In [1]:
```
# import keras
# from keras.datasets import cifar10
# from keras.models import Model, Sequential
# from keras.layers import Dense, Dropout, Flatten, Input, AveragePooling2D, merge, Activation
# from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
# from keras.layers import Concatenate
# from keras.optimizers import Adam
from tensorflow.keras import models, layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.optimizers import Adam
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
from keras import backend as K
import tensorflow as tf
```

In [4]:
```
# ! pip install tensorflow
```

In [5]:
```
import keras
config = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True,device_count = {'GPU': 1 , 'CPU': 56}) )
keras.backend.set_session(config)
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
```

In [6]:
```
# this part will prevent tensorflow to allocate all the avaliable GPU Memory
# backend
import tensorflow as tf
```

In [7]:
```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 10
l = 40
num_filter = 12
compression = 0.5
dropout_rate = 0.2
```

In [8]:
```
# Load CIFAR10 Data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
img_height, img_width, channel = X_train.shape[1],X_train.shape[2],X_train.shape[3]

# convert to one hot encoing
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 4s 0us/step
```

In [9]:
```
X_train.shape
```

Out[9]: (50000, 32, 32, 3)

In [10]:
```
X_test.shape
```

Out[10]: (10000, 32, 32, 3)

In [11]:
```
# Dense Block
def denseblock(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = layers.BatchNormalization()(temp)
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_3_3 = layers.Conv2D(int(num_filter*compression), (3,3), use_bias=False ,padding='same')(relu)
        if dropout_rate>0:
            Conv2D_3_3 = layers.Dropout(dropout_rate)(Conv2D_3_3)
        concat = layers.Concatenate(axis=-1)([temp,Conv2D_3_3])

        temp = concat

    return temp

## transition Blosck
def transition(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (1,1), use_bias=False ,padding='same')(relu)
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)
    return avg

#output layer
def output_layer(input):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    AvgPooling = layers.AveragePooling2D(pool_size=(2,2))(relu)
    flat = layers.Flatten()(AvgPooling)
    output = layers.Dense(num_classes, activation='softmax')(flat)
    return output
```

In [12]:

```
num_filter = 12
dropout_rate = 0.2
l = 12
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(num_filter, (3,3), use_bias=False ,padding='same')(input)

First_Block = denseblock(First_Conv2D, num_filter, dropout_rate)
First_Transition = transition(First_Block, num_filter, dropout_rate)

Second_Block = denseblock(First_Transition, num_filter, dropout_rate)
Second_Transition = transition(Second_Block, num_filter, dropout_rate)

Third_Block = denseblock(Second_Transition, num_filter, dropout_rate)
Third_Transition = transition(Third_Block, num_filter, dropout_rate)

Last_Block = denseblock(Third_Transition,  num_filter, dropout_rate)
output = output_layer(Last_Block)
```

In [ ]:

```
#https://arxiv.org/pdf/1608.06993.pdf
from IPython.display import IFrame, YouTubeVideo
YouTubeVideo(id='-W6y8xnd--U', width=600)
```

Out[9]:

In [13]:
```
model = Model(inputs=[input], outputs=[output])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 32, 32, 12)   324         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 32, 32, 12)   48          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 32, 32, 12)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 6)    648         activation[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 32, 32, 6)    0           conv2d_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 32, 32, 18)   0           conv2d[0][0]
                                                                 dropout[0][0]
__________________________________________________________________________________________________
```

In [ ]:
```
print(len(model.layers))
```

```
262
```

In [ ]:
```
# determine Loss function and Optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

In [ ]:
```
model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/10
391/391 [==============================] - 78s 102ms/step - loss: 1.7187 - accuracy: 0.3524 - val_loss: 1.6062 - val_accuracy: 0.4232
Epoch 2/10
391/391 [==============================] - 37s 94ms/step - loss: 1.3829 - accuracy: 0.4883 - val_loss: 1.3495 - val_accuracy: 0.5169
Epoch 3/10
391/391 [==============================] - 38s 96ms/step - loss: 1.2379 - accuracy: 0.5463 - val_loss: 1.1956 - val_accuracy: 0.5768
Epoch 4/10
391/391 [==============================] - 38s 97ms/step - loss: 1.1237 - accuracy: 0.5931 - val_loss: 1.2230 - val_accuracy: 0.5891
Epoch 5/10
391/391 [==============================] - 38s 97ms/step - loss: 1.0488 - accuracy: 0.6229 - val_loss: 1.5108 - val_accuracy: 0.5358
Epoch 6/10
391/391 [==============================] - 38s 97ms/step - loss: 0.9915 - accuracy: 0.6443 - val_loss: 1.0291 - val_accuracy: 0.6479
Epoch 7/10
391/391 [==============================] - 38s 97ms/step - loss: 0.9515 - accuracy: 0.6586 - val_loss: 1.1797 - val_accuracy: 0.6007
Epoch 8/10
391/391 [==============================] - 38s 97ms/step - loss: 0.9201 - accuracy: 0.6681 - val_loss: 1.0407 - val_accuracy: 0.6418
Epoch 9/10
391/391 [==============================] - 38s 97ms/step - loss: 0.8911 - accuracy: 0.6808 - val_loss: 0.9578 - val_accuracy: 0.6730
Epoch 10/10
391/391 [==============================] - 38s 97ms/step - loss: 0.8642 - accuracy: 0.6904 - val_loss: 0.9221 - val_accuracy: 0.6791
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x7f879036de10>
```

In [ ]:
```
# Test the model
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.9221 - accuracy: 0.6791
Test loss: 0.9220625758171082
Test accuracy: 0.679099977016449
```

In [ ]:
```
# Save the trained weights in to .h5 format
model.save_weights("DNST_model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

In [ ]:
```
del model
```

CNN on CIFR Assignment:

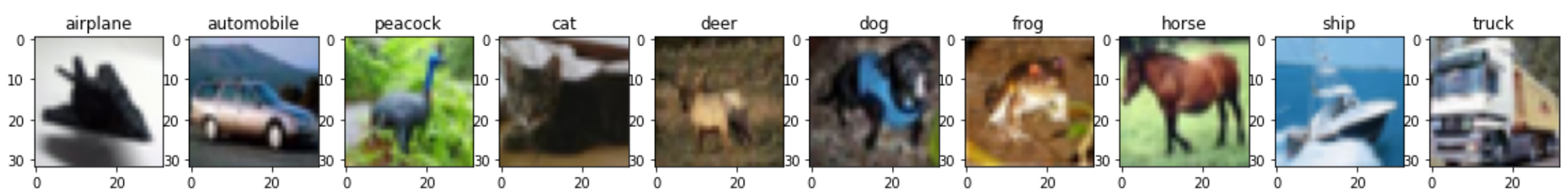
1. Please visit this link to access the state-of-art DenseNet code for reference - DenseNet - cifar10 notebook link
2. You need to create a copy of this and "retrain" this model to achieve 90+ test accuracy.
3. You cannot use DropOut layers.
4. You MUST use Image Augmentation Techniques.
5. You cannot use an already trained model as a beginning points, you have to initilize as your own

6. You cannot run the program for more than 300 Epochs, and it should be clear from your log, that you have only used 300 Epochs
7. You cannot use test images for training the model.
8. You cannot change the general architecture of DenseNet (which means you must use Dense Block, Transition and Output blocks as mentioned in the code)
9. You are free to change Convolution types (e.g. from 3x3 normal convolution to Depthwise Separable, etc)
10. You cannot have more than 1 Million parameters in total
11. You are free to move the code from Keras to Tensorflow, Pytorch, MXNET etc.
12. You can use any optimization algorithm you need.
13. You can checkpoint your model and retrain the model from that checkpoint so that no need of training the model from first if you lost at any epoch while training. You can directly load that model and Train from that epoch.

In [14]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [15]:
```
# Load CIFAR10 Data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
img_height, img_width, channel = X_train.shape[1],X_train.shape[2],X_train.shape[3]
```

In [16]:
```
## visualizing the classes
fig = plt.figure(figsize=(20,11))
classes = ['airplane','automobile','peacock','cat','deer','dog','frog','horse','ship','truck']
for i in range(10):
      plt.subplot(1,10,i+1)
      plt.imshow(X_train[np.where(y_train==i)[0][0]])
      plt.title(classes[i])
```



In [17]:
```
#### scale the pixels
## https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
# scale pixels
def scale_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255
    test_norm = test_norm / 255
    # return normalized images
    return train_norm, test_norm
```

In [18]:
```
X_train,X_test = scale_pixels(X_train,X_test)
```

In [19]:
```
# convert to one hot encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

let's change the dropout_rate to 0,alter strides to (5,5) and number of filters

In [20]:
```
# Dense Block
def denseblock(input, num_filter = 12, dropout_rate = 0):
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = layers.BatchNormalization()(temp)
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_5_5 = layers.Conv2D(int(num_filter*compression), (5,5),kernel_initializer='he_uniform' ,use_bias=False ,padding='same')(relu)
        if dropout_rate>0:
            Conv2D_5_5 = layers.Dropout(dropout_rate)(Conv2D_5_5)
        concat = layers.Concatenate(axis=-1)([temp,Conv2D_5_5])

        temp = concat

    return temp

## transition Blosck
def transition(input, num_filter = 12, dropout_rate = 0):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression),(5,5),kernel_initializer='he_uniform' ,use_bias=False ,padding='same')(relu)
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)
    return avg

