Build AI System For Analyzing Patient Data Project..

APPAS.B

8100211277 03

MECHANICAL ENGINEERING TAMIL

IV-YEAR

ANNA UNIVERSITY BIT CAMPUS
TRICHY

Review

Artificial Intelligence Applications in Smart Healthcare: A Survey

Xian Gao [1,†], Peixiong He [2,‡], Yi Zhou [1,*,†] and Xiao Qin [2,*,‡]

[1] TSYS School of Computer Science, Columbus State University, Columbus, GA 31907, USA; gao_xian@students.columbusstate.edu
[2] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA; pzh0029@auburn.edu
* Correspondence: yi_zhou@columbusstate.edu (Y.Z.); xqin@auburn.edu (X.Q.); Tel.: +1-706-507-8048 (Y.Z.)
† These authors contributed equally to this work.
‡ These authors contributed equally to this work.

Abstract: The rapid development of AI technology in recent years has led to its widespread use in daily life, where it plays an increasingly important role. In healthcare, AI has been integrated into the field to develop the new domain of smart healthcare. In smart healthcare, opportunities and challenges coexist. This article provides a comprehensive overview of past developments and recent progress in this area. First, we summarize the definition and characteristics of smart healthcare. Second, we explore the opportunities that AI technology brings to the smart healthcare field from a macro perspective. Third, we categorize specific AI applications in smart healthcare into ten domains and discuss their technological foundations individually. Finally, we identify ten key challenges these applications face and discuss the existing solutions for each.

Keywords: artificial intelligence; smart healthcare; real-world application

Citation: Gao, X.; He, P.; Zhou, Y.; Qin, X. Artificial Intelligence Applications in Smart Healthcare: A Survey. *Future Internet* **2024**, *16*, 308. https://doi.org/10.3390/fi16090308

Academic Editor: Petros Patias

Received: 11 July 2024
Revised: 13 August 2024
Accepted: 22 August 2024
Published: 27 August 2024

1. Introduction

In this age of information, massive amounts of data are continuously generated across various fields. Concurrently hardware technology has made it possible to process these data and derive valuable insights, which has fueled the rapid development of AI across numerous domains. Particularly in the healthcare field, people's pursuit of high-quality and efficient healthcare, along with the pressure on healthcare resources due to the growing population, has spurred the widespread integration of AI into healthcare. This integration has resulted in applications such as disease prediction and prevention, diagnostic imaging, and personalized treatment plans. The field encompassing these applications is smart healthcare, which originated from the concept of 'Smart Planet' proposed by IBM (Armonk, NY, USA) in 2009 [1]. Behind various types of AI applications in this field lies a common essential idea: after data are collected, ML (DL) techniques interpret them and present predictive analytics [2].

Smart healthcare differs from traditional healthcare because traditional healthcare follows a specialist-centered approach, while smart healthcare adopts a patient-centered approach by integrating AI technology [3]. In this approach, patients' needs, experience, and engagement are placed at the core of medical services through the incorporation of modern and intelligent healthcare solutions, thereby building a smart healthcare system.

There are several successful commercial implementations of smart healthcare. Merative L.P. provides smart healthcare products and services to nine of the top ten US hospitals [4]. Tempus collaborates with over 50% of all academic medical centers in the US [5]. Aidoc analyzes 2 million patients each month [6]. PathAI is utilized by 90% of the top 15 BioPharma companies to transform pathology [7]. Consequently, much research has focused on the applications and challenges of smart healthcare. Some research concludes

INTRODUCTION

In 2019, a deadly illness that causes a pandemic have been identified. Coronavirus or commonly known as Covid-19 is a disease that affect the respiratory system of the patient. Coronavirus disease (COVID-19) is an infectious disease caused by the SARS-CoV-2 virus [1]. A healthy person that are infected by the Covid-19 are supposedly to only symptoms such as fever, cough, fatigue and most noticeable, lost sense of taste and smell. The patients are recommended to seek medical attention to confirm the Covid-19 diagnosis when they lost their sense of taste and smell. Even though, the symptoms that are displayed are mild and less severe, patients that have contracted the disease could resulted in fatality. Older people and those with underlying medical conditions like cardiovascular disease, diabetes, chronic respiratory disease, or cancer are more likely to develop serious illness [1]. The symptoms below are addressed by the World Health Organization (WHO) for the civilians to take notice of whether they have contracted the disease.

Most common symptoms:

- fever
- cough
- tiredness
- loss of taste or smell.
- Less common symptoms:

Less common symptoms:

- sore throat
- headache
- aches and pains
- diarrhoea
- a rash on skin, or discolouration of fingers or toes
- red or irritated eyes.

Serious symptoms:

- difficulty breathing or shortness of breath
- loss of speech or mobility, or confusion
- chest pain.

