PR Group 4 Skin Disease Detection

```
import random
import os
import glob
import time

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras import layers, Sequential, regularizers
from tensorflow.keras.utils import plot_model

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50V2

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, precision_recall_fscore_support
from sklearn.metrics import accuracy_score, f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#from scikitplot.metrics import plot_roc



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

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

```
class CFG:
  EPOCHS=25
  BATCH_SIZE=32
  SEED=963
  TF_SEED=768
  HEIGHT=224
  WIDTH=224
  CHANNELS=3
  IMAGE_SIZE=(224,224,3)



DATASET_PATH='/content/drive/MyDrive/skin_dataset_resized'
TRAIN_PATH='/content/drive/MyDrive/skin_dataset_resized/train_set'
TEST_PATH='/content/drive/MyDrive/skin_dataset_resized/val_set'
VAL_PATH='/content/drive/MyDrive/skin_dataset_resized/test_set'



print('DATASET SUMMARY')
print('================\n')
for dirpath,dirnames,filenames in os.walk(DATASET_PATH):
  print("There are ",len(dirnames)," directories", "and ",len(filenames), "images in ",dirpath)
print('\n=============')
```

```
DATASET SUMMARY
================

There are  3  directories and  0 images in  /content/drive/MyDrive/skin_dataset_resized
There are  2  directories and  0 images in  /content/drive/MyDrive/skin_dataset_resized/val_set
There are  0  directories and  550 images in  /content/drive/MyDrive/skin_dataset_resized/val_set/benign
There are  0  directories and  550 images in  /content/drive/MyDrive/skin_dataset_resized/val_set/malignant
There are  2  directories and  0 images in  /content/drive/MyDrive/skin_dataset_resized/test_set
There are  0  directories and  550 images in  /content/drive/MyDrive/skin_dataset_resized/test_set/malignant
There are  0  directories and  550 images in  /content/drive/MyDrive/skin_dataset_resized/test_set/benign
There are  2  directories and  0 images in  /content/drive/MyDrive/skin_dataset_resized/train_set
There are  0  directories and  4001 images in  /content/drive/MyDrive/skin_dataset_resized/train_set/malignant
There are  0  directories and  5200 images in  /content/drive/MyDrive/skin_dataset_resized/train_set/benign

=============
```

```
%%time
train_images=glob.glob(TRAIN_PATH+'/**/*.jpg',recursive=True)
test_images =glob.glob(TEST_PATH+'/**/*.jpg',recursive=True)
val_images=glob.glob(VAL_PATH+'/**/*.jpg',recursive=True)
```

```
CPU times: user 49.4 ms, sys: 2.67 ms, total: 52.1 ms
Wall time: 366 ms
```

```
train_size=len(train_images)
test_size=len(test_images)

total=train_size + test_size

print(f"train samples count:\t\t {train_size}")
print('test samples count:\t\t',test_size)
print('==============================')
print('Total : \t\t\t',total)
```

```
train samples count:             9201
test samples count:              1100
==============================
Total :                          10301
```

create Panda Dataframe for paths and labels

```
def generate_labels(image_paths):
  return [_.split('/')[-2:][0] for _ in image_paths]

def build_df(image_paths,labels):
  df=pd.DataFrame({'image_path': image_paths, 'label':generate_labels(labels)})

  df['label_encoded'] = df.apply(lambda row:0 if row.label =='malignant' else 1, axis=1)

  return df.sample(frac=1, random_state=CFG.SEED).reset_index(drop=True)



train_df=build_df(train_images,generate_labels(train_images))
test_df=build_df(test_images,generate_labels(test_images))
val_df=build_df(val_images,generate_labels(val_images))
```