#output layer
def output_layer(input):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    AvgPooling = layers.AveragePooling2D(pool_size=(2,2))(relu)
    flat = layers.Flatten()(AvgPooling)
    output = layers.Dense(num_classes, activation='softmax')(flat)
    return output
```

In [21]:

```
num_filter = 10
dropout_rate = 0
l = 12
input = layers.Input(shape=(img_height, img_width, channel))
First_Conv2D = layers.Conv2D(num_filter, (5,5), use_bias=False ,padding='same')(input)
Batchnorm = layers.BatchNormalization()(First_Conv2D)

First_Block = denseblock(Batchnorm,32, dropout_rate)
First_Transition = transition(First_Block, num_filter, dropout_rate)

Second_Block = denseblock(First_Transition, 16, dropout_rate)
Second_Transition = transition(Second_Block, num_filter, dropout_rate)

Third_Block = denseblock(Second_Transition, num_filter, dropout_rate)
Third_Transition = transition(Third_Block, num_filter, dropout_rate)

Last_Block = denseblock(Third_Transition,  num_filter, dropout_rate)
output = output_layer(Last_Block)
```

In [22]:
```
model = Model(inputs=[input],outputs=[output])
model.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 32, 32, 3)]	0	
conv2d_52 (Conv2D)	(None, 32, 32, 10)	750	input_2[0][0]
batch_normalization_52 (BatchNo	(None, 32, 32, 10)	40	conv2d_52[0][0]
batch_normalization_53 (BatchNo	(None, 32, 32, 10)	40	batch_normalization_52[0][0]
activation_52 (Activation)	(None, 32, 32, 10)	0	batch_normalization_53[0][0]
conv2d_53 (Conv2D)	(None, 32, 32, 16)	4000	activation_52[0][0]
concatenate_48 (Concatenate)	(None, 32, 32, 26)	0	batch_normalization_52[0][0] conv2d_53[0][0]
batch_normalization_54 (BatchNo	(None, 32, 32, 26)	104	concatenate_48[0][0]
activation_53 (Activation)	(None, 32, 32, 26)	0	batch_normalization_54[0][0]
conv2d_54 (Conv2D)	(None, 32, 32, 16)	10400	activation_53[0][0]
concatenate_49 (Concatenate)	(None, 32, 32, 42)	0	concatenate_48[0][0] conv2d_54[0][0]
batch_normalization_55 (BatchNo	(None, 32, 32, 42)	168	concatenate_49[0][0]
activation_54 (Activation)	(None, 32, 32, 42)	0	batch_normalization_55[0][0]
conv2d_55 (Conv2D)	(None, 32, 32, 16)	16800	activation_54[0][0]
concatenate_50 (Concatenate)	(None, 32, 32, 58)	0	concatenate_49[0][0] conv2d_55[0][0]
batch_normalization_56 (BatchNo	(None, 32, 32, 58)	232	concatenate_50[0][0]
activation_55 (Activation)	(None, 32, 32, 58)	0	batch_normalization_56[0][0]
conv2d_56 (Conv2D)	(None, 32, 32, 16)	23200	activation_55[0][0]
concatenate_51 (Concatenate)	(None, 32, 32, 74)	0	concatenate_50[0][0] conv2d_56[0][0]
batch_normalization_57 (BatchNo	(None, 32, 32, 74)	296	concatenate_51[0][0]
activation_56 (Activation)	(None, 32, 32, 74)	0	batch_normalization_57[0][0]
conv2d_57 (Conv2D)	(None, 32, 32, 16)	29600	activation_56[0][0]
concatenate_52 (Concatenate)	(None, 32, 32, 90)	0	concatenate_51[0][0] conv2d_57[0][0]

batch_normalization_58 (BatchNo	(None, 32, 32, 90)	360	concatenate_52[0][0]
activation_57 (Activation)	(None, 32, 32, 90)	0	batch_normalization_58[0][0]
conv2d_58 (Conv2D)	(None, 32, 32, 16)	36000	activation_57[0][0]
concatenate_53 (Concatenate)	(None, 32, 32, 106)	0	concatenate_52[0][0] conv2d_58[0][0]
batch_normalization_59 (BatchNo	(None, 32, 32, 106)	424	concatenate_53[0][0]
activation_58 (Activation)	(None, 32, 32, 106)	0	batch_normalization_59[0][0]
conv2d_59 (Conv2D)	(None, 32, 32, 16)	42400	activation_58[0][0]
concatenate_54 (Concatenate)	(None, 32, 32, 122)	0	concatenate_53[0][0] conv2d_59[0][0]
batch_normalization_60 (BatchNo	(None, 32, 32, 122)	488	concatenate_54[0][0]
activation_59 (Activation)	(None, 32, 32, 122)	0	batch_normalization_60[0][0]
conv2d_60 (Conv2D)	(None, 32, 32, 16)	48800	activation_59[0][0]
concatenate_55 (Concatenate)	(None, 32, 32, 138)	0	concatenate_54[0][0] conv2d_60[0][0]
batch_normalization_61 (BatchNo	(None, 32, 32, 138)	552	concatenate_55[0][0]
activation_60 (Activation)	(None, 32, 32, 138)	0	batch_normalization_61[0][0]
conv2d_61 (Conv2D)	(None, 32, 32, 16)	55200	activation_60[0][0]
concatenate_56 (Concatenate)	(None, 32, 32, 154)	0	concatenate_55[0][0] conv2d_61[0][0]
batch_normalization_62 (BatchNo	(None, 32, 32, 154)	616	concatenate_56[0][0]
activation_61 (Activation)	(None, 32, 32, 154)	0	batch_normalization_62[0][0]
conv2d_62 (Conv2D)	(None, 32, 32, 16)	61600	activation_61[0][0]
concatenate_57 (Concatenate)	(None, 32, 32, 170)	0	concatenate_56[0][0] conv2d_62[0][0]
batch_normalization_63 (BatchNo	(None, 32, 32, 170)	680	concatenate_57[0][0]
activation_62 (Activation)	(None, 32, 32, 170)	0	batch_normalization_63[0][0]
conv2d_63 (Conv2D)	(None, 32, 32, 16)	68000	activation_62[0][0]
concatenate_58 (Concatenate)	(None, 32, 32, 186)	0	concatenate_57[0][0] conv2d_63[0][0]
batch_normalization_64 (BatchNo	(None, 32, 32, 186)	744	concatenate_58[0][0]
activation_63 (Activation)	(None, 32, 32, 186)	0	batch_normalization_64[0][0]
conv2d_64 (Conv2D)	(None, 32, 32, 16)	74400	activation_63[0][0]

concatenate_59 (Concatenate)	(None, 32, 32, 202)	0	concatenate_58[0][0] conv2d_64[0][0]
batch_normalization_65 (BatchNo	(None, 32, 32, 202)	808	concatenate_59[0][0]
activation_64 (Activation)	(None, 32, 32, 202)	0	batch_normalization_65[0][0]
conv2d_65 (Conv2D)	(None, 32, 32, 5)	25250	activation_64[0][0]
average_pooling2d_4 (AveragePoo	(None, 16, 16, 5)	0	conv2d_65[0][0]
batch_normalization_66 (BatchNo	(None, 16, 16, 5)	20	average_pooling2d_4[0][0]
activation_65 (Activation)	(None, 16, 16, 5)	0	batch_normalization_66[0][0]
conv2d_66 (Conv2D)	(None, 16, 16, 8)	1000	activation_65[0][0]
concatenate_60 (Concatenate)	(None, 16, 16, 13)	0	average_pooling2d_4[0][0] conv2d_66[0][0]
batch_normalization_67 (BatchNo	(None, 16, 16, 13)	52	concatenate_60[0][0]
activation_66 (Activation)	(None, 16, 16, 13)	0	batch_normalization_67[0][0]
conv2d_67 (Conv2D)	(None, 16, 16, 8)	2600	activation_66[0][0]
concatenate_61 (Concatenate)	(None, 16, 16, 21)	0	concatenate_60[0][0] conv2d_67[0][0]
batch_normalization_68 (BatchNo	(None, 16, 16, 21)	84	concatenate_61[0][0]
activation_67 (Activation)	(None, 16, 16, 21)	0	batch_normalization_68[0][0]
conv2d_68 (Conv2D)	(None, 16, 16, 8)	4200	activation_67[0][0]
concatenate_62 (Concatenate)	(None, 16, 16, 29)	0	concatenate_61[0][0] conv2d_68[0][0]
batch_normalization_69 (BatchNo	(None, 16, 16, 29)	116	concatenate_62[0][0]
activation_68 (Activation)	(None, 16, 16, 29)	0	batch_normalization_69[0][0]
conv2d_69 (Conv2D)	(None, 16, 16, 8)	5800	activation_68[0][0]
concatenate_63 (Concatenate)	(None, 16, 16, 37)	0	concatenate_62[0][0] conv2d_69[0][0]
batch_normalization_70 (BatchNo	(None, 16, 16, 37)	148	concatenate_63[0][0]
activation_69 (Activation)	(None, 16, 16, 37)	0	batch_normalization_70[0][0]
conv2d_70 (Conv2D)	(None, 16, 16, 8)	7400	activation_69[0][0]
concatenate_64 (Concatenate)	(None, 16, 16, 45)	0	concatenate_63[0][0] conv2d_70[0][0]
batch_normalization_71 (BatchNo	(None, 16, 16, 45)	180	concatenate_64[0][0]

