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.

With Dense Layer

In [1]:

```
from tensorflow.keras import models, layers
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras.callbacks import ModelCheckpoint, EarlyStopping
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.python.keras.callbacks import LearningRateScheduler, ReduceLROnPlateau
```

In [2]:

```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 100
l = 40
num_filter = 12
compression = 1
dropout_rate = 0.0
```

In [3]:

```
# Load CIFAR10 Data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
image_height, image_width, channel_size = 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
170500096/170498071 [==============================] - 2s 0us/step
```

In [4]:

```
X_train.shape, X_test.shape
```

Out[4]:

((50000, 32, 32, 3), (10000, 32, 32, 3))

In [5]:

```
y_train.shape, y_test.shape
```

Out[5]:

((50000, 10), (10000, 10))

In [6]:

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [7]:

```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:

```
(X_train_plot, y_train_plot), (X_test_plot, y_test_plot) = tf.keras.datasets.cifar10.load_data()
```

In [9]:

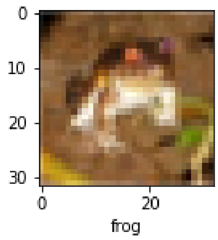
```
y_train_plot = y_train_plot.reshape(-1,)
y_train_plot[:5]
```

Out[9]:

array([6, 9, 9, 4, 1], dtype=uint8)

In [10]:

```
plot_sample(X_train_plot, y_train_plot, 0)
```



In [11]:

```
#Normalizing the data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [12]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from numpy import expand_dims
```

In [13]:

```
X_train[0].shape
```

Out[13]:

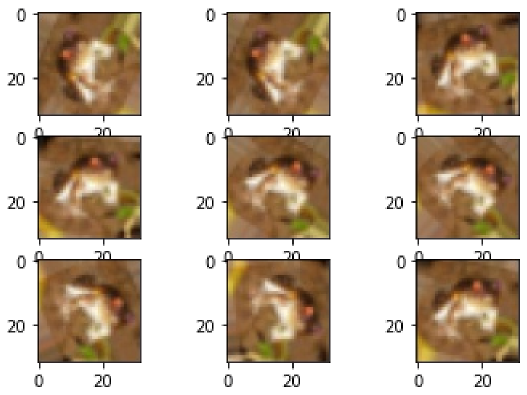
Out[13]:

```
(32, 32, 3)
```

In [14]:

```
#https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learnin
g-neural-networks/
image = expand_dims(X_train[0], 0)
dg = ImageDataGenerator(rotation_range=90)
it = dg.flow(image, batch_size=1)
for i in range(9):
    plt.subplot(330 + 1 + i) #330 means 3x3 grid and 1+i shifts the place of augmented image
    batch = it.next()
    img = batch[0];
    plt.imshow(img)

plt.show()
```



In [15]:

```
from keras import regularizers
```

DenseNet Architecture

In [16]:

```
# 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 Block
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
```

```python
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 [17]:

```python
tf.keras.backend.clear_session()
```

In [18]:

```python
dropout_rate = 0
num_filter = 30
l = 7
input = layers.Input(shape=(image_height, image_width, channel_size,))
First_Conv2D = layers.Conv2D(num_filter, (3,3), use_bias=False ,padding='same')(input)
#BatchNorm = layers.BatchNormalization()(First_Conv2D)

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 [19]:

```python
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, 30)   810         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 32, 32, 30)   120         conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 32, 32, 30)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 30)   8100        activation[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 32, 32, 60)   0           conv2d[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 32, 32, 60)   240         concatenate[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 32, 32, 60)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 32, 32, 30)   16200       activation_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32, 32, 90)   0           concatenate[0][0]
                                                                 conv2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 32, 90)   360         concatenate_1[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 32, 32, 90)   0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 32, 32, 30)   24300       activation_2[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 32, 32, 120)  0           concatenate_1[0][0]
                                                                 conv2d_3[0][0]
```

batch_normalization_3 (BatchNor	(None, 32, 32, 120)	480	concatenate_2[0][0]
activation_3 (Activation)	(None, 32, 32, 120)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 32, 32, 30)	32400	activation_3[0][0]
concatenate_3 (Concatenate)	(None, 32, 32, 150)	0	concatenate_2[0][0] conv2d_4[0][0]
batch_normalization_4 (BatchNor	(None, 32, 32, 150)	600	concatenate_3[0][0]
activation_4 (Activation)	(None, 32, 32, 150)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 30)	40500	activation_4[0][0]
concatenate_4 (Concatenate)	(None, 32, 32, 180)	0	concatenate_3[0][0] conv2d_5[0][0]
batch_normalization_5 (BatchNor	(None, 32, 32, 180)	720	concatenate_4[0][0]
activation_5 (Activation)	(None, 32, 32, 180)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 30)	48600	activation_5[0][0]
concatenate_5 (Concatenate)	(None, 32, 32, 210)	0	concatenate_4[0][0] conv2d_6[0][0]
batch_normalization_6 (BatchNor	(None, 32, 32, 210)	840	concatenate_5[0][0]
activation_6 (Activation)	(None, 32, 32, 210)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 30)	56700	activation_6[0][0]
concatenate_6 (Concatenate)	(None, 32, 32, 240)	0	concatenate_5[0][0] conv2d_7[0][0]
batch_normalization_7 (BatchNor	(None, 32, 32, 240)	960	concatenate_6[0][0]
activation_7 (Activation)	(None, 32, 32, 240)	0	batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 30)	7200	activation_7[0][0]
average_pooling2d (AveragePooli	(None, 16, 16, 30)	0	conv2d_8[0][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 30)	120	average_pooling2d[0][0]
activation_8 (Activation)	(None, 16, 16, 30)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 30)	8100	activation_8[0][0]
concatenate_7 (Concatenate)	(None, 16, 16, 60)	0	average_pooling2d[0][0] conv2d_9[0][0]
batch_normalization_9 (BatchNor	(None, 16, 16, 60)	240	concatenate_7[0][0]
activation_9 (Activation)	(None, 16, 16, 60)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 30)	16200	activation_9[0][0]
concatenate_8 (Concatenate)	(None, 16, 16, 90)	0	concatenate_7[0][0] conv2d_10[0][0]
batch_normalization_10 (BatchNo	(None, 16, 16, 90)	360	concatenate_8[0][0]
activation_10 (Activation)	(None, 16, 16, 90)	0	batch_normalization_10[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 30)	24300	activation_10[0][0]
concatenate_9 (Concatenate)	(None, 16, 16, 120)	0	concatenate_8[0][0] conv2d_11[0][0]
batch_normalization_11 (BatchNo	(None, 16, 16, 120)	480	concatenate_9[0][0]
activation_11 (Activation)	(None, 16, 16, 120)	0	batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 16, 16, 30)	32400	activation_11[0][0]

concatenate_10 (Concatenate)	(None, 16, 16, 150)	0	concatenate_9[0][0] conv2d_12[0][0]
batch_normalization_12 (BatchNo	(None, 16, 16, 150)	600	concatenate_10[0][0]
activation_12 (Activation)	(None, 16, 16, 150)	0	batch_normalization_12[0][0]
conv2d_13 (Conv2D)	(None, 16, 16, 30)	40500	activation_12[0][0]
concatenate_11 (Concatenate)	(None, 16, 16, 180)	0	concatenate_10[0][0] conv2d_13[0][0]
batch_normalization_13 (BatchNo	(None, 16, 16, 180)	720	concatenate_11[0][0]
activation_13 (Activation)	(None, 16, 16, 180)	0	batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 16, 16, 30)	48600	activation_13[0][0]
concatenate_12 (Concatenate)	(None, 16, 16, 210)	0	concatenate_11[0][0] conv2d_14[0][0]
