National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ayush Kumar
Student ID: 21208590

School of Computing
National College of Ireland

Supervisor: Dr. Abid Yaqoob

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ayush Kumar
Student ID:	21208590
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Abid Yaqoob
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	743
Page Count:	13

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:	
Date:	31st January 2024

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

Ayush Kumar
21208590

1 Introduction

Computer vision is an ongoing field of research, scientists and software engineers around the world are trying to optimize and fine-tune deep learning algorithms. In this configuration manual, a summarised outline of instructions is provided to replicate the results of the experiments. This manual discusses about both software and hardware specifications used in research experiments. It helps in understanding of python programming executed in the process of finding desirable results. Follow the instruction guide with each section to explain the programm.

2 System Configuration

In this research experiment, the hardware specification of the system is Windows 11 64-bit operating system, 16 GB RAM, 220 GB of ROM, 4GB NVIDIA GTX 1050 graphic processor and Intel i7 - 7700HQ is utilized. More hardware and operating system specifications are shown in Figure1.

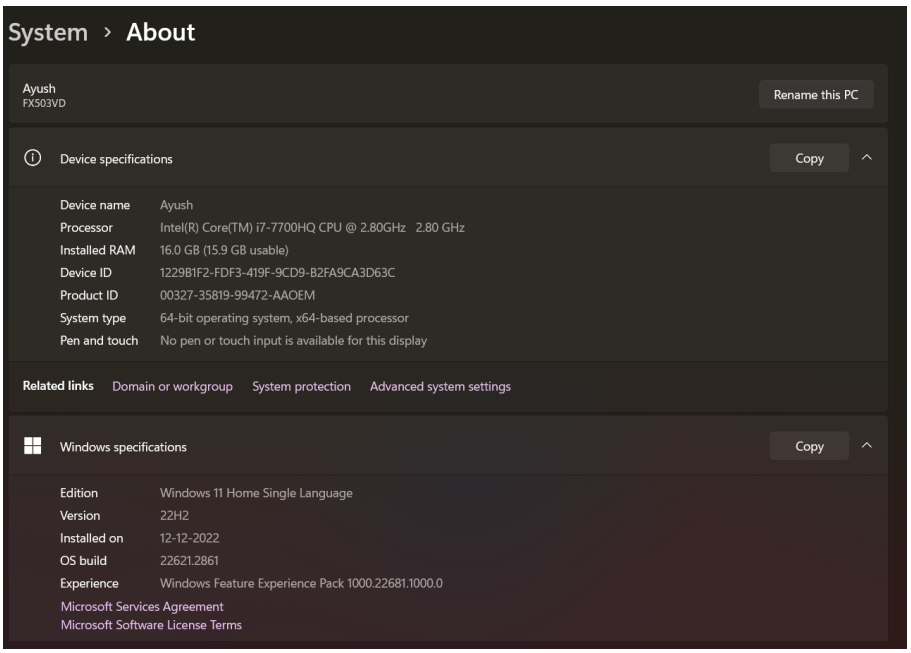

Figure 1: Hardware and Operating system specification

3 Environment Setup

3.1 Anaconda Navigator

In the research experiment python programming language is utilized for the implementation of deep learning model. The programming script is done with the help of Jupyter Notebook which is supported by Anaconda Navigator(present at link[1]) as shown in Figure 2

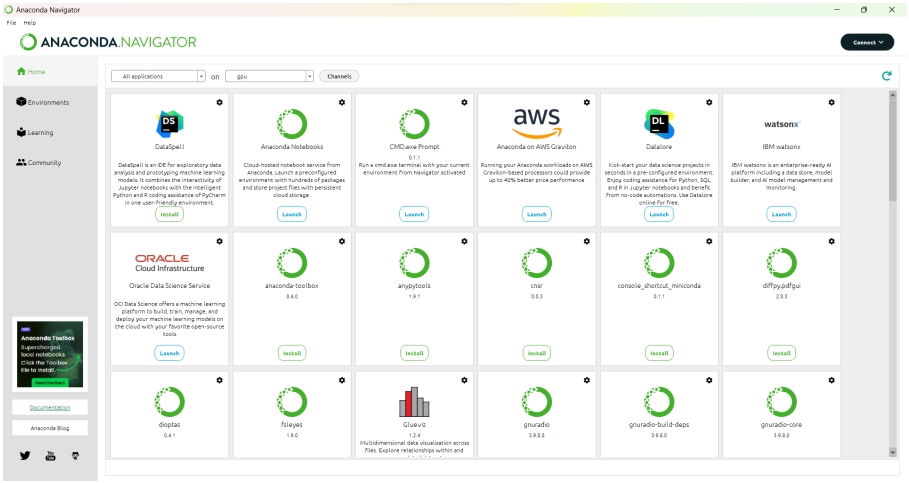

Figure 2: Anaconda Navigator

3.2 Environment

To meet the demand for python programm a new environment is created with the help of the Anaconda base command prompt. Python 3.10 is installed with multiple packages like tensorflow 2.10, cudatoolkit=11.2, cudnn=8.1 and other packages like mathplotlib, seaborn, numpy, pandas and more.

