Capstone Project

Software Technology 1 (4483)

Year 1 Semester 2

[GitHub Repo](https://github.com/vakkD/capstone/)

Dylan Colwill

26/10/2023



***Faculty of Science and Technology***

**Assignment Coversheet**

|  |  |
| --- | --- |
| **Student ID number &**  **Student Name** | Dylan Colwill u3261232 |
| **Unit name** | Software Technology 1 |
| **Unit number** | 4483 |
| **Unit Tutor** | Linda Ma |
| **Assignment name** | ST1 Capstone Project – Semester 1 2023 |
| **Due date** | 29/10/2023 |
| **Date submitted** | 29/10/2023 |

**You must keep a photocopy or electronic copy of your assignment.**

**Student declaration**

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

**Signature of student: Dylan Colwill Date: 29/10/2023**

Table of Contents

[Introduction 3](#_Toc149334932)

[Dataset Description 3](#_Toc149334933)

[Methodology 4](#_Toc149334934)

[Exploratory Data Analysis 4](#_Toc149334935)

# Introduction

In this report, I will cover the design and structure of my Capstone Project for Software Technology 1. The project is to create software to detect different types of lung and colon cancer. This project works of the basis of a dataset filled with histopathological images of these two different cancer types.

The histopathological images provide valuable insights into cellular structures and abnormalities. The aim is to develop accurate predictive models by harnessing machine learning techniques, that would improve both the efficiency and accuracy of cancer diagnosis.

The project journey involved extensive data exploration, analysis, and modelling in order to achieve meaningful results. Through experimentation learning techniques such as Convolutional Neural Networks, promising outcomes are shown in classifying different types of cancerous tissues based on visual patterns identified within these images.

# Dataset Description

This dataset is sourced online from Kaggle with a free license <https://www.kaggle.com/datasets/andrewmvd/lung-and-colon-cancer-histopathological-images>, with credit to (Borkowski AA, Bui MM, Thomas LB, Wilson CP, DeLand LA, Mastorides SM. Lung and Colon Cancer Histopathological Image Dataset (LC25000). arXiv:1912.12142v1 [eess.IV], 2019), can also be found originally on GitHub <https://github.com/tampapath/lung_colon_image_set>. The colon and lung cancer dataset consists of 25000 total images, comprised of five classes with 5000 images each. The dataset was created from 250 sample images from each class for a total of 1250, which was then expanded to 25000 using image augmentation methods, specifically the *Augmentor* Python package <https://github.com/mdbloice/Augmentor>, which is a technique to artificially generate images for the purpose of training a deep learning or machine learning model.

***Number of Classes:*** 5

***Classes:***

Colon Cancer:

* Colon Adenocarcinoma
* Colon Benign Tissue

Lung Cancer:

* Lung Adenocarcinoma
* Lung Benign Tissue
* Lung Squamous Cell Carcinoma

***Number of Images:*** 25000 total, 5000 per class

***File Type:*** jpeg

***Image Dimensions:*** 768 x 768 pixels

# Methodology

There are three major steps outlined to complete this project. Firstly, to perform Exploratory Data Analysis (EDA) on the dataset. Then creation of a prediction algorithm, Predictive Data Analytics (PDA) model. Finally to implement it into an application for ease of external users.

# Exploratory Data Analysis

The EDA stage is crucial to the development of a strong and accurate predictive model. It helps to get a deep understanding of the data and its patterns and outliers, as well as identify any errors.

In this stage, I decided to use *Jupyter Notebook* with *Visual Studio Code* as it gives a better approach for development and prototyping, allowing each code section to be edited and ran individually, which is incredibly useful in this case, as code snippets can occasionally take very long to complete. Additionally, *Jupyter Notebook* allows for better formatting as you can include all areas of the project into one place making it easier to demonstrate the process workings.

The *Jupyter Notebook* file can be found here: [*main.ipynb*](https://github.com/vakkD/capstone/blob/main/code/main.ipynb)*.*

Starting off, to figure out what to prove in EDA, questions about the dataset are needed. These are the recommended number of 5 questions used in this project, and how they can be solved.

