Configuration Manual

MSc Research Project
Data Analytics

Ekansh Kothiyal
Student ID: 23222417

School of Computing
National College of Ireland

Supervisor: Shubham Subhnil

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ekansh Kothiyal
Student ID:	23222417
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Shubham Subhnil
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	5

Signature:	Ekansh Kothiyal
Date:	7th December 2024

Configuration Manual

Ekansh Kothiyal
23222417

1 Introduction

This manual discusses in detail the instructions for configuring and deploying the tuberculosis detection system using pre-trained CNN models. A hybrid approach was implemented using deep learning models and transfer learning techniques to perfectly identifyTB infected from healthy cells.

2 System Requirements

Certain system requirements are needed for efficient model processing and minimizing the duration of time.

2.1 Hardware Requirements

The device used for the implementation was a Lenovo IdeaPad 330 with the below configurations:

1. Processor: Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
2. RAM: 8.00 GB
3. Hard Disk: 256GB SSD, 1 TB HDD
4. OS: Windows 10 Pro 64–bit

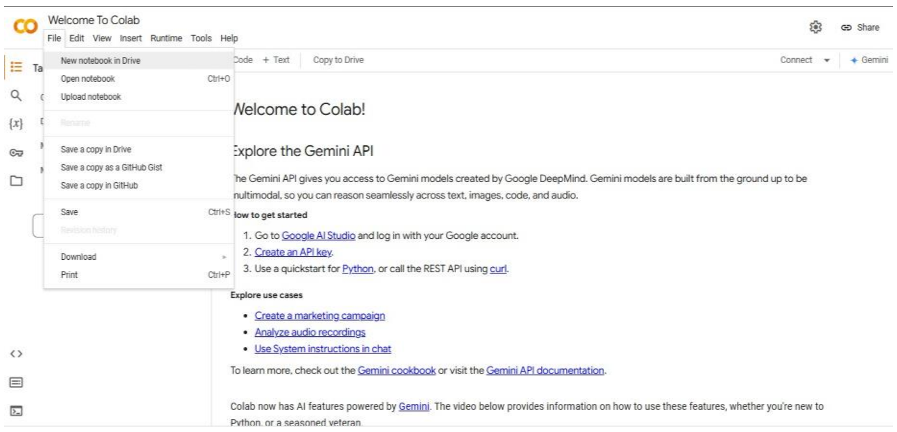
2.2 Software Requirements

Software	**Information**
Google Collab Platform	Cloud-based environment for running Python scripts.
Python	Google Collab uses version 3.8 or later.
TensorFlow/Keras	Building, training, and evaluating CNN models like MobileNetV2 and DenseNet121.
NumPy	For mathematical and array manipulation.
Pandas	For managing and processing datasets.
Matplotlib/Seaborn	For visualizing data.
sci-kit-learn	For generating metrics like confusion matrices, precision, recall, and F1-scores.

Table 1: Software and their Information

3 Implementation

3.1 Sign in to Google Collab, create a new notebook, and mount it to Google Drive.



Velcome to Colab!

xplore the Gemini API

he Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be nultimodal, so you can reason seamlessly across text, images, code, and audio.

low to get started

1. Go to Google AI Studio and log in with your Google account.
2. Create an API key.
3. Use a quickstart for Python, or call the REST API using curl.

