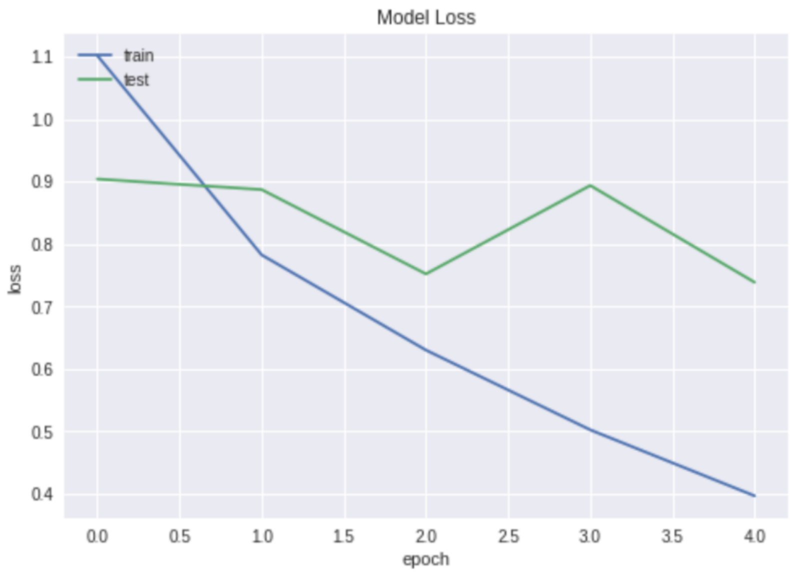
CNN Layers – 12 layers from the Densenet

FC layers – 512, 512, 10

Dropout – 0.35

The Testing accuracy was 76.98% and training accuracy was 86.06% with just a epoch of 5. Swish gave a better accuracy than Relu, as Swish works better with deeper networks like these. But Relu is computationally less expensive than Swish.

```
50000/50000 [==============================] - 423s 8ms/step - loss: 0.5023 - acc: 0.8232 - val_loss: 0.8935 - val_acc: 0.7152
Epoch 5/5
50000/50000 [==============================] - 423s 8ms/step - loss: 0.3967 - acc: 0.8606 - val_loss: 0.7386 - val_acc: 0.7698
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```




```
from keras.preprocessing import image
import matplotlib.pyplot as plt
# load model
#model = load_model("/content/drive/My Drive/Tar_File/saved_models/keras_MaxNet_trained_model.h5")

# load a single image
new_image = load_image("/content/drive/My Drive/MaxNet Files/prow-featured.jpg")

# check prediction
pred = trans_model.predict(new_image)
```

drive.google.com/drive/search?q=owner%3Ame %28type%3Aapplicat...

```
pred
```

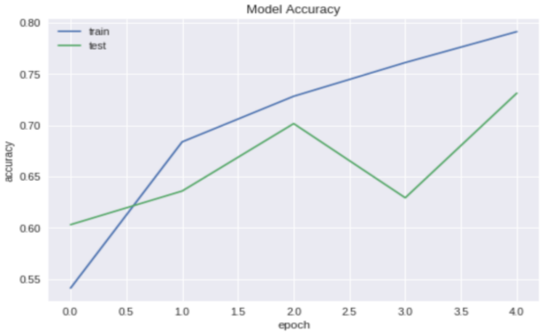
```
array([[1.03953965e-01, 1.15328515e-03, 7.83927321e-01, 5.04986756e-02,
        2.11512968e-02, 1.68351806e-03, 1.47230998e-02, 2.82732514e-03,
        1.94367226e-02, 6.44753978e-04]], dtype=float32)
```

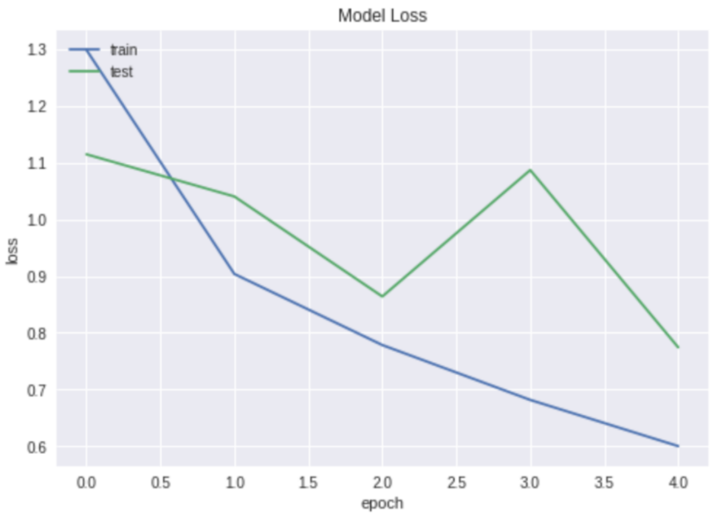
3rd class of the Cifar 10 dataset is a Bird.

The corresponding probability for Bird class is 78%, so it predicted the test image very closely.

Using Relu –