```
train_df.head()
```

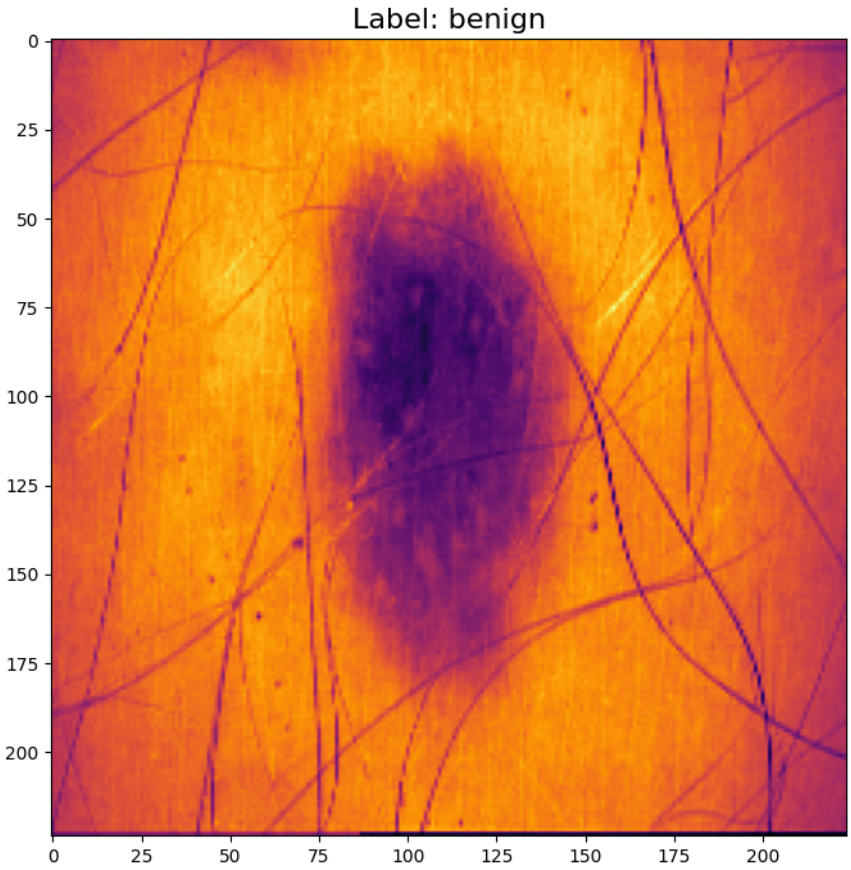
	image_path	label	label_encoded
0	/content/drive/MyDrive/skin_dataset_resized/tr...	malignant	0
1	/content/drive/MyDrive/skin_dataset_resized/tr...	benign	1
2	/content/drive/MyDrive/skin_dataset_resized/tr...	malignant	0
3	/content/drive/MyDrive/skin_dataset_resized/tr...	benign	1
4	/content/drive/MyDrive/skin_dataset_resized/tr...	malignant	0

