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.

```
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
from numpy import expand_dims
import numpy as np
import tensorflow as tf
import keras
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import regularizers
from matplotlib import pyplot

import warnings
warnings.filterwarnings("ignore")
```

```
# Hyperparameters
num_classes = 10
epochs = 10
compression = 0.5
```

```
(X_train_c, y_train_c), (X_test_c, y_test_c) = tf.keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 14s 0us/step
```

```
# 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 encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

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

```
print("X Train data shape",X_train.shape)
print("X Test data shape",y_train.shape)
print("y Train data shape",X_test.shape)
print("y Test data shape",y_test.shape)
```

```
X Train data shape (50000, 32, 32, 3)
X Test data shape (50000, 10)
y Train data shape (10000, 32, 32, 3)
y Test data shape (10000, 10)
```

```
y_train[:5]
```

```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

```
def denseblock(input, num_filter = 12, dropout_rate = 0.2):
    '''
    Create Dense Block
    '''
    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), 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


def transition(input, num_filter = 12, dropout_rate = 0.2):
    '''
    Create transition block
    '''
    global compression

    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)

    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (5,5), 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


def output_layer(input):
    '''
    Define output layer
    '''
    global compression

    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    AvgPooling = layers. MaxPooling2D(pool_size=(2,2))(relu)

    output = layers.Conv2D(filters=10,kernel_size=(2,2),activation='softmax')(AvgPooling)

    flat = layers.Flatten()(output)

    return flat
```

```
#https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
sample_image=X_train[1]
sample_image.shape
```

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

```
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)



model = Model(inputs=[input], outputs=[output])


model.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])


# Save the trained weights in to .h5 format
model.save_weights("DNST_model_with_dense_layer.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

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

```
50000
```

```
# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
# prepare iterator
iterator_train = datagen.flow(X_train, y_train, batch_size = 100)
# fit model
steps = int(X_train.shape[0] / 100)
#steps_per_epoch=steps,
history = model.fit_generator(iterator_train,epochs=50,validation_data = (X_test,y_test),verbose=1)
```

```
Epoch 20/50
500/500 [==============================] - 111s 222ms/step - loss: 0.2637 - accuracy: 0.9093 - val_loss: 0.6871 - val_accuracy:
Epoch 21/50
500/500 [==============================] - 111s 222ms/step - loss: 0.2479 - accuracy: 0.9144 - val_loss: 0.6671 - val_accuracy:
Epoch 22/50
500/500 [==============================] - 111s 222ms/step - loss: 0.2459 - accuracy: 0.9145 - val_loss: 0.4958 - val_accuracy:
Epoch 23/50
500/500 [==============================] - 111s 222ms/step - loss: 0.2387 - accuracy: 0.9178 - val_loss: 0.5553 - val_accuracy:
Epoch 24/50
500/500 [==============================] - 111s 223ms/step - loss: 0.2322 - accuracy: 0.9196 - val_loss: 0.4343 - val_accuracy:
Epoch 25/50
500/500 [==============================] - 112s 223ms/step - loss: 0.2312 - accuracy: 0.9203 - val_loss: 0.5163 - val_accuracy:
```

```
Epoch 45/50
500/500 [==============================] - 114s 229ms/step - loss: 0.1537 - accuracy: 0.9470 - val_loss: 0.5257 - val_accuracy:
Epoch 46/50
500/500 [==============================] - 111s 222ms/step - loss: 0.1551 - accuracy: 0.9462 - val_loss: 0.5208 - val_accuracy:
Epoch 47/50
500/500 [==============================] - 111s 221ms/step - loss: 0.1528 - accuracy: 0.9463 - val_loss: 0.5521 - val_accuracy:
```

```
# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
# prepare iterator
iterator_train = datagen.flow(X_train, y_train, batch_size = 10)
# fit model
steps = int(X_train.shape[0] / 10)
#steps_per_epoch=steps,
history = model.fit_generator(iterator_train,epochs= 100,validation_data = (X_test,y_test),verbose=1)
```

