Configuration Manual

MSc Research Project
MSc Data Analytics

Yash Kutaphale
Student ID: X21195960

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Yash Santosh Kutaphale

Student ID: x21195960

Programme: MSc in Data Analytics **Year:** 2022-23

Module: MSc Research Project

Lecturer: Furqan Rustam

Submission Due Date: 14th August 2023

Project Title: Lung Cancer Detection Using Machine Learning And Deep Learning

Word Count: 1239 **Page Count:** 24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Yash Kutaphale

Date: 14/08/23

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Yash Kutaphale
Student ID: X21195960

Contents

1. Hardware & Software

1.1 Device Specifications

Device specifications

Device name	DESKTOP-Q3MB2TD
Processor	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
Installed RAM	20.0 GB (17.9 GB usable)
Device ID	A12E6FDB-62AA-4C4D-A791-65C95B3E27FA
Product ID	00327-35869-69866-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Device Specifications

1.2 Windows Specifications

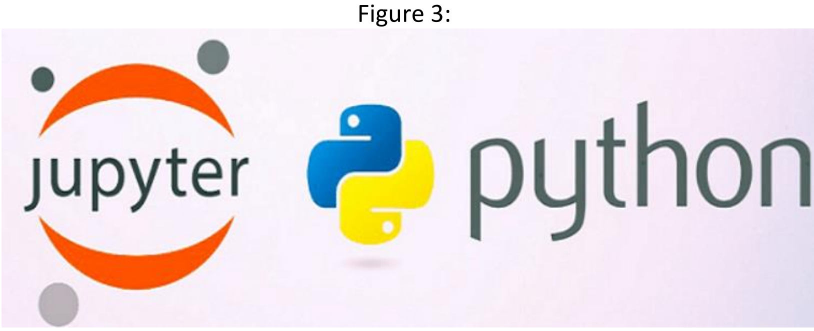
Windows specifications

Edition	Windows 11 Home Single Language
Version	22H2
Installed on	30-06-2023
OS build	22621.2070
Experience	Windows Feature Experience Pack 1000.22659.1000.0

Figure 2: Windows Specifications

1.3 Python and Jupyter

Figure 3:



Python Version: 3.10.1   Jupyter Version: 6.4.8

2 Data

2.1 Collection

The "IQOTHNCCD - Lung Cancer Dataset" dataset is provided from Kaggle. It provides a thorough set of information designed especially for lung cancer research.

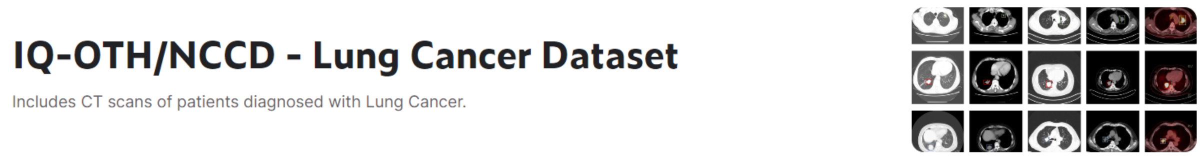


Figure 4: Dataset

2.2 Model Building

2.2.1 Approch 1

Figure 5: Python Code for importing all the required libraries:

Approach 1

```
#Loading all the required libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
from skimage.feature import graycomatrix, graycoprops
from skimage.io import imread_collection
from skimage.io import imread
from skimage.transform import resize
from sklearn.feature_selection import SelectKBest, f_classif
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
```

In this section all the necessary libraries are imported.

Figure 6: Python code for Loading Data:

Loading Data

```
# Set the path to your image dataset
dataset_directory = "C://Users//A//Desktop//CT Scan Dataset"

# Set the number of desired GLCM features to select
num_features = 20

# Set the number of classes in your dataset
num_classes = 3

# Set the batch size and number of training epochs
batch_size = 32
epochs = 10

# Load the image collection
image_collection = imread_collection(os.path.join(dataset_directory, '*', '*.jpg'))
```

Here first the data set path is set. Then the number of GLCM features is set to 20. The number of classes is set to 3 as there are three classes present in the data set. And later the image collection is loaded.

Figure 7: Python Code for Extracting and Performing Feature Selection:

Extracting GLCM Features

```
# Create an empty array to store the computed GLCM features
glcm_features = np.zeros((len(image_collection), num_features))

# Create an empty array to store the class labels
labels = []

# Loop over each image in the collection
for idx, image_path in enumerate(image_collection.files):

    # Read the image
    image = imread(image_path, as_gray=True)

    # Convert the image to unsigned integer type
    image = image.astype(np.uint8)

    # Compute the GLCM matrix
    glcm = graycomatrix(image, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)

    # Extract GLCM features
    features = np.ravel(graycoprops(glcm, prop='contrast'))

    # Store the features in the array
    glcm_features[idx, :] = features

    # Extract the class label from the image path
    label = os.path.basename(os.path.dirname(image_path))
    labels.append(label)
```

The computed GLCM features are stored in a variable and then each image runs through a loop Which extracts the GLCM features and stores in an array

Performing Feature Selection

```
# Convert labels to numerical values
label_mapping = {label: i for i, label in enumerate(set(labels))}
numerical_labels = np.array([label_mapping[label] for label in labels])

# Perform feature selection using ANOVA F-value
selector = SelectKBest(f_classif, k=num_features)
selected_features = selector.fit_transform(glcm_features, numerical_labels)

# Get the names of the selected features
selected_feature_indices = selector.get_support(indices=True)
selected_feature_names = [f'Feature {idx+1}' for idx in selected_feature_indices]

# Print the names of the selected features
print('Selected Features:')
for feature_name in selected_feature_names:
    print(feature_name)
```

All the labels are converted to numerical values and feature selection is performed using ANOVA F value. The name of the selected features is fetched.