The new environment is selected by activating gpu environment from base as shown in Figure 3. A new environment is created to run research experiments with the help of GPU present in the system for faster execution.

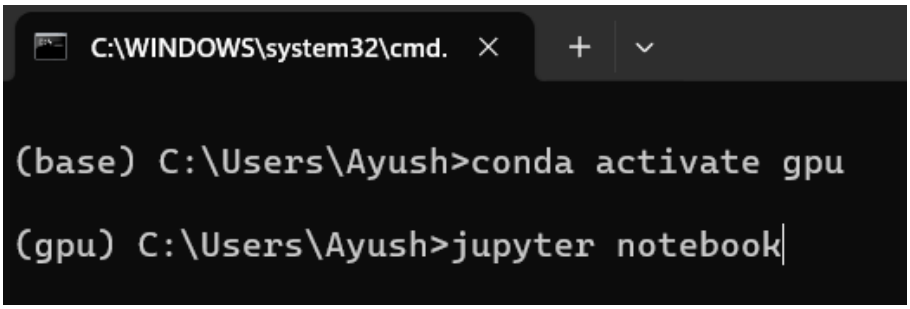

Figure 3: New GPU environment for research

[1] https://www.anaconda.com/download

4 Research Experiment

For a better understanding of the aspect of code, the program code is segmented into sections.

4.1 Importing Libraries

The initial step is to load essential libraries and packages into Python file as shown in Figure 4

Figure 4: Importing Libraries

4.2 GPU Threshold

While working on the research experiment, the GPU used is only 4 GB, hence a threshold is applied of 50% usage. This is done to avoid the issue of Resource exhaustion errors. code used in limiting the usage of GPU is shown in Figure 5.

Figure 5: Normal and Pneumonia x-rays

4.3 Data Loading

The dataset used in this experiment is present on the Kaggle website [2]. The dataset is professionally labeled by Kermany (2018). The zip file of 2.3 GB is stored in the local drive which is loaded into the dataframe as shown in Figure 6

[2] https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

Medical image dataset loading into dataframe

```
train_dir = "chest_xray/train"
test_dir = "chest_xray/test"
val_dir = "chest_xray/val"

print("Train set:\n")
num_pneumonia = len(os.listdir(os.path.join(train_dir, 'PNEUMONIA')))
num_normal = len(os.listdir(os.path.join(train_dir, 'NORMAL')))
print(f"PNEUMONIA={num_pneumonia}")
print(f"NORMAL={num_normal}")
print("**********************************************")
print("Test set:\n")
print(f"PNEUMONIA={len(os.listdir(os.path.join(test_dir, 'PNEUMONIA')))}")
print(f"NORMAL={len(os.listdir(os.path.join(test_dir, 'NORMAL')))}")
print("**********************************************")
print("Validation set:\n")
print(f"PNEUMONIA={len(os.listdir(os.path.join(val_dir, 'PNEUMONIA')))}")
print(f"NORMAL={len(os.listdir(os.path.join(val_dir, 'NORMAL')))}")
```

Figure 6: Loading data into pandas dataframe

4.4 Exploratory data analysis

The code snippet shown in Figure 7, is used to display the image present in the dataset. Few medical images are produced as output to see differences with human eyes.

EDA

```
pneumonia = os.listdir("chest_xray/train/PNEUMONIA")
pneumonia_dir = "chest_xray/train/PNEUMONIA"

plt.figure(figsize=(20, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(pneumonia_dir, pneumonia[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```

Figure 7: frontal chest xray images

In Figure 8, this code snippet is used to check the pixel intensity in X-ray images. As a pneumonia patient chest x-ray has some white fading present in x-ray images. The statistical analysis of image pixels is also produced as output.