1. What is the distribution of the dataset classes (lung and colon) in the dataset?
   1. Generate simple bar chart separated by cancer types
2. Are there any identifiable patterns or clusters when comparing features across different tissue classifications?
   1. Image analysis, t-SNE, create a scatterplot. If groups are visually separated, this indicates physical differences are present across classes.
3. Are there any significant differences between variations of cancerous tissues?
   1. Similar to before, highlight differences using violin plots.
4. How are the classes different visually depending under different contrast conditions?
   1. Display images under different contrast levels and lighting artifacts
5. What does an image from each class look like?
   1. Display a sample image from each class

Firstly, before the EDA, some preparation has to be done, as seen below.

Importing necessary libraries

import os

import pandas as pd

import matplotlib.pyplot as plt

import cv2

import numpy as np

import seaborn as sns

import pickle

from PIL import Image

from sklearn.preprocessing import StandardScaler

import time

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adamax

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from sklearn.preprocessing import LabelEncoder

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

Because my dataset did not include a structured csv file or similar directory, and since the dataset is laid out in different folders and sub folders, I had to sort and store the image directories. Show below, starting by defining the root dataset folder. As I was working on this project across two machines, I had to check both paths for easy transition from one station to another. The code iterates through subfolders and files in the specified directory, collecting file paths and assigning labels based on the folder names. Then lastly creating a *Pandas DataFrame* with two columns, *filepaths* and *labels*.

# import and label images

data\_dir ='L:/!school/!uni/!classes/sem2-2023/software technology/assignments/assignment 2/lung\_colon\_image\_set'

try:os.listdir(data\_dir)

except:data\_dir ='C:/Users/dylan/OneDrive/school/sem2-2023/software technology/assignments/assignment 2/lung\_colon\_image\_set'

labels\_mapping = {

    'colon\_aca': 'Colon Adenocarcinoma',

    'colon\_n': 'Colon Benign Tissue',

    'lung\_aca': 'Lung Adenocarcinoma',

    'lung\_n': 'Lung Benign Tissue',

    'lung\_scc': 'Lung Squamous Cell Carcinoma'

}

filepaths = []

labels = []

for fold in os.listdir(data\_dir):

    foldpath = os.path.join(data\_dir, fold)

    for f in os.listdir(foldpath):

        f\_path = os.path.join(foldpath, f)

        for file in os.listdir(f\_path):

            fpath = os.path.join(f\_path, file)

            filepaths.append(fpath)

            labels.append(labels\_mapping.get(f, ''))

df = pd.DataFrame({'filepaths': filepaths, 'labels': labels})

data\_dir

df.info()

df

***Output:***

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 25000 entries, 0 to 24999

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 filepaths 25000 non-null object

1 labels 25000 non-null object

dtypes: object(2)

memory usage: 390.8+ KB

|  | **filepaths** | **labels** |
| --- | --- | --- |
| 0 | L:/!school/!uni/!classes/sem2-2023/software te... | Colon Adenocarcinoma |
| 1 | L:/!school/!uni/!classes/sem2-2023/software te... | Colon Adenocarcinoma |
| 2 | L:/!school/!uni/!classes/sem2-2023/software te... | Colon Adenocarcinoma |
| 3 | L:/!school/!uni/!classes/sem2-2023/software te... | Colon Adenocarcinoma |
| 4 | L:/!school/!uni/!classes/sem2-2023/software te... | Colon Adenocarcinoma |
| ... | ... | ... |
| 24995 | L:/!school/!uni/!classes/sem2-2023/software te... | Lung Squamous Cell Carcinoma |
| 24996 | L:/!school/!uni/!classes/sem2-2023/software te... | Lung Squamous Cell Carcinoma |
| 24997 | L:/!school/!uni/!classes/sem2-2023/software te... | Lung Squamous Cell Carcinoma |
| 24998 | L:/!school/!uni/!classes/sem2-2023/software te... | Lung Squamous Cell Carcinoma |
| 24999 | L:/!school/!uni/!classes/sem2-2023/software te... | Lung Squamous Cell Carcinoma |

The next areas are fairly straight forward and use the *df* *DataFrame* variable to access the images, which is defined before. The first section (indicated by the comments) creates a bar plot to display the number of images in each class. Using the *labels* column from the *DataFrame df* variable to count the occurrences of each class. The second section displays a sample imaegs from each class in the dataset. It identifies the first unique class name from the *labels* column, and for each class it retrieves the file path of the first image in that class. Reading and displaying the image using *OpenCV cv2*.