Figure 8: Python Code to Split Data into Train and Test:

Train - Test Split

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(selected_features, numerical_labels, test_size=0.2, random_state=42)

# Normalize the selected features
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

The data set is split into trained and test and the selecyed features are normalise using standard scaler.

Figure 9: Python code to build the CNN Model:

Developing CNN Model

```
# Create the CNN model
model = Sequential([
    Dense(128, activation='relu', input_shape=(num_features,)),
    Dense(num_classes, activation='softmax')
])

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

# Convert the labels to one-hot encoding
y_train_encoded = to_categorical(y_train, num_classes=num_classes)
y_test_encoded = to_categorical(y_test, num_classes=num_classes)

# Train the model
model.fit(X_train_normalized, y_train_encoded,
          validation_data=(X_test_normalized, y_test_encoded),
          batch_size=batch_size,
          epochs=epochs)

# Save the trained model
model.save('lung_cancer_model.keras')

# Evaluate the model on the test dataset
_, test_accuracy = model.evaluate(X_test_normalized, y_test_encoded)

# Predict the classes for the test dataset
y_pred_probabilities = model.predict(X_test_normalized)
y_pred = np.argmax(y_pred_probabilities, axis=1)

# Compute evaluation metrics for the CNN model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Print the evaluation metrics for the CNN model
print('CNN Test Accuracy:', test_accuracy)
print('CNN Accuracy:', accuracy)
print('CNN Precision:', precision)
print('CNN Recall:', recall)
print('CNN F1-score:', f1)
```

In this part of the code the CNN model is Built and compiled. Then the lables are converted to one hot encoding and the model is trained using the trained data set. Later the model is use for prediction and evalution matrix are computed.

Figure 10: Python Code to build the SVM Model:

Developing SVM Model

```
# Build the SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the SVM model
svm_pred = svm_model.predict(X_test_normalized)

# Compute evaluation metrics for the SVM model
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred, average='weighted', zero_division=0)
svm_recall = recall_score(y_test, svm_pred, average='weighted')
svm_f1 = f1_score(y_test, svm_pred, average='weighted')

# Print the evaluation metrics for the SVM model
print('SVM Accuracy:', svm_accuracy)
print('SVM Precision:', svm_precision)
print('SVM Recall:', svm_recall)
print('SVM F1-score:', svm_f1)
```

```
SVM Accuracy: 0.6045454545454545
SVM Precision: 0.5282170608028943
SVM Recall: 0.6045454545454545
SVM F1-score: 0.5493479044700212
```

In this part of the code the SVM model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 11: Python Code to build the Random Forest Model:

Developing Random Forest Model

```
# Build the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the Random Forest model
rf_pred = rf_model.predict(X_test_normalized)

# Compute evaluation metrics for the Random Forest model
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_precision = precision_score(y_test, rf_pred, average='weighted', zero_division=0)
rf_recall = recall_score(y_test, rf_pred, average='weighted')
rf_f1 = f1_score(y_test, rf_pred, average='weighted')

# Print the evaluation metrics for the Random Forest model
print('Random Forest Accuracy:', rf_accuracy)
print('Random Forest Precision:', rf_precision)
print('Random Forest Recall:', rf_recall)
print('Random Forest F1-score:', rf_f1)
```

```
Random Forest Accuracy: 0.5772727272727273
Random Forest Precision: 0.5731013779814579
Random Forest Recall: 0.5772727272727273
Random Forest F1-score: 0.5748873543107312
```

In this part of the code the random forest model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 12: Python Code for building the Decision Tree Model:

Developing Decesion Tree Model

```
# Build the Decision Tree model
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the Decision Tree model
dt_pred = dt_model.predict(X_test_normalized)

# Compute evaluation metrics for the Decision Tree model
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred, average='weighted', zero_division=0)
dt_recall = recall_score(y_test, dt_pred, average='weighted')
dt_f1 = f1_score(y_test, dt_pred, average='weighted')

# Print the evaluation metrics for the Decision Tree model
print('Decision Tree Accuracy:', dt_accuracy)
print('Decision Tree Precision:', dt_precision)
print('Decision Tree Recall:', dt_recall)
print('Decision Tree F1-score:', dt_f1)
```

```
Decision Tree Accuracy: 0.5727272727272728
Decision Tree Precision: 0.5640756556740163
Decision Tree Recall: 0.5727272727272728
Decision Tree F1-score: 0.5664226682408501
```

In this part of the code the Decision tree model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 13: Python Code for Developing the Gradient Boosting Machine Model:

Developing Gradient Boosting Machine Model

```
# Build the Gradient Boosting Machine (GBM) model
gbm_model = GradientBoostingClassifier(n_estimators=100)
gbm_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the GBM model
gbm_pred = gbm_model.predict(X_test_normalized)

# Compute evaluation metrics for the GBM model
gbm_accuracy = accuracy_score(y_test, gbm_pred)
gbm_precision = precision_score(y_test, gbm_pred, average='weighted', zero_division=0)
gbm_recall = recall_score(y_test, gbm_pred, average='weighted')
gbm_f1 = f1_score(y_test, gbm_pred, average='weighted')

# Print the evaluation metrics for the GBM model
print('Gradient Boosting Machine Accuracy:', gbm_accuracy)
print('Gradient Boosting Machine Precision:', gbm_precision)
print('Gradient Boosting Machine Recall:', gbm_recall)
print('Gradient Boosting Machine F1-score:', gbm_f1)
```

```
Gradient Boosting Machine Accuracy: 0.6409090909090909
Gradient Boosting Machine Precision: 0.6317883302319494
Gradient Boosting Machine Recall: 0.6409090909090909
Gradient Boosting Machine F1-score: 0.6227518225735617
```

In this part of the code the Gradient boosting machine model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 14: Python Code for developing the KNN Model:

Developing KNN Model

```
# Build the k-Nearest Neighbors (KNN) model
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the KNN model
knn_pred = knn_model.predict(X_test_normalized)

# Compute evaluation metrics for the KNN model
knn_accuracy = accuracy_score(y_test, knn_pred)
knn_precision = precision_score(y_test, knn_pred, average='weighted', zero_division=0)
knn_recall = recall_score(y_test, knn_pred, average='weighted')
knn_f1 = f1_score(y_test, knn_pred, average='weighted')

# Print the evaluation metrics for the KNN model
print('k-Nearest Neighbors Accuracy:', knn_accuracy)
print('k-Nearest Neighbors Precision:', knn_precision)
print('k-Nearest Neighbors Recall:', knn_recall)
print('k-Nearest Neighbors F1-score:', knn_f1)
```

```
k-Nearest Neighbors Accuracy: 0.6545454545454545
k-Nearest Neighbors Precision: 0.6508510638297873
k-Nearest Neighbors Recall: 0.6545454545454545
k-Nearest Neighbors F1-score: 0.6323060991988145
```

In this part of the code the KNN model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 15: Python Code for building Logistic Regression Model:

Developing Logistic Regression Model

```
# Build the Logistic Regression model
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_normalized, y_train)

# Predict the classes for the test dataset using the Logistic Regression model
logreg_pred = logreg_model.predict(X_test_normalized)

# Compute evaluation metrics for the Logistic Regression model
logreg_accuracy = accuracy_score(y_test, logreg_pred)
logreg_precision = precision_score(y_test, logreg_pred, average='weighted', zero_division=0)
logreg_recall = recall_score(y_test, logreg_pred, average='weighted')
logreg_f1 = f1_score(y_test, logreg_pred, average='weighted')

# Print the evaluation metrics for the Logistic Regression model
print('Logistic Regression Accuracy:', logreg_accuracy)
print('Logistic Regression Precision:', logreg_precision)
print('Logistic Regression Recall:', logreg_recall)
print('Logistic Regression F1-score:', logreg_f1)
```

```
Logistic Regression Accuracy: 0.6136363636363636
Logistic Regression Precision: 0.531367490956532
Logistic Regression Recall: 0.6136363636363636
Logistic Regression F1-score: 0.5638501191348974
```

In this part of the code the Logistic regression model is Built. Later the model is use for prediction and evalution matrix are computed.

Figure 16: Python Code to develop the ResNet Model:

Developing ResNet Model

```
# Create a fully connected neural network
fc_model = Sequential([
    Dense(128, activation='relu', input_shape=(num_features,)),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])

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

# Train the model
fc_model.fit(X_train_normalized, y_train_encoded,
             validation_data=(X_test_normalized, y_test_encoded),
             batch_size=batch_size,
             epochs=epochs)

# Evaluate the model on the test dataset
_, fc_test_accuracy = fc_model.evaluate(X_test_normalized, y_test_encoded)

# Predict the classes for the test dataset using the fully connected model
fc_y_pred_probabilities = fc_model.predict(X_test_normalized)
fc_y_pred = np.argmax(fc_y_pred_probabilities, axis=1)

# Compute evaluation metrics for the fully connected model
fc_accuracy = accuracy_score(y_test, fc_y_pred)
fc_precision = precision_score(y_test, fc_y_pred, average='weighted', zero_division=0)
fc_recall = recall_score(y_test, fc_y_pred, average='weighted')
fc_f1 = f1_score(y_test, fc_y_pred, average='weighted')

# Print the evaluation metrics for the fully connected model
print('Fully Connected Test Accuracy:', fc_test_accuracy)
print('Fully Connected Accuracy:', fc_accuracy)
print('Fully Connected Precision:', fc_precision)
print('Fully Connected Recall:', fc_recall)
print('Fully Connected F1-score:', fc_f1)
```

```
Fully Connected Test Accuracy: 0.6409090757369995
Fully Connected Accuracy: 0.6409090909090909
Fully Connected Precision: 0.5548832897290247
Fully Connected Recall: 0.6409090909090909
Fully Connected F1-score: 0.594112181524769
```

In this part of the code the ResNet model is Built and compiled. The model is then trained and Later the model is use for prediction and evalution matrix are computed.