batch_normalization_14 (BatchNo	(None, 16, 16, 210)	840	concatenate_12[0][0]
activation_14 (Activation)	(None, 16, 16, 210)	0	batch_normalization_14[0][0]
conv2d_15 (Conv2D)	(None, 16, 16, 30)	56700	activation_14[0][0]
concatenate_13 (Concatenate)	(None, 16, 16, 240)	0	concatenate_12[0][0] conv2d_15[0][0]
batch_normalization_15 (BatchNo	(None, 16, 16, 240)	960	concatenate_13[0][0]
activation_15 (Activation)	(None, 16, 16, 240)	0	batch_normalization_15[0][0]
conv2d_16 (Conv2D)	(None, 16, 16, 30)	7200	activation_15[0][0]
average_pooling2d_1 (AveragePoo	(None, 8, 8, 30)	0	conv2d_16[0][0]
batch_normalization_16 (BatchNo	(None, 8, 8, 30)	120	average_pooling2d_1[0][0]
activation_16 (Activation)	(None, 8, 8, 30)	0	batch_normalization_16[0][0]
conv2d_17 (Conv2D)	(None, 8, 8, 30)	8100	activation_16[0][0]
concatenate_14 (Concatenate)	(None, 8, 8, 60)	0	average_pooling2d_1[0][0] conv2d_17[0][0]
batch_normalization_17 (BatchNo	(None, 8, 8, 60)	240	concatenate_14[0][0]
activation_17 (Activation)	(None, 8, 8, 60)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 8, 8, 30)	16200	activation_17[0][0]
concatenate_15 (Concatenate)	(None, 8, 8, 90)	0	concatenate_14[0][0] conv2d_18[0][0]
batch_normalization_18 (BatchNo	(None, 8, 8, 90)	360	concatenate_15[0][0]
activation_18 (Activation)	(None, 8, 8, 90)	0	batch_normalization_18[0][0]
conv2d_19 (Conv2D)	(None, 8, 8, 30)	24300	activation_18[0][0]
concatenate_16 (Concatenate)	(None, 8, 8, 120)	0	concatenate_15[0][0] conv2d_19[0][0]
batch_normalization_19 (BatchNo	(None, 8, 8, 120)	480	concatenate_16[0][0]
activation_19 (Activation)	(None, 8, 8, 120)	0	batch_normalization_19[0][0]
conv2d_20 (Conv2D)	(None, 8, 8, 30)	32400	activation_19[0][0]
concatenate_17 (Concatenate)	(None, 8, 8, 150)	0	concatenate_16[0][0] conv2d_20[0][0]
batch_normalization_20 (BatchNo	(None, 8, 8, 150)	600	concatenate_17[0][0]

activation_20 (Activation)	(None, 8, 8, 150)	0	batch_normalization_20[0][0]
conv2d_21 (Conv2D)	(None, 8, 8, 30)	40500	activation_20[0][0]
concatenate_18 (Concatenate)	(None, 8, 8, 180)	0	concatenate_17[0][0] conv2d_21[0][0]
batch_normalization_21 (BatchNo	(None, 8, 8, 180)	720	concatenate_18[0][0]
activation_21 (Activation)	(None, 8, 8, 180)	0	batch_normalization_21[0][0]
conv2d_22 (Conv2D)	(None, 8, 8, 30)	48600	activation_21[0][0]
concatenate_19 (Concatenate)	(None, 8, 8, 210)	0	concatenate_18[0][0] conv2d_22[0][0]
batch_normalization_22 (BatchNo	(None, 8, 8, 210)	840	concatenate_19[0][0]
activation_22 (Activation)	(None, 8, 8, 210)	0	batch_normalization_22[0][0]
conv2d_23 (Conv2D)	(None, 8, 8, 30)	56700	activation_22[0][0]
concatenate_20 (Concatenate)	(None, 8, 8, 240)	0	concatenate_19[0][0] conv2d_23[0][0]
batch_normalization_23 (BatchNo	(None, 8, 8, 240)	960	concatenate_20[0][0]
activation_23 (Activation)	(None, 8, 8, 240)	0	batch_normalization_23[0][0]
conv2d_24 (Conv2D)	(None, 8, 8, 30)	7200	activation_23[0][0]
average_pooling2d_2 (AveragePoo	(None, 4, 4, 30)	0	conv2d_24[0][0]
batch_normalization_24 (BatchNo	(None, 4, 4, 30)	120	average_pooling2d_2[0][0]
activation_24 (Activation)	(None, 4, 4, 30)	0	batch_normalization_24[0][0]
conv2d_25 (Conv2D)	(None, 4, 4, 30)	8100	activation_24[0][0]
concatenate_21 (Concatenate)	(None, 4, 4, 60)	0	average_pooling2d_2[0][0] conv2d_25[0][0]
batch_normalization_25 (BatchNo	(None, 4, 4, 60)	240	concatenate_21[0][0]
activation_25 (Activation)	(None, 4, 4, 60)	0	batch_normalization_25[0][0]
conv2d_26 (Conv2D)	(None, 4, 4, 30)	16200	activation_25[0][0]
concatenate_22 (Concatenate)	(None, 4, 4, 90)	0	concatenate_21[0][0] conv2d_26[0][0]
batch_normalization_26 (BatchNo	(None, 4, 4, 90)	360	concatenate_22[0][0]
activation_26 (Activation)	(None, 4, 4, 90)	0	batch_normalization_26[0][0]
conv2d_27 (Conv2D)	(None, 4, 4, 30)	24300	activation_26[0][0]
concatenate_23 (Concatenate)	(None, 4, 4, 120)	0	concatenate_22[0][0] conv2d_27[0][0]
batch_normalization_27 (BatchNo	(None, 4, 4, 120)	480	concatenate_23[0][0]
activation_27 (Activation)	(None, 4, 4, 120)	0	batch_normalization_27[0][0]
conv2d_28 (Conv2D)	(None, 4, 4, 30)	32400	activation_27[0][0]
concatenate_24 (Concatenate)	(None, 4, 4, 150)	0	concatenate_23[0][0] conv2d_28[0][0]
batch_normalization_28 (BatchNo	(None, 4, 4, 150)	600	concatenate_24[0][0]
activation_28 (Activation)	(None, 4, 4, 150)	0	batch_normalization_28[0][0]
conv2d_29 (Conv2D)	(None, 4, 4, 30)	40500	activation_28[0][0]
concatenate_25 (Concatenate)	(None, 4, 4, 180)	0	concatenate_24[0][0] conv2d_29[0][0]