activation_70 (Activation)	(None, 16, 16, 45)	0	batch_normalization_71[0][0]
conv2d_71 (Conv2D)	(None, 16, 16, 8)	9000	activation_70[0][0]
concatenate_65 (Concatenate)	(None, 16, 16, 53)	0	concatenate_64[0][0] conv2d_71[0][0]
batch_normalization_72 (BatchNo	(None, 16, 16, 53)	212	concatenate_65[0][0]
activation_71 (Activation)	(None, 16, 16, 53)	0	batch_normalization_72[0][0]
conv2d_72 (Conv2D)	(None, 16, 16, 8)	10600	activation_71[0][0]
concatenate_66 (Concatenate)	(None, 16, 16, 61)	0	concatenate_65[0][0] conv2d_72[0][0]
batch_normalization_73 (BatchNo	(None, 16, 16, 61)	244	concatenate_66[0][0]
activation_72 (Activation)	(None, 16, 16, 61)	0	batch_normalization_73[0][0]
conv2d_73 (Conv2D)	(None, 16, 16, 8)	12200	activation_72[0][0]
concatenate_67 (Concatenate)	(None, 16, 16, 69)	0	concatenate_66[0][0] conv2d_73[0][0]
batch_normalization_74 (BatchNo	(None, 16, 16, 69)	276	concatenate_67[0][0]
activation_73 (Activation)	(None, 16, 16, 69)	0	batch_normalization_74[0][0]
conv2d_74 (Conv2D)	(None, 16, 16, 8)	13800	activation_73[0][0]
concatenate_68 (Concatenate)	(None, 16, 16, 77)	0	concatenate_67[0][0] conv2d_74[0][0]
batch_normalization_75 (BatchNo	(None, 16, 16, 77)	308	concatenate_68[0][0]
activation_74 (Activation)	(None, 16, 16, 77)	0	batch_normalization_75[0][0]
conv2d_75 (Conv2D)	(None, 16, 16, 8)	15400	activation_74[0][0]
concatenate_69 (Concatenate)	(None, 16, 16, 85)	0	concatenate_68[0][0] conv2d_75[0][0]
batch_normalization_76 (BatchNo	(None, 16, 16, 85)	340	concatenate_69[0][0]
activation_75 (Activation)	(None, 16, 16, 85)	0	batch_normalization_76[0][0]
conv2d_76 (Conv2D)	(None, 16, 16, 8)	17000	activation_75[0][0]
concatenate_70 (Concatenate)	(None, 16, 16, 93)	0	concatenate_69[0][0] conv2d_76[0][0]
batch_normalization_77 (BatchNo	(None, 16, 16, 93)	372	concatenate_70[0][0]
activation_76 (Activation)	(None, 16, 16, 93)	0	batch_normalization_77[0][0]
conv2d_77 (Conv2D)	(None, 16, 16, 8)	18600	activation_76[0][0]
concatenate_71 (Concatenate)	(None, 16, 16, 101)	0	concatenate_70[0][0]

			conv2d_77[0][0]
batch_normalization_78 (BatchNo	(None, 16, 16, 101)	404	concatenate_71[0][0]
activation_77 (Activation)	(None, 16, 16, 101)	0	batch_normalization_78[0][0]
conv2d_78 (Conv2D)	(None, 16, 16, 5)	12625	activation_77[0][0]
average_pooling2d_5 (AveragePoo	(None, 8, 8, 5)	0	conv2d_78[0][0]
batch_normalization_79 (BatchNo	(None, 8, 8, 5)	20	average_pooling2d_5[0][0]
activation_78 (Activation)	(None, 8, 8, 5)	0	batch_normalization_79[0][0]
conv2d_79 (Conv2D)	(None, 8, 8, 5)	625	activation_78[0][0]
concatenate_72 (Concatenate)	(None, 8, 8, 10)	0	average_pooling2d_5[0][0] conv2d_79[0][0]
batch_normalization_80 (BatchNo	(None, 8, 8, 10)	40	concatenate_72[0][0]
activation_79 (Activation)	(None, 8, 8, 10)	0	batch_normalization_80[0][0]
conv2d_80 (Conv2D)	(None, 8, 8, 5)	1250	activation_79[0][0]
concatenate_73 (Concatenate)	(None, 8, 8, 15)	0	concatenate_72[0][0] conv2d_80[0][0]
batch_normalization_81 (BatchNo	(None, 8, 8, 15)	60	concatenate_73[0][0]
activation_80 (Activation)	(None, 8, 8, 15)	0	batch_normalization_81[0][0]
conv2d_81 (Conv2D)	(None, 8, 8, 5)	1875	activation_80[0][0]
concatenate_74 (Concatenate)	(None, 8, 8, 20)	0	concatenate_73[0][0] conv2d_81[0][0]
batch_normalization_82 (BatchNo	(None, 8, 8, 20)	80	concatenate_74[0][0]
activation_81 (Activation)	(None, 8, 8, 20)	0	batch_normalization_82[0][0]
conv2d_82 (Conv2D)	(None, 8, 8, 5)	2500	activation_81[0][0]
concatenate_75 (Concatenate)	(None, 8, 8, 25)	0	concatenate_74[0][0] conv2d_82[0][0]
batch_normalization_83 (BatchNo	(None, 8, 8, 25)	100	concatenate_75[0][0]
activation_82 (Activation)	(None, 8, 8, 25)	0	batch_normalization_83[0][0]
conv2d_83 (Conv2D)	(None, 8, 8, 5)	3125	activation_82[0][0]
concatenate_76 (Concatenate)	(None, 8, 8, 30)	0	concatenate_75[0][0] conv2d_83[0][0]
batch_normalization_84 (BatchNo	(None, 8, 8, 30)	120	concatenate_76[0][0]
activation_83 (Activation)	(None, 8, 8, 30)	0	batch_normalization_84[0][0]

conv2d_84 (Conv2D)	(None, 8, 8, 5)	3750	activation_83[0][0]
concatenate_77 (Concatenate)	(None, 8, 8, 35)	0	concatenate_76[0][0] conv2d_84[0][0]
batch_normalization_85 (BatchNo	(None, 8, 8, 35)	140	concatenate_77[0][0]
activation_84 (Activation)	(None, 8, 8, 35)	0	batch_normalization_85[0][0]
conv2d_85 (Conv2D)	(None, 8, 8, 5)	4375	activation_84[0][0]
concatenate_78 (Concatenate)	(None, 8, 8, 40)	0	concatenate_77[0][0] conv2d_85[0][0]
batch_normalization_86 (BatchNo	(None, 8, 8, 40)	160	concatenate_78[0][0]
activation_85 (Activation)	(None, 8, 8, 40)	0	batch_normalization_86[0][0]
conv2d_86 (Conv2D)	(None, 8, 8, 5)	5000	activation_85[0][0]
concatenate_79 (Concatenate)	(None, 8, 8, 45)	0	concatenate_78[0][0] conv2d_86[0][0]
batch_normalization_87 (BatchNo	(None, 8, 8, 45)	180	concatenate_79[0][0]
activation_86 (Activation)	(None, 8, 8, 45)	0	batch_normalization_87[0][0]
conv2d_87 (Conv2D)	(None, 8, 8, 5)	5625	activation_86[0][0]
concatenate_80 (Concatenate)	(None, 8, 8, 50)	0	concatenate_79[0][0] conv2d_87[0][0]
batch_normalization_88 (BatchNo	(None, 8, 8, 50)	200	concatenate_80[0][0]
activation_87 (Activation)	(None, 8, 8, 50)	0	batch_normalization_88[0][0]
conv2d_88 (Conv2D)	(None, 8, 8, 5)	6250	activation_87[0][0]
concatenate_81 (Concatenate)	(None, 8, 8, 55)	0	concatenate_80[0][0] conv2d_88[0][0]
batch_normalization_89 (BatchNo	(None, 8, 8, 55)	220	concatenate_81[0][0]
activation_88 (Activation)	(None, 8, 8, 55)	0	batch_normalization_89[0][0]
conv2d_89 (Conv2D)	(None, 8, 8, 5)	6875	activation_88[0][0]
concatenate_82 (Concatenate)	(None, 8, 8, 60)	0	concatenate_81[0][0] conv2d_89[0][0]
batch_normalization_90 (BatchNo	(None, 8, 8, 60)	240	concatenate_82[0][0]
activation_89 (Activation)	(None, 8, 8, 60)	0	batch_normalization_90[0][0]
conv2d_90 (Conv2D)	(None, 8, 8, 5)	7500	activation_89[0][0]
concatenate_83 (Concatenate)	(None, 8, 8, 65)	0	concatenate_82[0][0] conv2d_90[0][0]