Figure 17: Python Code to build Efficient Model:

Developing EfficientNet Model

```python
# After Normalizing the selected features:
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

# Function to pad the features to 32x32
def pad_features(features, target_size=(32, 32)):
    padded_features = np.zeros((features.shape[0], target_size[0], target_size[1]))
    offset_x = (target_size[0] - features.shape[1]) // 2
    offset_y = (target_size[1] - features.shape[2]) // 2
    padded_features[:, offset_x:offset_x+features.shape[1], offset_y:offset_y+features.shape[2]] = features
    return padded_features

# Reshape the data to (samples, 20, 20)
X_train_reshaped = X_train_normalized.reshape(-1, num_features, 1)
X_test_reshaped = X_test_normalized.reshape(-1, num_features, 1)

# Pad the reshaped data
X_train_padded = pad_features(X_train_reshaped)
X_test_padded = pad_features(X_test_reshaped)

# Add an extra dimension for the channel (since EfficientNet expects it)
X_train_padded = X_train_padded[..., np.newaxis]
X_test_padded = X_test_padded[..., np.newaxis]

# Now use the padded data for EfficientNetB0:
efficientnet_base = EfficientNetB0(weights=None, include_top=False, input_shape=(32, 32, 1))
x = efficientnet_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
efficientnet_model = Model(inputs=efficientnet_base.input, outputs=predictions)
efficientnet_model.compile(optimizer='adam',
                           loss='categorical_crossentropy',
                           metrics=['accuracy'])

efficientnet_model.fit(X_train_padded, y_train_encoded,
                       validation_data=(X_test_padded, y_test_encoded),
                       batch_size=batch_size,
                       epochs=epochs)

_, efficientnet_test_accuracy = efficientnet_model.evaluate(X_test_padded, y_test_encoded)
efficientnet_y_pred_probabilities = efficientnet_model.predict(X_test_padded)
efficientnet_y_pred = np.argmax(efficientnet_y_pred_probabilities, axis=1)

efficientnet_accuracy = accuracy_score(y_test, efficientnet_y_pred)
efficientnet_precision = precision_score(y_test, efficientnet_y_pred, average='weighted', zero_division=0)
efficientnet_recall = recall_score(y_test, efficientnet_y_pred, average='weighted')
efficientnet_f1 = f1_score(y_test, efficientnet_y_pred, average='weighted')

print('EfficientNet Test Accuracy:', efficientnet_test_accuracy)
print('EfficientNet Accuracy:', efficientnet_accuracy)
print('EfficientNet Precision:', efficientnet_precision)
print('EfficientNet Recall:', efficientnet_recall)
print('EfficientNet F1-score:', efficientnet_f1)
```

```
EfficientNet Test Accuracy: 0.4954545497894287
EfficientNet Accuracy: 0.4954545454545455
EfficientNet Precision: 0.24547520661157027
EfficientNet Recall: 0.4954545454545455
EfficientNet F1-score: 0.32829510914617294
```

After normalising the selected features using standard scaler tye features are reshaped into 32x32. The data is reshaped to 20x20 and padded. After that extra dimension is added as the model used is EfficientNet Then the padded data is used to built and compiled model. The evaluation metrices are printed in the end.

Figure 18: Python Code to generate comparison of results:

Comparison of the Results

```
# Compare the evaluation metrics of the models
print('Evaluation Metrics:')
print('-----------------------')
print('CNN Accuracy:', accuracy)
print('SVM Accuracy:', svm_accuracy)
print('Random Forest Accuracy:', rf_accuracy)
print('Decision Tree Accuracy:', dt_accuracy)
print('Gradient Boosting Machine Accuracy:', gbm_accuracy)
print('k-Nearest Neighbors Accuracy:', knn_accuracy)
print('Logistic Regression Accuracy:', logreg_accuracy)
print('ResNet Accuracy:', fc_accuracy)
print('EfficientNet Accuracy:', efficientnet_accuracy)
print('-----------------------')
print('CNN Precision:', precision)
print('SVM Precision:', svm_precision)
print('Random Forest Precision:', rf_precision)
print('Decision Tree Precision:', dt_precision)
print('Gradient Boosting Machine Precision:', gbm_precision)
print('k-Nearest Neighbors Precision:', knn_precision)
print('Logistic Regression Precision:', logreg_precision)
print('ResNet Precision:', fc_precision)
print('EfficientNet Precision:', efficientnet_precision)
print('-----------------------')
print('CNN Recall:', recall)
print('SVM Recall:', svm_recall)
print('Random Forest Recall:', rf_recall)
print('Decision Tree Recall:', dt_recall)
print('Gradient Boosting Machine Recall:', gbm_recall)
print('k-Nearest Neighbors Recall:', knn_recall)
print('Logistic Regression Recall:', logreg_recall)
print('ResNet Recall:', fc_recall)
print('EfficientNet Recall:', efficientnet_recall)
print('-----------------------')
print('-----------------------')
print('CNN F1-score:', f1)
print('SVM F1-score:', svm_f1)
print('Random Forest F1-score:', rf_f1)
print('Decision Tree F1-score:', dt_f1)
print('Gradient Boosting Machine F1-score:', gbm_f1)
print('k-Nearest Neighbors F1-score:', knn_f1)
print('Logistic Regression F1-score:', logreg_f1)
print('ResNet F1-score:', fc_f1)
print('EfficientNet F1-score:', efficientnet_f1)
```

```
Evaluation Metrics:
-----------------------
CNN Accuracy: 0.6272727272727273
SVM Accuracy: 0.6045454545454545
Random Forest Accuracy: 0.5772727272727273
Decision Tree Accuracy: 0.5727272727272728
Gradient Boosting Machine Accuracy: 0.6409090909090909
k-Nearest Neighbors Accuracy: 0.6545454545454545
Logistic Regression Accuracy: 0.6136363636363636
ResNet Accuracy: 0.6409090909090909
EfficientNet Accuracy: 0.4954545454545455
-----------------------
CNN Precision: 0.5621333417713508
SVM Precision: 0.5282170608028943
Random Forest Precision: 0.5731013779814579
Decision Tree Precision: 0.5640756556740163
Gradient Boosting Machine Precision: 0.6317883302319494
k-Nearest Neighbors Precision: 0.6508510638297873
Logistic Regression Precision: 0.531367490956532
ResNet Precision: 0.5548832897290247
EfficientNet Precision: 0.24547520661157027
-----------------------
CNN Recall: 0.6272727272727273
SVM Recall: 0.6045454545454545
Random Forest Recall: 0.5772727272727273
Decision Tree Recall: 0.5727272727272728
Gradient Boosting Machine Recall: 0.6409090909090909
k-Nearest Neighbors Recall: 0.6545454545454545
Logistic Regression Recall: 0.6136363636363636
ResNet Recall: 0.6409090909090909
EfficientNet Recall: 0.4954545454545455
-----------------------
CNN F1-score: 0.5653481316109108
SVM F1-score: 0.5493479044700212
Random Forest F1-score: 0.5748873543107312
Decision Tree F1-score: 0.5664226682408501
Gradient Boosting Machine F1-score: 0.6227518225735617
k-Nearest Neighbors F1-score: 0.6323060991988145
Logistic Regression F1-score: 0.5638501191348974
ResNet F1-score: 0.594112181524769
EfficientNet F1-score: 0.32829510914617294
```