```
                                                                  conv2d_29[0][0]
__________________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 4, 4, 180)    720         concatenate_25[0][0]
__________________________________________________________________________________________________
activation_29 (Activation)      (None, 4, 4, 180)    0           batch_normalization_29[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 4, 4, 30)     48600       activation_29[0][0]
__________________________________________________________________________________________________
concatenate_26 (Concatenate)    (None, 4, 4, 210)    0           concatenate_25[0][0]
                                                                  conv2d_30[0][0]
__________________________________________________________________________________________________
batch_normalization_30 (BatchNo (None, 4, 4, 210)    840         concatenate_26[0][0]
__________________________________________________________________________________________________
activation_30 (Activation)      (None, 4, 4, 210)    0           batch_normalization_30[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 4, 4, 30)     56700       activation_30[0][0]
__________________________________________________________________________________________________
concatenate_27 (Concatenate)    (None, 4, 4, 240)    0           concatenate_26[0][0]
                                                                  conv2d_31[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 4, 4, 240)    960         concatenate_27[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 4, 4, 240)    0           batch_normalization_31[0][0]
__________________________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 2, 2, 240)    0           activation_31[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 960)          0           average_pooling2d_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 10)           9610        flatten[0][0]
==================================================================================================
Total params: 956,500
Trainable params: 947,860
Non-trainable params: 8,640
__________________________________________________________________________________________________
```

In [20]:

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

In [21]:

```
model.metrics_names
```

Out[21]:

```
[]
```

In [22]:

```
dg = ImageDataGenerator(height_shift_range=0.1, width_shift_range= 0.1, shear_range=0.2, zoom_range=0.2
, horizontal_flip=True)
```

In [23]:

```
dg.fit(X_train)
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
ve", force_remount=True).
```

In [ ]:

```
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
```

```
#filepath="/content/drive/MyDrive/NLP_data/weights-{epoch:02d}-{val_accuracy:.2f}.hdf5"
```

In [ ]:

```
#https://keras.io/api/callbacks/reduce_lr_on_plateau/
#checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, mode='max')
#reduceLR = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,verbose = 1)
#earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
```

In [ ]:

```
#callbacksList = [checkpoint, reduceLR, earlystop]
```

In [24]:

```
it = dg.flow(x=X_train, y=y_train, batch_size=batch_size)
step_sze = X_train.shape[0]//batch_size
model1 = model.fit_generator(it, steps_per_epoch=step_sze, epochs=100, verbose=1, validation_data=(X_te
st, y_test))
```

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

```
Epoch 1/100
390/390 [==============================] - 98s 158ms/step - loss: 1.4657 - accuracy: 0.4670 - val_loss:
1.6231 - val_accuracy: 0.4257
Epoch 2/100
390/390 [==============================] - 58s 149ms/step - loss: 1.0403 - accuracy: 0.6266 - val_loss:
0.8988 - val_accuracy: 0.6819
Epoch 3/100
390/390 [==============================] - 58s 149ms/step - loss: 0.8194 - accuracy: 0.7099 - val_loss:
1.1710 - val_accuracy: 0.6204
Epoch 4/100
390/390 [==============================] - 58s 149ms/step - loss: 0.7038 - accuracy: 0.7549 - val_loss:
1.0499 - val_accuracy: 0.6746
Epoch 5/100
390/390 [==============================] - 58s 149ms/step - loss: 0.6311 - accuracy: 0.7777 - val_loss:
0.7402 - val_accuracy: 0.7492
Epoch 6/100
390/390 [==============================] - 58s 149ms/step - loss: 0.5767 - accuracy: 0.7981 - val_loss:
0.7257 - val_accuracy: 0.7592
Epoch 7/100
390/390 [==============================] - 58s 149ms/step - loss: 0.5262 - accuracy: 0.8171 - val_loss:
1.0612 - val_accuracy: 0.6766
Epoch 8/100
390/390 [==============================] - 58s 149ms/step - loss: 0.4913 - accuracy: 0.8287 - val_loss:
0.5737 - val_accuracy: 0.8069
Epoch 9/100
390/390 [==============================] - 58s 149ms/step - loss: 0.4645 - accuracy: 0.8396 - val_loss:
0.7512 - val_accuracy: 0.7633
Epoch 10/100
390/390 [==============================] - 58s 149ms/step - loss: 0.4281 - accuracy: 0.8509 - val_loss:
0.7095 - val_accuracy: 0.7783
Epoch 11/100
390/390 [==============================] - 58s 149ms/step - loss: 0.4113 - accuracy: 0.8560 - val_loss:
0.5642 - val_accuracy: 0.8103
Epoch 12/100
390/390 [==============================] - 58s 150ms/step - loss: 0.3962 - accuracy: 0.8609 - val_loss:
1.0863 - val_accuracy: 0.7003
Epoch 13/100
390/390 [==============================] - 61s 155ms/step - loss: 0.3773 - accuracy: 0.8690 - val_loss:
0.5168 - val_accuracy: 0.8261
Epoch 14/100
390/390 [==============================] - 59s 150ms/step - loss: 0.3567 - accuracy: 0.8761 - val_loss:
0.6752 - val_accuracy: 0.7857
Epoch 15/100
390/390 [==============================] - 58s 149ms/step - loss: 0.3469 - accuracy: 0.8784 - val_loss:
0.5157 - val_accuracy: 0.8323
Epoch 16/100
390/390 [==============================] - 58s 149ms/step - loss: 0.3348 - accuracy: 0.8826 - val_loss:
0.5224 - val_accuracy: 0.8290
```