```
normal_img = os.listdir("chest_xray/train/NORMAL")[1]
normal_dir = "chest_xray/train/NORMAL"
sample_img = plt.imread(os.path.join(normal_dir, normal_img))
plt.imshow(sample_img, cmap='gray')
plt.colorbar()
plt.title('Normal Chest X Ray Image')

print(f"The dimensions of the image are {sample_img.shape[0]} pixels width and {sample_img.shape[1]} pixels height, one single color channel.")
print(f"The maximum pixel value is {sample_img.max():.4f} and the minimum is {sample_img.min():.4f}")
print(f"The mean value of the pixels is {sample_img.mean():.4f} and the standard deviation is {sample_img.std():.4f}")
```

```
sns.distplot(sample_img.ravel(),
             label=f"Pixel Mean {np.mean(sample_img):.4f} & Standard Deviation {np.std(sample_img):.4f}", kde=False)
plt.legend(loc='upper center')
plt.title('Distribution of Pixel Intensities in Normal X-Ray')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

Figure 8: Normal and Pneumonia x-rays

To check class balance, the Python code is shown in Figure 9. This done to check the biases present in the dataset.

```
weight_for_0 = num_pneumonia / (num_normal + num_pneumonia)
weight_for_1 = num_normal / (num_normal + num_pneumonia)

class_weight = {0: weight_for_0, 1: weight_for_1}

print(f"Weight for class 0: {weight_for_0:.2f}")
print(f"Weight for class 1: {weight_for_1:.2f}")
```

```
labels = ['Pneumonia', 'Normal']
weights = [weight_for_0, weight_for_1]

plt.bar(labels, weights, color=['blue', 'green'])
plt.xlabel('Class')
plt.ylabel('Weight')
plt.title('Class Weights')
plt.show()
```

Figure 9: Normal and Pneumonia x-rays

4.5 Data processing

Using *ImageDataGenerator* function, the data is preprocessed for later use. This experiment is conducted in two phases in phase one no augmentation techniques are applied as shown in Figure 10

Data Preprocessing

```
image_generator = ImageDataGenerator()
```

Figure 10: Phase one without augmentation

In phase 2, data augmentation techniques are applied such as random flipping, zoom,

shear range and splitwise parameters are given to *ImageDataGenrator* function shown in Figure 11

```
image_generator = ImageDataGenerator(rotation_range=20,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    samplewise_center=True,
    samplewise_std_normalization=True)
```

Figure 11: Data Augmentation with parameters

Later data is prepared by providing all training parameters like batch size, shuffle, class mode, target size and seed value. The data is not required to be split into test, train and validation, as it is already segmented into different folders.

```
train = image_generator.flow_from_directory(train_dir,
                                            batch_size=8,
                                            shuffle=True,
                                            class_mode='binary',
                                            target_size=(180, 180),
                                            seed=42)

validation = image_generator.flow_from_directory(val_dir,
                                                 batch_size=1,
                                                 shuffle=True,
                                                 class_mode='binary',
                                                 target_size=(180, 180),
                                                  seed=42)

test = image_generator.flow_from_directory(test_dir,
                                           batch_size=8,
                                           shuffle=True,
                                           class_mode='binary',
                                           target_size=(180, 180),
                                           seed=42)
```

Figure 12: Data Preprocessing

4.6 Model Implementation

4.6.1 ResNet50

Model Implementation

Resnet50

```
resnet_base_model = ResNet50(input_shape=(180,180,3), include_top=False, weights='imagenet')
```

```
resnet_base_model.summary()
```

```
  resnet_model = tf.keras.Sequential([
        resnet_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
resnet_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
r = resnet_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 13: Model Implementation

```
restest =resnet_model.evaluate(test)
rtesta=restest[1] * 100
rtestp=restest[2] * 100
rtestr=restest[3] * 100
rtestl=restest[0]
print("Test Accuracy:", rtesta,"%")
print("Precision:", rtestp,"%")
print("Recall:", rtestr,"%")
print("Loss Value:", rtestl)

restrain = resnet_model.evaluate(train)
rtraina=restrain[1] * 100
rtrainp=restrain[2] * 100
rtrainr=restrain[3] * 100
rtrainl=restrain[0]
print("Train Accuracy:", rtraina,"%")
print("Precision:", rtrainp,"%")
print("Recall:", rtrainr,"%")
print("Loss Value:", rtrainl)
```

```
accuracy = restest[1]
recall =restest[3]
precision =restest[2]


tp = recall * (accuracy + 1) / (precision + recall - 1)
fp = precision * (accuracy + 1) / (precision + recall - 1) - tp
tn = accuracy - tp
fn = 1 - recall - tn
conf_matrix = [[tn, fp],
               [fn, tp]]
f1_scorer = 2 * (precision * recall) / (precision + recall)
print('Dice Score is ',f1_scorer)
# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt='.0f', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
```

Figure 14: Evaluation

4.6.2 EfficeintNetB0

EfficientNetB0