In the end the results of all the models are compared.

Figure 19: Python Code to Show visualization of the Results:

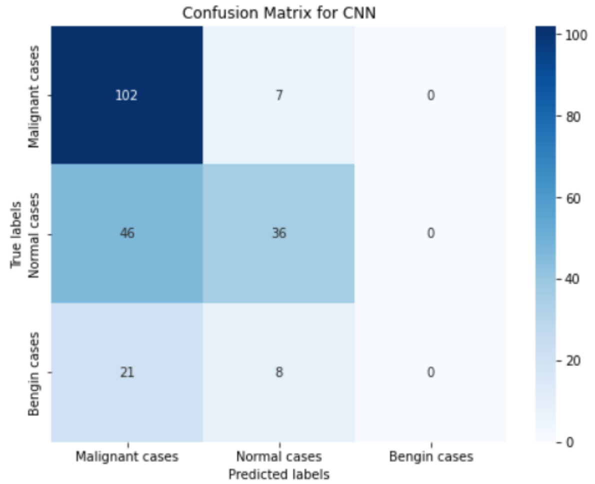
Visual Representation of Results

```
#Accuracy, Precision, Recall, and F1-score of Models
models = ['CNN', 'SVM', 'Random Forest', 'Decision Tree', 'Gradient Boosting Machine', 'k-Nearest Neighbors', 'Logistic Regressic
accuracies = [accuracy, svm_accuracy, rf_accuracy, dt_accuracy, gbm_accuracy, knn_accuracy, logreg_accuracy, fc_accuracy, efficie
precisions = [precision, svm_precision, rf_precision, dt_precision, gbm_precision, knn_precision, logreg_precision, fc_precision,
recalls = [recall, svm_recall, rf_recall, dt_recall, gbm_recall, knn_recall, logreg_recall, fc_recall, efficientnet_recall]
f1_scores = [f1, svm_f1, rf_f1, dt_f1, gbm_f1, knn_f1, logreg_f1, fc_f1, efficientnet_f1]

plt.figure(figsize=(15, 7))
x = np.arange(len(models))
width = 0.2
plt.bar(x - width*1.5, accuracies, width, label='Accuracy')
plt.bar(x - width/2, precisions, width, label='Precision')
plt.bar(x + width/2, recalls, width, label='Recall')
plt.bar(x + width*1.5, f1_scores, width, label='F1-score')
plt.xlabel("Models")
plt.ylabel("Score")
plt.title("Model Evaluation Metrics")
plt.xticks(x, models)
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

Here the evaluation metrices of all the models is compared visually using bar graph.

Figure 20: Python Code to plot Confusion Matrix for CNN Model:

```
# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=label_mapping.keys(), yticklabels=label_mapping.keys())
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()

# Plotting confusion matrix for CNN
plot_confusion_matrix(y_test, y_pred, 'CNN')
```



The Confusion matrix of CNN model is plotted here in this section.

2.2.2 Approch 2

Figure 21: Python Code to import all the required libraries in Approach 2

Approach 2

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
import seaborn as sns
import cv2
import random
import os
import imageio
import plotly.graph_objects as go
import plotly.express as px
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from collections import Counter
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import accuracy_score, recall_score, precision_score, classification_report, confusion_matrix, plot_confusic
from sklearn.model_selection import RandomizedSearchCV, cross_val_score, RepeatedStratifiedKFold
from imblearn.over_sampling import SMOTE
import tensorflow as tf
import tensorflow_addons as tfa
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, BatchNormalization
from keras.applications import resnet
from tensorflow.keras.applications import EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, Efficie
from keras.applications.resnet import ResNet50
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
```

All the necessary libraries required for the second approach are imported in this section.