```
Epoch 4/5
50000/50000 [==============================] - 206s 4ms/step - loss: 0.6813 - acc: 0.7612 - val_loss: 1.0870 - val_acc: 0.6292
Epoch 5/5
50000/50000 [==============================] - 206s 4ms/step - loss: 0.5998 - acc: 0.7913 - val_loss: 0.7742 - val_acc: 0.7313
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





```
[ ] pred
```

```
array([[1.03953965e-01, 1.15328515e-03, 7.83927321e-01, 5.04986756e-02,
        2.11512968e-02, 1.68351806e-03, 1.47230998e-02, 2.82732514e-03,
        1.94367226e-02, 6.44753978e-04]], dtype=float32)
```

3rd class of the Cifar 10 dataset is a Bird.

3. Used AutoKeras to Tune Hyperparameters –

Since the dataset is huge and requires very high computational capacity, we could only run a few epochs even after letting the Autokeras run for 10 hours on GPU. We compared the performance and speed on ImageNet and Cifar10.

Image Net -

Epochs – 20, 28 (2 Models)

The best accuracy was 24.4% on ImageNet and 88.6% on Cifar10 , This shows that Image net is a way more complex dataset than Cifar10 and it will take much denser network configurations to learn the representations.

```
Preprocessing the images.
Preprocessing finished.

Initializing search.
Initialization finished.


+----------------------------------------------+
|               Training model 0               |
+----------------------------------------------+

No loss decrease after 5 epochs.


Saving model.
+--------------------------------------------------------------------------+
|        Model ID        |          Loss          |      Metric Value      |
+--------------------------------------------------------------------------+
|           0            |   17.306082820892335   |         0.1048         |
+--------------------------------------------------------------------------+


+----------------------------------------------+
|               Training model 1               |
+----------------------------------------------+
Epoch-20, Current Metric - 0.196:  63%|██████████████          | 490/778 [04:58<02:58,  1.61 batch/s]Time is out.
```

```
[ ] Saving model.
+--------------------------------------------------------------------------+
|          Model ID          |          Loss          |     Metric Value     |
+--------------------------------------------------------------------------+
|             0              |   3.9897366404533385   |        0.6532        |
+--------------------------------------------------------------------------+


+----------------------------------------------+
|               Training model 1               |
+----------------------------------------------+

No loss decrease after 5 epochs.


Saving model.
+--------------------------------------------------------------------------+
|          Model ID          |          Loss          |     Metric Value     |
+--------------------------------------------------------------------------+
|             1              |   1.6496357917785645   |  0.8592000000000001  |
+--------------------------------------------------------------------------+


+----------------------------------------------+
|               Training model 2               |
+----------------------------------------------+
Epoch-3, Current Metric - 0.544:  67%|██████████████        | 260/387 [02:48<01:25,  1.49 batch/s]
```

```
+--------------------------------------------------------------------------+
[ ] |             0              |   17.599685287475587   |  0.10400000000000001  |
+--------------------------------------------------------------------------+


+----------------------------------------------+
|               Training model 1               |
+----------------------------------------------+

No loss decrease after 5 epochs.


Saving model.
+--------------------------------------------------------------------------+
|          Model ID          |          Loss          |     Metric Value     |
+--------------------------------------------------------------------------+
|             1              |   14.157835245132446   |        0.2536        |
+--------------------------------------------------------------------------+


+----------------------------------------------+
|               Training model 2               |
+----------------------------------------------+
Epoch-14, Current Metric - 0.114:   4%|█                | 30/778 [00:14<05:01,  2.48 batch/s]Time is out.
```

```
[ ]  from sklearn.metrics import accuracy_score
     y_prediction = clf.predict(X_test)
     accuracy_score(y_true=y_test, y_pred=y_prediction)

     y = clf.evaluate(X_test, y_test)
     print(y * 100)
```

```
30.15
```