```
model = EfficientNetB0(input_shape=(180, 180, 3), include_top=False, weights='imagenet')
model.summary()
```

```
  eff_model = tf.keras.Sequential([
        model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
eff_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
e = eff_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 15: Model Implementation

```
[ ]: efftest =eff_model.evaluate(test)

     etesta=efftest[1] * 100
     etestp=efftest[2] * 100
     etestr=efftest[3] * 100
     etestl=efftest[0]
     print("Test Accuracy:", etesta,"%")
     print("Precision:", etestp,"%")
     print("Recall:", etestr,"%")
     print("Loss Value:", etestl)

     efftrain = eff_model.evaluate(train)
     etraina=efftrain[1] * 100
     etrainp=efftrain[2] * 100
     etrainr=efftrain[3] * 100
     etrainl=efftrain[0]
     print("Train Accuracy:", etraina,"%")
     print("Precision:", etrainp,"%")
     print("Recall:", etrainr,"%")
     print("Loss Value:", etrainl)
```

```
[ ]: accuracy = efftest[1]
     recall =efftest[3]
     precision =efftest[2]


     tp = recall * (accuracy + 1) / (precision + recall - 1)
     fp = precision * (accuracy + 1) / (precision + recall - 1) - tp
     tn = accuracy - tp
     fn = 1 - recall - tn
     conf_matrix = [[tn, fp],
                    [fn, tp]]
     f1_scoree = 2 * (precision * recall) / (precision + recall)
     print('Dice Score is ',f1_scoree)
     # Plot confusion matrix using seaborn
     sns.heatmap(conf_matrix, annot=True, fmt='.0f', cmap='Blues',
                 xticklabels=['Predicted Negative', 'Predicted Positive'],
                 yticklabels=['Actual Negative', 'Actual Positive'])

     plt.xlabel('Predicted')
     plt.ylabel('Actual')
     plt.title('Confusion Matrix')
     plt.show()
```

Figure 16: Evaluation

4.6.3 VGG16

VGG16

```
[ ]: vgg16_base_model = VGG16(input_shape=(180,180,3),include_top=False,weights='imagenet')
vgg16_base_model.summary()
```

```
[ ]: vgg16_model = tf.keras.Sequential([
        vgg16_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])
```

```
[ ]: opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
vgg16_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
[ ]: vgg = vgg16_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 17: Model Implementation

```
[ ]: vtest =vgg16_model.evaluate(test)

vtesta=vtest[1] * 100
vtestp=vtest[2] * 100
vtestr=vtest[3] * 100
vtestl=vtest[0]
print("Test Accuracy:", vtesta,"%")
print("Precision:", vtestp,"%")
print("Recall:", vtestr,"%")
print("Loss Value:", vtestl)


vtrain = vgg16_model.evaluate(train)
vtraina=vtrain[1] * 100
vtrainp=vtrain[2] * 100
vtrainr=vtrain[3] * 100
vtrainl=vtrain[0]
print("Train Accuracy:", vtraina,"%")
print("Precision:", vtrainp,"%")
print("Recall:", vtrainr,"%")
print("Loss Value:", vtrainl)
```

```
[ ]: accuracy = vtest[1]
recall =vtest[3]
precision =vtest[2]


tp = recall * (accuracy + 1) / (precision + recall - 1)
fp = precision * (accuracy + 1) / (precision + recall - 1) - tp
tn = accuracy - tp
fn = 1 - recall - tn
conf_matrix = [[tn, fp],
               [fn, tp]]
f1_scorev = 2 * (precision * recall) / (precision + recall)
print('Dice Score is ',f1_scorev)
# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt='.0f', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Figure 18: Evaluation

4.6.4 InceptionV3

InceptionV3

```
[ ]: inceptionv3_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(180, 180, 3))

     inceptionv3_model.summary()
```

```
[ ]: inceptionv3_model = tf.keras.Sequential([
         inceptionv3_model,
         GlobalAveragePooling2D(),
         Dense(512, activation="relu"),
         BatchNormalization(),
         Dropout(0.6),
         Dense(128, activation="relu"),
         BatchNormalization(),
         Dropout(0.4),
         Dense(64,activation="relu"),
         BatchNormalization(),
         Dropout(0.3),
         Dense(1,activation="sigmoid")
     ])
```

```
[ ]: opt = tf.keras.optimizers.Adam(learning_rate=0.001)
     METRICS = [
         'accuracy',
         tf.keras.metrics.Precision(name='precision'),
         tf.keras.metrics.Recall(name='recall')
     ]
     inceptionv3_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
[ ]: icp = inceptionv3_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 19: Model Implementation

InceptionV3