```
Epoch 1/100
5000/5000 [==============================] - 211s 40ms/step - loss: 1.8105 - accuracy: 0.3266 - val_loss: 1.5348 - val_accuracy:
Epoch 2/100
5000/5000 [==============================] - 196s 39ms/step - loss: 1.4140 - accuracy: 0.4821 - val_loss: 1.2505 - val_accuracy:
Epoch 3/100
5000/5000 [==============================] - 194s 39ms/step - loss: 1.1522 - accuracy: 0.5865 - val_loss: 1.1883 - val_accuracy:
Epoch 4/100
5000/5000 [==============================] - 197s 39ms/step - loss: 1.0003 - accuracy: 0.6451 - val_loss: 0.9088 - val_accuracy:
Epoch 5/100
5000/5000 [==============================] - 195s 39ms/step - loss: 0.9043 - accuracy: 0.6813 - val_loss: 0.8052 - val_accuracy:
Epoch 6/100
5000/5000 [==============================] - 195s 39ms/step - loss: 0.8272 - accuracy: 0.7085 - val_loss: 0.7996 - val_accuracy:
Epoch 7/100
5000/5000 [==============================] - 204s 41ms/step - loss: 0.7664 - accuracy: 0.7292 - val_loss: 0.7340 - val_accuracy:
Epoch 8/100
5000/5000 [==============================] - 207s 41ms/step - loss: 0.7164 - accuracy: 0.7494 - val_loss: 0.6790 - val_accuracy:
Epoch 9/100
5000/5000 [==============================] - 198s 40ms/step - loss: 0.6766 - accuracy: 0.7644 - val_loss: 0.6476 - val_accuracy:
Epoch 10/100
5000/5000 [==============================] - 196s 39ms/step - loss: 0.6404 - accuracy: 0.7792 - val_loss: 0.8122 - val_accuracy:
Epoch 11/100
5000/5000 [==============================] - 197s 39ms/step - loss: 0.6140 - accuracy: 0.7867 - val_loss: 0.6367 - val_accuracy:
Epoch 12/100
5000/5000 [==============================] - 197s 39ms/step - loss: 0.5865 - accuracy: 0.7952 - val_loss: 0.6111 - val_accuracy:
Epoch 13/100
5000/5000 [==============================] - 196s 39ms/step - loss: 0.5602 - accuracy: 0.8051 - val_loss: 0.7277 - val_accuracy:
Epoch 14/100
5000/5000 [==============================] - 200s 40ms/step - loss: 0.5375 - accuracy: 0.8138 - val_loss: 0.5556 - val_accuracy:
Epoch 15/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.5196 - accuracy: 0.8199 - val_loss: 0.6659 - val_accuracy:
Epoch 16/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.5029 - accuracy: 0.8256 - val_loss: 0.5858 - val_accuracy:
Epoch 17/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.4845 - accuracy: 0.8337 - val_loss: 0.5862 - val_accuracy:
Epoch 18/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.4720 - accuracy: 0.8369 - val_loss: 0.5617 - val_accuracy:
Epoch 19/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.4567 - accuracy: 0.8429 - val_loss: 0.5360 - val_accuracy:
Epoch 20/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.4425 - accuracy: 0.8448 - val_loss: 0.5274 - val_accuracy:
Epoch 21/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.4331 - accuracy: 0.8514 - val_loss: 0.5390 - val_accuracy:
Epoch 22/100
5000/5000 [==============================] - 192s 38ms/step - loss: 0.4211 - accuracy: 0.8553 - val_loss: 0.5870 - val_accuracy:
Epoch 23/100
5000/5000 [==============================] - 194s 39ms/step - loss: 0.4118 - accuracy: 0.8585 - val_loss: 0.5673 - val_accuracy:
Epoch 24/100
5000/5000 [==============================] - 192s 38ms/step - loss: 0.3978 - accuracy: 0.8610 - val_loss: 0.5079 - val_accuracy:
Epoch 25/100
5000/5000 [==============================] - 193s 38ms/step - loss: 0.3947 - accuracy: 0.8639 - val_loss: 0.4963 - val_accuracy:
Epoch 26/100
5000/5000 [==============================] - 197s 39ms/step - loss: 0.3833 - accuracy: 0.8681 - val_loss: 0.4941 - val_accuracy:
Epoch 27/100
5000/5000 [==============================] - 193s 39ms/step - loss: 0.3755 - accuracy: 0.8699 - val_loss: 0.4637 - val_accuracy:
Epoch 28/100
5000/5000 [==============================] - 196s 39ms/step - loss: 0.3660 - accuracy: 0.8722 - val_loss: 0.5115 - val_accuracy:
Epoch 29/100
```