```
test_df.head()
```

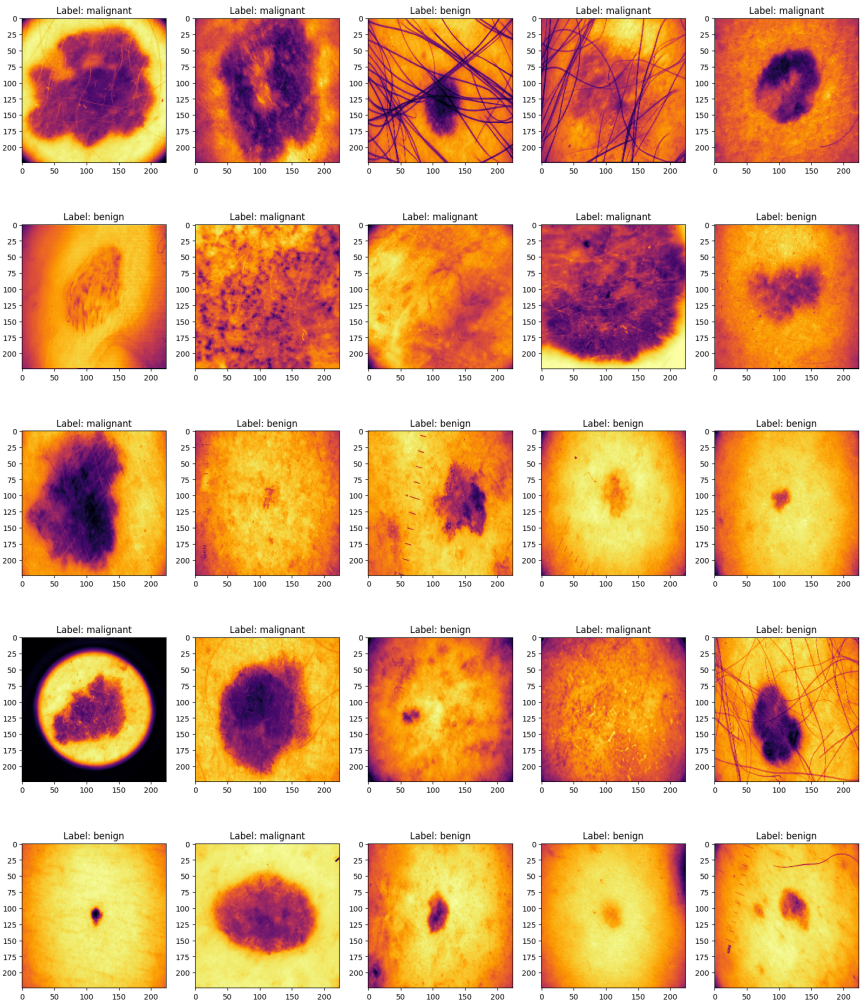
	image_path	label	label_encoded
0	/content/drive/MyDrive/skin_dataset_resized/va...	malignant	0
1	/content/drive/MyDrive/skin_dataset_resized/va...	malignant	0
2	/content/drive/MyDrive/skin_dataset_resized/va...	malignant	0
3	/content/drive/MyDrive/skin_dataset_resized/va...	benign	1
4	/content/drive/MyDrive/skin_dataset_resized/va...	benign	1

```
def _load(image_path):
  #reading and decoding image file to unit8 tensor
  image=tf.io.read_file(image_path)
  image=tf.io.decode_jpeg(image,channels=3)

  #resizing images
  image=tf.image.resize(image,[CFG.HEIGHT,CFG.WIDTH],method=tf.image.ResizeMethod.LANCZOS3)

  #covert image dtype to float32 and normalize
  image=tf.cast(image,tf.float32)/255
  return image


def view_sample(image,label, color_map='rgb', fig_size=(8,10)):
  plt.figure(figsize=fig_size)
```

```
  if color_map=='rgb':
    plt.imshow(image)
  else:
    plt.imshow(tf.image.rgb_to_grayscale(image),cmap=color_map)

  plt.title(f'Label: {label}', fontsize=16)
  return


#select random sample from train_df
idx=random.sample(train_df.index.to_list(),1)[0]

sample_image,sample_label=_load(train_df.image_path[idx]),train_df.label[idx]
#view the random sample
view_sample(sample_image,sample_label,color_map='inferno')
```



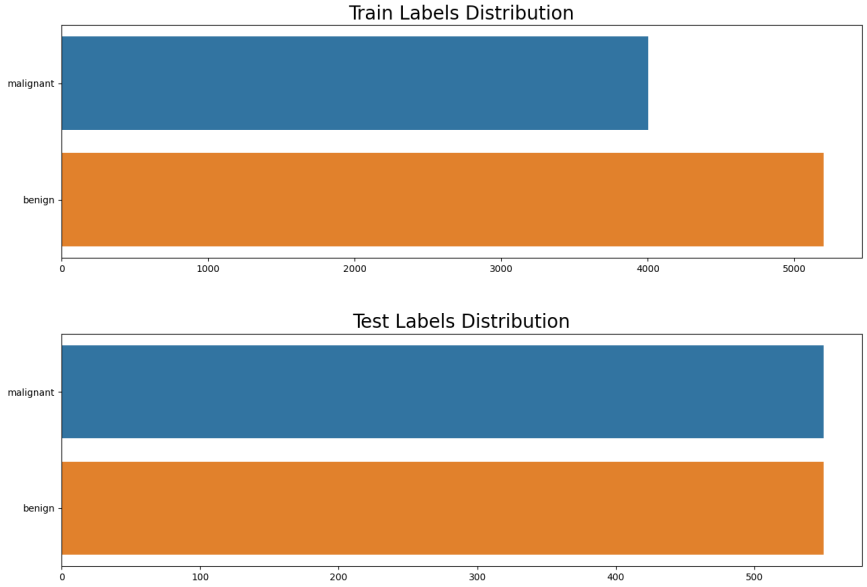