#bar plot

plt.figure(figsize=(10, 6))

df['labels'].value\_counts().plot(kind='bar')

plt.title('Number of Images per Class')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.show()

#show sample image per class

classes = df['labels'].unique()

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

for i, class\_name in enumerate(classes):

    img\_path = df[df['labels']==class\_name]['filepaths'].values[0]

    img = cv2.imread(img\_path)

    img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

    plt.subplot(4, 4, i + 1)

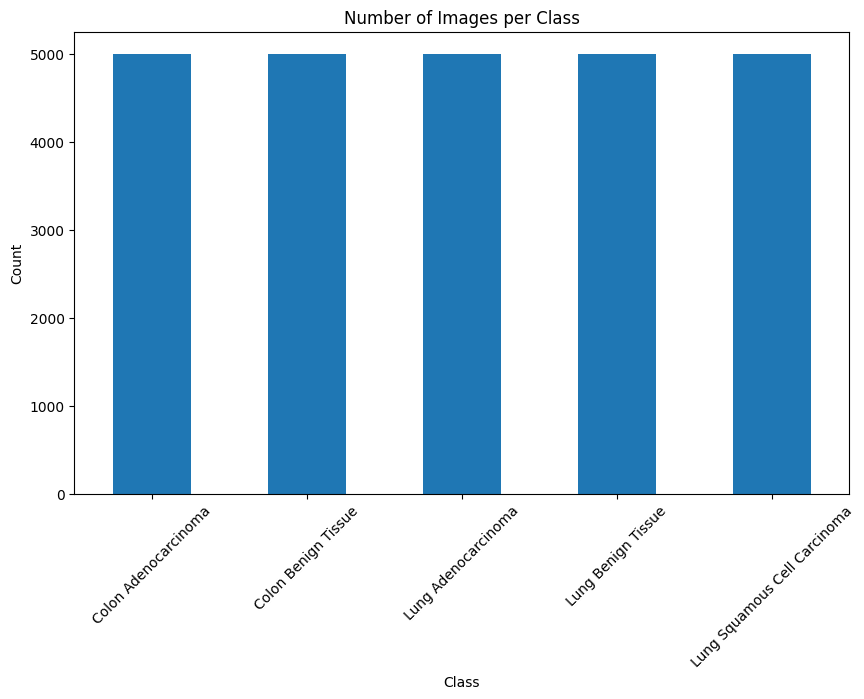
    plt.imshow(img\_rgb)

    plt.title(class\_name, color='blue', fontsize=12)

    plt.axis('off')

plt.show()

***Output:***



A close-up of a microscope

Description automatically generated

After this, examining lighting artifacts from the exposure is useful to show how these factors may affect the quality and characteristics of the images. Using techniques such as adaptive histogram equalisation, by enhancing image contrast, helps to address issues that may be caused by uneven brightness. Here I am using the example code provided from unit material. The code firstly loads and displays two images, one from the *Colon Adenocarcinoma* class and another from the *Lung Adenocarcinoma* class, using the *data\_dir* variable defined before when importing images. It then proceeds to analyse and visualise the illumination and lighting artifacts in these images. Creating a function *show(img)* to display images and their histograms. Applying adaptive histogram equalisation using *skie.equalize\_adapthist* to enhance image contrast. Then computes the sobel edge detection on both the original and equalised images using *skif.sobel*. All these images are displayed using the *show(img)* function.

img\_path\_1 = f'{data\_dir}/colon\_image\_sets/colon\_aca/colonca1.jpeg'

img\_1 = cv2.imread(img\_path\_1)

img\_path\_2 = f'{data\_dir}/lung\_image\_sets/lung\_aca/lungaca1.jpeg'

img\_2 = cv2.imread(img\_path\_2)

# Analysing illumination and lighting artefacts by examining the camera effects/exposure of an #image

import cv2

import matplotlib.pyplot as plt

import numpy as np

import skimage

import skimage.color as skic

import skimage.filters as skif

import skimage.data as skid

import skimage.util as sku

import skimage.exposure as skie

# %matplotlib inline

def show(img):

    # Display the image.

    fig, (ax1, ax2) = plt.subplots(1, 2,

                                figsize=(12, 3))

    ax1.imshow(img, cmap=plt.cm.gray)

    ax1.set\_axis\_off()