```
# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
# prepare iterator
iterator_train = datagen.flow(X_train, y_train, batch_size = 25)
# fit model
steps = int(X_train.shape[0] / 25)
#steps_per_epoch=steps,
history = model.fit_generator(iterator_train,epochs= 50,validation_data = (X_test,y_test),verbose=1)
```

```
2000/2000 [==============================] - 139s 69ms/step - loss: 0.3820 - accuracy: 0.8673 - val_loss: 0.5525 - val_accuracy:
Epoch 25/50
2000/2000 [==============================] - 142s 71ms/step - loss: 0.3700 - accuracy: 0.8719 - val_loss: 0.6144 - val_accuracy:
Epoch 26/50
2000/2000 [==============================] - 139s 69ms/step - loss: 0.3624 - accuracy: 0.8727 - val_loss: 0.7257 - val_accuracy:
Epoch 27/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.3556 - accuracy: 0.8773 - val_loss: 0.5117 - val_accuracy:
Epoch 28/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.3442 - accuracy: 0.8806 - val_loss: 0.5483 - val_accuracy:
Epoch 29/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.3369 - accuracy: 0.8822 - val_loss: 0.6513 - val_accuracy:
Epoch 30/50
2000/2000 [==============================] - 141s 70ms/step - loss: 0.3314 - accuracy: 0.8849 - val_loss: 0.5871 - val_accuracy:
Epoch 31/50
2000/2000 [==============================] - 141s 70ms/step - loss: 0.3197 - accuracy: 0.8903 - val_loss: 0.4856 - val_accuracy:
Epoch 32/50
2000/2000 [==============================] - 141s 71ms/step - loss: 0.3145 - accuracy: 0.8921 - val_loss: 0.5289 - val_accuracy:
Epoch 33/50
2000/2000 [==============================] - 142s 71ms/step - loss: 0.3104 - accuracy: 0.8931 - val_loss: 0.6828 - val_accuracy:
Epoch 34/50
2000/2000 [==============================] - 142s 71ms/step - loss: 0.3005 - accuracy: 0.8963 - val_loss: 0.5385 - val_accuracy:
Epoch 35/50
2000/2000 [==============================] - 141s 70ms/step - loss: 0.2945 - accuracy: 0.8976 - val_loss: 0.5471 - val_accuracy:
Epoch 36/50
2000/2000 [==============================] - 141s 71ms/step - loss: 0.2903 - accuracy: 0.8986 - val_loss: 0.5769 - val_accuracy:
Epoch 37/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.2827 - accuracy: 0.9010 - val_loss: 0.4977 - val_accuracy:
Epoch 38/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.2789 - accuracy: 0.9028 - val_loss: 0.5050 - val_accuracy:
Epoch 39/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.2709 - accuracy: 0.9057 - val_loss: 0.4599 - val_accuracy:
Epoch 40/50
2000/2000 [==============================] - 139s 70ms/step - loss: 0.2691 - accuracy: 0.9060 - val_loss: 0.5113 - val_accuracy:
Epoch 41/50
2000/2000 [==============================] - 144s 72ms/step - loss: 0.2611 - accuracy: 0.9089 - val_loss: 0.5233 - val_accuracy:
Epoch 42/50
2000/2000 [==============================] - 143s 71ms/step - loss: 0.2581 - accuracy: 0.9114 - val_loss: 0.5044 - val_accuracy:
Epoch 43/50
2000/2000 [==============================] - 140s 70ms/step - loss: 0.2499 - accuracy: 0.9121 - val_loss: 0.4260 - val_accuracy:
Epoch 44/50
2000/2000 [==============================] - 142s 71ms/step - loss: 0.2491 - accuracy: 0.9138 - val_loss: 0.4523 - val_accuracy:
Epoch 45/50
2000/2000 [==============================] - 142s 71ms/step - loss: 0.2419 - accuracy: 0.9160 - val_loss: 0.4832 - val_accuracy:
Epoch 46/50
2000/2000 [==============================] - 141s 71ms/step - loss: 0.2425 - accuracy: 0.9153 - val_loss: 0.5380 - val_accuracy:
Epoch 47/50
2000/2000 [==============================] - 143s 71ms/step - loss: 0.2363 - accuracy: 0.9175 - val_loss: 0.5826 - val_accuracy:
Epoch 48/50
2000/2000 [==============================] - 144s 72ms/step - loss: 0.2337 - accuracy: 0.9183 - val_loss: 0.5212 - val_accuracy:
Epoch 49/50
2000/2000 [==============================] - 144s 72ms/step - loss: 0.2232 - accuracy: 0.9213 - val_loss: 0.5096 - val_accuracy:
Epoch 50/50
```