The serious symptoms are the signs of the disease have progressed further into the respiratory system, mostly the lung. COVID-19 can cause lung complications such as pneumonia and, in the most severe cases, acute respiratory distress syndrome, or ARDS. Sepsis, another possible complication of COVID-19, can also cause lasting harm to the lungs and other organs [2]. Therefore, the lungs of Covid-19 patients have been X-Rayed to gain better understanding of this disease.

Technically, there are five stages of the Covid-19 progression. For starters, positive COVID-19 patients are categorised based on their symptoms and management plans are tailored according to their clinical stage as below; [2]

Clinical stage		Condition of COVID-19 Patients
MILD	1	Asymptomatic
	2	Symptomatic, No Pneumonia
	3	Symptomatic, Pneumonia
Clinical stage		**Condition of COVID-19 Patients**
SEVERE	4	Symptomatic, Pneumonia, Requiring supplemental oxygen
	5	Symptomatic, No Pneumonia

The stages of lung damages caused by this disease are roughly split into mild, moderate, and severe. The moderate phase has similar symptoms as pneumonia, whereas the severe phase is difficult to distinguish from pulmonary cancer [3]. It also can be classified into Early Stage, Normal Stage and Severe Stage.

- Early Stage, categorized as the same as the first clinical stage; asymptomatic.
- Normal Stage, categorized as the same as the third clinical stage; resulting in symptomatic and suffers from pneumonia.
- Severe Stage, categorized as the fourth clinical stage; symptomatic, suffer from pneumonia and require oxygen mask.

The X-Ray images of lungs have been classified based on the categorization mentioned above.

METHODOLOGY

The Artificial Intelligence are modelled, and the programming are executed by using Google Collab. The data of X-Ray images are stored in a cloud storage (Google Drive) due to easier modification and accessibility by Google Collab. There are 600 images as data that have been given, 200 images each. The data are stored in the Google Drive and each of the class have been divided into corresponding subdirectories. The data images have been converted to PNG format for better images quality, as it is to classify medical images. Precision is prioritized. The programme and explanation are as below:

1. Library Import of TensorFlow Keras

This code is to install the Tensorflow Keras, a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np
import os
import seaborn as sns
from sklearn.metrics import confusion_matrix , classification_report
import pandas as pd
from tensorflow.keras.callbacks import
EarlyStopping,ReduceLROnPlateau
from google.colab import drive
drive.mount('/content/drive')
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```
import tensorflow as tf
```

```
from tensorflow.keras import models,layer
```

- TensorFlow: An open-source library usually used for machine learning.
- Keras: High-level API in TensorFlow to build and train models.
    - models: Used to create and manipulate neural network models.
    - layers: Used to add layers to a model.

```
import matplotlib.pyplot as plt
```

- matplotlib.pyplot: For plotting graphs and visualizations.

```
import numpy as np
```

- numpy: For numerical operations.

```
import os
```

- os: For interacting with the operating system, such as file and directory operations

```
import seaborn as sns
```

- seaborn: For making statistical graphics

```
from sklearn.metrics
```

- sklearn.metrics: For performance metrics, particularly confusion matrix and classification report.

```
import confusion_matrix, classification_report
```

- sklearn.metrics: For performance metrics, particularly confusion matrix and classification report.
    - confusion matrix: to plot the confusion matrix
    - classification report: to give the classification report that will show the accuracy and others analysis data

```
import pandas as pd
```

- pandas: For data manipulation and analysis.

```
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
```

- tensorflow.keras.callbacks: For callbacks function
    - EarlyStopping: Keras callback to stop training when a monitored metric has stopped improving.
    - ReduceLROnPlateau: Keras callback to reduce learning rate when a metric has stopped improving.

```
from google.colab import drive

drive.mount('/content/drive')
```

- google.colab import drive: module for google colab to interface with the google drive
- drive.mount('/content./drive'): to access the file that are stored in the google drive

```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

- This sets an environment variable to avoid conflicts caused by the Keras library trying to load multiple instances of certain libraries. This is a workaround to handle a specific issue related to Intel's MKL library.

2. Defining the Data Parameters

The data that are uploaded on Google Drive are needed to be defined so that the model will be able to read the images properly and clearly.

```
BATCH_SIZE = 20
IMAGE_WIDTH = 300
IMAGE_HEIGHT = 300
CHANNELS= 3
EPOCHS= 200
```

- BATCH_SIZE = 20: define the number of samples per iteration, passing through the network one at a time during training stage
- IMAGE_WIDTH = 300: define the image width
- IMAGE_HEIGHT = 300: define the image height
- CHANNELS= 3: RGB colour identification; the image are in RGB format
- EPOCHS= 200: the entire data will be trained for 200 times to complete the training stage

3. Images Acquisition from Google Drive

The images are needed to be extracted from the Google Drive and is generated into the model. The data is also needed to be separated in different folder that represents each class.