```
[ ]: inceptionv3_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(180, 180, 3))

     inceptionv3_model.summary()
```

```
[ ]: inceptionv3_model = tf.keras.Sequential([
         inceptionv3_model,
         GlobalAveragePooling2D(),
         Dense(512, activation="relu"),
         BatchNormalization(),
         Dropout(0.6),
         Dense(128, activation="relu"),
         BatchNormalization(),
         Dropout(0.4),
         Dense(64,activation="relu"),
         BatchNormalization(),
         Dropout(0.3),
         Dense(1,activation="sigmoid")
     ])
```

```
[ ]: opt = tf.keras.optimizers.Adam(learning_rate=0.001)
     METRICS = [
         'accuracy',
         tf.keras.metrics.Precision(name='precision'),
         tf.keras.metrics.Recall(name='recall')
     ]
     inceptionv3_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
[ ]: icp = inceptionv3_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 20: Evaluation

4.6.5 DenseNet121

DenseNet121

```
[ ]:
densenet_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(180, 180, 3))

densenet_model.summary()
```

```
[ ]:
 densenet_model = tf.keras.Sequential([
        densenet_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])
```

```
[ ]:
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
densenet_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
[ ]:
dnm = densenet_model.fit(train,
          epochs=15,
          validation_data=validation,
          class_weight=class_weight)
```

Figure 21: Model Implementation

```
[ ]:
dtest =densenet_model.evaluate(test)

dtesta=dtest[1] * 100
dtestp=dtest[2] * 100
dtestr=dtest[3] * 100
dtestl=dtest[0]
print("Test Accuracy:", dtesta,"%")
print("Precision:", dtestp,"%")
print("Recall:", dtestr,"%")
print("Loss Value:", dtestl)

dtrain = densenet_model.evaluate(train)

dtraina=dtrain[1] * 100
dtrainp=dtrain[2] * 100
dtrainr=dtrain[3] * 100
dtrainl=dtrain[0]
print("Train Accuracy:", dtraina,"%")
print("Precision:", dtrainp,"%")
print("Recall:", dtrainr,"%")
print("Loss Value:", dtrainl)
```

```
[ ]:
accuracy = dtest[1]
recall =dtest[3]
precision =dtest[2]


tp = recall * (accuracy + 1) / (precision + recall - 1)
fp = precision * (accuracy + 1) / (precision + recall - 1) - tp
tn = accuracy - tp
fn = 1 - recall - tn
conf_matrix = [[tn, fp],
               [fn, tp]]
f1_scored = 2 * (precision * recall) / (precision + recall)
print('Dice Score is ',f1_scored)
# Plot confusion matrix using seaborn
sns.heatmap(conf_matrix, annot=True, fmt='.0f', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Figure 22: Evaluation

5 Final Evaluation

Results Evaluation

```
model_names = ['Resnet', 'EfficientNet', 'VGG', 'Inception','DenseNet']
accuracies = [rtesta,etesta,vtesta,itesta,dtesta]
precisions = [rtestp,etestp,vtestp,itestp,dtestp]
dices = [f1_scorer,f1_scoree,f1_scorev,f1_scorei,f1_scored]
recalls = [rtestr,etestr,vtestr,itestr,dtestr]

data = {'Model': model_names, 'Accuracy': accuracies, 'Precision': precisions, 'Dice Score':dices, 'Recall': recalls}
table = pd.DataFrame(data)


print(table)
```

```
K.clear_session()
```

```
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 8))

plt.plot(r.history['accuracy'], label='ResNet50', color='red')

plt.plot(e.history['accuracy'], label='EfficientNetB0', color='yellow')

plt.plot(vgg.history['accuracy'], label='VGG16', color='green')

plt.plot(icp.history['accuracy'], label='InceptionV3', color='blue')

plt.plot(dnm.history['accuracy'], label='DenseNet121', color='black')


plt.legend()
plt.title(' Training Accuracy Evolution ')
plt.xlabel('Epochs')
plt.ylabel('Metrics Value')

plt.show()
```

Figure 23: Normal and Pneumonia x-rays

References

Kermany, D. (2018). Labeled optical coherence tomography (oct) and chest x-ray images for classification, Mendeley Data. Available at: `https://data.mendeley.com/datasets/rscbjbr9sj/2`.