```
0.5224 - val_accuracy: 0.8290
Epoch 17/100
390/390 [==============================] - 58s 149ms/step - loss: 0.3198 - accuracy: 0.8893 - val_loss:
0.7572 - val_accuracy: 0.7817
Epoch 18/100
390/390 [==============================] - 58s 149ms/step - loss: 0.3029 - accuracy: 0.8941 - val_loss:
0.5433 - val_accuracy: 0.8325
Epoch 19/100
390/390 [==============================] - 60s 155ms/step - loss: 0.2945 - accuracy: 0.8959 - val_loss:
0.7383 - val_accuracy: 0.7795
Epoch 20/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2832 - accuracy: 0.9004 - val_loss:
0.5343 - val_accuracy: 0.8378
Epoch 21/100
390/390 [==============================] - 59s 150ms/step - loss: 0.2741 - accuracy: 0.9042 - val_loss:
0.6022 - val_accuracy: 0.8231
Epoch 22/100
390/390 [==============================] - 60s 154ms/step - loss: 0.2674 - accuracy: 0.9049 - val_loss:
0.5279 - val_accuracy: 0.8392
Epoch 23/100
390/390 [==============================] - 58s 150ms/step - loss: 0.2586 - accuracy: 0.9099 - val_loss:
0.4455 - val_accuracy: 0.8598
Epoch 24/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2457 - accuracy: 0.9142 - val_loss:
0.5913 - val_accuracy: 0.8251
Epoch 25/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2417 - accuracy: 0.9153 - val_loss:
0.4759 - val_accuracy: 0.8542
Epoch 26/100
390/390 [==============================] - 58s 150ms/step - loss: 0.2326 - accuracy: 0.9179 - val_loss:
0.5034 - val_accuracy: 0.8498
Epoch 27/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2256 - accuracy: 0.9198 - val_loss:
0.6847 - val_accuracy: 0.8106
Epoch 28/100
390/390 [==============================] - 58s 150ms/step - loss: 0.2180 - accuracy: 0.9238 - val_loss:
0.5278 - val_accuracy: 0.8452
Epoch 29/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2159 - accuracy: 0.9242 - val_loss:
0.5945 - val_accuracy: 0.8273
Epoch 30/100
390/390 [==============================] - 58s 149ms/step - loss: 0.2057 - accuracy: 0.9263 - val_loss:
0.4679 - val_accuracy: 0.8596
Epoch 31/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1963 - accuracy: 0.9306 - val_loss:
0.4512 - val_accuracy: 0.8623
Epoch 32/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1984 - accuracy: 0.9305 - val_loss:
0.4454 - val_accuracy: 0.8626
Epoch 33/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1901 - accuracy: 0.9338 - val_loss:
0.4757 - val_accuracy: 0.8603
Epoch 34/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1854 - accuracy: 0.9354 - val_loss:
0.4380 - val_accuracy: 0.8669
Epoch 35/100
390/390 [==============================] - 61s 156ms/step - loss: 0.1807 - accuracy: 0.9364 - val_loss:
0.4568 - val_accuracy: 0.8688
Epoch 36/100
390/390 [==============================] - 61s 155ms/step - loss: 0.1771 - accuracy: 0.9371 - val_loss:
0.4040 - val_accuracy: 0.8809
Epoch 37/100
390/390 [==============================] - 58s 150ms/step - loss: 0.1712 - accuracy: 0.9387 - val_loss:
0.4221 - val_accuracy: 0.8735
Epoch 38/100
390/390 [==============================] - 58s 150ms/step - loss: 0.1698 - accuracy: 0.9401 - val_loss:
0.5234 - val_accuracy: 0.8588
Epoch 39/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1634 - accuracy: 0.9426 - val_loss:
0.4830 - val_accuracy: 0.8645
Epoch 40/100
390/390 [==============================] - 60s 155ms/step - loss: 0.1596 - accuracy: 0.9421 - val_loss:
0.5885 - val_accuracy: 0.8420
Epoch 41/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1573 - accuracy: 0.9442 - val_loss:
0.3682 - val_accuracy: 0.8900
Epoch 42/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1530 - accuracy: 0.9458 - val_loss:
```