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes',
    seed = 42,
    image_size = (IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size = BATCH_SIZE,
    class_names=['Early', 'Normal', 'Severe']
)
class_names = dataset.class_names
class_names
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(

'/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes',

seed = 42,

image_size = (IMAGE_WIDTH, IMAGE_HEIGHT),

batch_size = BATCH_SIZE,

class_names=['Early','Normal','Severe']

)
```

4. Defining the Training Dataset, Validation Dataset and Testing Dataset

   The dataset is needed to separate into three datasets: Training Dataset, Validation Dataset and Testing Dataset. The partition for the datasets is as follows: Testing Dataset = 80%, Validation Dataset = 10%, Testing Dataset = 10%.

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1,
test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed = 42)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

`def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):`

- `def get_dataset_partition_tf` is the function code to separate the data
  - `ds` is the tensorflow data that are to be separated
  - `train_split=0.8 is` to separate the training data by 80% of the total data
  - `val_split=0.1` is to separate the validation data by 10% of the total data
  - `test_split=0.1 is` to separate the testing data by 10% of the total data
  - `shuffle=True` is to shuffle the data before splitting
  - `shuffle_size=10000` is the buffer size for shuffling

`ds_size = len(ds)`

- The size of the dataset (ds_size) is determined using len(ds).

```
if shuffle:
        ds = ds.shuffle(shuffle_size, seed = 42)


    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)


    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)


return train_ds, val_ds, test_ds
```

- if shuffle: if shuffle is True, the dataset is shuffled using the shuffle method with the specified shuffle_size and a fixed seed (seed=42) for reproducibility.
    - `train_size` is the number of elements in the training set, computed as `train_split * ds_size`.
    - `val_size` is the number of elements in the validation set, computed as `val_split * ds_size`.
    - `train_ds` is created by taking the first `train_size` elements from the dataset.
    - `val_ds` is created by skipping the first `train_size` elements and taking the next `val_size` elements.
    - `test_ds` is created by skipping the first `train_size` elements and the next `val_size` elements, taking the remaining elements.
- `Return train_ds, val_ds, test_ds:` The function returns the three partitions: train_ds, val_ds, and test_ds.

5. Data Sample Calculation and Verification

The data sample are calculated and verified to avoid data misconfigure or misloading during the data acquisition. This is a debug process.

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

train_size = 0
for batch in train_ds:
    train_size += batch[0].shape[0]
val_size = 0
for batch in val_ds:
    val_size += batch[0].shape[0]
test_size = 0
for batch in test_ds:
    test_size += batch[0].shape[0]

print(f"Train size: {train_size}")
print(f"Validation size: {val_size}")
print(f"Test size: {test_size}")
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

- `train_ds, val_ds, test_ds` are the dataset partition that are defined by tensorflow partition function

```
train_size = 0
```

```
for batch in train_ds:
```

```
    train_size += batch[0].shape[0]
```

```
val_size = 0
```

```
for batch in val_ds:
```

```
    val_size += batch[0].shape[0]
```

```
test_size = 0
```

```
for batch in test_ds:
```

```
    test_size += batch[0].shape[0]
```

- this code is to count the batch of training samples, validation samples and testing sample.

```
print(f"Train size: {train_size}")
```

```
print(f"Validation size: {val_size}")
```

```
print(f"Test size: {test_size}")
```

- to display and verify the samples of each dataset

6. Data Modification and Enhancement

The given code is enhancing the performance of the training, validation, and test datasets by adding caching, shuffling, and prefetching to the data pipeline. This will increase the model accuracy and predict even better.

```
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOT
UNE)
val_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUN
E)
test_ds =
test_ds.cache().shuffle(1000,reshuffle_each_iteration=False).pref
etch(buffer_size=tf.data.AUTOTUNE)
```

- Caching stores the dataset in memory after the first epoch, which can significantly speed up data loading in subsequent epochs, especially if the dataset fits into memory.
- Shuffling the dataset ensures that the data is mixed, which helps prevent the model from learning the order of the data. This is particularly important for training datasets to ensure that batches are varied and representative.
    - `1000`: The buffer size for shuffling. This means the dataset will maintain a buffer of 1000 elements and randomly sample from this buffer.
    - `reshuffle_each_iteration=False` (for the test dataset): This ensures that the test dataset is shuffled only once and not reshuffled in every iteration, which is useful for consistent evaluation during model validation.
- Prefetching overlaps the data preprocessing and model execution, which helps improve the performance of the training process by ensuring that data is ready and waiting when the model needs it.
    - `buffer_size=tf.data.AUTOTUNE`: This allows TensorFlow to dynamically tune the prefetch buffer size for optimal performance.

7. Resizing and Rescaling the Data

This Sequential model can be used as a preprocessing step in your data pipeline. When you pass an image through resize_and_rescale, it will first resize the image to the specified dimensions and then rescale the pixel values to the [0, 1] range.