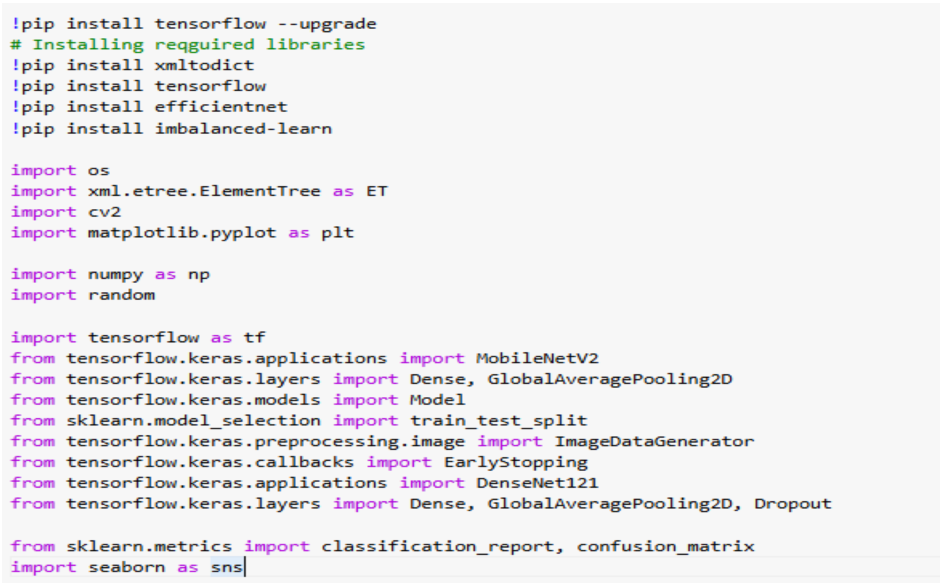
Explore use cases

- Create a marketing campaign
- Analyze audio recordings
- Use System instructions in chat

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

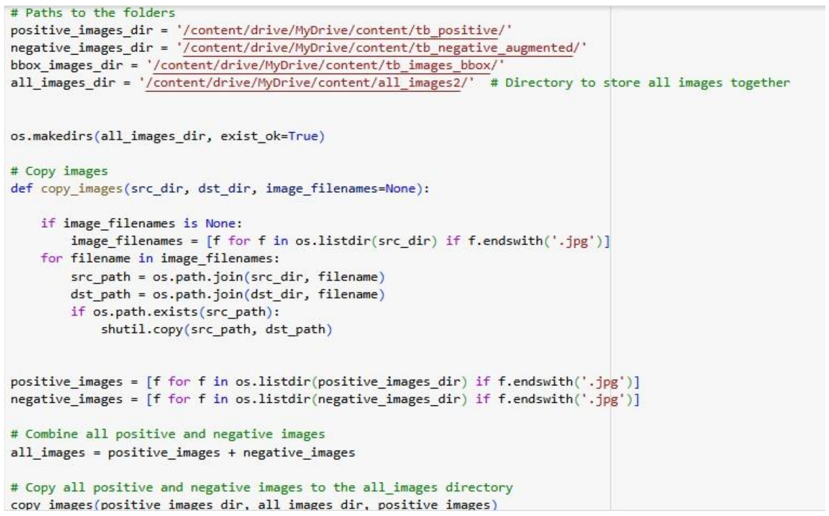
Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.