```
390/390 [==============================] - 58s 149ms/step - loss: 0.1530 - accuracy: 0.9458 - val_loss:
0.4089 - val_accuracy: 0.8829
Epoch 43/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1433 - accuracy: 0.9487 - val_loss:
0.4672 - val_accuracy: 0.8695
Epoch 44/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1462 - accuracy: 0.9480 - val_loss:
0.4427 - val_accuracy: 0.8785
Epoch 45/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1405 - accuracy: 0.9501 - val_loss:
0.4438 - val_accuracy: 0.8737
Epoch 46/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1364 - accuracy: 0.9513 - val_loss:
0.4384 - val_accuracy: 0.8778
Epoch 47/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1393 - accuracy: 0.9503 - val_loss:
0.4249 - val_accuracy: 0.8812
Epoch 48/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1282 - accuracy: 0.9538 - val_loss:
0.5045 - val_accuracy: 0.8674
Epoch 49/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1339 - accuracy: 0.9522 - val_loss:
0.4412 - val_accuracy: 0.8780
Epoch 50/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1257 - accuracy: 0.9559 - val_loss:
0.4522 - val_accuracy: 0.8761
Epoch 51/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1241 - accuracy: 0.9561 - val_loss:
0.4238 - val_accuracy: 0.8847
Epoch 52/100
390/390 [==============================] - 61s 155ms/step - loss: 0.1226 - accuracy: 0.9560 - val_loss:
0.4220 - val_accuracy: 0.8831
Epoch 53/100
390/390 [==============================] - 61s 155ms/step - loss: 0.1190 - accuracy: 0.9581 - val_loss:
0.4887 - val_accuracy: 0.8739
Epoch 54/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1188 - accuracy: 0.9573 - val_loss:
0.6411 - val_accuracy: 0.8419
Epoch 55/100
390/390 [==============================] - 61s 155ms/step - loss: 0.1134 - accuracy: 0.9586 - val_loss:
0.5002 - val_accuracy: 0.8767
Epoch 56/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1120 - accuracy: 0.9600 - val_loss:
0.4275 - val_accuracy: 0.8888
Epoch 57/100
390/390 [==============================] - 58s 150ms/step - loss: 0.1096 - accuracy: 0.9604 - val_loss:
0.5486 - val_accuracy: 0.8685
Epoch 58/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1059 - accuracy: 0.9623 - val_loss:
0.4240 - val_accuracy: 0.8862
Epoch 59/100
390/390 [==============================] - 59s 150ms/step - loss: 0.1025 - accuracy: 0.9634 - val_loss:
0.5625 - val_accuracy: 0.8632
Epoch 60/100
390/390 [==============================] - 58s 150ms/step - loss: 0.1054 - accuracy: 0.9620 - val_loss:
0.4277 - val_accuracy: 0.8897
Epoch 61/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1007 - accuracy: 0.9640 - val_loss:
0.4738 - val_accuracy: 0.8822
Epoch 62/100
390/390 [==============================] - 58s 149ms/step - loss: 0.0993 - accuracy: 0.9645 - val_loss:
0.4201 - val_accuracy: 0.8864
Epoch 63/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0990 - accuracy: 0.9643 - val_loss:
0.4261 - val_accuracy: 0.8950
Epoch 64/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0944 - accuracy: 0.9662 - val_loss:
0.5048 - val_accuracy: 0.8771
Epoch 65/100
390/390 [==============================] - 58s 149ms/step - loss: 0.1023 - accuracy: 0.9639 - val_loss:
0.5716 - val_accuracy: 0.8646
Epoch 66/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0932 - accuracy: 0.9673 - val_loss:
0.4430 - val_accuracy: 0.8891
Epoch 67/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0919 - accuracy: 0.9682 - val_loss:
0.4030 - val_accuracy: 0.8970
Epoch 68/100
```