4. Implemented the AlexNet Architechture –

The network configuration of the AlexNet and outcome is given below –

Epochs – 10

Activation – Relu

Optimizer – Adamax

Batch Size – 128

CNN Layers – 96. 256, 384, 384, 256

FC layers – 4096, 4096, 1000, 200

Dropout – 0.4

The Alexnet uses Dropout for avoiding overfitting, Batch normalization was developed after this so that was not used in the architecture. As we can see the computation is huge as it involves some 76 million trainable parameters.

The Accuracy we got was 0.05 with only 1 epoch. This can be a great model if we have good capacity to run on multiple GPUs. This was how it was ran in the original paper as they distributed the layer computations on different GPUs.

```
[ ]  Layer (type)                 Output Shape              Param #
     =================================================================
     conv2d_1 (Conv2D)            (None, 56, 56, 96)        34944
     _________________________________________________________________
     activation_1 (Activation)    (None, 56, 56, 96)        0
     _________________________________________________________________
     max_pooling2d_1 (MaxPooling2 (None, 28, 28, 96)        0
     _________________________________________________________________
     conv2d_2 (Conv2D)            (None, 28, 28, 256)       2973952
     _________________________________________________________________
     activation_2 (Activation)    (None, 28, 28, 256)       0
     _________________________________________________________________
     max_pooling2d_2 (MaxPooling2 (None, 14, 14, 256)       0
     _________________________________________________________________
     conv2d_3 (Conv2D)            (None, 14, 14, 384)       885120
     _________________________________________________________________
     activation_3 (Activation)    (None, 14, 14, 384)       0
     _________________________________________________________________
     conv2d_4 (Conv2D)            (None, 14, 14, 384)       1327488
     _________________________________________________________________
     activation_4 (Activation)    (None, 14, 14, 384)       0
     _________________________________________________________________
     conv2d_5 (Conv2D)            (None, 14, 14, 256)       884992
     _________________________________________________________________
     activation_5 (Activation)    (None, 14, 14, 256)       0
     _________________________________________________________________
     max_pooling2d_3 (MaxPooling2 (None, 7, 7, 256)         0
     _________________________________________________________________
```

```
[ ] dense_2 (Dense)              (None, 4096)              16781312
    _________________________________________________________________
    activation_7 (Activation)    (None, 4096)              0
    _________________________________________________________________
    dropout_2 (Dropout)          (None, 4096)              0
    _________________________________________________________________
    dense_3 (Dense)              (None, 1000)              4097000
    _________________________________________________________________
    activation_8 (Activation)    (None, 1000)              0
    _________________________________________________________________
    dropout_3 (Dropout)          (None, 1000)              0
    _________________________________________________________________
    dense_4 (Dense)              (None, 200)               200200
    _________________________________________________________________
    activation_9 (Activation)    (None, 200)               0
    =================================================================
    Total params: 76,210,032
    Trainable params: 76,210,032
    Non-trainable params: 0
    _________________________________________________________________
    Found 100000 images belonging to 200 classes.
    Found 10000 images belonging to 200 classes.
    WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066:
    Instructions for updating:
    Use tf.cast instead.
    Epoch 1/10
        6/10000 [..............................] - ETA: 12:13:49 - loss: 14.2061 - acc: 0.0000e+00
```

5. Implemented the DenseNet121 Architechture –

The Densenet121 consists of Convolution Blocks which have many convolution layers and pooling layers. The total depth of the network is 121 hence the name DenseNet121

The volume after every Dense Block increase by the growth rate times the number of Dense Layers within that Dense Block

DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters changes between them. These layers between them are called Transition Layers and take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers.

