**Practical Assignment**

**Objective: - Image Classification with Indoor Scenes Images**

From MIT, this dataset contains over 15,000 images of indoor locations. The dataset was originally built to tackle the problem of indoor scene recognition. All images are in JPEG format and have been divided into 67 categories. The number of images per category vary. However, there are at least 100 images for each category.

**Dataset Link: -**

[**https://www.kaggle.com/itsahmad/indoor-scenes-cvpr-2019**](https://www.kaggle.com/itsahmad/indoor-scenes-cvpr-2019)

**Task: -** Create a Web Application using Flask. Use the end user should be able to upload an image and get results with the prediction score. Use any CNN architecture launched after 2014.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link.

**Importing required libraries**

*# Importing required libraries*

import os

import pathlib

import math

import matplotlib.pyplot as plt

import cv2

from google.colab.patches import cv2\_imshow

import numpy as np

np.random.seed(42)

import random as rn

rn.seed(42)

from keras import backend as K

import tensorflow as tf

tf.random.set\_seed(42)

import keras

from keras.models import Sequential, Model

from keras.layers import Input, Lambda, Dense, Flatten, GlobalMaxPool3D, BatchNormalization, Dropout, Activation

from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from keras import optimizers

from keras.regularizers import l2

from keras.models import load\_model

from keras.applications.inception\_v3 import InceptionV3

from keras.applications.xception import Xception

from keras.applications.inception\_resnet\_v2 import InceptionResNetV2

*# Checking available Nvidia GPU*

!nvidia-smi

**Preparing data**

*# Unzip files*

from zipfile import ZipFile

with ZipFile('/content/drive/MyDrive/colab\_data/indoorCVPR/data\_10\_cats.zip', 'r') as zipobj:

zipobj.extractall('/content/drive/MyDrive/colab\_data/indoorCVPR')

print('Files are unzipped')

### Spliting data for training and validation

*# Training dataset*

data\_path= '/content/drive/MyDrive/colab\_data/indoorCVPR'

train\_data\_dir= pathlib.Path(data\_path)

image\_count= len(list(train\_data\_dir.glob('\*/\*.jpg')))

print(image\_count)

*# Setting image height, width and batch size*

img\_height= 150

img\_width= 150

batch\_size= 64

*# Training dataset*

train\_ds= tf.keras.preprocessing.image\_dataset\_from\_directory(train\_data\_dir, seed= 42, validation\_split= 0.2,

subset= 'training',

batch\_size= batch\_size,

image\_size= (img\_height, img\_width)

)

*# Validation dataset*

val\_ds= tf.keras.preprocessing.image\_dataset\_from\_directory(train\_data\_dir, seed= 42, validation\_split= 0.2,

subset= 'validation',

batch\_size= batch\_size,

image\_size= (img\_height, img\_width)

)

### Vizualizing training data

*# Checking all class names*

class\_names = train\_ds.class\_names

print(class\_names)

*# Checking training data*

plt.figure(figsize=(9, 9))

for images, labels in train\_ds.take(1):

for i in range(16):

ax = plt.subplot(4, 4, i + 1)

plt.imshow(images[i].numpy().astype('uint8'))

plt.title(class\_names[labels[i]])

plt.axis('off')

### Data Augmentation

*# ## Creating data augmentation layer*

augmentation\_layer= Sequential(

[tf.keras.layers.InputLayer(input\_shape= (img\_height, img\_width, 3)),

tf.keras.layers.RandomFlip('horizontal', seed= 42),

tf.keras.layers.RandomRotation(0.2, fill\_mode= 'wrap', seed= 42),

tf.keras.layers.RandomZoom(0.2, seed= 42),

tf.keras.layers.RandomTranslation(.2, .2, fill\_mode='wrap', interpolation='bilinear', seed= 42),

tf.keras.layers.RandomContrast(0.2, seed= 42)]

)

*# Plotting different augmented version of a random image from training dataset*

plt.figure(figsize=(9, 9))

a= np.random.randint(42)

for images, labels in train\_ds.take(1):

for i in range(12):

aug\_img= augmentation\_layer(images)

ax = plt.subplot(3, 4, i + 1)

plt.imshow(aug\_img[a].numpy().astype(np.int32))

plt.axis('off')

## Model building

### InceptionV3 Finetuning

*# Creating function to fine tune InceptionV3*

def inceptionv3(inp\_shape, dropout\_rate, train\_layers\_after):

inceptionv3= InceptionV3(weights= 'imagenet', include\_top= False)

for layer in inceptionv3.layers[:train\_layers\_after]:

layer.trainable= False

input\_layer= Input(shape= inp\_shape)

data\_aug\_layer= augmentation\_layer(input\_layer)

norm\_layer= tf.keras.layers.Rescaling(1./255)(data\_aug\_layer)

cnn\_layers= inceptionv3(norm\_layer)

flatten\_layer= Flatten()(cnn\_layers)

dropout\_layer1= Dropout(dropout\_rate)(flatten\_layer)

dense\_layer= Dense(1024, activation= 'relu', kernel\_initializer= 'he\_normal', kernel\_regularizer='l2')(dropout\_layer1)

dropout\_layer2= Dropout(dropout\_rate)(dense\_layer)

dense\_layer\_1= Dense(512, activation= 'relu', kernel\_initializer= 'he\_normal', kernel\_regularizer='l2')(dropout\_layer2)

dropout\_layer3= Dropout(dropout\_rate)(dense\_layer\_1)

output\_layer= Dense(10, activation= 'softmax')(dropout\_layer3)

model= Model(input\_layer, output\_layer)

model.compile(optimizer= 'adam', loss= 'sparse\_categorical\_crossentropy', metrics= ['sparse\_categorical\_accuracy'])

return model

*# Creating model*

inp\_shape= (150, 150, 3)

dropout\_rate= .25

train\_layers\_after= 249

incv3\_model= inceptionv3(inp\_shape, dropout\_rate, train\_layers\_after)

incv3\_model.summary()