Figure 22: Python Code to Load the dataset for approach 2:

Loading Data for Approach 2

```
# Set the directory and categories
directory = r"C:\Users\A\Desktop\CT Scan Dataset"
categories = ['Bengin cases', 'Malignant cases', 'Normal cases']

# Analyzing image size variations
size_data = {}

for i in categories:
    path = os.path.join(directory, i)
    class_num = categories.index(i)
    temp_dict = {}
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        height, width, channels = imageio.imread(filepath).shape
        if str(height) + ' x ' + str(width) in temp_dict:
            temp_dict[str(height) + ' x ' + str(width)] += 1
        else:
            temp_dict[str(height) + ' x ' + str(width)] = 1

    size_data[i] = temp_dict

size_data
```

```
{'Bengin cases': {'512 x 512': 120},
 'Malignant cases': {'512 x 512': 501,
  '404 x 511': 1,
  '512 x 801': 28,
  '512 x 623': 31},
 'Normal cases': {'512 x 512': 415, '331 x 506': 1}}
```

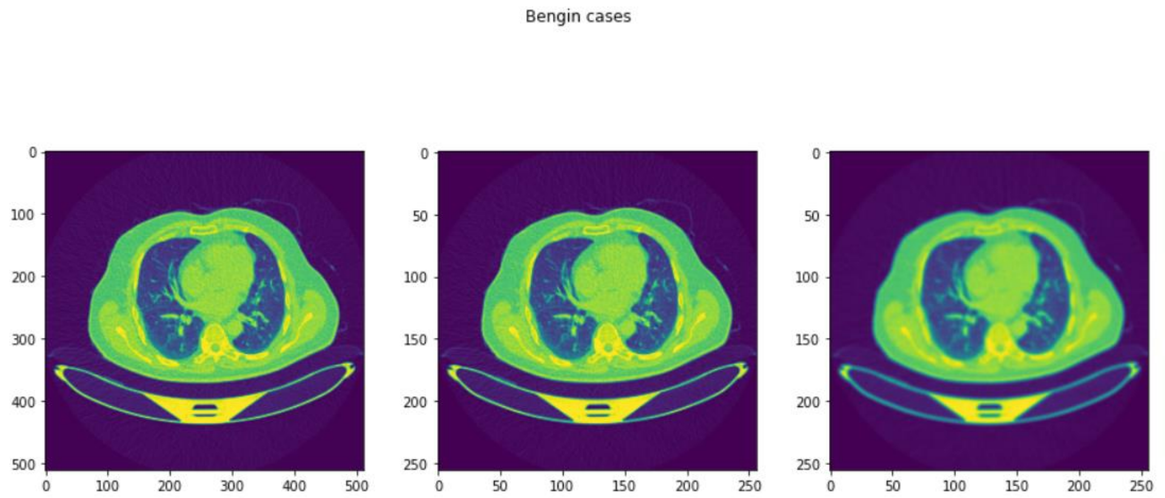
In this part of the code data set directory and categories are set.

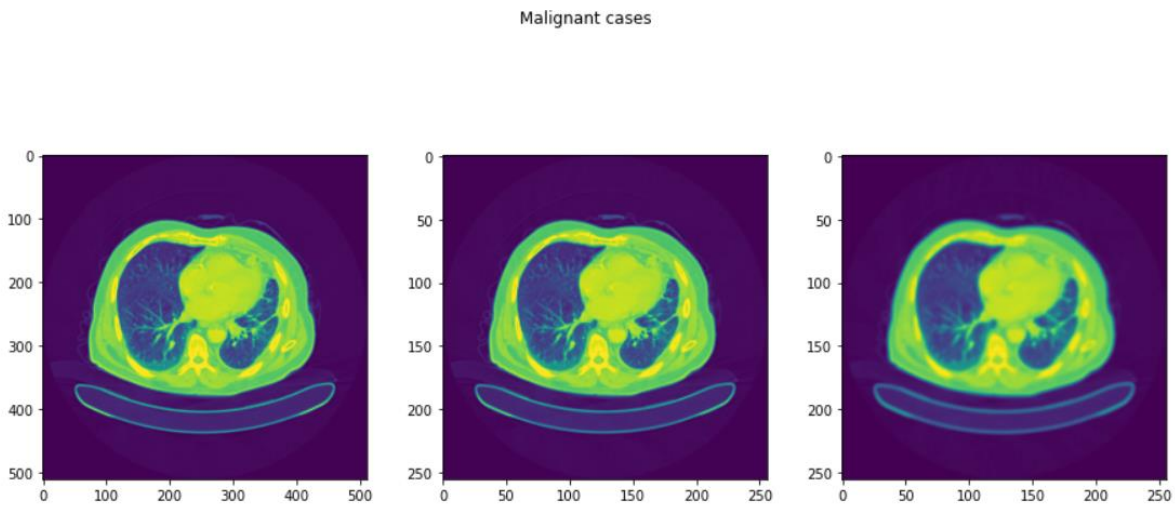
Figure 23: Python Code for preprocessing and visualization of data:

Preprocessing and visualizing images

```python
# Preprocessing and visualization of images
img_size = 256
for i in categories:
    cnt, samples = 0, 3
    fig, ax = plt.subplots(samples, 3, figsize=(15, 15))
    fig.suptitle(i)

    path = os.path.join(directory, i)
    class_num = categories.index(i)
    for curr_cnt, file in enumerate(os.listdir(path)):
        filepath = os.path.join(path, file)
        img = cv2.imread(filepath, 0)

        img0 = cv2.resize(img, (img_size, img_size))

        img1 = cv2.GaussianBlur(img0, (5, 5), 0)

        ax[cnt, 0].imshow(img)
        ax[cnt, 1].imshow(img0)
        ax[cnt, 2].imshow(img1)
        cnt += 1
        if cnt == samples:
            break

plt.show()
```

The preprocessing step is carried out here. The images are resize and GaussianBlur effect is used. And the results are visualized.