```
def view_multiple_samples(df,sample_loader,count=10,color_map='rgb', fig_size=(14,10)):
  rows=count//5
  if count%5>0:
    rows+=1

  idx=random.sample(df.index.to_list(),count)
  fig=plt.figure(figsize=fig_size)

  for column, _ in enumerate(idx):
    plt.subplot(rows,5,column+1)
    plt.title(f'Label: {df.label[_]}')
    if color_map=='rgb':
      plt.imshow(sample_loader(df.image_path[_]))
    else:
      plt.imshow(tf.image.rgb_to_grayscale(sample_loader(df.image_path[_])),cmap=color_map)

  return

view_multiple_samples(train_df, _load, count=25, color_map='inferno',fig_size=(20,24))
```


Label: malignant
Label: malignant
Label: benign
Label: malignant
Label: malignant
Label: benign
Label: malignant
Label: malignant
Label: malignant
Label: benign
Label: malignant
Label: benign
Label: benign
Label: benign
Label: benign
Label: malignant
Label: malignant
Label: benign
Label: malignant
Label: benign
Label: benign
Label: malignant
Label: benign
Label: benign
Label: benign

View Train Labels Distribution

```
fig, (ax1,ax2) = plt.subplots(2,figsize=(14,10))

fig.tight_layout(pad=6.0)

ax1.set_title('Train Labels Distribution', fontsize=20)
train_distribution = train_df['label'].value_counts().sort_values()
sns.barplot(x=train_distribution.values,
            y=list(train_distribution.keys()),
            orient='h',
            ax=ax1)

ax2.set_title('Test Labels Distribution', fontsize=20)
test_distribution = test_df['label'].value_counts().sort_values()
sns.barplot(x=test_distribution.values,
            y=list(test_distribution.keys()),
            orient='h',
            ax=ax2)
```

```
<Axes: title={'center': 'Test Labels Distribution'}>
```



Balancing Benign and Malign Data in Train Set

```
benign_idx=train_df[train_df['label']=='benign'].index
malignant_idx=train_df[train_df['label']=='malignant'].index
new_train_df=train_df[train_df['label']=='benign'].sample(n=len(malignant_idx))
malignant_train_df=train_df[train_df['label']=='malignant']

new_train_df=pd.concat([new_train_df,malignant_train_df])
new_train_df['label'].value_counts()
```

```
benign       4001
malignant    4001
Name: label, dtype: int64
```

```
fig, (ax1,ax2,ax3) = plt.subplots(3,figsize=(14,10))

fig.tight_layout(pad=6.0)

ax1.set_title('Revised Train Labels Distribution', fontsize=20)
train_distribution = new_train_df['label'].value_counts().sort_values()
sns.barplot(x=train_distribution.values,
            y=list(train_distribution.keys()),
            orient='h',
            ax=ax1)
ax2.set_title('Validation Labels Distribution', fontsize=20)
val_distribution = val_df['label'].value_counts().sort_values()
sns.barplot(x=val_distribution.values,
            y=list(val_distribution.keys()),
            orient='h',
            ax=ax2)

ax3.set_title('Test Labels Distribution', fontsize=20)
test_distribution = test_df['label'].value_counts().sort_values()
sns.barplot(x=test_distribution.values,
            y=list(test_distribution.keys()),
            orient='h',
            ax=ax3)
```