batch_normalization_91 (BatchNo	(None, 8, 8, 65)	260	concatenate_83[0][0]
activation_90 (Activation)	(None, 8, 8, 65)	0	batch_normalization_91[0][0]
conv2d_91 (Conv2D)	(None, 8, 8, 5)	8125	activation_90[0][0]
average_pooling2d_6 (AveragePoo	(None, 4, 4, 5)	0	conv2d_91[0][0]
batch_normalization_92 (BatchNo	(None, 4, 4, 5)	20	average_pooling2d_6[0][0]
activation_91 (Activation)	(None, 4, 4, 5)	0	batch_normalization_92[0][0]
conv2d_92 (Conv2D)	(None, 4, 4, 5)	625	activation_91[0][0]
concatenate_84 (Concatenate)	(None, 4, 4, 10)	0	average_pooling2d_6[0][0] conv2d_92[0][0]
batch_normalization_93 (BatchNo	(None, 4, 4, 10)	40	concatenate_84[0][0]
activation_92 (Activation)	(None, 4, 4, 10)	0	batch_normalization_93[0][0]
conv2d_93 (Conv2D)	(None, 4, 4, 5)	1250	activation_92[0][0]
concatenate_85 (Concatenate)	(None, 4, 4, 15)	0	concatenate_84[0][0] conv2d_93[0][0]
batch_normalization_94 (BatchNo	(None, 4, 4, 15)	60	concatenate_85[0][0]
activation_93 (Activation)	(None, 4, 4, 15)	0	batch_normalization_94[0][0]
conv2d_94 (Conv2D)	(None, 4, 4, 5)	1875	activation_93[0][0]
concatenate_86 (Concatenate)	(None, 4, 4, 20)	0	concatenate_85[0][0] conv2d_94[0][0]
batch_normalization_95 (BatchNo	(None, 4, 4, 20)	80	concatenate_86[0][0]
activation_94 (Activation)	(None, 4, 4, 20)	0	batch_normalization_95[0][0]
conv2d_95 (Conv2D)	(None, 4, 4, 5)	2500	activation_94[0][0]
concatenate_87 (Concatenate)	(None, 4, 4, 25)	0	concatenate_86[0][0] conv2d_95[0][0]
batch_normalization_96 (BatchNo	(None, 4, 4, 25)	100	concatenate_87[0][0]
activation_95 (Activation)	(None, 4, 4, 25)	0	batch_normalization_96[0][0]
conv2d_96 (Conv2D)	(None, 4, 4, 5)	3125	activation_95[0][0]
concatenate_88 (Concatenate)	(None, 4, 4, 30)	0	concatenate_87[0][0] conv2d_96[0][0]
batch_normalization_97 (BatchNo	(None, 4, 4, 30)	120	concatenate_88[0][0]
activation_96 (Activation)	(None, 4, 4, 30)	0	batch_normalization_97[0][0]
conv2d_97 (Conv2D)	(None, 4, 4, 5)	3750	activation_96[0][0]

concatenate_89 (Concatenate)	(None, 4, 4, 35)	0	concatenate_88[0][0] conv2d_97[0][0]
batch_normalization_98 (BatchNo	(None, 4, 4, 35)	140	concatenate_89[0][0]
activation_97 (Activation)	(None, 4, 4, 35)	0	batch_normalization_98[0][0]
conv2d_98 (Conv2D)	(None, 4, 4, 5)	4375	activation_97[0][0]
concatenate_90 (Concatenate)	(None, 4, 4, 40)	0	concatenate_89[0][0] conv2d_98[0][0]
batch_normalization_99 (BatchNo	(None, 4, 4, 40)	160	concatenate_90[0][0]
activation_98 (Activation)	(None, 4, 4, 40)	0	batch_normalization_99[0][0]
conv2d_99 (Conv2D)	(None, 4, 4, 5)	5000	activation_98[0][0]
concatenate_91 (Concatenate)	(None, 4, 4, 45)	0	concatenate_90[0][0] conv2d_99[0][0]
batch_normalization_100 (BatchN	(None, 4, 4, 45)	180	concatenate_91[0][0]
activation_99 (Activation)	(None, 4, 4, 45)	0	batch_normalization_100[0][0]
conv2d_100 (Conv2D)	(None, 4, 4, 5)	5625	activation_99[0][0]
concatenate_92 (Concatenate)	(None, 4, 4, 50)	0	concatenate_91[0][0] conv2d_100[0][0]
batch_normalization_101 (BatchN	(None, 4, 4, 50)	200	concatenate_92[0][0]
activation_100 (Activation)	(None, 4, 4, 50)	0	batch_normalization_101[0][0]
conv2d_101 (Conv2D)	(None, 4, 4, 5)	6250	activation_100[0][0]
concatenate_93 (Concatenate)	(None, 4, 4, 55)	0	concatenate_92[0][0] conv2d_101[0][0]
batch_normalization_102 (BatchN	(None, 4, 4, 55)	220	concatenate_93[0][0]
activation_101 (Activation)	(None, 4, 4, 55)	0	batch_normalization_102[0][0]
conv2d_102 (Conv2D)	(None, 4, 4, 5)	6875	activation_101[0][0]
concatenate_94 (Concatenate)	(None, 4, 4, 60)	0	concatenate_93[0][0] conv2d_102[0][0]
batch_normalization_103 (BatchN	(None, 4, 4, 60)	240	concatenate_94[0][0]
activation_102 (Activation)	(None, 4, 4, 60)	0	batch_normalization_103[0][0]
conv2d_103 (Conv2D)	(None, 4, 4, 5)	7500	activation_102[0][0]
concatenate_95 (Concatenate)	(None, 4, 4, 65)	0	concatenate_94[0][0] conv2d_103[0][0]
batch_normalization_104 (BatchN	(None, 4, 4, 65)	260	concatenate_95[0][0]

```
activation_103 (Activation)     (None, 4, 4, 65)     0           batch_normalization_104[0][0]
__________________________________________________________________________________________________
average_pooling2d_7 (AveragePoo (None, 2, 2, 65)     0           activation_103[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 260)          0           average_pooling2d_7[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           2610        flatten_1[0][0]
==================================================================================================
Total params: 746,808
Trainable params: 740,834
Non-trainable params: 5,974
__________________________________________________________________________________________________
```

we have 0.8 million trainable params

In [24]:
```python
from keras.preprocessing.image import ImageDataGenerator

###Data Augmentation
### https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/
# rotation_range = 15, horizontal_flip = True, width_shift_range = 0.1, height_shift_range = 0.1, zoom_range = 0.2, shear_range = 15
def augment_data():
    datagen = ImageDataGenerator(
      rotation_range=15,
      width_shift_range=0.1,
      height_shift_range=0.09,
      horizontal_flip=True,
      zoom_range=0.2,
      shear_range=15
      )
    return datagen
datagen = augment_data()
datagen.fit(X_train)
```

In [ ]:
```python
### https://keras.io/api/callbacks/model_checkpoint/
```

In [ ]:
```python
# from google.colab import drive
# drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```python
path = 'best_weights_epoch20.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:
```python
# from google.colab import drive
# drive.mount('/content/drive')
```

In [31]:

```
batch_size=64
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=20,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/20
781/781 [==============================] - 153s 148ms/step - loss: 1.7250 - accuracy: 0.3659 - val_loss: 1.6352 - val_accuracy: 0.4372
Epoch 2/20
781/781 [==============================] - 114s 145ms/step - loss: 1.4152 - accuracy: 0.4820 - val_loss: 1.3549 - val_accuracy: 0.5060
Epoch 3/20
781/781 [==============================] - 116s 148ms/step - loss: 1.2497 - accuracy: 0.5505 - val_loss: 1.5288 - val_accuracy: 0.4906
Epoch 4/20
781/781 [==============================] - 119s 153ms/step - loss: 1.1252 - accuracy: 0.5960 - val_loss: 1.1729 - val_accuracy: 0.6021
Epoch 5/20
781/781 [==============================] - 116s 148ms/step - loss: 1.0303 - accuracy: 0.6335 - val_loss: 1.1565 - val_accuracy: 0.6027
Epoch 6/20
781/781 [==============================] - 116s 149ms/step - loss: 0.9658 - accuracy: 0.6551 - val_loss: 1.0424 - val_accuracy: 0.6391
Epoch 7/20
781/781 [==============================] - 116s 149ms/step - loss: 0.9110 - accuracy: 0.6756 - val_loss: 0.8783 - val_accuracy: 0.6937
Epoch 8/20
781/781 [==============================] - 116s 148ms/step - loss: 0.8595 - accuracy: 0.6953 - val_loss: 0.9983 - val_accuracy: 0.6614
Epoch 9/20
781/781 [==============================] - 115s 147ms/step - loss: 0.8199 - accuracy: 0.7094 - val_loss: 1.0162 - val_accuracy: 0.6650
Epoch 10/20
781/781 [==============================] - 115s 147ms/step - loss: 0.7833 - accuracy: 0.7249 - val_loss: 1.1882 - val_accuracy: 0.6387
Epoch 11/20
781/781 [==============================] - 116s 149ms/step - loss: 0.7541 - accuracy: 0.7358 - val_loss: 0.9855 - val_accuracy: 0.6864
Epoch 12/20
781/781 [==============================] - 116s 149ms/step - loss: 0.7270 - accuracy: 0.7451 - val_loss: 0.8677 - val_accuracy: 0.7156
Epoch 13/20
781/781 [==============================] - 117s 149ms/step - loss: 0.6991 - accuracy: 0.7536 - val_loss: 0.6496 - val_accuracy: 0.7756
Epoch 14/20
781/781 [==============================] - 115s 147ms/step - loss: 0.6761 - accuracy: 0.7640 - val_loss: 0.7735 - val_accuracy: 0.7373
Epoch 15/20
781/781 [==============================] - 115s 147ms/step - loss: 0.6579 - accuracy: 0.7697 - val_loss: 0.6855 - val_accuracy: 0.7683
Epoch 16/20
781/781 [==============================] - 119s 152ms/step - loss: 0.6426 - accuracy: 0.7749 - val_loss: 0.8081 - val_accuracy: 0.7320
Epoch 17/20
781/781 [==============================] - 116s 149ms/step - loss: 0.6255 - accuracy: 0.7820 - val_loss: 0.6836 - val_accuracy: 0.7732
Epoch 18/20
781/781 [==============================] - 118s 151ms/step - loss: 0.6110 - accuracy: 0.7869 - val_loss: 0.6480 - val_accuracy: 0.7792
Epoch 19/20
781/781 [==============================] - 115s 148ms/step - loss: 0.5909 - accuracy: 0.7941 - val_loss: 0.5855 - val_accuracy: 0.8045
Epoch 20/20
781/781 [==============================] - 116s 148ms/step - loss: 0.5787 - accuracy: 0.7957 - val_loss: 0.6022 - val_accuracy: 0.7945
```

In [ ]:

```
# Test the model
model.load_weights(path)
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 20s 65ms/step - loss: 0.6417 - accuracy: 0.7849
Test loss: 0.641657829284668
Test accuracy: 0.7849000096321106
```

In [ ]:

```
path = 'best_weights_epoch55.h5'
```

In [ ]:

```
model.load_weights('best_weights_epoch40.h5')
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=15,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/15
781/781 [==============================] - 156s 151ms/step - loss: 0.4178 - accuracy: 0.8550 - val_loss: 0.5698 - val_accuracy: 0.8134
Epoch 2/15
781/781 [==============================] - 115s 147ms/step - loss: 0.4090 - accuracy: 0.8568 - val_loss: 0.5355 - val_accuracy: 0.8202
Epoch 3/15
781/781 [==============================] - 115s 147ms/step - loss: 0.4031 - accuracy: 0.8618 - val_loss: 0.5208 - val_accuracy: 0.8301
Epoch 4/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3976 - accuracy: 0.8615 - val_loss: 0.5456 - val_accuracy: 0.8216
Epoch 5/15
781/781 [==============================] - 115s 148ms/step - loss: 0.3936 - accuracy: 0.8625 - val_loss: 0.4491 - val_accuracy: 0.8510
Epoch 6/15
781/781 [==============================] - 115s 148ms/step - loss: 0.3928 - accuracy: 0.8623 - val_loss: 0.5136 - val_accuracy: 0.8268
Epoch 7/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3825 - accuracy: 0.8661 - val_loss: 0.4635 - val_accuracy: 0.8492
Epoch 8/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3789 - accuracy: 0.8672 - val_loss: 0.5166 - val_accuracy: 0.8311
Epoch 9/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3761 - accuracy: 0.8695 - val_loss: 0.5255 - val_accuracy: 0.8289
Epoch 10/15
781/781 [==============================] - 118s 151ms/step - loss: 0.3741 - accuracy: 0.8690 - val_loss: 0.5462 - val_accuracy: 0.8299
Epoch 11/15
781/781 [==============================] - 115s 148ms/step - loss: 0.3642 - accuracy: 0.8716 - val_loss: 0.4804 - val_accuracy: 0.8439
Epoch 12/15
781/781 [==============================] - 117s 150ms/step - loss: 0.3646 - accuracy: 0.8729 - val_loss: 0.4253 - val_accuracy: 0.8579
Epoch 13/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3571 - accuracy: 0.8751 - val_loss: 0.6388 - val_accuracy: 0.8069
Epoch 14/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3539 - accuracy: 0.8780 - val_loss: 0.5496 - val_accuracy: 0.8301
Epoch 15/15
781/781 [==============================] - 115s 147ms/step - loss: 0.3511 - accuracy: 0.8766 - val_loss: 0.4427 - val_accuracy: 0.8541
```

In [ ]:

```
!cp -r '/content/best_weights_epoch55.h5'  '/content/drive/MyDrive/'
```

In [ ]:

```
path = 'best_weights_epoch85.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:

```
model.load_weights('best_weights_epoch55.h5')
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=30,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/30
781/781 [==============================] - 157s 153ms/step - loss: 0.3638 - accuracy: 0.8736 - val_loss: 0.4842 - val_accuracy: 0.8436
Epoch 2/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3564 - accuracy: 0.8768 - val_loss: 0.6951 - val_accuracy: 0.7971
Epoch 3/30
781/781 [==============================] - 118s 151ms/step - loss: 0.3524 - accuracy: 0.8761 - val_loss: 0.5342 - val_accuracy: 0.8328
Epoch 4/30
781/781 [==============================] - 115s 148ms/step - loss: 0.3473 - accuracy: 0.8776 - val_loss: 0.4074 - val_accuracy: 0.8682
Epoch 5/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3444 - accuracy: 0.8798 - val_loss: 0.4847 - val_accuracy: 0.8424
Epoch 6/30
781/781 [==============================] - 117s 150ms/step - loss: 0.3414 - accuracy: 0.8816 - val_loss: 0.4684 - val_accuracy: 0.8437
Epoch 7/30
781/781 [==============================] - 117s 150ms/step - loss: 0.3393 - accuracy: 0.8811 - val_loss: 0.4696 - val_accuracy: 0.8452
Epoch 8/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3337 - accuracy: 0.8816 - val_loss: 0.4637 - val_accuracy: 0.8458
Epoch 9/30
781/781 [==============================] - 114s 146ms/step - loss: 0.3331 - accuracy: 0.8840 - val_loss: 0.4954 - val_accuracy: 0.8412
Epoch 10/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3297 - accuracy: 0.8843 - val_loss: 0.5015 - val_accuracy: 0.8437
Epoch 11/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3295 - accuracy: 0.8850 - val_loss: 0.5027 - val_accuracy: 0.8419
Epoch 12/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3216 - accuracy: 0.8875 - val_loss: 0.4757 - val_accuracy: 0.8480
Epoch 13/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3182 - accuracy: 0.8879 - val_loss: 0.5190 - val_accuracy: 0.8341
Epoch 14/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3188 - accuracy: 0.8900 - val_loss: 0.5112 - val_accuracy: 0.8456
Epoch 15/30
781/781 [==============================] - 114s 146ms/step - loss: 0.3218 - accuracy: 0.8868 - val_loss: 0.3919 - val_accuracy: 0.8691
Epoch 16/30
781/781 [==============================] - 114s 146ms/step - loss: 0.3119 - accuracy: 0.8889 - val_loss: 0.3960 - val_accuracy: 0.8684
Epoch 17/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3085 - accuracy: 0.8924 - val_loss: 0.5192 - val_accuracy: 0.8417
Epoch 18/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3053 - accuracy: 0.8922 - val_loss: 0.5063 - val_accuracy: 0.8392
Epoch 19/30
781/781 [==============================] - 114s 146ms/step - loss: 0.3090 - accuracy: 0.8928 - val_loss: 0.4406 - val_accuracy: 0.8595
Epoch 20/30
781/781 [==============================] - 115s 147ms/step - loss: 0.3005 - accuracy: 0.8957 - val_loss: 0.4997 - val_accuracy: 0.8426
Epoch 21/30
781/781 [==============================] - 115s 147ms/step - loss: 0.2984 - accuracy: 0.8957 - val_loss: 0.4546 - val_accuracy: 0.8527
Epoch 22/30
781/781 [==============================] - 115s 147ms/step - loss: 0.2953 - accuracy: 0.8966 - val_loss: 0.6197 - val_accuracy: 0.8173
Epoch 23/30
781/781 [==============================] - 114s 146ms/step - loss: 0.2953 - accuracy: 0.8970 - val_loss: 0.5026 - val_accuracy: 0.8380
Epoch 24/30
```

```
781/781 [==============================] - 115s 147ms/step - loss: 0.2935 - accuracy: 0.8977 - val_loss: 0.4265 - val_accuracy: 0.8623
Epoch 25/30
781/781 [==============================] - 115s 148ms/step - loss: 0.2885 - accuracy: 0.8975 - val_loss: 0.4517 - val_accuracy: 0.8579
Epoch 26/30
781/781 [==============================] - 116s 148ms/step - loss: 0.2894 - accuracy: 0.8981 - val_loss: 0.4063 - val_accuracy: 0.8677
Epoch 27/30
781/781 [==============================] - 115s 147ms/step - loss: 0.2833 - accuracy: 0.8997 - val_loss: 0.5102 - val_accuracy: 0.8440
Epoch 28/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2824 - accuracy: 0.9003 - val_loss: 0.4280 - val_accuracy: 0.8647
Epoch 29/30
781/781 [==============================] - 115s 148ms/step - loss: 0.2812 - accuracy: 0.9013 - val_loss: 0.3993 - val_accuracy: 0.8751
Epoch 30/30
781/781 [==============================] - 115s 148ms/step - loss: 0.2799 - accuracy: 0.9020 - val_loss: 0.4679 - val_accuracy: 0.8547
```

In [ ]:

```
!cp -r '/content/best_weights_epoch85.h5'  '/content/drive/MyDrive/'
```

In [ ]:

```
path = 'best_weights_epoch115.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:

```
model.load_weights('best_weights_epoch85.h5')
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=30,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