```
model.summary()
```

```
 ormalization)

 activation_50 (Activation)     (None, 4, 4, 60)     0           ['batch_normalization_51[0][0]']

 conv2d_51 (Conv2D)             (None, 4, 4, 5)      7500        ['activation_50[0][0]']

 concatenate_47 (Concatenate)   (None, 4, 4, 65)     0           ['concatenate_46[0][0]',
                                                                  'conv2d_51[0][0]']

 batch_normalization_52 (BatchN  (None, 4, 4, 65)    260         ['concatenate_47[0][0]']
 ormalization)

 activation_51 (Activation)     (None, 4, 4, 65)     0           ['batch_normalization_52[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 2, 2, 65)     0           ['activation_51[0][0]']

 conv2d_52 (Conv2D)             (None, 1, 1, 10)     2610        ['max_pooling2d[0][0]']

 flatten (Flatten)              (None, 10)           0           ['conv2d_52[0][0]']

==================================================================================================
Total params: 746,808
Trainable params: 740,834
Non-trainable params: 5,974
__________________________________________________________________________________________________
```

```
!rm -rf ./model1_logs/
```

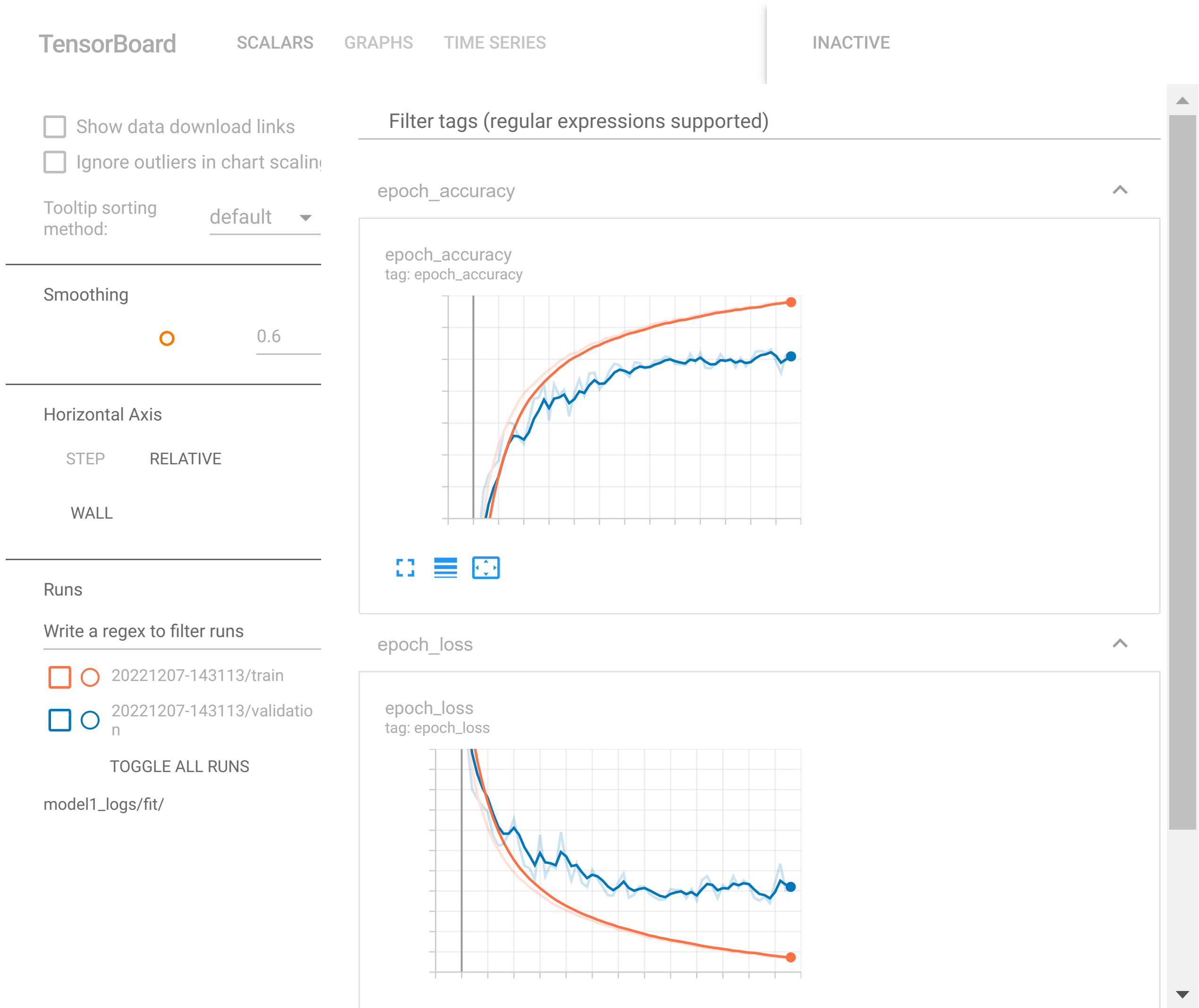
```
from tensorflow.keras.callbacks import EarlyStopping
import datetime