```
<Axes: title={'center': 'Test Labels Distribution'}>
```


```
train_df=new_train_df.sample(frac=1)
```

Implementing Preprocessing. Building an input data Pipeline

```
train_df.shape,test_df.shape,val_df.shape
```

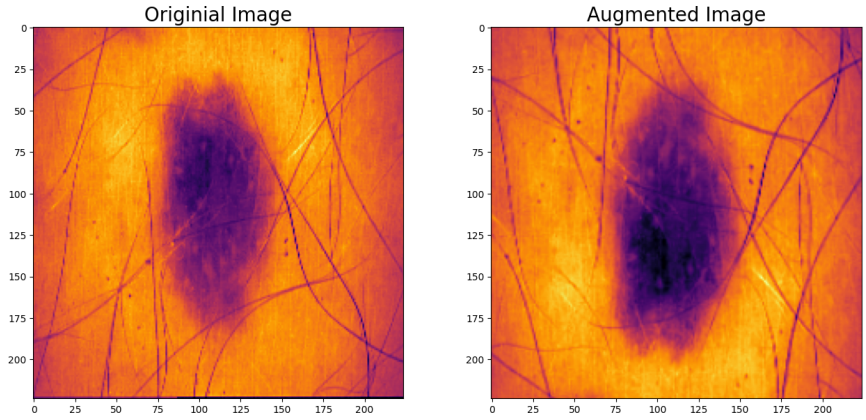
```
((8002, 3), (1100, 3), (1100, 3))
```

Create an Image Data Augmentation Layer

```
#Build Augmentation layer
augmentation_layer=Sequential([
    layers.RandomFlip(mode='horizontal_and_vertical',seed=CFG.TF_SEED),
    layers.RandomZoom(height_factor=(-0.1,0.1),width_factor=(-0.1,0.1),seed=CFG.TF_SEED)],
    name='augmentation_layer'
)
```

```
image=tf.image.rgb_to_grayscale(sample_image)
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(14,10))

#set the spacing between subplots
fig.tight_layout(pad=6.0)
#view orignial Image
ax1.set_title('Originial Image',fontsize=20)
ax1.imshow(image,cmap='inferno')
#view augmented image
ax2.set_title('Augmented Image',fontsize=20)
ax2.imshow(augmentation_layer(image),cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x7f9f302d9dc0>
```



Implementing ResNet50

Load Images in numpy array from dataframes

```
def encode_labels(labels,encode_depth=2):
  return tf.one_hot(labels,depth=encode_depth).numpy()

def create_pipeline(df,load_function,augment=False,batch_size=32,shuffle=False,cache=None,prefetch=False):
  '''
  Generates an input pipeline using tf.data api given a pandas Dataframe and image loading functions
  @parameters
  df : pd.Dataframes
  load_function : function used to load images given their paths
  augment: bool: condition for applying augmentation
  batch_size : int : size for batched
  shuffle : bool: condition for shuffling
  cache :(str) : cache path for caching data, data is not cached when None
  prefetch : bool: condition for prefetching data
```

```
    returns:
    dataset : tf.data.dataset
    '''
    # Get image paths and labels from DatFrame
    image_paths = df.image_path
    image_labels= encode_labels(df.label_encoded)
    AUTOTUNE=tf.data.AUTOTUNE

    #create dataset with raw data from DataFrame
    ds= tf.data.Dataset.from_tensor_slices((image_paths,image_labels))

    #map augmentation layer and load fucnction to dataset input if augment is true
    if augment:
      ds=ds.map(lambda x,y :(augmentation_layer(load_function(x)),y),num_parallel_calls=AUTOTUNE)
    else:
      ds=ds.map(lambda x,y:(load_function(x),y),num_parallel_calls=AUTOTUNE)

    #applying shuffle
    if shuffle:
      ds=ds.shuffle(buffer_size=1000)

    #apply batching
    ds=ds.batch(batch_size)