```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_WIDTH,
IMAGE_HEIGHT),
  layers.experimental.preprocessing.Rescaling(1.0/255),
])
```

- `tf.keras.Sequential` is a linear stack of layers where each layer has exactly one input tensor and one output tensor.
- `layers.experimental.preprocessing.Resizing(IMAGE_WIDTH, IMAGE_HEIGHT)` is a preprocessing layer that resizes the input images to the specified width (`IMAGE_WIDTH`) and height (`IMAGE_HEIGHT`). This is useful for standardizing the input image sizes before feeding them into a neural network.
- `layers.experimental.preprocessing.Rescaling(1.0/255)` is a preprocessing layer that rescales the pixel values of the images. Since the original pixel values are typically in the range [0, 255], dividing by 255 scales these values to the range [0, 1]. This normalization helps the neural network train more effectively, as it ensures the input values are on a similar scale.

8. Model Specification and Architecture

This stage is when the model is design with the architecture of Convolutional Neural Network (CNN) by utilizing Keras imported from TensorFlow. In this section, all of the hyperparameters are defined accordingly to increase the accuracy and precision of the model.

```
input_shape = (BATCH_SIZE, IMAGE_WIDTH, IMAGE_HEIGHT, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape = input_shape)
```

```
input_shape = (BATCH_SIZE, IMAGE_WIDTH, IMAGE_HEIGHT, CHANNELS)
n_classes = 3
```

- `BATCH_SIZE`: Number of images in each batch.
- `IMAGE_WIDTH` and `IMAGE_HEIGHT`: Dimensions to which each image will be resized.
- `CHANNELS`: Number of color channels in the image (e.g., 3 for RGB images).

```
n_classes = 3
```

- The number of output classes for the classification task.

9. Model Execution (Training Stage)

This section explains the execution of the model, the verification of the model beforehand and the additional model function during training. Additional function that are added are the EarlyStopping function and ReduceLROnPlateau that will help in preserving the accuracy of the model and avoiding overfitting.

```
model.summary()

model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logit
s=False),
    metrics=['accuracy']
)
```

```
early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
patience = 3, verbose=1,factor=0.3, min_lr=0.0000001)
```

```
callbacks_list = [ early, learning_rate_reduction]

history = model.fit(
    train_ds,
```

```
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS, #epochs
    callbacks = callbacks_list,
)
```

`model.summary()`

- This method prints a summary of the model, displaying the architecture, including each layer's type, output shape, and number of parameters.

```
model.compile(

    optimizer='adam',


loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Fa
lse),

    metrics=['accuracy']

)
```

- `'adam'` is an adaptive learning rate optimization algorithm that's widely used for training deep learning models. Also known as Adaptive Moment Estimation
- `SparseCategoricalCrossentropy(from_logits=False)` is used for multi-class classification problems where the target labels are integers. `from_logits=False`

10. Model Analysis (Accuracy and Loss Graph)

After the training have been completed, the model will be analysed based on its accuracy and the performance will be determined. In this section, the loss and accuracy are assessed and plotted on a graph to gain better understanding of the accuracy and loss during training.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(acc) + 1), acc, label='Training Accuracy')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation
Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, len(loss) + 1), loss, label='Training Loss')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation
Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

- `history.history['accuracy']`: Retrieves the accuracy values for each epoch during training.
- `acc`: A list of accuracy values for each epoch on the training data.
- `history.history['val_accuracy']`: Retrieves the accuracy values for each epoch on the validation data.
- `val_acc`: A list of accuracy values for each epoch on the validation data.
- **`history.history['loss']`**: Retrieves the loss values for each epoch during training.

11. Model Analysis (Accuracy, Confusion Matrix and Classification Report)

The model is also analysed with the display of accuracy and confusion matrix display. The confusion matrix functions as to show how the data are categorized according to the classes that are defined. The classification is to show additional analysis data that will determine the model efficiency and precision.

```
scores = model.evaluate(test_ds)
print("Testing Accuracy: %.2f%%\n" % (scores[1]*100))

y_true = np.concatenate([labels_batch.numpy() for _, labels_batch
in test_ds])

y_test = model.predict(test_ds)
y_pred = np.argmax(y_test, axis=1)
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.xticks(ticks=range(len(class_names)), labels=class_names)
plt.yticks(ticks=range(len(class_names)), labels=class_names)
plt.show()

print(classification_report(y_true,y_pred,target_names
=['Early','Normal','Severe']))
```