    # Display the histogram.

    ax2.hist(img.ravel(), lw=0, bins=256)

    ax2.set\_xlim(0, img.max())

    ax2.set\_yticks([])

    plt.show()

show(img\_1)

# adaptive histogram equalisation

show(skie.equalize\_adapthist(img\_1))

show(img\_2)

# adaptive histogram equalisation

show(skie.equalize\_adapthist(img\_2))

#class 1 image

img = skic.rgb2gray(img\_1)

sobimg\_nheq= skif.sobel(img)

show(sobimg\_nheq)

img = skic.rgb2gray(skie.equalize\_adapthist(img\_1))

sobimg\_heq\_1 = skif.sobel(img)

show(sobimg\_heq\_1)

#class 2 image

img = skic.rgb2gray(img\_2)

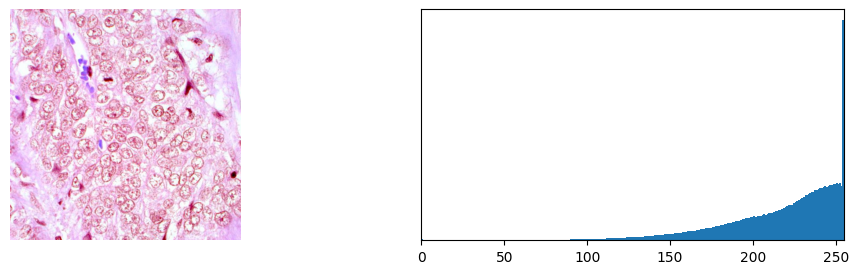
sobimg\_nheq= skif.sobel(img)

show(sobimg\_nheq)

img = skic.rgb2gray(skie.equalize\_adapthist(img\_2))

sobimg\_heq\_2 = skif.sobel(img)

show(sobimg\_heq\_2)

***Output:***  


A graph with a blue line

Description automatically generated

A blue line graph with a white background

Description automatically generated

A blue graph with numbers

Description automatically generated

A blue graph with white background

Description automatically generated

A blue graph with white background

Description automatically generatedA blue graph with white background

Description automatically generated

A blue graph with white background

Description automatically generated

Finally, to show image dimensionality *principal component analysis* *(PCA)* and *t-Distributed Stochastic Neighbour Embedding* *(t-SNE)* techniques were applied to the dataset. Starting by checking if the data is already calculated and saved. This is done as the calculations take some time to go through each image and compute, mainly due to debugging reasons, it was beneficial to take this step. The image processing step includes converting the images to greyscale, resizing them to 128x128 pixels, and flattening them into 1D arrays. Then performs *PCA* on the processed image data with 50 components, which are the new variables or features that are created to represent the original data. In the PCA case, components represent the directions in the original data space along which there is most variation. *PCA* is applied to reduce the dimensionality of the processed image data while retaining essential information. It also stores the class labels from *df* into *labels\_encoded* using the *scikit-learn* *LabelEncoder* function. Then applying *t-SNE* with 2 components, which is used for further dimensionality reduction, reducing *PCA* to a 2D representation, while focussing on preserving the similarity relationships between data points, allowing for effective visualisation and clustering of data. In this case, components refer to the transformed dimensions in the lower dimensional space (2D) where the data points are mapped. Then saving all this data into files for later use.

try:

    with open('pca\_result.pkl', 'rb') as f:

        pca\_result = pickle.load(f)

    with open('tsne\_result.pkl', 'rb') as f:

        tsne\_result = pickle.load(f)

    with open('labels\_encoded.pkl', 'rb') as f:

        labels\_encoded = pickle.load(f)

except:

    images = []

    for i in df['filepaths']:

        img = Image.open(i).convert('L')

        img = img.resize((128, 128), Image.LANCZOS)

        img = np.array(img).flatten()

        images.append(img)

    le = LabelEncoder()

    labels\_encoded = le.fit\_transform(df['labels'])

    pca = PCA(n\_components=50)

    pca\_result = pca.fit\_transform(images)

    tsne = TSNE(n\_components=2)

    tsne\_result = tsne.fit\_transform(pca\_result)

    with open('pca\_result.pkl', 'wb') as f:

        pickle.dump(pca\_result, f)

    with open('tsne\_result.pkl', 'wb') as f:

        pickle.dump(tsne\_result, f)

    with open('labels\_encoded.pkl', 'wb') as f:

        pickle.dump(labels\_encoded, f)