    #apply caching based on condition
    #Note: use cache in memory(cache='') if the data is small enough to fit in memory
    if cache !=None:
      ds=ds.cache(cache)

    if prefetch:
      ds=ds.prefetch(buffer_size=AUTOTUNE)
    return ds


#Generate Train Input Pipeline
train_ds=create_pipeline(train_df,_load,augment=True,batch_size=CFG.BATCH_SIZE,
                         shuffle=False,
                         prefetch=True)
val_ds=create_pipeline(val_df,_load, augment=False,batch_size=CFG.BATCH_SIZE,
                       shuffle=False,
                       prefetch=False)
test_ds=create_pipeline(test_df,_load,
                        batch_size=CFG.BATCH_SIZE,
                        shuffle=False,
                        prefetch=False)


resnet=ResNet50V2(input_shape=CFG.IMAGE_SIZE, weights='imagenet',include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_ker
94668760/94668760 [==============================] - 6s 0us/step
```

```
for layer in resnet.layers:
  layer.trainable=True
n=len(resnet.layers)
#resnet.layers[0].trainable=True

#resnet.layers[n-2].trainable=True
resnet.layers[n-1].trainable=True


model_name='ResNet50_local'
tf.random.set_seed(CFG.SEED)
def resnet_model():
  initializer=tf.keras.initializers.GlorotNormal()

  resnet_sequential=Sequential([
      layers.Input(shape=CFG.IMAGE_SIZE,dtype=tf.float32,name='input_image'),
      resnet,
      layers.MaxPooling2D(),
      layers.BatchNormalization(),
      layers.Dropout(0.5),
      Flatten(),
      layers.Dense(256,activation='relu',kernel_initializer=initializer, kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4), bias_reg
      #layers.MaxPooling2D(),
      layers.BatchNormalization(),
      tf.keras.layers.LeakyReLU(alpha=0.08),
      layers.Dropout(0.5),
      layers.Dense(2,dtype=tf.float32,activation='sigmoid',kernel_initializer=initializer)],
      name='ResNet50_Sequential'
  )
```

```
  return resnet_sequential


#generating model
model_resnet_local=resnet_model()
model_resnet_local.summary()
```

```
Model: "ResNet50_Sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 2048)       0
 2D)

 batch_normalization (BatchN  (None, 3, 3, 2048)       8192
 ormalization)

 dropout (Dropout)           (None, 3, 3, 2048)        0

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 256)               4718848

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 leaky_re_lu (LeakyReLU)     (None, 256)               0

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 2)                 514

=================================================================
Total params: 28,293,378
Trainable params: 28,243,330
Non-trainable params: 50,048
_________________________________________________________________
/usr/local/lib/python3.9/dist-packages/keras/initializers/initializers.py:120: UserWarning: The initializer GlorotNormal is unseede
  warnings.warn(
```

```
plot_model(model_resnet_local,dpi=60,show_shapes=True)
```

```
def train_model(model,num_epochs,callbacks_list,tf_train_data,tf_valid_data=None,shuffling=False):
  '''
  Trains a TensorFlow Model and returns a dict object containing the model metrics history data
  Input
  model: to be trained
  num_epochs,
  callbacks_list : list containing callback functions for model,
  tf_train_data
  tf_valid_data
  shuffling
  Output
  model_history : dict : conatining loss and metrics values tarcked during trg
  '''
  model_histroy={}
  if tf_valid_data !=None:
    model_history=model.fit(tf_train_data,
                            epochs=num_epochs,
                            validation_data=tf_valid_data,
                            validation_steps=int(len(tf_valid_data)),
                            callbacks=callbacks_list,
                            shuffle=shuffling)
  if tf_valid_data==None:
    model_history=model.fit(tf_train_data,
                            epochs=num_epochs,
                            validation_data=tf_valid_data,
                            callbacks=callbacks_list,
                            shuffle=shuffling)
  return model_history
```

| leaky re lu | input: | (None, 256) |