```
scores = model.evaluate(test_ds)
print("Testing Accuracy: %.2f%%\n" % (scores[1]*100))
```

- `scores = model.evaluate(test_ds)`: Evaluates the model on the test dataset (`test_ds`). Returns a list containing loss and accuracy.
- `print("Testing Accuracy: %.2f%%\n" % (scores[1]*100))`: Prints the testing accuracy as a percentage.

RESULT

The trained model has been executed and analysed. Due to the EarlyStopping function, the training stage requires less than half the time needed to run all the epoch. 40 epoch are enough as the model has stabilised and reached the peak accuracy. The learning rate is updated only once by the ReduceLROnPlateau function that causes the training stage to run smoothly and faster.

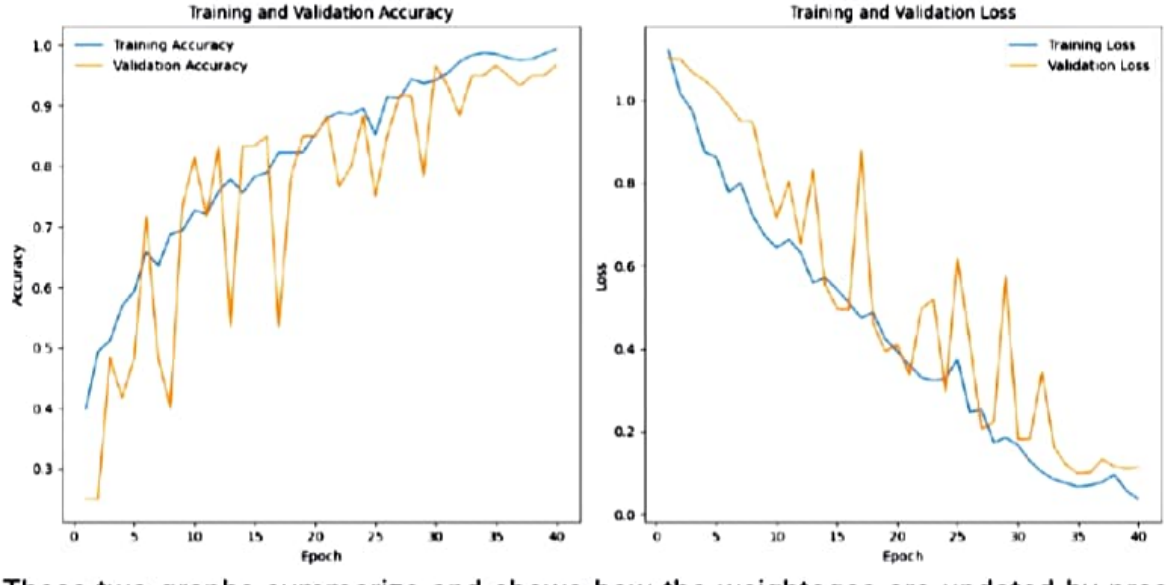
The analysis data is compiled and shown below:

1. Accuracy Display
   3/3 [==============================] - 4s 22ms/step - loss: 0.0215 - accuracy: 1.0000
   Testing Accuracy: 100.00%

   The accuracy is calculated with the formula $Accuracy = \frac{Number\ of\ Correct\ Prediction}{Total\ Number\ of\ Prediction}$.
   However, in this case, the accuracy is calculated by the programming instead to gain correct calculation as to diminish human error during calculation.

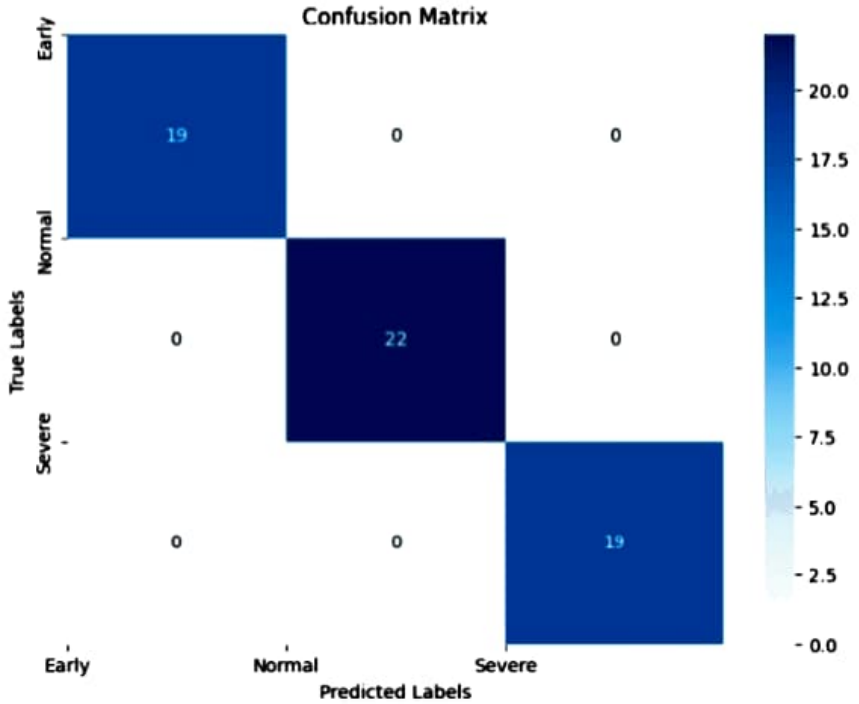
   The result of the model is 100% accurate, giving no error during testing stage; in other words, all the testing datasets are classified correctly according to their own class.

2. Training and Validation: Accuracy and Loss



These two graphs summarize and shows how the weightages are updated by presenting accuracy and loss of both training and validation stage. By observing both graphs, it is concluded that accuracy are in reciprocal relationship. As the accuracy increases, the loss decreases.

3. Confusion Matrix



The confusion matrix is plotted with the shade of blue for easier reading. The y-axis represents the true value for each of the class and the x-axis represents the predicted value for each of the class. Through observation, all datasets are predicted accurately as there is no other value but the diagonal value. The data also can be summarized as below:

- Early Stage: 19 data has been classified correctly and 0 data is classified as the other two class.
- Normal Stage: 22 data has been classified correctly and 0 data is classified as the other two class
- Severe Stage: 19 data has been classified correctly and 0 data is classified as the other two class.

4. Classification Report

A classification report is a report showing the information that are needed to be know during AI model analysis. The classification report is as below:

	Precision	recall	f-1 score	support
Early	1.0	1.0	1.0	19
Normal	1.0	1.0	1.0	22
Severe	1.0	1.0	1.0	19
Accuracy			1.0	60
Macro avg	1.0	1.0	1.0	60
Weighted avg	1.0	1.0	1.0	60

The classification report can be interpreted by category:

For the x-axis;

- Precision
  - The Early Stage are trained with 100% precision, inferring that all the Early sample are classified as Early sample
  - The Normal Stage are trained with 100% precision, inferring that all the Normal sample are classified as Normal sample
  - The Severe Stage are trained with 100% precision, inferring that all the Severe sample are classified as Severe sample
- recall