The scatter plot shows how the data points from different classes are distributed and clustered in this lower dimensional space, showing the grouping and separation. The violin plot here shows the first and second principal components, which explain the maximum variance in the data, first most and second most.

df\_tsne = pd.DataFrame({'X':tsne\_result[:,0], 'Y':tsne\_result[:,1], 'labels':labels\_encoded})

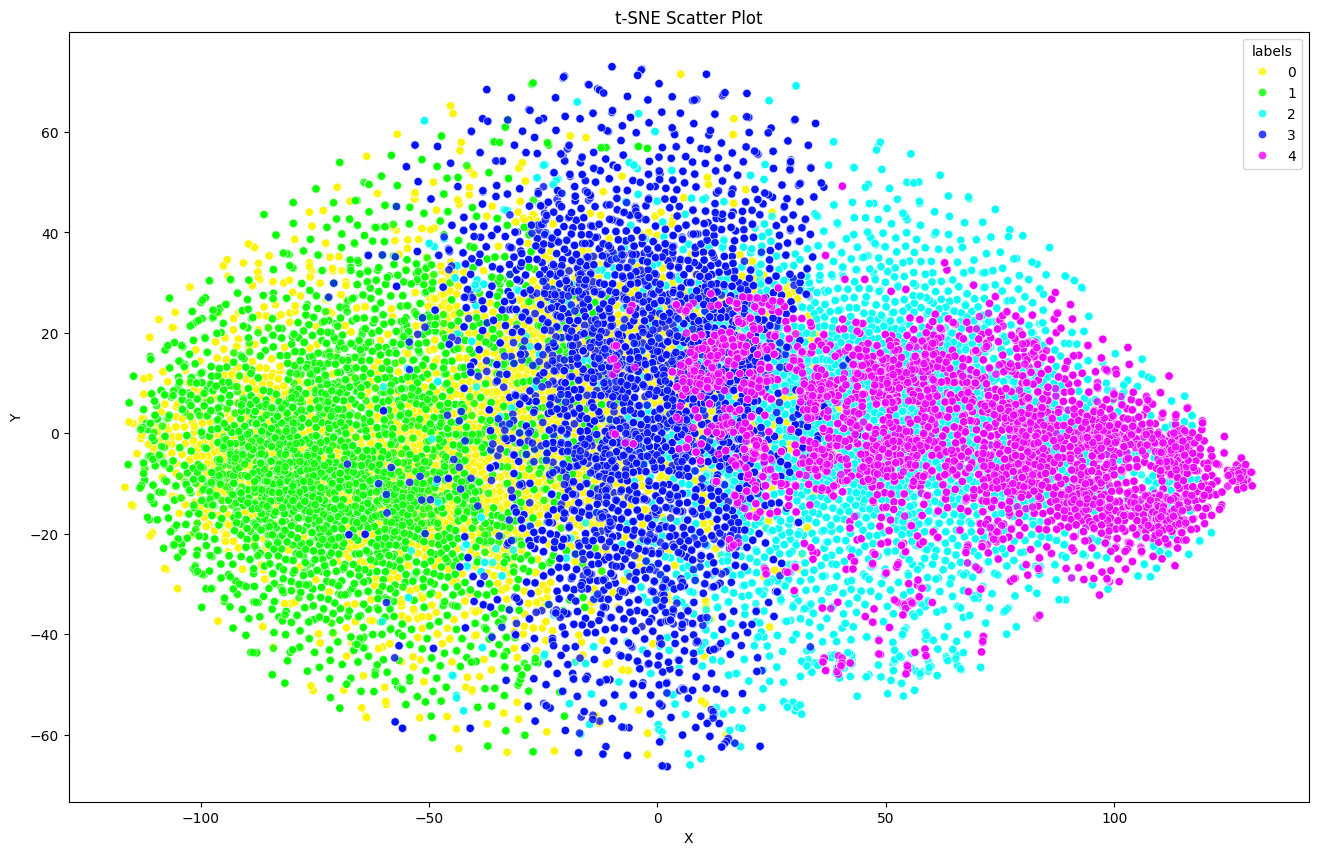
plt.figure(figsize=(16,10))

sns.scatterplot(data=df\_tsne, x="X", y="Y", hue="labels", palette=sns.color\_palette("hsv", 5), legend="full", alpha=0.8)

plt.title('t-SNE Scatter Plot')

plt.show()