```
!pip install tensorflow-addons==0.16.1
import tensorflow_addons as tfa
from tensorflow.keras import metrics
from tensorflow_addons.metrics import F1Score
from tensorflow.keras.optimizers import Adam
from tensorflow_addons.metrics import MatthewsCorrelationCoefficient
#defining early stopping

early_stopping_callback=tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=4,
    restore_best_weights=True
)
#defining reduce learning rate callback
reduce_lr_callback=tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    patience=3,
    factor=0.1,
    verbose=1
)

#define callbacks and metrics
#CALLBACKS=[early_stopping_callback,reduce_lr_callback]
CALLBACKS=[reduce_lr_callback]
METRICS=['accuracy',
         metrics.Precision(name='precision'),
         metrics.Recall(name='recall'),
         F1Score(num_classes=2,name='f1'),
         MatthewsCorrelationCoefficient(num_classes=2,name='mc'),
         tf.keras.metrics.TruePositives(name='tp'),
         tf.keras.metrics.TrueNegatives(name='tn'),
         tf.keras.metrics.FalseNegatives(name='fn'),
         tf.keras.metrics.FalsePositives(name='fp')]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow-addons==0.16.1
  Downloading tensorflow_addons-0.16.1-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 49.4 MB/s eta 0:00:00
Collecting typeguard>=2.7
  Downloading typeguard-3.0.2-py3-none-any.whl (30 kB)
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.9/dist-packages (from typeguard>=2.7->tensorflow-
Requirement already satisfied: importlib-metadata>=3.6 in /usr/local/lib/python3.9/dist-packages (from typeguard>=2.7->tensorflow-a
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=3.6->typeguard>=2.7->t
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.16.1 typeguard-3.0.2
/usr/local/lib/python3.9/dist-packages/tensorflow_addons/utils/ensure_tf_install.py:53: UserWarning: Tensorflow Addons supports usi
 The versions of TensorFlow you are currently using is 2.12.0 and is not supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

```
tf.random.set_seed(CFG.SEED)

model_resnet_local.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy']
)

print(f'Training {model_resnet_local.name}')
print(f'Train on {len(train_df)} samples, validate on {len(val_df)} samples.')
print('-----------------------------------')

model_resnet_local_history=train_model(model_resnet_local,
                                       CFG.EPOCHS,
                                       CALLBACKS,
                                       train_ds,
                                       val_ds,
                                       shuffling=False)
```

```
Training ResNet50_Sequential
Train on 8002 samples, validate on 1100 samples.
-----------------------------------
Epoch 1/25
251/251 [==============================] - 2090s 8s/step - loss: 0.8315 - accuracy: 0.8667 - val_loss: 31.4341 - val_accuracy: 0.
Epoch 2/25
251/251 [==============================] - 149s 592ms/step - loss: 0.7302 - accuracy: 0.8630 - val_loss: 8.2627 - val_accuracy: 0
Epoch 3/25
251/251 [==============================] - 149s 592ms/step - loss: 0.6616 - accuracy: 0.8685 - val_loss: 69.0139 - val_accuracy:
Epoch 4/25
251/251 [==============================] - 147s 584ms/step - loss: 0.5380 - accuracy: 0.8908 - val_loss: 2.0772 - val_accuracy: 0
Epoch 5/25
251/251 [==============================] - 149s 594ms/step - loss: 0.4722 - accuracy: 0.8940 - val_loss: 0.8927 - val_accuracy: 0
Epoch 6/25
251/251 [==============================] - 152s 605ms/step - loss: 0.4895 - accuracy: 0.8914 - val_loss: 1.4981 - val_accuracy: 0
Epoch 7/25