Experiment 1 :

The network configuration of the Densenet121 and outcome is given below –

Epochs – 5

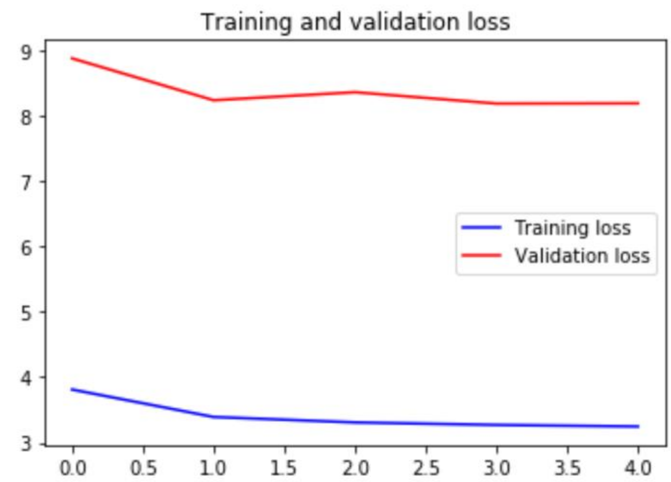
Activation – Relu

Optimizer – Adamax

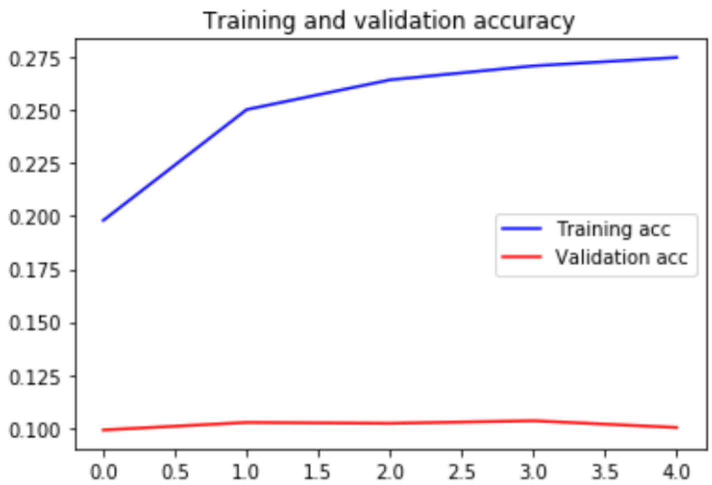
Batch Size – 128

Dropout – 0.4

```
[ ] <keras.engine.input_layer.InputLayer object at 0x7f2c867ea9b0> False
<keras.layers.convolutional.ZeroPadding2D object at 0x7f2c86776fd0> False
<keras.layers.convolutional.Conv2D object at 0x7f2c86776e10> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c867921d0> False
<keras.layers.core.Activation object at 0x7f2c867926a0> False
<keras.layers.convolutional.ZeroPadding2D object at 0x7f2c846f8d30> False
<keras.layers.pooling.MaxPooling2D object at 0x7f2c8675be10> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c86792eb8> False
<keras.layers.core.Activation object at 0x7f2c841f2eb8> False
<keras.layers.convolutional.Conv2D object at 0x7f2caf033898> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c841c5438> False
<keras.layers.core.Activation object at 0x7f2c84126080> False
<keras.layers.convolutional.Conv2D object at 0x7f2c841039b0> False
<keras.layers.merge.Concatenate object at 0x7f2c840cf278> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c8409b748> False
<keras.layers.core.Activation object at 0x7f2c72fd9828> False
<keras.layers.convolutional.Conv2D object at 0x7f2c72f50e48> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c72fafba8> False
<keras.layers.core.Activation object at 0x7f2c72f75e80> False
<keras.layers.convolutional.Conv2D object at 0x7f2c72e7e828> False
<keras.layers.merge.Concatenate object at 0x7f2c72e4a0f0> False
<keras.layers.normalization.BatchNormalization object at 0x7f2c72e6afd0> False
<keras.layers.core.Activation object at 0x7f2c72e17748> False
```