***Output:***



#violin plots

df\_plot = pd.DataFrame(pca\_result[:, :2], columns=['PC1', 'PC2'])

df\_plot['Label'] = labels\_encoded

plt.figure(figsize=(15, 10))

sns.violinplot(x='Label', y='PC1', data=df\_plot)

plt.title('Violin plot of the first principal component')

plt.show()

plt.figure(figsize=(15, 10))

sns.violinplot(x='Label', y='PC2', data=df\_plot)

plt.title('Violin plot of the second principal component')

plt.show()

***Output:***

A graph of blue and black rhombus shapes

Description automatically generatedA diagram of a graph

Description automatically generated with medium confidence

# Predictive Data Analytics

This section doesn’t require any initial setup as everything is already done in *EDA*.

The code starts by splitting the original *DataFrame* into training (80%) and testing (20%) sets. Then setting up data augmentation for the training using the *Keras* *ImageDataGenerator* function. Data augmentation includes operations such as zooming, shifting, image rotation and flipping. To train the data a *Convolutional Neural Network (CNN)* is created using *Keras*. The model consists of several convolutional layers followed by max pooling layers, flattening and fully connected layers. The final output layer has five units with a softmax activation for multi class classification. The model is compiled using the *Adamax* optimiser from *Keras*. It is trained to classify images into one of the five classes, trained on 100 epochs.

train\_df, test\_df = train\_test\_split(df, test\_size=0.2, random\_state=42)

#data augmentation

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    zoom\_range=0.2,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    horizontal\_flip=True

)

test\_datagen = ImageDataGenerator(rescale=1./255)

img\_size = (224, 224)

batch\_size = 16

#traingina and testing data

train\_gen = train\_datagen.flow\_from\_dataframe(

    train\_df,

    x\_col='filepaths',

    y\_col='labels',

    target\_size=img\_size,

    class\_mode='categorical',

    color\_mode='rgb',

    shuffle=True,

    batch\_size=batch\_size

)

test\_gen = test\_datagen.flow\_from\_dataframe(

    test\_df,

    x\_col='filepaths',

    y\_col='labels',

    target\_size=img\_size,

    class\_mode='categorical',

    color\_mode='rgb',

    shuffle=False,

    batch\_size=batch\_size

)

#create model

model = Sequential([

    Conv2D(16, (3, 3), activation='relu', input\_shape=(img\_size[0], img\_size[1], 3)),

    MaxPooling2D(pool\_size=(2, 2)),

    Conv2D(32, (3, 3), activation='relu'),

    MaxPooling2D(pool\_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D(pool\_size=(2, 2)),

    Flatten(),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(5, activation='softmax')

])

model.compile(optimizer=Adamax(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

start\_time =time.time()

# train model

history = model.fit(train\_gen, epochs=100, validation\_data=test\_gen)

overall\_time=time.time()-start\_time

***Output:***

Epoch 1/100

1250/1250 [==============================] - 435s 347ms/step - loss: 0.6653 - accuracy: 0.7010 - val\_loss: 0.3664 - val\_accuracy: 0.8366

Epoch 2/100

1250/1250 [==============================] - 311s 249ms/step - loss: 0.3808 - accuracy: 0.8457 - val\_loss: 0.2545 - val\_accuracy: 0.8958

Epoch 3/100

1250/1250 [==============================] - 310s 248ms/step - loss: 0.3017 - accuracy: 0.8810 - val\_loss: 0.1953 - val\_accuracy: 0.9274

Epoch 4/100

1250/1250 [==============================] - 325s 260ms/step - loss: 0.2640 - accuracy: 0.8985 - val\_loss: 0.1957 - val\_accuracy: 0.9204

Epoch 5/100

1250/1250 [==============================] - 323s 258ms/step - loss: 0.2342 - accuracy: 0.9105 - val\_loss: 0.2754 - val\_accuracy: 0.9052