3.2 Import all the necessary libraries



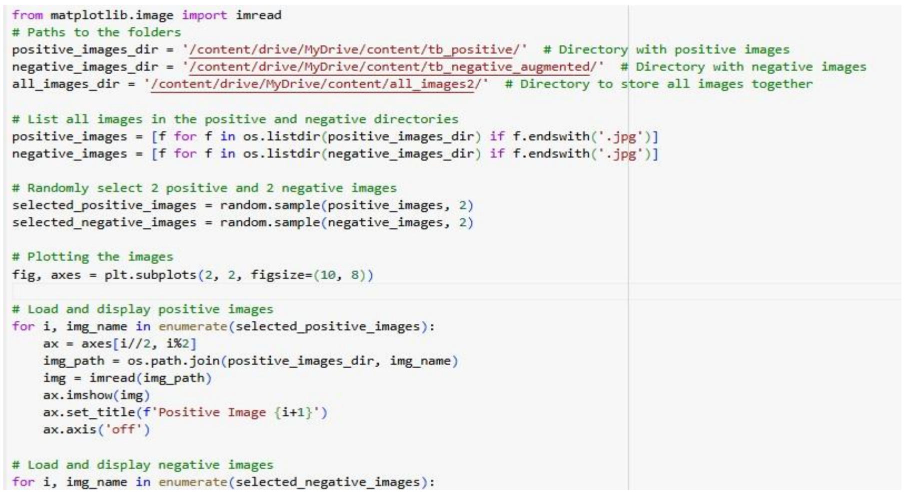
```
!pip install tensorflow --upgrade
# Installing reqguired libraries
!pip install xmltodict
!pip install tensorflow
!pip install efficientnet
!pip install imbalanced-learn

import os
import xml.etree.ElementTree as ET
import cv2
import matplotlib.pyplot as plt

import numpy as np
import random

import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

3.3 Importing the dataset with positive and negative images and combining them into a single folder with all the images.



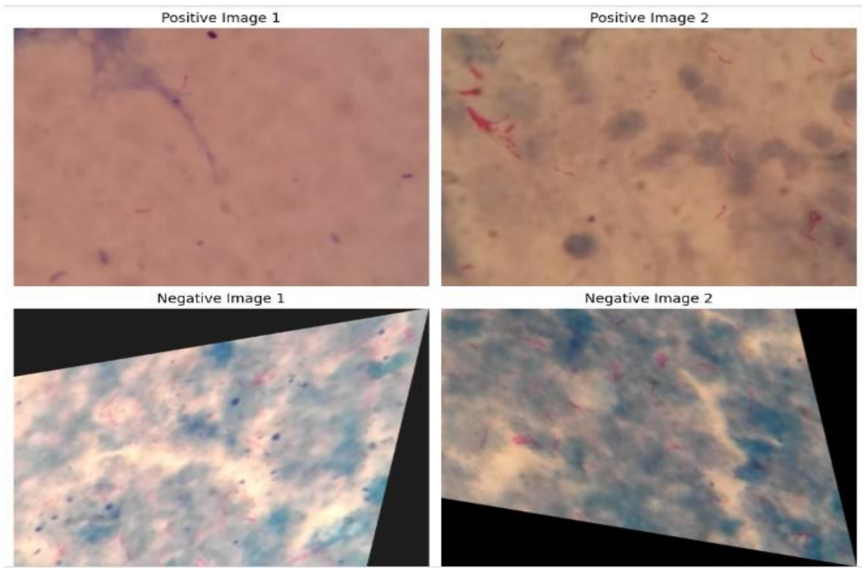
```
# Paths to the folders
positive_images_dir = '/content/drive/MyDrive/content/tb_positive/'
negative_images_dir = '/content/drive/MyDrive/content/tb_negative_augmented/'
bbox_images_dir = '/content/drive/MyDrive/content/tb_images_bbox/'
all_images_dir = '/content/drive/MyDrive/content/all_images2/'  # Directory to store all images together


os.makedirs(all_images_dir, exist_ok=True)

# Copy images
def copy_images(src_dir, dst_dir, image_filenames=None):

    if image_filenames is None:
        image_filenames = [f for f in os.listdir(src_dir) if f.endswith('.jpg')]
    for filename in image_filenames:
        src_path = os.path.join(src_dir, filename)
        dst_path = os.path.join(dst_dir, filename)
        if os.path.exists(src_path):
            shutil.copy(src_path, dst_path)


positive_images = [f for f in os.listdir(positive_images_dir) if f.endswith('.jpg')]
negative_images = [f for f in os.listdir(negative_images_dir) if f.endswith('.jpg')]