*# Loading data into cache to overcome data bottleneck during training.*

AUTOTUNE= tf.data.AUTOTUNE

AUTOTUNE= tf.data.experimental.AUTOTUNE

*# Shuffling data before starting of each epoch*

train\_ds= train\_ds.cache().shuffle(1000).prefetch(buffer\_size= AUTOTUNE)

val\_ds= val\_ds.cache().prefetch(buffer\_size= AUTOTUNE)

*# Setting callbacks*

base\_path= '/content/drive/MyDrive/colab\_data/indoorCVPR/models/'

filepath= base\_path + 'model-{epoch:05d}-{loss:.5f}-{sparse\_categorical\_accuracy:.5f}-{val\_loss:.5f}-{val\_sparse\_categorical\_accuracy:.5f}.h5'

checkpoint= ModelCheckpoint(filepath, monitor= 'val\_sparse\_categorical\_accuracy', verbose= 1,

save\_best\_only= True, save\_weights\_only= False, mode= 'auto')

LR= ReduceLROnPlateau(monitor= 'val\_loss', factor= 0.1, patience= 30, verbose= 1)

callbacks\_list= [checkpoint, LR]

*# Model training*

epochs= 200

steps\_per\_epoch= math.ceil(2652/batch\_size)

validation\_steps= math.ceil(663/batch\_size)

history= incv3\_model.fit(train\_ds, validation\_data= val\_ds, steps\_per\_epoch= steps\_per\_epoch, epochs= epochs, callbacks= callbacks\_list, validation\_steps= validation\_steps)

acc= history.history['sparse\_categorical\_accuracy']

val\_acc= history.history['val\_sparse\_categorical\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.show()

### Xception Finetunning

*# Creating function to fine tune Xception*

def xception\_cnn(inp\_shape, dropout\_rate, train\_layers\_after):

xcp= Xception(weights= 'imagenet', include\_top= False)

for layer in xcp.layers[:train\_layers\_after]:

layer.trainable= False

input\_layer= Input(shape= inp\_shape)

data\_aug\_layer= augmentation\_layer(input\_layer)

norm\_layer= tf.keras.layers.Rescaling(1./255)(data\_aug\_layer)

cnn\_layers= xcp(norm\_layer)

flatten\_layer= Flatten()(cnn\_layers)

dropout\_layer1= Dropout(dropout\_rate)(flatten\_layer)

dense\_layer= Dense(1024, activation= 'relu', kernel\_initializer= 'he\_normal', kernel\_regularizer='l2')(dropout\_layer1)

dropout\_layer2= Dropout(dropout\_rate)(dense\_layer)

dense\_layer\_1= Dense(512, activation= 'relu', kernel\_initializer= 'he\_normal', kernel\_regularizer='l2')(dropout\_layer2)

dropout\_layer3= Dropout(dropout\_rate)(dense\_layer\_1)

output\_layer= Dense(10, activation= 'softmax')(dropout\_layer3)

model= Model(input\_layer, output\_layer)

model.compile(optimizer= 'adam', loss= 'sparse\_categorical\_crossentropy', metrics= ['sparse\_categorical\_accuracy'])

return model

*# Creating model*

inp\_shape= (150, 150, 3)

dropout\_rate= .25

train\_layers\_after= 114

xcp\_model= xception\_cnn(inp\_shape, dropout\_rate, train\_layers\_after)

xcp\_model.summary()

*# Model training*

epochs= 200

steps\_per\_epoch= math.ceil(2652/batch\_size)

validation\_steps= math.ceil(663/batch\_size)

history= xcp\_model.fit(train\_ds, validation\_data= val\_ds, steps\_per\_epoch= steps\_per\_epoch, epochs= epochs, callbacks= callbacks\_list, validation\_steps= validation\_steps)

## Inference Script

*# Combining both the models for inference*

final\_model1= tf.keras.models.load\_model('/content/drive/MyDrive/colab\_data/models/model-00088-0.07750-0.99321-0.49403-0.89140.h5')

final\_model2= tf.keras.models.load\_model('/content/drive/MyDrive/colab\_data/models/model-00109-0.08159-0.99170-0.67942-0.85068.h5')

models= [final\_model1, final\_model2]

*# Single inference function*

def single\_inference(path, models):

classes= ['airport\_inside', 'auditorium', 'bakery', 'bathroom', 'bookstore', 'casino', 'church\_inside', 'grocerystore', 'operating\_room', 'warehouse']

img= cv2.imread(path, cv2.IMREAD\_UNCHANGED)

cv2\_imshow(img)

img= cv2.resize(img, (150, 150), interpolation = cv2.INTER\_AREA)

img= img.reshape((1,150,150,3))

pred1= models[0].predict(img)

pred2= models[1].predict(img)

pred= (pred1+pred2)/2

print ('Predicted class:', classes[np.argmax(pred)])

*# Predicting a random image from internet*

single\_inference('/content/test1.jpg', models)

*# Predicting a random image from internet*

single\_inference('/content/test2.jpg', models)

*# Predicting a random image from internet*

single\_inference('/content/test3.jpg', models)

*# Predicting a random image from internet*

single\_inference('/content/test4.jpg', models)