Epoch 6/100

1250/1250 [==============================] - 317s 254ms/step - loss: 0.2096 - accuracy: 0.9194 - val\_loss: 0.2527 - val\_accuracy: 0.9088

Epoch 7/100

1250/1250 [==============================] - 303s 242ms/step - loss: 0.1927 - accuracy: 0.9277 - val\_loss: 0.1854 - val\_accuracy: 0.9308

Epoch 8/100

1250/1250 [==============================] - 307s 245ms/step - loss: 0.1814 - accuracy: 0.9298 - val\_loss: 0.1236 - val\_accuracy: 0.9562

Epoch 9/100

1250/1250 [==============================] - 306s 244ms/step - loss: 0.1697 - accuracy: 0.9362 - val\_loss: 0.2360 - val\_accuracy: 0.9214

Epoch 10/100

1250/1250 [==============================] - 305s 244ms/step - loss: 0.1600 - accuracy: 0.9386 - val\_loss: 0.1196 - val\_accuracy: 0.9538

Epoch 11/100

1250/1250 [==============================] - 306s 245ms/step - loss: 0.1463 - accuracy: 0.9444 - val\_loss: 0.1443 - val\_accuracy: 0.9414

Epoch 12/100

1250/1250 [==============================] - 307s 245ms/step - loss: 0.1445 - accuracy: 0.9470 - val\_loss: 0.1256 - val\_accuracy: 0.9466

Epoch 13/100

...

Epoch 99/100

1250/1250 [==============================] - 313s 251ms/step - loss: 0.0325 - accuracy: 0.9886 - val\_loss: 0.0133 - val\_accuracy: 0.9942

Epoch 100/100

1250/1250 [==============================] - 312s 250ms/step - loss: 0.0358 - accuracy: 0.9872 - val\_loss: 0.0185 - val\_accuracy: 0.9948

model.save('newmodel.h5')

accuracy = model.evaluate(test\_gen)[1]

print("Accuracy:", accuracy)

overall\_time/60

***Output:***

[c:\Users\dylan\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\training.py:3079](file:///C:\Users\dylan\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\training.py:3079): UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

313/313 [==============================] - 18s 57ms/step - loss: 0.0185 - accuracy: 0.9948

Accuracy: 0.9947999715805054

520.1634558677673

The trained model can be used to make predictions on the test data, results in a probability distribution over the classes for each test sample. Extracting the class with the highest probability from the predicted probability distribution for each test sample. Then generating a classification report by comparing the true labels and the predicted labels. A confusion matrix is used to evaluate the performance of the model. Creating a heatmap visualisation of the confusion matrix helps in understanding how well the model has identified data into different classes. It provides insights into the models strengths and weaknesses.

***Output:***

Classification Report:

precision recall f1-score support

0 1.00 0.99 0.99 1021

1 0.99 1.00 0.99 1000

2 1.00 0.99 0.99 985

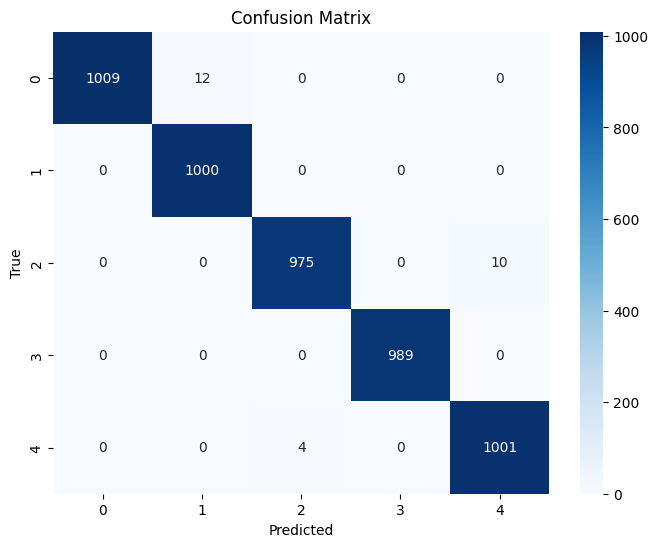
3 1.00 1.00 1.00 989

4 0.99 1.00 0.99 1005

accuracy 0.99 5000

macro avg 0.99 0.99 0.99 5000

weighted avg 0.99 0.99 0.99 5000

****