log_dir="model1_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
model_save_path='/tf_model.h5'
Early_stop=EarlyStopping(monitor='val_accuracy',min_delta=0.0001,patience=3,verbose=1)

checkpoint= tf.keras.callbacks.ModelCheckpoint(filepath=model_save_path,save_weights_only=True,monitor='val_accuracy',mode='auto',save_be
ten_brd = keras.callbacks.TensorBoard(log_dir=log_dir)
callbacks = [checkpoint,ten_brd]
```

```
# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
# prepare iterator
iterator_train = datagen.flow(X_train, y_train, batch_size = 32)
#steps_per_epoch=steps,
history = model.fit_generator(iterator_train,epochs= 64,validation_data = (X_test,y_test),verbose=1,callbacks=callbacks)
```

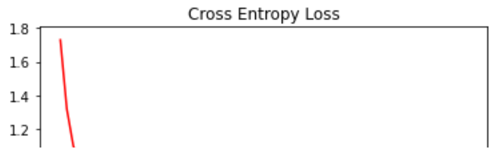
```
Epoch 1/64
1563/1563 [==============================] - 142s 81ms/step - loss: 1.7281 - accuracy: 0.3546 - val_loss: 1.5348 - val_accuracy:
Epoch 2/64
1563/1563 [==============================] - 123s 79ms/step - loss: 1.3200 - accuracy: 0.5191 - val_loss: 1.2627 - val_accuracy:
Epoch 3/64
1563/1563 [==============================] - 123s 79ms/step - loss: 1.1032 - accuracy: 0.6050 - val_loss: 1.0013 - val_accuracy:
Epoch 4/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.9722 - accuracy: 0.6566 - val_loss: 0.9558 - val_accuracy:
Epoch 5/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.8871 - accuracy: 0.6878 - val_loss: 0.9194 - val_accuracy:
Epoch 6/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.8099 - accuracy: 0.7160 - val_loss: 0.8891 - val_accuracy:
Epoch 7/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.7504 - accuracy: 0.7365 - val_loss: 0.7744 - val_accuracy:
Epoch 8/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.7035 - accuracy: 0.7541 - val_loss: 0.7232 - val_accuracy:
Epoch 9/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.6610 - accuracy: 0.7699 - val_loss: 0.7310 - val_accuracy:
Epoch 10/64
1563/1563 [==============================] - 128s 82ms/step - loss: 0.6241 - accuracy: 0.7850 - val_loss: 0.7837 - val_accuracy:
Epoch 11/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.5920 - accuracy: 0.7960 - val_loss: 0.8550 - val_accuracy:
Epoch 12/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.5633 - accuracy: 0.8050 - val_loss: 0.7250 - val_accuracy:
Epoch 13/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.5442 - accuracy: 0.8113 - val_loss: 0.6249 - val_accuracy:
Epoch 14/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.5182 - accuracy: 0.8198 - val_loss: 0.6119 - val_accuracy:
Epoch 15/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.5021 - accuracy: 0.8270 - val_loss: 0.5589 - val_accuracy:
Epoch 16/64
1563/1563 [==============================] - 123s 78ms/step - loss: 0.4809 - accuracy: 0.8336 - val_loss: 0.7761 - val_accuracy:
Epoch 17/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.4636 - accuracy: 0.8393 - val_loss: 0.5731 - val_accuracy:
Epoch 18/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.4449 - accuracy: 0.8470 - val_loss: 0.6304 - val_accuracy:
Epoch 19/64
1563/1563 [==============================] - 123s 78ms/step - loss: 0.4299 - accuracy: 0.8509 - val_loss: 0.6125 - val_accuracy:
Epoch 20/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.4183 - accuracy: 0.8576 - val_loss: 0.7900 - val_accuracy:
Epoch 21/64
```