  - The Early Stage are trained with 100% recall, inferring that all the actual Early sample are classified as Early Stage
  - The Normal Stage are trained with 100% recall, inferring that all the actual Normal sample are classified as Normal Stage
  - The Severe Stage are trained with 100% recall, inferring that all the actual Severe Stage are classified as Severe Stage
- F-1 Score (weighted average of precision and recall)
  - 100% indicate perfect precision and recall for Early Stage
  - 100% indicate perfect precision and recall for Normal Stage
  - 100% indicate perfect precision and recall for Severe Stage
- Support
  - Number of actual occurrences for Early is 19
  - Number of actual occurrences for Normal is 22
  - Number of actual occurrences for Severe is 19

DISCUSSION

Based on the model analysis data, a few topics can be discussed, including the accuracy and the support value of each class. Referring to the classification report, we can see that the overall performance is 100% for each class, however there is an anomaly in the support value. The support value refers to the total data of that are set for testing the model. In this case, the value is 60 sample data, 20 data samples each class. The data that are instanced in the class are incorrect as some of the data from Early and Severe have been classified correctly as the Normal. Therefore, revision are made and the programme has been modified with the objective to correct the value of the training of each class. The revision are as below:

Data Assertion:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/EMJ37403/Deep Learning/Project 1/Classes',
    seed = 42,
    image_size = (IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size = BATCH_SIZE,
    class_names=['Early', 'Normal', 'Severe']
)
dataset_dir = '/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes'
class_names = ['Early', 'Normal', 'Severe']
class_names = dataset.class_names
datasets = {class_name: load_class_data(class_name) for class_name in
class_names}

def split_dataset(ds, train_split=0.8, val_split=0.1, test_split=0.1):
    ds_size = len(list(ds))
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    ds = ds.shuffle(ds_size, seed=42)
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds


def get_dataset_partitions_tf(ds, train_split=0.6, val_split=0.2,
test_split=0.2, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
```

```
        ds = ds.shuffle(shuffle_size, seed = 42)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
splits = {class_name: get_dataset_partitions_tf(datasets[class_name])
for class_name in class_names}

# Combine class-specific datasets into a single dataset for each split
train_datasets = [splits[class_name][0] for class_name in class_names]
val_datasets = [splits[class_name][1] for class_name in class_names]
test_datasets = [splits[class_name][2] for class_name in class_names]

# Function to load images from paths
def load_image(image_path, label):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [IMAGE_WIDTH, IMAGE_HEIGHT])
    return image, label

# Combine the datasets and map the image loading function
train_ds =
tf.data.experimental.sample_from_datasets(train_datasets).map(load_imag
e)
val_ds =
tf.data.experimental.sample_from_datasets(val_datasets).map(load_image)
test_ds =
tf.data.experimental.sample_from_datasets(test_datasets).map(load_image
)
```