# Combine all positive and negative images
all_images = positive_images + negative_images

# Copy all positive and negative images to the all_images directory
copy_images(positive_images_dir, all_images_dir, positive_images)
```

3.4 Preprocessing the dataset



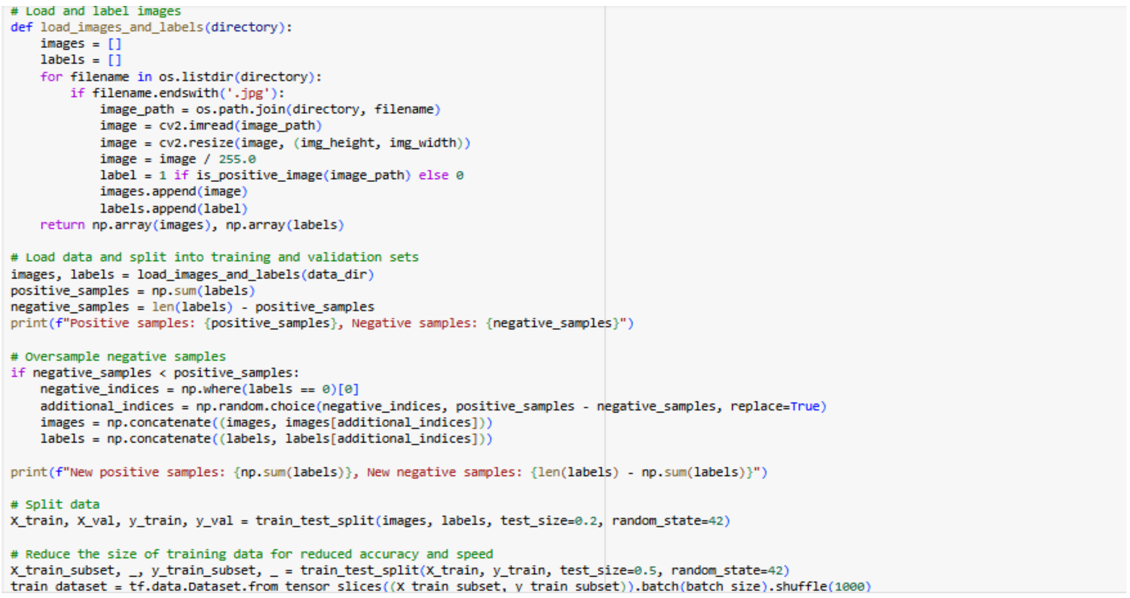
```
from matplotlib.image import imread
# Paths to the folders
positive_images_dir = '/content/drive/MyDrive/content/tb_positive/'  # Directory with positive images
negative_images_dir = '/content/drive/MyDrive/content/tb_negative_augmented/'  # Directory with negative images
all_images_dir = '/content/drive/MyDrive/content/all_images2/'  # Directory to store all images together

# List all images in the positive and negative directories
positive_images = [f for f in os.listdir(positive_images_dir) if f.endswith('.jpg')]
negative_images = [f for f in os.listdir(negative_images_dir) if f.endswith('.jpg')]

# Randomly select 2 positive and 2 negative images
selected_positive_images = random.sample(positive_images, 2)
selected_negative_images = random.sample(negative_images, 2)

# Plotting the images
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# Load and display positive images
for i, img_name in enumerate(selected_positive_images):
    ax = axes[i//2, i%2]
    img_path = os.path.join(positive_images_dir, img_name)
    img = imread(img_path)
    ax.imshow(img)
    ax.set_title(f'Positive Image {i+1}')
    ax.axis('off')