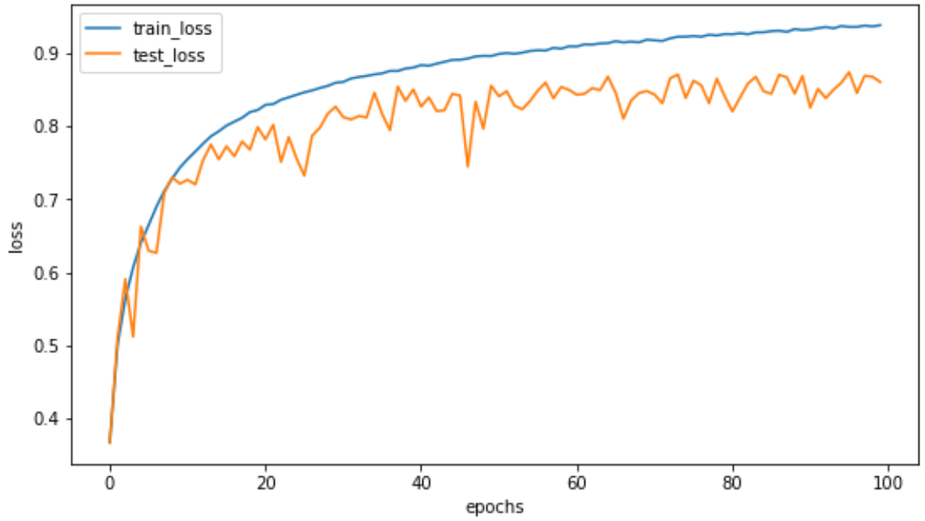
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/30
781/781 [==============================] - 159s 155ms/step - loss: 0.2775 - accuracy: 0.9017 - val_loss: 0.4250 - val_accuracy: 0.8604
Epoch 2/30
781/781 [==============================] - 121s 154ms/step - loss: 0.2778 - accuracy: 0.9017 - val_loss: 0.3903 - val_accuracy: 0.8730
Epoch 3/30
781/781 [==============================] - 120s 154ms/step - loss: 0.2780 - accuracy: 0.9028 - val_loss: 0.4269 - val_accuracy: 0.8666
Epoch 4/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2742 - accuracy: 0.9043 - val_loss: 0.4757 - val_accuracy: 0.8550
Epoch 5/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2723 - accuracy: 0.9037 - val_loss: 0.4582 - val_accuracy: 0.8588
Epoch 6/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2720 - accuracy: 0.9042 - val_loss: 0.4763 - val_accuracy: 0.8529
Epoch 7/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2643 - accuracy: 0.9076 - val_loss: 0.4387 - val_accuracy: 0.8646
Epoch 8/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2738 - accuracy: 0.9043 - val_loss: 0.5215 - val_accuracy: 0.8448
Epoch 9/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2650 - accuracy: 0.9074 - val_loss: 0.4282 - val_accuracy: 0.8688
Epoch 10/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2647 - accuracy: 0.9085 - val_loss: 0.4375 - val_accuracy: 0.8617
Epoch 11/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2594 - accuracy: 0.9094 - val_loss: 0.5873 - val_accuracy: 0.8325
Epoch 12/30
781/781 [==============================] - 120s 154ms/step - loss: 0.2656 - accuracy: 0.9068 - val_loss: 0.4245 - val_accuracy: 0.8668
Epoch 13/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2659 - accuracy: 0.9062 - val_loss: 0.4918 - val_accuracy: 0.8467
Epoch 14/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2585 - accuracy: 0.9099 - val_loss: 0.3979 - val_accuracy: 0.8731
Epoch 15/30
781/781 [==============================] - 120s 154ms/step - loss: 0.2574 - accuracy: 0.9093 - val_loss: 0.4473 - val_accuracy: 0.8671
Epoch 16/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2533 - accuracy: 0.9107 - val_loss: 0.3749 - val_accuracy: 0.8799
Epoch 17/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2496 - accuracy: 0.9116 - val_loss: 0.4054 - val_accuracy: 0.8706
Epoch 18/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2510 - accuracy: 0.9122 - val_loss: 0.4819 - val_accuracy: 0.8544
Epoch 19/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2489 - accuracy: 0.9118 - val_loss: 0.4347 - val_accuracy: 0.8655
Epoch 20/30
781/781 [==============================] - 120s 154ms/step - loss: 0.2465 - accuracy: 0.9130 - val_loss: 0.4244 - val_accuracy: 0.8721
Epoch 21/30
781/781 [==============================] - 117s 150ms/step - loss: 0.2492 - accuracy: 0.9132 - val_loss: 0.4077 - val_accuracy: 0.8744
Epoch 22/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2456 - accuracy: 0.9139 - val_loss: 0.3849 - val_accuracy: 0.8797
Epoch 23/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2471 - accuracy: 0.9132 - val_loss: 0.4045 - val_accuracy: 0.8753
Epoch 24/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2456 - accuracy: 0.9129 - val_loss: 0.3963 - val_accuracy: 0.8789
```

```
Epoch 25/30
781/781 [==============================] - 118s 151ms/step - loss: 0.2396 - accuracy: 0.9155 - val_loss: 0.4283 - val_accuracy: 0.8662
Epoch 26/30
781/781 [==============================] - 121s 154ms/step - loss: 0.2410 - accuracy: 0.9154 - val_loss: 0.4251 - val_accuracy: 0.8658
Epoch 27/30
781/781 [==============================] - 117s 149ms/step - loss: 0.2362 - accuracy: 0.9173 - val_loss: 0.4279 - val_accuracy: 0.8692
Epoch 28/30
781/781 [==============================] - 118s 150ms/step - loss: 0.2357 - accuracy: 0.9173 - val_loss: 0.4325 - val_accuracy: 0.8688
Epoch 29/30
781/781 [==============================] - 118s 150ms/step - loss: 0.2387 - accuracy: 0.9164 - val_loss: 0.4364 - val_accuracy: 0.8709
Epoch 30/30
781/781 [==============================] - 120s 154ms/step - loss: 0.2322 - accuracy: 0.9190 - val_loss: 0.3871 - val_accuracy: 0.8788
```

In [ ]:

```python
#### visualize loss and layers for 100 epochs
import seaborn as sns
def plot_accuracy(train_loss,test_loss):
    fig = plt.figure(figsize=(9,5))
    sns.lineplot(np.arange(100),train_loss,label='train_loss')
    sns.lineplot(np.arange(100),test_loss,label='test_loss')
    plt.title('Test-Train Accuracy')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.show()
plot_accuracy(history['accuracy'][:100],history['val_accuracy'][:100])
```



In [ ]:

```python
path = 'best_weights_epoch135.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:

```
model.load_weights('best_weights_epoch115.h5')
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=20,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/20
781/781 [==============================] - 123s 152ms/step - loss: 0.2464 - accuracy: 0.9125 - val_loss: 0.4448 - val_accuracy: 0.8660
Epoch 2/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2394 - accuracy: 0.9154 - val_loss: 0.4264 - val_accuracy: 0.8678
Epoch 3/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2428 - accuracy: 0.9146 - val_loss: 0.4666 - val_accuracy: 0.8594
Epoch 4/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2419 - accuracy: 0.9137 - val_loss: 0.4198 - val_accuracy: 0.8711
Epoch 5/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2370 - accuracy: 0.9173 - val_loss: 0.4397 - val_accuracy: 0.8677
Epoch 6/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2396 - accuracy: 0.9161 - val_loss: 0.4658 - val_accuracy: 0.8606
Epoch 7/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2361 - accuracy: 0.9169 - val_loss: 0.4918 - val_accuracy: 0.8523
Epoch 8/20
781/781 [==============================] - 120s 154ms/step - loss: 0.2391 - accuracy: 0.9155 - val_loss: 0.3854 - val_accuracy: 0.8811
Epoch 9/20
781/781 [==============================] - 121s 154ms/step - loss: 0.2364 - accuracy: 0.9173 - val_loss: 0.3571 - val_accuracy: 0.8845
Epoch 10/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2327 - accuracy: 0.9168 - val_loss: 0.4488 - val_accuracy: 0.8686
Epoch 11/20
781/781 [==============================] - 118s 150ms/step - loss: 0.2303 - accuracy: 0.9194 - val_loss: 0.4265 - val_accuracy: 0.8689
Epoch 12/20
781/781 [==============================] - 121s 154ms/step - loss: 0.2298 - accuracy: 0.9200 - val_loss: 0.4837 - val_accuracy: 0.8580
Epoch 13/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2275 - accuracy: 0.9199 - val_loss: 0.4764 - val_accuracy: 0.8589
Epoch 14/20
781/781 [==============================] - 120s 153ms/step - loss: 0.2247 - accuracy: 0.9207 - val_loss: 0.4889 - val_accuracy: 0.8556
Epoch 15/20
781/781 [==============================] - 120s 153ms/step - loss: 0.2259 - accuracy: 0.9202 - val_loss: 0.3972 - val_accuracy: 0.8767
Epoch 16/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2285 - accuracy: 0.9189 - val_loss: 0.4497 - val_accuracy: 0.8646
Epoch 17/20
781/781 [==============================] - 118s 150ms/step - loss: 0.2281 - accuracy: 0.9196 - val_loss: 0.4538 - val_accuracy: 0.8671
Epoch 18/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2272 - accuracy: 0.9210 - val_loss: 0.4015 - val_accuracy: 0.8732
Epoch 19/20
781/781 [==============================] - 120s 154ms/step - loss: 0.2240 - accuracy: 0.9225 - val_loss: 0.3444 - val_accuracy: 0.8910
Epoch 20/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2217 - accuracy: 0.9205 - val_loss: 0.4281 - val_accuracy: 0.8700
```

