ASSIGNMENT-3

```
import numpy as np  import pandas as pd


from PIL import ImageFile

from tqdm import tqdm

import h5py import cv2


import matplotlib.pyplot as plt

%matplotlib inline


import seaborn as sns


from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix from

sklearn.metrics import plot_confusion_matrix


from tensorflow.keras.utils import to_categorical from

tensorflow.keras.preprocessing import image as keras_image from

tensorflow.keras.models import Sequential, load_model from

tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.layers import Activation, Dropout from
tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D from
tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint from
tensorflow.keras.layers import LeakyReLU def model():

    model = Sequential()

    model.add(Conv2D(128, (3, 3), input_shape=x_train.shape[1:]))
model.add(LeakyReLU(alpha=0.02))

    model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

    model.add(Conv2D(128, (3, 3)))
model.add(LeakyReLU(alpha=0.02))

    model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

    model.add(GlobalMaxPooling2D())
```

```
    model.add(Dense(512))
model.add(LeakyReLU(alpha=0.02))
model.add(Dropout(0.5))      model.add(Dense(10))
model.add(Activation('softmax'))


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


    return model


model = model()


# To save the best model checkpointer = ModelCheckpoint(filepath='weights.best.model.hdf5',
verbose=2, save_best_only=True)


# To reduce learning rate dynamically lr_reduction =
ReduceLROnPlateau(monitor='val_loss', patience=5, verbose=2, factor=0.2)

# Train the model history = model.fit(x_train, y_train, epochs=75,
batch_size=32, verbose=2,

                validation_data=(x_valid, y_valid),
callbacks=[checkpointer,

data_generator = keras_image.ImageDataGenerator(shear_range=0.3,
```

```
                                zoom_range=0.3,
rotation_range=30,
horizontal_flip=True)


dg_history = model.fit_generator(data_generator.flow(x_train, y_train, batch_size=64),
steps_per_epoch = len(x_train)//64, epochs=7, verbose=2, validation_data=(x_valid,
y_valid), callbacks=[checkpointer,lr_reduction])
```