This partial code is to assert the programme that each class have 20 data images as the testing dataset. The dataset can be verified by the debug code that are as below:

Debug Code:

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

train_size = 0
for batch in train_ds:
    train_size += batch[0].shape[0]
val_size = 0
for batch in val_ds:
    val_size += batch[0].shape[0]
test_size = 0
```

```
for batch in test_ds:
    test_size += batch[0].shape[0]

print(f"Train size: {train_size}")
print(f"Validation size: {val_size}")
print(f"Test size: {test_size}")
# Path to the dataset directory
dataset_dir = '/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes'

# Get the list of all files and directories
all_classes = os.listdir(dataset_dir)

# Initialize a Counter to keep track of class counts in the test
dataset
test_class_counts = Counter()

# Iterate over the test dataset
for images, labels in test_ds:
    test_class_counts.update([labels.numpy()])

# Map class indices to class names
class_names = dataset.class_names
test_class_counts_named = {class_names[k]: v for k, v in
test_class_counts.items()}

print("Test class counts:", test_class_counts_named)
```

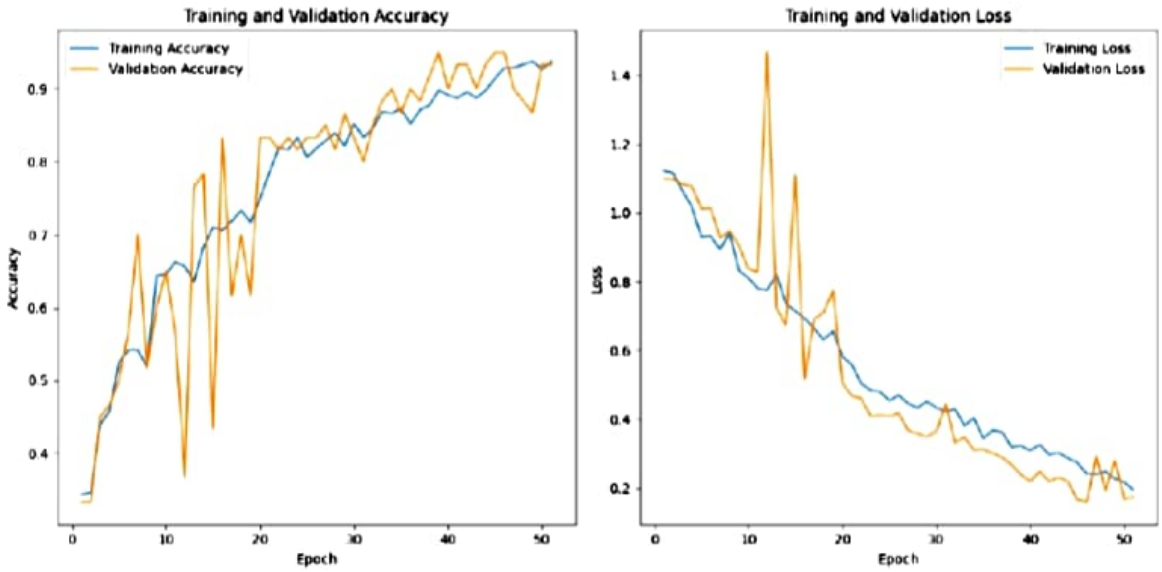
And the outputs are as follows:

```
Train size: 467
Validation size: 60
Test size: 60
Test class counts: {'Early': 20, 'Normal': 20, 'Severe': 20}
```

The output shows that the number of data that are calculated and set as the training dataset, validation dataset and testing dataset are correct and accordingly. This prevents the programme to evaluate and classify correctly of the dataset in the wrong class. Therefore, that are compiled and executed given new results.

New Results:

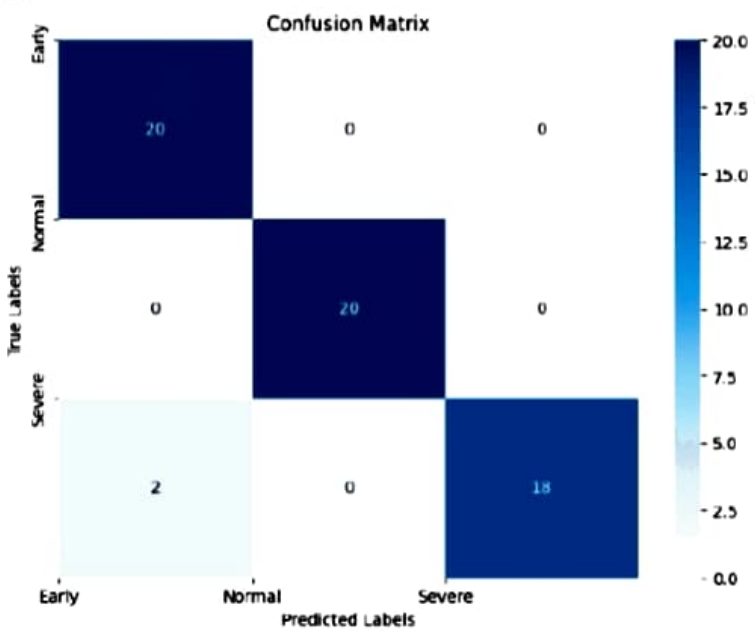
Accuracy and Loss Graph



Accuracy