None



Experiment 2 :

The network configuration of the Densenet121 and outcome is given below –

Epochs – 10

Activation – Relu

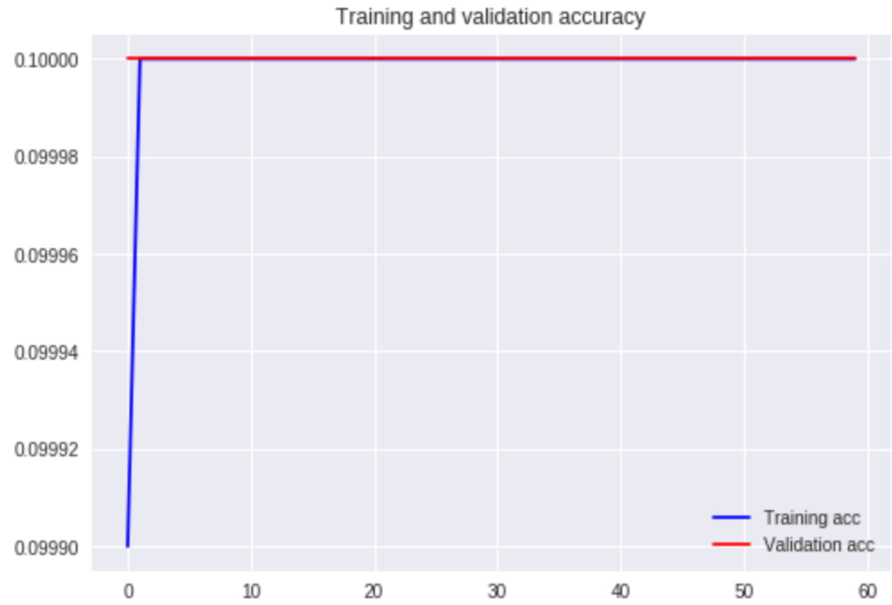
Optimizer – Adamax

Batch Size – 128

Dropout – 0.2

FC – 1024, 200

```
Instructions for updating:
Use tf.cast instead.
Train on 100000 samples, validate on 10000 samples
Epoch 1/10
100000/100000 [==============================] - 284s 3ms/step - loss: 4.7411 - acc: 0.0505 - val_loss: 5.7974 - val_acc: 0.0406
Epoch 2/10
100000/100000 [==============================] - 236s 2ms/step - loss: 4.3481 - acc: 0.0899 - val_loss: 4.6958 - val_acc: 0.0625
Epoch 3/10
100000/100000 [==============================] - 236s 2ms/step - loss: 4.2331 - acc: 0.1045 - val_loss: 4.3370 - val_acc: 0.0937
Epoch 4/10
100000/100000 [==============================] - 236s 2ms/step - loss: 4.0588 - acc: 0.1274 - val_loss: 4.5161 - val_acc: 0.0815
Epoch 5/10
100000/100000 [==============================] - 236s 2ms/step - loss: 3.9863 - acc: 0.1383 - val_loss: 4.4380 - val_acc: 0.1005
Epoch 6/10
100000/100000 [==============================] - 236s 2ms/step - loss: 3.9438 - acc: 0.1459 - val_loss: 4.6768 - val_acc: 0.1007
Epoch 7/10
100000/100000 [==============================] - 236s 2ms/step - loss: 3.8588 - acc: 0.1573 - val_loss: 4.0263 - val_acc: 0.1404
Epoch 8/10
100000/100000 [==============================] - 235s 2ms/step - loss: 3.8177 - acc: 0.1625 - val_loss: 4.5948 - val_acc: 0.1214
Epoch 9/10
100000/100000 [==============================] - 235s 2ms/step - loss: 3.7689 - acc: 0.1725 - val_loss: 5.7159 - val_acc: 0.0254
Epoch 10/10
100000/100000 [==============================] - 236s 2ms/step - loss: 3.8892 - acc: 0.1546 - val_loss: 3.9044 - val_acc: 0.1578
```