```
Epoch 68/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0909 - accuracy: 0.9675 - val_loss:
0.5844 - val_accuracy: 0.8618
Epoch 69/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0919 - accuracy: 0.9672 - val_loss:
0.5367 - val_accuracy: 0.8691
Epoch 70/100
390/390 [==============================] - 58s 149ms/step - loss: 0.0861 - accuracy: 0.9692 - val_loss:
0.4539 - val_accuracy: 0.8962
Epoch 71/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0890 - accuracy: 0.9683 - val_loss:
0.4543 - val_accuracy: 0.8883
Epoch 72/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0837 - accuracy: 0.9702 - val_loss:
0.4327 - val_accuracy: 0.8895
Epoch 73/100
390/390 [==============================] - 59s 152ms/step - loss: 0.0848 - accuracy: 0.9690 - val_loss:
0.4939 - val_accuracy: 0.8782
Epoch 74/100
390/390 [==============================] - 61s 156ms/step - loss: 0.0852 - accuracy: 0.9704 - val_loss:
0.4616 - val_accuracy: 0.8849
Epoch 75/100
390/390 [==============================] - 59s 151ms/step - loss: 0.0822 - accuracy: 0.9704 - val_loss:
0.4621 - val_accuracy: 0.8891
Epoch 76/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0807 - accuracy: 0.9711 - val_loss:
0.3815 - val_accuracy: 0.9061
Epoch 77/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0814 - accuracy: 0.9713 - val_loss:
0.3839 - val_accuracy: 0.9033
Epoch 78/100
390/390 [==============================] - 59s 151ms/step - loss: 0.0766 - accuracy: 0.9728 - val_loss:
0.5298 - val_accuracy: 0.8775
Epoch 79/100
390/390 [==============================] - 61s 156ms/step - loss: 0.0794 - accuracy: 0.9723 - val_loss:
0.4992 - val_accuracy: 0.8851
Epoch 80/100
390/390 [==============================] - 59s 151ms/step - loss: 0.0772 - accuracy: 0.9736 - val_loss:
0.4498 - val_accuracy: 0.8928
Epoch 81/100
390/390 [==============================] - 59s 151ms/step - loss: 0.0754 - accuracy: 0.9731 - val_loss:
0.4563 - val_accuracy: 0.8924
Epoch 82/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0740 - accuracy: 0.9739 - val_loss:
0.4337 - val_accuracy: 0.8947
Epoch 83/100
390/390 [==============================] - 59s 151ms/step - loss: 0.0737 - accuracy: 0.9740 - val_loss:
0.5310 - val_accuracy: 0.8732
Epoch 84/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0764 - accuracy: 0.9739 - val_loss:
0.4260 - val_accuracy: 0.8907
Epoch 85/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0702 - accuracy: 0.9755 - val_loss:
0.5415 - val_accuracy: 0.8823
Epoch 86/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0664 - accuracy: 0.9762 - val_loss:
0.4852 - val_accuracy: 0.8831
Epoch 87/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0685 - accuracy: 0.9754 - val_loss:
0.5536 - val_accuracy: 0.8806
Epoch 88/100
390/390 [==============================] - 61s 156ms/step - loss: 0.0725 - accuracy: 0.9734 - val_loss:
0.4474 - val_accuracy: 0.8897
Epoch 89/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0687 - accuracy: 0.9760 - val_loss:
0.4571 - val_accuracy: 0.8927
Epoch 90/100
390/390 [==============================] - 58s 149ms/step - loss: 0.0662 - accuracy: 0.9760 - val_loss:
0.4709 - val_accuracy: 0.8911
Epoch 91/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0639 - accuracy: 0.9769 - val_loss:
0.5811 - val_accuracy: 0.8697
Epoch 92/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0652 - accuracy: 0.9766 - val_loss:
0.4900 - val_accuracy: 0.8906
Epoch 93/100
390/390 [==============================] - 61s 155ms/step - loss: 0.0640 - accuracy: 0.9774 - val_loss:
0.5053 - val_accuracy: 0.8879
```

```
0.5053 - val_accuracy: 0.8872
Epoch 94/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0702 - accuracy: 0.9750 - val_loss:
0.4913 - val_accuracy: 0.8891
Epoch 95/100
390/390 [==============================] - 58s 150ms/step - loss: 0.0685 - accuracy: 0.9756 - val_loss:
0.4079 - val_accuracy: 0.9026
Epoch 96/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0634 - accuracy: 0.9781 - val_loss:
0.4989 - val_accuracy: 0.8852
Epoch 97/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0599 - accuracy: 0.9790 - val_loss:
0.4577 - val_accuracy: 0.8919
Epoch 98/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0614 - accuracy: 0.9774 - val_loss:
0.4599 - val_accuracy: 0.8988
Epoch 99/100
390/390 [==============================] - 59s 150ms/step - loss: 0.0634 - accuracy: 0.9782 - val_loss:
0.4380 - val_accuracy: 0.8986
Epoch 100/100
390/390 [==============================] - 58s 149ms/step - loss: 0.0571 - accuracy: 0.9799 - val_loss:
0.4412 - val_accuracy: 0.9025
```

In [25]:

```
metrics = model.evaluate(x= X_test, y= y_test, verbose=1)
```

```
313/313 [==============================] - 3s 11ms/step - loss: 0.4412 - accuracy: 0.9025
```

In [26]:

```
print("test accuracy: ", metrics[1])
```

```
test accuracy:  0.9024999737739563
```