```
3/3 [==============================] - 2s 646ms/step - loss: 0.1372 -
accuracy: 0.9667
Testing Accuracy: 96.67%
```

Confusion Matrix



APPENDIX

Full Code:

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np
import os
import seaborn as sns
from sklearn.metrics import confusion_matrix , classification_report
import pandas as pd
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
from google.colab import drive
drive.mount('/content/drive')
os.environ['KMP_DUPLICATE_LIB_OK']='True'
from collections import Counter

BATCH_SIZE = 20
IMAGE_WIDTH = 300
IMAGE_HEIGHT = 300
CHANNELS= 3
EPOCHS= 200

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/EMJ37403/Deep Learning/Project 1/Classes',
    seed = 42,
    image_size = (IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size = BATCH_SIZE,
    class_names=['Early', 'Normal', 'Severe']
)
dataset_dir = '/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes'
class_names = ['Early', 'Normal', 'Severe']
class_names = dataset.class_names
datasets = {class_name: load_class_data(class_name) for class_name in
class_names}

def split_dataset(ds, train_split=0.8, val_split=0.1, test_split=0.1):
    ds_size = len(list(ds))
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    ds = ds.shuffle(ds_size, seed=42)
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
```

```
    return train_ds, val_ds, test_ds


def get_dataset_partitions_tf(ds, train_split=0.6, val_split=0.2,
test_split=0.2, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed = 42)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
splits = {class_name: get_dataset_partitions_tf(datasets[class_name])
for class_name in class_names}


train_datasets = [splits[class_name][0] for class_name in class_names]
val_datasets = [splits[class_name][1] for class_name in class_names]
test_datasets = [splits[class_name][2] for class_name in class_names]

def load_image(image_path, label):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [IMAGE_WIDTH, IMAGE_HEIGHT])
    return image, label

train_ds =
tf.data.experimental.sample_from_datasets(train_datasets).map(load_imag
e)
val_ds =
tf.data.experimental.sample_from_datasets(val_datasets).map(load_image)
test_ds =
tf.data.experimental.sample_from_datasets(test_datasets).map(load_image
)

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

train_size = 0
for batch in train_ds:
    train_size += batch[0].shape[0]
```

```
val_size = 0
for batch in val_ds:
    val_size += batch[0].shape[0]
test_size = 0
for batch in test_ds:
    test_size += batch[0].shape[0]

print(f"Train size: {train_size}")
print(f"Validation size: {val_size}")
print(f"Test size: {test_size}")

dataset_dir = '/content/drive/MyDrive/EMJ37403/Deep Learning/Project
1/Classes'

all_classes = os.listdir(dataset_dir)

test_class_counts = Counter()

for images, labels in test_ds:
    test_class_counts.update([labels.numpy()])

class_names = dataset.class_names
test_class_counts_named = {class_names[k]: v for k, v in
test_class_counts.items()}

print("Test class counts:", test_class_counts_named)

train_ds =
tf.data.experimental.sample_from_datasets(train_datasets).map(load_imag
e).batch(BATCH_SIZE).cache().shuffle(1000).prefetch(buffer_size=tf.data
.AUTOTUNE)
val_ds =
tf.data.experimental.sample_from_datasets(val_datasets).map(load_image)
.batch(BATCH_SIZE).cache().shuffle(1000).prefetch(buffer_size=tf.data.A
UTOTUNE)
test_ds =
tf.data.experimental.sample_from_datasets(test_datasets).map(load_image
).batch(BATCH_SIZE).cache().prefetch(buffer_size=tf.data.AUTOTUNE)

resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_WIDTH,
IMAGE_HEIGHT),
  layers.experimental.preprocessing.Rescaling(1.0/255),
])

input_shape = (BATCH_SIZE, IMAGE_WIDTH, IMAGE_HEIGHT, CHANNELS)
n_classes = 3
```

```
model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape = input_shape)

model.summary()

model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Fals
e),
    metrics=['accuracy']
)

early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
patience = 3, verbose=1,factor=0.3, min_lr=0.0000001)
callbacks_list = [ early, learning_rate_reduction]

history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS, #epochs
    callbacks = callbacks_list,
)
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(acc) + 1), acc, label='Training Accuracy')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation
Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, len(loss) + 1), loss, label='Training Loss')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation
Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

scores = model.evaluate(test_ds)
print("Testing Accuracy: %.2f%%\n" % (scores[1]*100))

y_true = np.concatenate([labels_batch.numpy() for _, labels_batch in
test_ds])

y_test = model.predict(test_ds)
y_pred = np.argmax(y_test, axis=1)
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d',
xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.xticks(ticks=range(len(class_names)), labels=class_names)
plt.yticks(ticks=range(len(class_names)), labels=class_names)
plt.show()
```

```
print(classification_report(y_true,y_pred,target_names
=['Early','Normal','Severe']))
```