Training and validation accuracy
0.10000
0.09998
0.09996
0.09994
0.09992
0.09990
0
10
20
30
40
50
60
Training acc
Validation acc

Training and validation loss
+1.45e1
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
10
20
30
40
50
60
Training loss
Validation loss

6. **Implemented the Fine Tuning on Cifar10 using Pretrained DenseNet121 –**

Fine tuning is just about making some fine adjustments to further improve performance. For example, during transfer learning, you can unfreeze the pre-trained model and let it adapt more to the task at hand. Fine tuning on the other hand is just about making some fine adjustments to further improve performance. For example, during transfer learning, you can unfreeze the pre-trained model and let it adapt more to the task at hand.

Experiment 1 –

The network configuration of the Densenet121 and outcome is given below –

Epochs – 15

Activation – Swish

Optimizer – Adamax

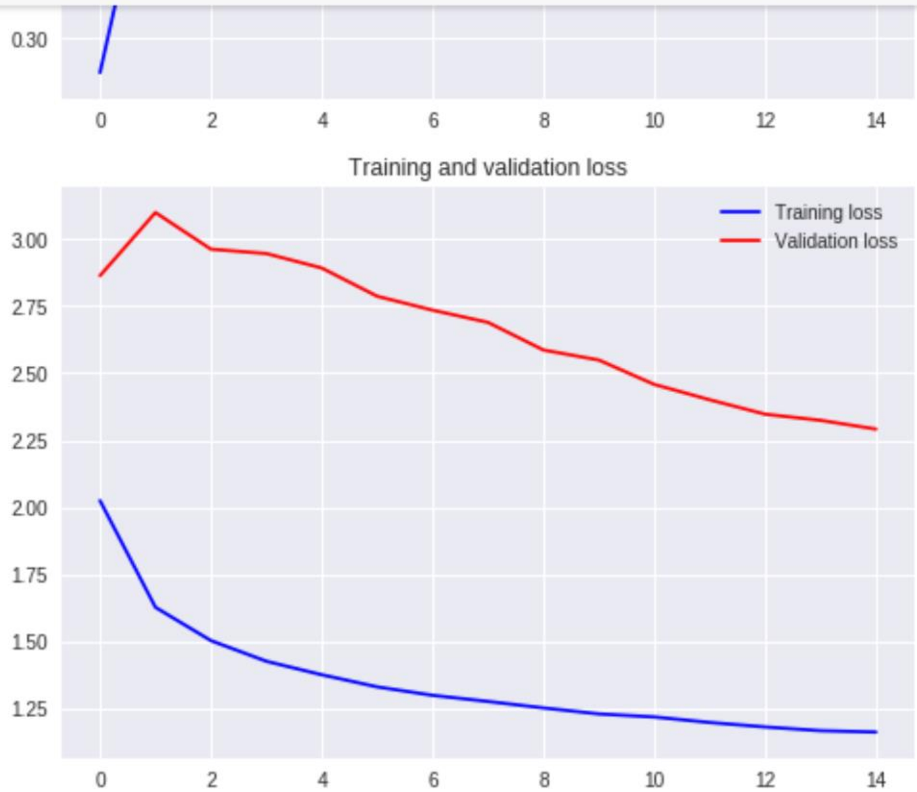
Batch Size – 64

Dropout – 0.4

FC – 512, 512, 10

```
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
2
4
6
8
10
12
14


0.30
Training and validation loss
Training loss
Validation loss
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
0
2
4
6
8
10
12
14