251/251 [==============================] - 146s 583ms/step - loss: 0.4915 - accuracy: 0.8873 - val_loss: 0.9241 - val_accuracy: 0
Epoch 8/25
250/251 [============================>.] - ETA: 0s - loss: 0.6360 - accuracy: 0.8462
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
251/251 [==============================] - 149s 593ms/step - loss: 0.6360 - accuracy: 0.8463 - val_loss: 2.8663 - val_accuracy: 0
Epoch 9/25
251/251 [==============================] - 146s 583ms/step - loss: 0.6127 - accuracy: 0.8697 - val_loss: 0.8416 - val_accuracy: 0
Epoch 10/25
251/251 [==============================] - 149s 592ms/step - loss: 0.5729 - accuracy: 0.8777 - val_loss: 0.7835 - val_accuracy: 0
Epoch 11/25
251/251 [==============================] - 149s 593ms/step - loss: 0.5391 - accuracy: 0.8835 - val_loss: 0.7944 - val_accuracy: 0
Epoch 12/25
251/251 [==============================] - 147s 585ms/step - loss: 0.5179 - accuracy: 0.8837 - val_loss: 0.7579 - val_accuracy: 0
Epoch 13/25
251/251 [==============================] - 144s 575ms/step - loss: 0.4916 - accuracy: 0.8880 - val_loss: 0.7340 - val_accuracy: 0
Epoch 14/25
251/251 [==============================] - 146s 584ms/step - loss: 0.4701 - accuracy: 0.8904 - val_loss: 0.7183 - val_accuracy: 0
Epoch 15/25
251/251 [==============================] - 144s 574ms/step - loss: 0.4540 - accuracy: 0.8964 - val_loss: 0.7510 - val_accuracy: 0
Epoch 16/25
251/251 [==============================] - 145s 576ms/step - loss: 0.4421 - accuracy: 0.8928 - val_loss: 0.7210 - val_accuracy: 0
Epoch 17/25
250/251 [============================>.] - ETA: 0s - loss: 0.4157 - accuracy: 0.9019
Epoch 17: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
251/251 [==============================] - 145s 579ms/step - loss: 0.4157 - accuracy: 0.9019 - val_loss: 0.7916 - val_accuracy: 0
Epoch 18/25
251/251 [==============================] - 150s 598ms/step - loss: 0.4065 - accuracy: 0.9001 - val_loss: 0.7565 - val_accuracy: 0
Epoch 19/25
251/251 [==============================] - 144s 575ms/step - loss: 0.3978 - accuracy: 0.9008 - val_loss: 0.7536 - val_accuracy: 0
Epoch 20/25
251/251 [==============================] - 145s 579ms/step - loss: 0.3994 - accuracy: 0.9023 - val_loss: 0.7060 - val_accuracy: 0
Epoch 21/25
251/251 [==============================] - 147s 584ms/step - loss: 0.3951 - accuracy: 0.9035 - val_loss: 0.7091 - val_accuracy: 0
Epoch 22/25
251/251 [==============================] - 148s 591ms/step - loss: 0.3913 - accuracy: 0.9061 - val_loss: 0.7217 - val_accuracy: 0
Epoch 23/25
251/251 [==============================] - 145s 577ms/step - loss: 0.3921 - accuracy: 0.9031 - val_loss: 0.6914 - val_accuracy: 0
Epoch 24/25
251/251 [==============================] - 146s 581ms/step - loss: 0.3839 - accuracy: 0.9065 - val_loss: 0.6981 - val_accuracy: 0
Epoch 25/25
251/251 [==============================] - 148s 590ms/step - loss: 0.3796 - accuracy: 0.9063 - val_loss: 0.7024 - val_accuracy: 0
```

```
eval_result=model_resnet_local.evaluate(test_ds)
```

```
35/35 [==============================] - 463s 13s/step - loss: 0.4120 - accuracy: 0.8955
```

```
model_prediction=model_resnet_local.predict(test_ds,verbose=1)
model_prediction
```

```
35/35 [==============================] - 8s 191ms/step
array([[0.9732631 , 0.02533531],
       [0.5921483 , 0.39677164],
       [0.97719157, 0.02632946],
       ...,
       [0.08346026, 0.9153587 ],
       [0.05139947, 0.95040286],
       [0.04322391, 0.95905143]], dtype=float32)
```

```
history=model_resnet_local_history
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```