# Load and display negative images
for i, img_name in enumerate(selected_negative_images):
```

3.5 Visualizing the images in the dataset



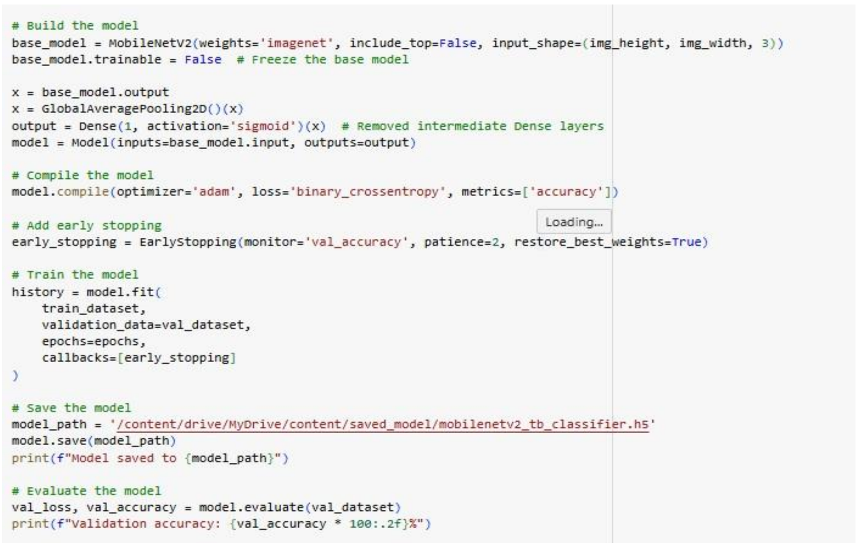
3.6 Oversampling the minority class (Negative sample)

Due to class imbalance (less sample in negative class) oversampling was done in negative class with positive samples as 1863 and negative as 1863, and then splitting the dataset into train and validation for training the models.



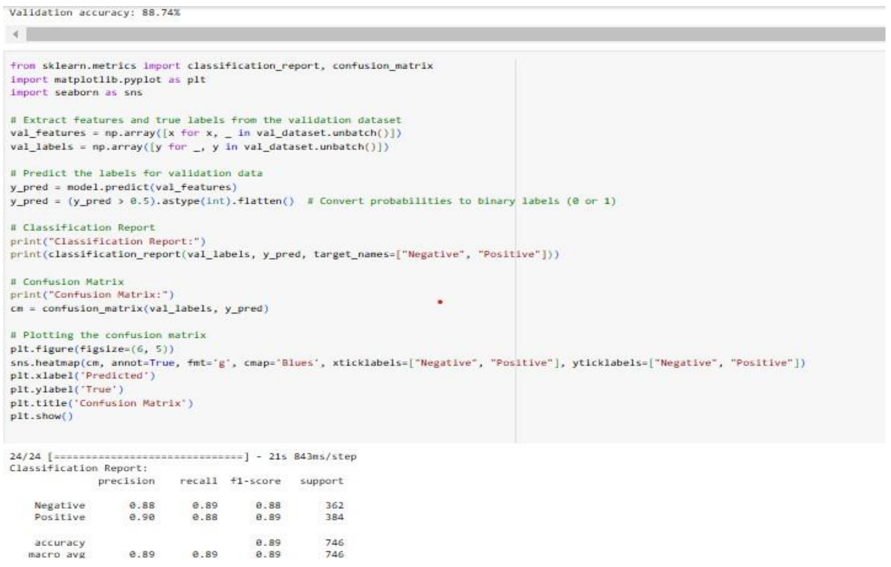
```python
# Load and label images
def load_images_and_labels(directory):
    images = []
    labels = []
    for filename in os.listdir(directory):
        if filename.endswith('.jpg'):
            image_path = os.path.join(directory, filename)
            image = cv2.imread(image_path)
            image = cv2.resize(image, (img_height, img_width))
            image = image / 255.0
            label = 1 if is_positive_image(image_path) else 0
            images.append(image)
            labels.append(label)
    return np.array(images), np.array(labels)

# Load data and split into training and validation sets
images, labels = load_images_and_labels(data_dir)
positive_samples = np.sum(labels)
negative_samples = len(labels) - positive_samples
print(f"Positive samples: {positive_samples}, Negative samples: {negative_samples}")

# Oversample negative samples
if negative_samples < positive_samples:
    negative_indices = np.where(labels == 0)[0]
    additional_indices = np.random.choice(negative_indices, positive_samples - negative_samples, replace=True)
    images = np.concatenate((images, images[additional_indices]))
    labels = np.concatenate((labels, labels[additional_indices]))

print(f"New positive samples: {np.sum(labels)}, New negative samples: {len(labels) - np.sum(labels)}")