```
Epoch 7/15
50000/50000 [==============================] - 441s 9ms/step - loss: 1.3003 - acc: 0.5408 - val_loss: 2.7343 - val_acc: 0.4487
Epoch 8/15
50000/50000 [==============================] - 572s 11ms/step - loss: 1.2774 - acc: 0.5480 - val_loss: 2.6887 - val_acc: 0.4496
Epoch 9/15
50000/50000 [==============================] - 582s 12ms/step - loss: 1.2533 - acc: 0.5592 - val_loss: 2.5859 - val_acc: 0.4540
Epoch 10/15
50000/50000 [==============================] - 426s 9ms/step - loss: 1.2310 - acc: 0.5647 - val_loss: 2.5486 - val_acc: 0.4540
Epoch 11/15
50000/50000 [==============================] - 411s 8ms/step - loss: 1.2197 - acc: 0.5685 - val_loss: 2.4578 - val_acc: 0.4585
Epoch 12/15
50000/50000 [==============================] - 410s 8ms/step - loss: 1.1992 - acc: 0.5774 - val_loss: 2.4007 - val_acc: 0.4602
Epoch 13/15
50000/50000 [==============================] - 414s 8ms/step - loss: 1.1825 - acc: 0.5825 - val_loss: 2.3470 - val_acc: 0.4669
Epoch 14/15
50000/50000 [==============================] - 414s 8ms/step - loss: 1.1686 - acc: 0.5858 - val_loss: 2.3242 - val_acc: 0.4636
Epoch 15/15
50000/50000 [==============================] - 416s 8ms/step - loss: 1.1634 - acc: 0.5900 - val_loss: 2.2916 - val_acc: 0.4684
Saved trained model at /content/saved_models/keras_cifar10FineTune_trained_model1.h5
10000/10000 [==============================] - 82s 8ms/step
Test loss: 2.291573031616211
Test accuracy: 0.4684
```

Experiment 2–

The network configuration of the Densenet121 and outcome is given below –

Epochs – 4

Activation – Relu

Optimizer – Adamax

Batch Size – 16

Dropout – 0.4

FC – 12, 24, 16, 10

CNN – 64,64, 64, 64

Lr – 0.001

```
Train on 3000 samples, validate on 100 samples
Epoch 1/4
3000/3000 [==============================] - 2774s - loss: 1.0577 - acc: 0.6363 - val_loss: 0.6477 - val_acc: 0.7900
Epoch 2/4
3000/3000 [==============================] - 2770s - loss: 0.3847 - acc: 0.8740 - val_loss: 0.5647 - val_acc: 0.8000
Epoch 3/4
3000/3000 [==============================] - 2770s - loss: 0.1814 - acc: 0.9473 - val_loss: 0.4912 - val_acc: 0.8300
Epoch 4/4
3000/3000 [==============================] - 2764s - loss: 0.0927 - acc: 0.9780 - val_loss: 0.4171 - val_acc: 0.8600
100/100 [==============================] - 43s
```

Accuracy Test – 86%

Training Acc – 97.8

This is the best model we got after fine tuning the Pretrained DenseNet121 model.

Experiment 3 –

The network configuration of the Densenet121 and outcome is given below –

Epochs – 4

Activation – Relu

Optimizer – SGD

Batch Size – 16

Dropout – 0.4

Learning rate – 0.0001

FC – 12, 24, 16, 10

CNN – 64,64, 64, 64

```
Train on 3000 samples, validate on 100 samples
Epoch 1/4
3000/3000 [==============================] - 4786s - loss: 2.4331 - acc: 0.2587 - val_loss: 14.8286 - val_acc: 0.0800
Epoch 2/4
3000/3000 [==============================] - 4784s - loss: 1.9965 - acc: 0.3253 - val_loss: 2.6960 - val_acc: 0.3100
Epoch 3/4
3000/3000 [==============================] - 4785s - loss: 1.7944 - acc: 0.3847 - val_loss: 11.3192 - val_acc: 0.1300
Epoch 4/4
3000/3000 [==============================] - 4786s - loss: 1.5556 - acc: 0.4507 - val_loss: 1.8677 - val_acc: 0.3700
100/100 [==============================] - 76s
```