In [ ]:

```
path = 'best_weights_epoch155.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:

```
model.load_weights('best_weights_epoch135.h5')
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=20,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/20
781/781 [==============================] - 123s 152ms/step - loss: 0.2206 - accuracy: 0.9218 - val_loss: 0.3823 - val_accuracy: 0.8819
Epoch 2/20
781/781 [==============================] - 120s 153ms/step - loss: 0.2188 - accuracy: 0.9247 - val_loss: 0.3703 - val_accuracy: 0.8842
Epoch 3/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2201 - accuracy: 0.9229 - val_loss: 0.4306 - val_accuracy: 0.8683
Epoch 4/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2195 - accuracy: 0.9221 - val_loss: 0.4011 - val_accuracy: 0.8799
Epoch 5/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2172 - accuracy: 0.9235 - val_loss: 0.5441 - val_accuracy: 0.8507
Epoch 6/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2171 - accuracy: 0.9238 - val_loss: 0.3852 - val_accuracy: 0.8819
Epoch 7/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2157 - accuracy: 0.9233 - val_loss: 0.3958 - val_accuracy: 0.8797
Epoch 8/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2126 - accuracy: 0.9238 - val_loss: 0.3593 - val_accuracy: 0.8885
Epoch 9/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2091 - accuracy: 0.9266 - val_loss: 0.4084 - val_accuracy: 0.8785
Epoch 10/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2155 - accuracy: 0.9242 - val_loss: 0.3320 - val_accuracy: 0.8986
Epoch 11/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2121 - accuracy: 0.9251 - val_loss: 0.4551 - val_accuracy: 0.8667
Epoch 12/20
781/781 [==============================] - 117s 149ms/step - loss: 0.2092 - accuracy: 0.9267 - val_loss: 0.3868 - val_accuracy: 0.8774
Epoch 13/20
781/781 [==============================] - 116s 149ms/step - loss: 0.2101 - accuracy: 0.9244 - val_loss: 0.4343 - val_accuracy: 0.8744
Epoch 14/20
781/781 [==============================] - 120s 153ms/step - loss: 0.2075 - accuracy: 0.9261 - val_loss: 0.5316 - val_accuracy: 0.8532
Epoch 15/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2021 - accuracy: 0.9279 - val_loss: 0.4122 - val_accuracy: 0.8747
Epoch 16/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2048 - accuracy: 0.9273 - val_loss: 0.4393 - val_accuracy: 0.8698
Epoch 17/20
781/781 [==============================] - 120s 154ms/step - loss: 0.2031 - accuracy: 0.9291 - val_loss: 0.4312 - val_accuracy: 0.8744
Epoch 18/20
781/781 [==============================] - 118s 150ms/step - loss: 0.2036 - accuracy: 0.9278 - val_loss: 0.3904 - val_accuracy: 0.8876
Epoch 19/20
781/781 [==============================] - 118s 151ms/step - loss: 0.2002 - accuracy: 0.9280 - val_loss: 0.4365 - val_accuracy: 0.8758
Epoch 20/20
781/781 [==============================] - 117s 150ms/step - loss: 0.2048 - accuracy: 0.9276 - val_loss: 0.4986 - val_accuracy: 0.8578
```

In [ ]:

```
path = 'best_weights_epoch185.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

In [ ]:

```
model.load_weights('best_weights_epoch155.h5')
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),\
                    steps_per_epoch=X_train.shape[0] // batch_size,epochs=30,\
                    verbose=1,validation_data=(X_test,y_test),callbacks=[checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/30
781/781 [==============================] - 289s 325ms/step - loss: 0.2166 - accuracy: 0.9234 - val_loss: 0.3505 - val_accuracy: 0.8853
Epoch 2/30
781/781 [==============================] - 252s 322ms/step - loss: 0.2083 - accuracy: 0.9269 - val_loss: 0.4370 - val_accuracy: 0.8720
Epoch 3/30
781/781 [==============================] - 252s 322ms/step - loss: 0.2087 - accuracy: 0.9259 - val_loss: 0.3576 - val_accuracy: 0.8885
Epoch 4/30
781/781 [==============================] - 251s 321ms/step - loss: 0.2052 - accuracy: 0.9279 - val_loss: 0.4450 - val_accuracy: 0.8709
Epoch 5/30
781/781 [==============================] - 251s 322ms/step - loss: 0.2071 - accuracy: 0.9259 - val_loss: 0.3825 - val_accuracy: 0.8850
Epoch 6/30
781/781 [==============================] - 250s 320ms/step - loss: 0.2048 - accuracy: 0.9282 - val_loss: 0.4575 - val_accuracy: 0.8689
Epoch 7/30
781/781 [==============================] - 251s 322ms/step - loss: 0.2066 - accuracy: 0.9261 - val_loss: 0.4060 - val_accuracy: 0.8728
Epoch 8/30
781/781 [==============================] - 252s 322ms/step - loss: 0.2004 - accuracy: 0.9304 - val_loss: 0.4743 - val_accuracy: 0.8632
Epoch 9/30
781/781 [==============================] - 252s 323ms/step - loss: 0.2040 - accuracy: 0.9277 - val_loss: 0.5136 - val_accuracy: 0.8573
Epoch 10/30
781/781 [==============================] - 252s 323ms/step - loss: 0.2023 - accuracy: 0.9283 - val_loss: 0.4570 - val_accuracy: 0.8673
Epoch 11/30
781/781 [==============================] - 252s 322ms/step - loss: 0.2016 - accuracy: 0.9285 - val_loss: 0.3484 - val_accuracy: 0.8919
Epoch 12/30
781/781 [==============================] - 252s 322ms/step - loss: 0.2013 - accuracy: 0.9304 - val_loss: 0.4652 - val_accuracy: 0.8676
Epoch 13/30
781/781 [==============================] - 251s 322ms/step - loss: 0.1974 - accuracy: 0.9303 - val_loss: 0.4059 - val_accuracy: 0.8796
Epoch 14/30
781/781 [==============================] - 251s 322ms/step - loss: 0.1969 - accuracy: 0.9296 - val_loss: 0.3836 - val_accuracy: 0.8852
Epoch 15/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1967 - accuracy: 0.9305 - val_loss: 0.4415 - val_accuracy: 0.8747
Epoch 16/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1957 - accuracy: 0.9318 - val_loss: 0.4962 - val_accuracy: 0.8616
Epoch 17/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1955 - accuracy: 0.9305 - val_loss: 0.4071 - val_accuracy: 0.8809
Epoch 18/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1949 - accuracy: 0.9318 - val_loss: 0.3903 - val_accuracy: 0.8855
Epoch 19/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1969 - accuracy: 0.9311 - val_loss: 0.5107 - val_accuracy: 0.8593
Epoch 20/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1943 - accuracy: 0.9310 - val_loss: 0.4738 - val_accuracy: 0.8632
Epoch 21/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1913 - accuracy: 0.9332 - val_loss: 0.4201 - val_accuracy: 0.8777
Epoch 22/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1975 - accuracy: 0.9299 - val_loss: 0.4235 - val_accuracy: 0.8777
Epoch 23/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1936 - accuracy: 0.9306 - val_loss: 0.4826 - val_accuracy: 0.8654
Epoch 24/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1891 - accuracy: 0.9334 - val_loss: 0.4679 - val_accuracy: 0.8737
```

```
Epoch 25/30
781/781 [==============================] - 252s 323ms/step - loss: 0.1905 - accuracy: 0.9321 - val_loss: 0.4436 - val_accuracy: 0.8726
Epoch 26/30
781/781 [==============================] - 251s 322ms/step - loss: 0.1893 - accuracy: 0.9331 - val_loss: 0.4268 - val_accuracy: 0.8731
Epoch 27/30
781/781 [==============================] - 251s 322ms/step - loss: 0.1901 - accuracy: 0.9319 - val_loss: 0.4381 - val_accuracy: 0.8742
Epoch 28/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1898 - accuracy: 0.9334 - val_loss: 0.4332 - val_accuracy: 0.8778
Epoch 29/30
781/781 [==============================] - 252s 322ms/step - loss: 0.1888 - accuracy: 0.9336 - val_loss: 0.4167 - val_accuracy: 0.8814
Epoch 30/30
781/781 [==============================] - 250s 320ms/step - loss: 0.1852 - accuracy: 0.9344 - val_loss: 0.4013 - val_accuracy: 0.8836
```

let's use SGD optimizer with momentum 0.6 and learning rate 0.001 to converge faster to optimal solution