# Split data
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Reduce the size of training data for reduced accuracy and speed
X_train_subset, _, y_train_subset, _ = train_test_split(X_train, y_train, test_size=0.5, random_state=42)
train_dataset = tf.data.Dataset.from_tensor_slices((X_train_subset, y_train_subset)).batch(batch_size).shuffle(1000)
```

3.7 Building a MobileNetV2 model



```python
# Build the model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False  # Freeze the base model

x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(1, activation='sigmoid')(x)  # Removed intermediate Dense layers
model = Model(inputs=base_model.input, outputs=output)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Add early stopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=2, restore_best_weights=True)

# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs,
    callbacks=[early_stopping]
)

# Save the model
model_path = '/content/drive/MyDrive/content/saved_model/mobilenetv2_tb_classifier.h5'
model.save(model_path)
print(f"Model saved to {model_path}")

# Evaluate the model
val_loss, val_accuracy = model.evaluate(val_dataset)
print(f"Validation accuracy: {val_accuracy * 100:.2f}%")
```

3.8 After Model implementation validation Accuracy is considered as evaluation factor

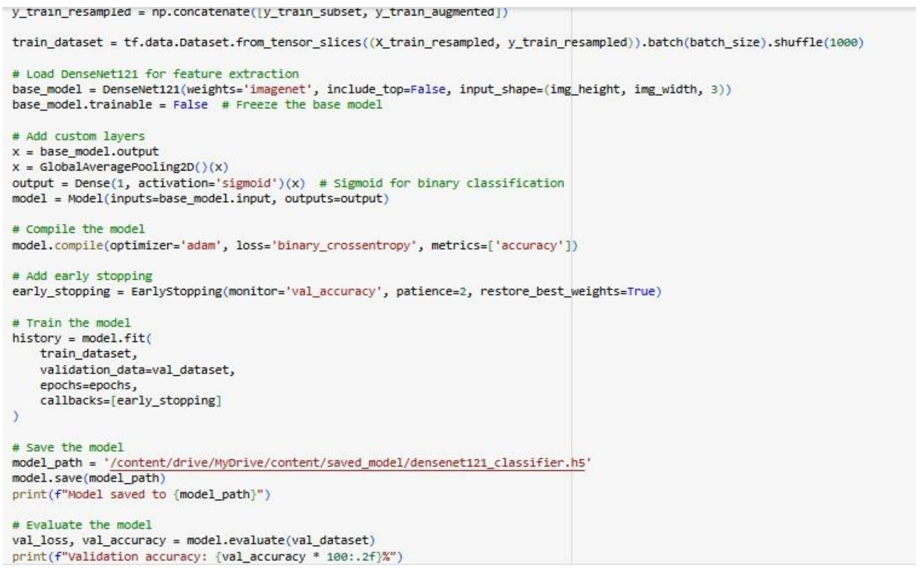


```
Validation accuracy: 88.74%
```

```python
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Extract features and true labels from the validation dataset
val_features = np.array([x for x, _ in val_dataset.unbatch()])
val_labels = np.array([y for _, y in val_dataset.unbatch()])

# Predict the labels for validation data
y_pred = model.predict(val_features)
y_pred = (y_pred > 0.5).astype(int).flatten()  # Convert probabilities to binary labels (0 or 1)

# Classification Report
print("Classification Report:")
print(classification_report(val_labels, y_pred, target_names=["Negative", "Positive"]))

# Confusion Matrix
print("Confusion Matrix:")
cm = confusion_matrix(val_labels, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=["Negative", "Positive"], yticklabels=["Negative", "Positive"])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