```
1563/1563 [==============================] - 126s 81ms/step - loss: 0.4054 - accuracy: 0.8598 - val_loss: 0.6388 - val_accuracy:
Epoch 22/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.3949 - accuracy: 0.8625 - val_loss: 0.5519 - val_accuracy:
Epoch 23/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.3823 - accuracy: 0.8678 - val_loss: 0.6350 - val_accuracy:
Epoch 24/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.3714 - accuracy: 0.8726 - val_loss: 0.5408 - val_accuracy:
Epoch 25/64
1563/1563 [==============================] - 126s 81ms/step - loss: 0.3625 - accuracy: 0.8758 - val_loss: 0.5198 - val_accuracy:
Epoch 26/64
1563/1563 [==============================] - 123s 79ms/step - loss: 0.3557 - accuracy: 0.8759 - val_loss: 0.6050 - val_accuracy:
Epoch 27/64
1563/1563 [==============================] - 123s 78ms/step - loss: 0.3416 - accuracy: 0.8826 - val_loss: 0.5608 - val_accuracy:
Epoch 28/64
```

```
%load_ext tensorboard
%tensorboard --logdir model1_logs/fit/
```




```
def model_plots(history):
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='red', label='Loss')
  pyplot.plot(history.history['accuracy'], color='blue', label='Accuracy')
  pyplot.xlabel('Epochs')
  pyplot.ylabel('Loss')
  pyplot.show()


model_plots(history)
```




```
# Train the model
score = model.evaluate(X_train, y_train, verbose=1)
print('Train loss:', score[0])
print('Train accuracy:', score[1])
```

```
1563/1563 [==============================] - 37s 24ms/step - loss: 0.1620 - accuracy: 0.9427
Train loss: 0.16198216378688812
Train accuracy: 0.9426800012588501
```

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

```
313/313 [==============================] - 7s 24ms/step - loss: 0.5034 - accuracy: 0.8617
Test loss: 0.5034124255180359
Test accuracy: 0.8616999983787537
```

```
y_pred = model.predict(X_test)
```

```
313/313 [==============================] - 7s 21ms/step
```

```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

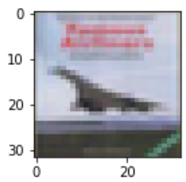
```
[3, 8, 8, 8, 6]
```

```
classes = list(set(y_classes))
classes
```

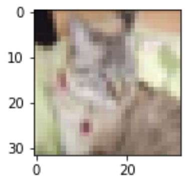
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

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

```
plot_sample(X_test, y_test,3)
```



```
plot_sample(X_test, y_test,8)
```



0s completed at 10:23 PM