In [32]:
```
from keras.optimizers import  SGD
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
model.load_weights('best_weights_epoch185.h5')


path = 'best_weights_epoch205.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

model.compile(loss='categorical_crossentropy', optimizer=SGD(momentum=0.6,learning_rate=0.001),
              metrics=['accuracy'])
model.fit_generator(datagen.flow(X_train, y_train, batch_size),
                    steps_per_epoch = X_train.shape[0]//batch_size,
                    epochs = 20, validation_data =(X_test, y_test), callbacks = [checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/20
390/390 [==============================] - 156s 297ms/step - loss: 0.1792 - accuracy: 0.9370 - val_loss: 0.3536 - val_accuracy: 0.8930
Epoch 2/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1709 - accuracy: 0.9398 - val_loss: 0.3498 - val_accuracy: 0.8952
Epoch 3/20
390/390 [==============================] - 111s 284ms/step - loss: 0.1693 - accuracy: 0.9402 - val_loss: 0.3473 - val_accuracy: 0.8960
Epoch 4/20
390/390 [==============================] - 111s 285ms/step - loss: 0.1648 - accuracy: 0.9417 - val_loss: 0.3457 - val_accuracy: 0.8964
Epoch 5/20
390/390 [==============================] - 114s 291ms/step - loss: 0.1633 - accuracy: 0.9437 - val_loss: 0.3446 - val_accuracy: 0.8973
Epoch 6/20
390/390 [==============================] - 111s 284ms/step - loss: 0.1607 - accuracy: 0.9442 - val_loss: 0.3433 - val_accuracy: 0.8976
Epoch 7/20
390/390 [==============================] - 111s 283ms/step - loss: 0.1602 - accuracy: 0.9438 - val_loss: 0.3416 - val_accuracy: 0.8979
Epoch 8/20
390/390 [==============================] - 110s 283ms/step - loss: 0.1598 - accuracy: 0.9446 - val_loss: 0.3435 - val_accuracy: 0.8974
Epoch 9/20
390/390 [==============================] - 110s 282ms/step - loss: 0.1581 - accuracy: 0.9447 - val_loss: 0.3417 - val_accuracy: 0.8988
Epoch 10/20
390/390 [==============================] - 110s 282ms/step - loss: 0.1569 - accuracy: 0.9449 - val_loss: 0.3411 - val_accuracy: 0.8988
Epoch 11/20
390/390 [==============================] - 109s 280ms/step - loss: 0.1530 - accuracy: 0.9462 - val_loss: 0.3409 - val_accuracy: 0.8986
Epoch 12/20
390/390 [==============================] - 110s 282ms/step - loss: 0.1561 - accuracy: 0.9458 - val_loss: 0.3421 - val_accuracy: 0.8988
Epoch 13/20
390/390 [==============================] - 110s 283ms/step - loss: 0.1568 - accuracy: 0.9443 - val_loss: 0.3422 - val_accuracy: 0.8991
Epoch 14/20
390/390 [==============================] - 109s 280ms/step - loss: 0.1578 - accuracy: 0.9441 - val_loss: 0.3421 - val_accuracy: 0.8994
Epoch 15/20
390/390 [==============================] - 111s 283ms/step - loss: 0.1524 - accuracy: 0.9466 - val_loss: 0.3418 - val_accuracy: 0.8994
Epoch 16/20
390/390 [==============================] - 110s 283ms/step - loss: 0.1519 - accuracy: 0.9466 - val_loss: 0.3425 - val_accuracy: 0.8995
Epoch 17/20
390/390 [==============================] - 109s 280ms/step - loss: 0.1509 - accuracy: 0.9479 - val_loss: 0.3415 - val_accuracy: 0.8997
Epoch 18/20
390/390 [==============================] - 110s 282ms/step - loss: 0.1516 - accuracy: 0.9476 - val_loss: 0.3397 - val_accuracy: 0.9006
Epoch 19/20
```

```
390/390 [==============================] - 110s 283ms/step - loss: 0.1492 - accuracy: 0.9478 - val_loss: 0.3425 - val_accuracy: 0.8990
Epoch 20/20
390/390 [==============================] - 109s 280ms/step - loss: 0.1478 - accuracy: 0.9485 - val_loss: 0.3391 - val_accuracy: 0.9001
```

Out[20]:
```
<tensorflow.python.keras.callbacks.History at 0x7f857abd6d50>
```

In [ ]:
```
model.load_weights('best_weights_epoch205.h5')

path = 'best_weights_epoch215.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

model.compile(loss='categorical_crossentropy', optimizer=SGD(momentum=0.6,learning_rate=0.001),
              metrics=['accuracy'])
model.fit_generator(datagen.flow(X_train, y_train, batch_size),
                    steps_per_epoch = X_train.shape[0]//batch_size,
                    epochs = 10, validation_data =(X_test, y_test), callbacks = [checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
390/390 [==============================] - 110s 274ms/step - loss: 0.1534 - accuracy: 0.9461 - val_loss: 0.3416 - val_accuracy: 0.8999
Epoch 2/10
390/390 [==============================] - 108s 278ms/step - loss: 0.1512 - accuracy: 0.9466 - val_loss: 0.3415 - val_accuracy: 0.9004
Epoch 3/10
390/390 [==============================] - 112s 286ms/step - loss: 0.1474 - accuracy: 0.9488 - val_loss: 0.3395 - val_accuracy: 0.8999
Epoch 4/10
390/390 [==============================] - 112s 287ms/step - loss: 0.1508 - accuracy: 0.9481 - val_loss: 0.3398 - val_accuracy: 0.9005
Epoch 5/10
390/390 [==============================] - 110s 283ms/step - loss: 0.1464 - accuracy: 0.9482 - val_loss: 0.3400 - val_accuracy: 0.9002
Epoch 6/10
390/390 [==============================] - 112s 288ms/step - loss: 0.1506 - accuracy: 0.9477 - val_loss: 0.3434 - val_accuracy: 0.8999
Epoch 7/10
390/390 [==============================] - 109s 280ms/step - loss: 0.1479 - accuracy: 0.9475 - val_loss: 0.3406 - val_accuracy: 0.8999
Epoch 8/10
390/390 [==============================] - 109s 280ms/step - loss: 0.1476 - accuracy: 0.9479 - val_loss: 0.3425 - val_accuracy: 0.8997
Epoch 9/10
390/390 [==============================] - 109s 280ms/step - loss: 0.1484 - accuracy: 0.9467 - val_loss: 0.3421 - val_accuracy: 0.8995
Epoch 10/10
390/390 [==============================] - 111s 284ms/step - loss: 0.1478 - accuracy: 0.9488 - val_loss: 0.3413 - val_accuracy: 0.8997
```

Out[21]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8577e4a4d0>
```

In [33]:
```
model.load_weights('best_weights_epoch215.h5')
# model.optimizer = SGD()
# keras.backend.set_value(model.optimizer.momentum, 0.7)
# keras.backend.set_value(model.optimizer.lr, 0.001)

path = 'best_weights_epoch225.h5'
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath=path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

model.compile(loss='categorical_crossentropy', optimizer=SGD(momentum=0.6,learning_rate=0.001),
              metrics=['accuracy'])
model.fit_generator(datagen.flow(X_train, y_train, batch_size),
                    steps_per_epoch = X_train.shape[0]//batch_size,
                    epochs = 10, validation_data =(X_test, y_test), callbacks = [checkpoint])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
390/390 [==============================] - 158s 302ms/step - loss: 0.1525 - accuracy: 0.9464 - val_loss: 0.3418 - val_accuracy: 0.8997
Epoch 2/10
390/390 [==============================] - 112s 287ms/step - loss: 0.1480 - accuracy: 0.9478 - val_loss: 0.3422 - val_accuracy: 0.8995
Epoch 3/10
390/390 [==============================] - 112s 286ms/step - loss: 0.1490 - accuracy: 0.9479 - val_loss: 0.3419 - val_accuracy: 0.8997
Epoch 4/10
390/390 [==============================] - 112s 287ms/step - loss: 0.1480 - accuracy: 0.9484 - val_loss: 0.3413 - val_accuracy: 0.9009
Epoch 5/10
390/390 [==============================] - 112s 286ms/step - loss: 0.1488 - accuracy: 0.9481 - val_loss: 0.3405 - val_accuracy: 0.9009
Epoch 6/10
390/390 [==============================] - 112s 286ms/step - loss: 0.1487 - accuracy: 0.9476 - val_loss: 0.3414 - val_accuracy: 0.8996
Epoch 7/10
390/390 [==============================] - 111s 286ms/step - loss: 0.1461 - accuracy: 0.9488 - val_loss: 0.3421 - val_accuracy: 0.8998
Epoch 8/10
390/390 [==============================] - 112s 286ms/step - loss: 0.1444 - accuracy: 0.9498 - val_loss: 0.3410 - val_accuracy: 0.9006
Epoch 9/10
390/390 [==============================] - 111s 286ms/step - loss: 0.1465 - accuracy: 0.9492 - val_loss: 0.3383 - val_accuracy: 0.9012
Epoch 10/10
390/390 [==============================] - 114s 291ms/step - loss: 0.1496 - accuracy: 0.9467 - val_loss: 0.3405 - val_accuracy: 0.9009
```

Out[33]: <tensorflow.python.keras.callbacks.History at 0x7fbe28df7c50>

In [34]:
```
# Test the model
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 8s 26ms/step - loss: 0.3405 - accuracy: 0.9009
Test loss: 0.340536504983902
Test accuracy: 0.9009000062942505
```

In [ ]:
```

```

In [ ]:
```
# Save the trained weights in to .h5 formath
# model.save_weights("DNST_model_100epochs.h6")
# print("Saved model to disk")
```

```
Saved model to disk
```

After training with 215 epochs we could get val-accuracy of 0.9009

In [ ]:
```

```