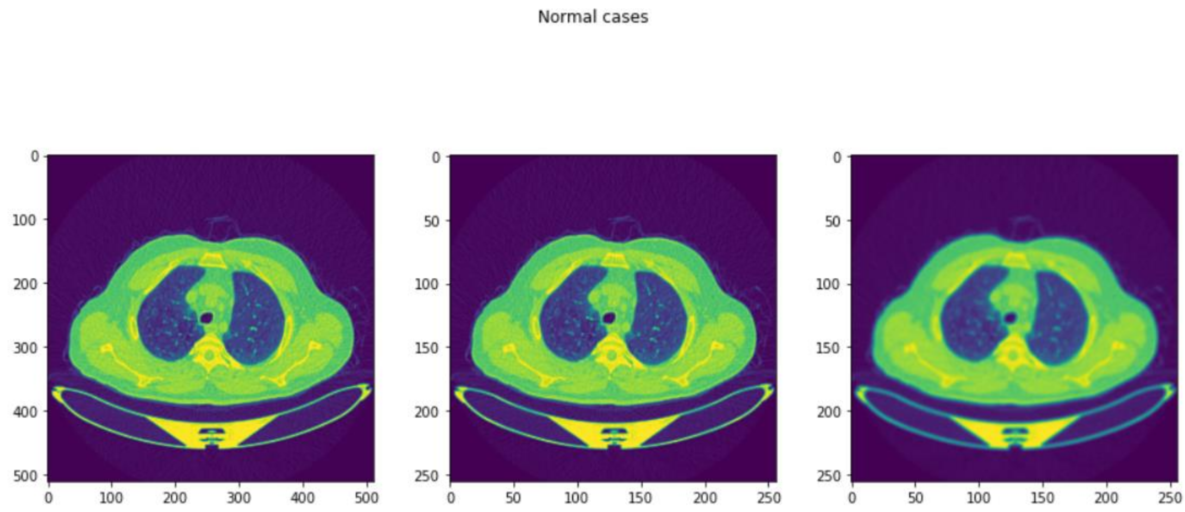


Figure 24: Python code to prepare the code for model building:

Preparing Data for Model Building

```
# Preparing data
data = []
img_size = 256

for i in categories:
    path = os.path.join(directory, i)
    class_num = categories.index(i)
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        img = cv2.imread(filepath, 0)
        # preprocess here
        img = cv2.resize(img, (img_size, img_size))
        data.append([img, class_num])

random.shuffle(data)

X, y = [], []
for feature, label in data:
    X.append(feature)
    y.append(label)

print('X length:', len(X))
print('y counts:', Counter(y))

# normalize
X = np.array(X).reshape(-1, img_size, img_size, 1)
X = X / 255.0
y = np.array(y)
```

```
X length: 1097
y counts: Counter({1: 561, 2: 416, 0: 120})
```

Here the data is prepared and normalised for building the model.

Figure 26: Python Code to split into Train and Test and reshape Data using SMOTE

```python
# Splitting the data into training and validation sets
# Using stratification to maintain the distribution of the target variable in both training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state=10, stratify=y)

# Displaying the distribution of classes in the training and validation sets
print(Counter(y_train), Counter(y_valid))

# Checking the shape of the training data
print(len(X_train), X_train.shape)

# Reshaping the training data to a 2D array in preparation for SMOTE oversampling
X_train = X_train.reshape(X_train.shape[0], img_size*img_size*1)
print(len(X_train), X_train.shape)

# Using SMOTE (Synthetic Minority Over-sampling Technique) to address class imbalance
# This will create synthetic samples for the minority classes
print('Before SMOTE:', Counter(y_train))
smote = SMOTE()
X_train_sampled, y_train_sampled = smote.fit_resample(X_train, y_train)
print('After SMOTE:', Counter(y_train_sampled))

# Reshaping the data back to its original shape after SMOTE oversampling
X_train = X_train.reshape(X_train.shape[0], img_size, img_size, 1)
X_train_sampled = X_train_sampled.reshape(X_train_sampled.shape[0], img_size, img_size, 1)

# Checking the shape of the training data after SMOTE oversampling
print(len(X_train), X_train.shape)
print(len(X_train_sampled), X_train_sampled.shape)
```

```
Counter({1: 420, 2: 312, 0: 90}) Counter({1: 141, 2: 104, 0: 30})
822 (822, 256, 256, 1)
822 (822, 65536)
Before SMOTE: Counter({1: 420, 2: 312, 0: 90})
After SMOTE: Counter({2: 420, 1: 420, 0: 420})
822 (822, 256, 256, 1)
1260 (1260, 256, 256, 1)
```

The dataset is split to train and test and synthetic samples are created for minority classes using SMOTE.