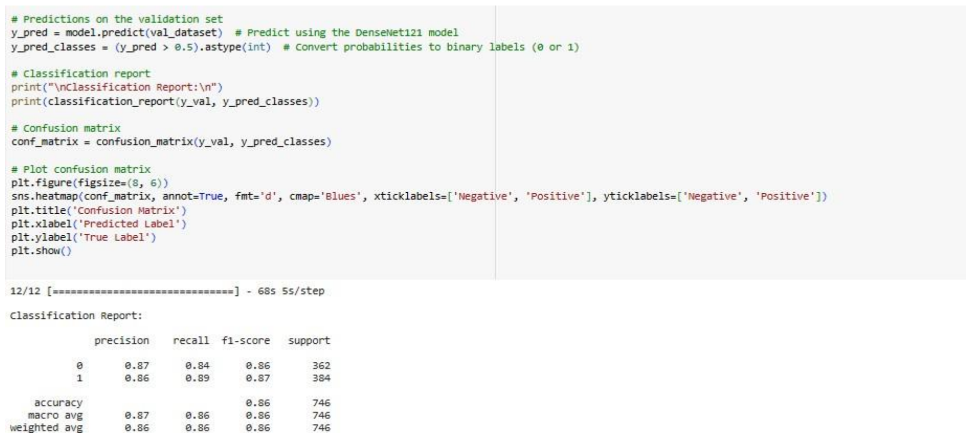
```
24/24 [==============================] - 21s 843ms/step
Classification Report:
              precision    recall  f1-score   support

    Negative       0.88      0.89      0.88       362
    Positive       0.90      0.88      0.89       384

    accuracy                           0.89       746
   macro avg       0.89      0.89      0.89       746
```

3.9 Building a DenseNet121 model



```python
y_train_resampled = np.concatenate([y_train_subset, y_train_augmented])

train_dataset = tf.data.Dataset.from_tensor_slices((X_train_resampled, y_train_resampled)).batch(batch_size).shuffle(1000)

# Load DenseNet121 for feature extraction
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False  # Freeze the base model

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(1, activation='sigmoid')(x)  # Sigmoid for binary classification
model = Model(inputs=base_model.input, outputs=output)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Add early stopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=2, restore_best_weights=True)

# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs,
    callbacks=[early_stopping]
)

# Save the model
model_path = '/content/drive/MyDrive/content/saved_model/densenet121_classifier.h5'
model.save(model_path)
print(f"Model saved to {model_path}")

# Evaluate the model
val_loss, val_accuracy = model.evaluate(val_dataset)
print(f"Validation accuracy: {val_accuracy * 100:.2f}%")
```

3.10 After Model implementation validation Accuracy is considered as evaluation factor



```
# Predictions on the validation set
y_pred = model.predict(val_dataset)  # Predict using the DenseNet121 model
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary labels (0 or 1)

# Classification report
print("\nClassification Report:\n")
print(classification_report(y_val, y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
12/12 [==============================] - 68s 5s/step

Classification Report:

              precision    recall  f1-score   support

           0       0.87      0.84      0.86       362
           1       0.86      0.89      0.87       384

    accuracy                           0.86       746
   macro avg       0.87      0.86      0.86       746
weighted avg       0.86      0.86      0.86       746
```

Total time taken:
MobileNetV2 model overall took 1009 seconds for the training time and evaluation time with the accuracy of 88.74% , and DenseNet121 model overall took 3274 seconds for the training time and evaluation time with the accuracy of 86.46%.

DenseNet121 is significantly slower, taking approx. 3.25x longer than MobileNetV2 for the same task.

To conclude, this code file includes one project file titled TB_detection.ipynb which includes both the models training and evaluation with the visualizations.

References

NumPy.org. 2023. NumPy 2.1.0 Release Notes. [online] Available at: https://numpy.org/doc/stable/release/2.1.0-notes.html

Scikit-learn.org. 2023. Scikit-learn: Machine Learning in Python — Scikit-learn 1.3.1 Documentation. [online] Available at: https://scikit-learn.org/stable/

TensorFlow.org. 2023. TensorFlow: An End-to-End Open Source Machine Learning Platform. [online] Available at: https://www.tensorflow.org/.