Figure 28: Python Code to Build CNN Model:

Building CNN Model

```python
# Initialize the CNN model using Keras Sequential API
cnn_model = Sequential()

# Add the first convolutional layer with 64 filters of size 3x3
# Using the ReLU activation function
cnn_model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
cnn_model.add(Activation('relu'))

# Add a max-pooling layer to down-sample the feature maps
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

# Add the second convolutional layer with 64 filters of size 3x3
# Using the ReLU activation function directly within the Conv2D layer
cnn_model.add(Conv2D(64, (3, 3), activation='relu'))

# Add another max-pooling layer
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the feature maps into a one-dimensional vector to prepare for the fully connected layers
cnn_model.add(Flatten())

# Add a dense (fully connected) layer with 16 neurons
cnn_model.add(Dense(16))

# Add the output layer with 3 neurons (one for each class)
# Using the softmax activation function to produce a probability distribution
cnn_model.add(Dense(3, activation='softmax'))

# Display a summary of the model's architecture
cnn_model.summary()

# Compile the CNN model specifying the loss function, optimizer, and evaluation metric
cnn_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

The CNN model is built in this part of the code. First convolutional layer with 64 filters is added using the ReLU activation function. Later a max-pooling layer is added to down sample the feature maps.

Again a convolutional layer is added followed by a max-pooling layer. In the end dense layersare added.

```
# Train the CNN model using the training data
# Here, we also specify a batch size, number of epochs, and validation data
history = cnn_model.fit(X_train_sampled, y_train_sampled, batch_size=8, epochs=10, validation_data=(X_valid, y_valid))

# Predict the classes for the validation data
y_pred = cnn_model.predict(X_valid, verbose=1)

# Convert the predicted probabilities into class labels
y_pred_bool = np.argmax(y_pred, axis=1)

# Display a detailed classification report
print(classification_report(y_valid, y_pred_bool))
```

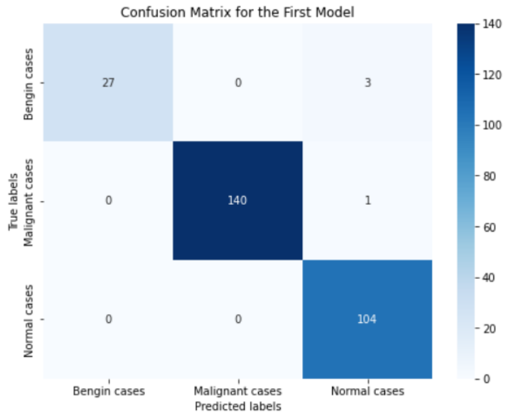
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 64)      640       
                                                                 
 activation (Activation)     (None, 254, 254, 64)      0         
                                                                 
 max_pooling2d (MaxPooling2  (None, 127, 127, 64)      0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 125, 125, 64)      36928     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 246016)            0         
                                                                 
 dense_7 (Dense)             (None, 16)                3936272   
                                                                 
 dense_8 (Dense)             (None, 3)                 51        
                                                                 
=================================================================
Total params: 3973891 (15.16 MB)
Trainable params: 3973891 (15.16 MB)
Non-trainable params: 0 (0.00 Byte)
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        30
           1       1.00      0.99      1.00       141
           2       0.96      1.00      0.98       104

    accuracy                           0.99       275
   macro avg       0.99      0.96      0.97       275
weighted avg       0.99      0.99      0.99       275
```

The model is trained and the classes are predicted. Classification report of the model is fetched.

Figure 29: Python code to plot Confusion Matrix:

Confusion Matrix for the Model

```python
# Confusion matrix for the CNN model
plt.figure(figsize=(8,6))
conf_matrix = confusion_matrix(y_true=y_valid, y_pred=y_pred_bool)
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=categories, yticklabels=categories)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for the First Model')
plt.show()
```



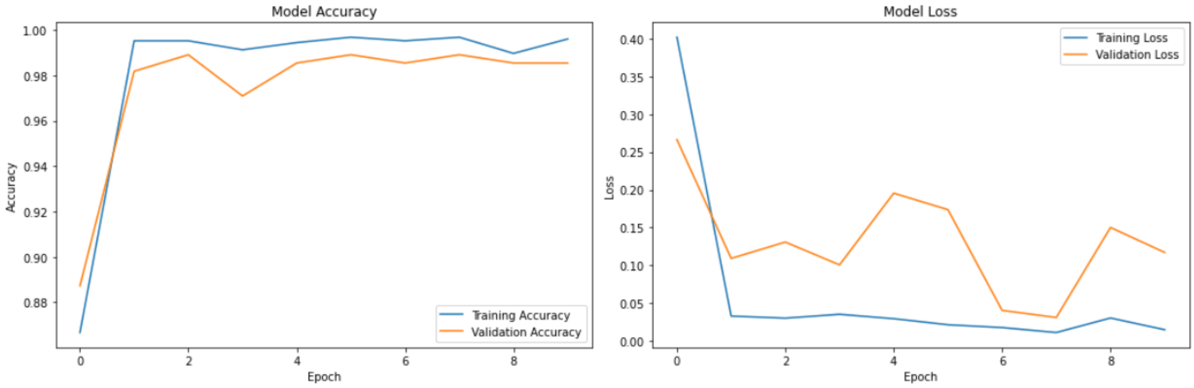
Confusion matrix is generated for the CNN model.

Figure 30: Python Code to Visualize the Results:

Visualization of the Results

```python
# Training and validation accuracy and loss curves
plt.figure(figsize=(15,5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Line graph is printed for the model accuracy and loss is plotted using Matplotlib.


Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
Model Loss
Training Loss
Validation Loss
Loss
Epoch