The Application of Artificial intelligence in medical field

Abstract

This study delves into the application of Artificial Intelligence (AI) in the medical field, with a specific focus on lung cancer classification. Beginning with an exploration of AI fundamentals, including its impact across various sectors, the research outlines different AI techniques and algorithms crucial for medical applications. Within healthcare, AI plays a pivotal role in diagnostics, personalized medicine, and treatment planning. The study highlights the significance of AI in lung cancer classification, detailing its current state of research, including advancements in deep learning methods and challenges faced in implementation. Through the development of a Convolutional Neural Network (CNN) model trained on lung cancer datasets, the research aims to address key limitations and enhance the efficiency of AI systems in healthcare. Overall, the study underscores the transformative potential of AI in revolutionizing medical diagnosis and treatment strategies, paving the way for more precise and personalized patient care.

Table of Contents

2.1.1.

1. INTRODUCTION

1.1 Fundamentals of AI

Artificial Intelligence (AI) refers to the subset of computer science where computer systems are developed to perform tasks that typically require human intelligence and thought process, machine learning ML & Deep learning are subfields of AI used to carry out different types of tasks. These tasks include reasoning, problem-solving, speech recognition, and natural language understanding. AI systems aim to mimic human cognitive functions and, in some cases, surpass human capabilities in specific subdomains. The rapid development in AI over the past decades has been exponential and nothing short of mesmerizing, especially with advancements in the availability of data and the computing power needed to process this huge amount of information. [1]

There are different types of AI based on either their functionality (specified tasks) or their capabilities (cognitive abilities)

Functionality: This category includes narrow, general, and super AI models. Narrow AI models are specific and task-oriented, carrying out custom training instructions during their development which focuses on applying their skills to solving a particular task, this type of AI would be considered less intelligent than humans. General AI models have a wider scope of reasoning allowing them to carry out multi-step processes and use their toolkit of information to solve a task, in a similar manner to how their training data provides information on solving that task or parts of it, this type of AI is considered to be on par with human intelligence and still not fully achieved, however, researchers have never been closer. Super AI is a computer scientist's fantasy, this model is theoretically more capable than any human and is designed to mimic peak human fluid intelligence across various domains. [2]

Capability: This category of AI models is split into 4 main types, reactive machines have their outputs confined within a specific set of predetermined rules, for every specific input, there is a specific output, an example of this would be chess bots. Limited memory models have a more data-driven learning approach, they make decisions based on past recent experiences, an example of this would be text completion suggestion algorithms that learn from previous user inputs. Theory of mind (ToM) is another type of model, it's like general AI models in the sense that it understands how humans process emotions & thoughts, and hence acts that out when faced with issues to tackle, this type of model still hasn't fully been realized just yet. The last model is the Self-aware AI , this type of model can grasp the concept of self, it is aware of it's existence and it's surroundings, has the ability to be goal oriented, asses it's current state of being and make up it's own mind on where it wants to be next using algorithms such as Q* and others, this is the closest thing to a super intelligent AI, and developing such a model must be handled with extra care to ensure safety and responsible use for humankind. [3]

1.2 AI Impact on Different Sectors

AI has impacted many fields in various ways, to name a few, in the healthcare sector AI is used for diagnostics, personalized medicine based on assessing patients' unique cases and which treatments would work best for them based on their medical history as well as drug discovery and research when it comes to testing different chemical compounds and how they can be directed towards treating specific diseases.

In finance, AI is used to assess financial risk when approaching a new investment, basing its outputs on company resources, and market conditions & factoring in the macro-economic landscape, it is also used to detect fraud by analyzing the behavioral history of accounts and individuals.

In the customer service field, AI is deployed across each customer interaction, whether through a personalized assistant responding to text, or voice recognition systems that transcribe calls, analyze sentiment and provide summaries with action items that would help representatives ensure the highest customer satisfaction rates.

In the manufacturing and retail sector, AI is used all across the supply chain from the beginning of the manufacturing process itself, to quality control, demand forecasting, logistics, price optimization and personalized recommendations to buyers. [4]

1.3 AI Techniques & Algorithms

AI techniques & algorithm choices are an essential component to guarantee that the model performs the desired output we design it for. The 3 main categories are classic AI techniques, machine learning algorithms, and deep learning algorithms.

Classic AI techniques include symbolic AI also known as rule-based AI which we discussed in the capability section as the set of conditions used by reactive machines to determine their outputs simulating a slightly inferior decision-making ability than that of a human due to it's limited factor consideration capabilities. Another similar technique to the rule-based AI in decision-making procedures is the formal logic concept, which emulates the step-by-step reasoning humans go through, considering more situational factors to come to conclusions. A mathematical and statistical approach can also be taken to solve problems with incomplete information by arbitrarily exploring variables and their possible outcomes through trial-and-error analysis, all while being goal-oriented. [5]

Machine Learning

Supervised learning encompasses the following algorithms

Linear Regression: An algorithm used to predict the best fitting plane or polynomial regression to find the best fitting curve that can complement a set of data. This method can be useful when forecasting the expected sales of a company.

Support Vector Machines: An algorithm used in classification tasks, it finds the data points with the largest margins of separation specifically in larger data sets.

Decision Trees: An algorithm used in both classification & regression tasks that helps determine output values by applying simple decision-making statements, the algorithm does not take into consideration arbitrary assumptions.

Random Forest: An algorithm that works by constructing multiple decision trees and putting together their predictions to solve a larger task with improved accuracy and less overfitting, it is known to be an effective method when dealing with larger datasets. [6]

Unsupervised learning encompasses the following algorithms

K-Means Clustering: A portion-based clustering algorithm that groups similar data points into predefined clusters based on the Euclidean distance between 2 points

***Principal Component Analysis (PCA)**:* A dimensionality reduction technique that transforms data from high dimensions to lower dimensions, essentially simplifying the data, but still keeping the essential component of the data that makes it unique.

Hierarchical Clustering: This method combines K-Means Clustering and PCA by clustering similar shortened data points to each other, this allows for scalable clustering of large datasets in both linear and non-linear distributions and provides a visual representation.

Autoencoders: A form of neural network used to encode the compressed representation of data transformed from higher to lower dimensions, it's a useful method when compressing data and detecting anomalies. [6]

Reinforcement learning

Q-Learning: A method of reinforcing the best path of action towards a goal by implementing the highest reward value at a desired outcome (Q), this method gives the model the flexibility to select the optimal path of action (policy) to achieve that highest reward. [7]

Reinforcement learning from human feedback (RLHF): A ML approach that combines the rewards and penalty system with human guidance to train AI models, it is primarily used in fine-tuning natural language processing (NLP) models ensuring the outputs are more context-oriented and logical, the downside however is that it's time intensive

Reinforcement Learning from AI Feedback (RLAIF): A solution proposed to solve the limiting factor of RLHF (time), it uses feedback from a pre-trained AI system in a specified domain or more powerful large language models (LLMs) to reinforce rewards and penalties to models in training, this saves time and has results similar or superior to RLHF. [8]

Deep Learning

Convolutional Neural Networks (CNNs): A model that utilizes a series of layers that apply filters to input data, pools and connects related layers to decrease spatial dimensions, best used for the classification of "grid-like" images and text.

Recurrent Neural Networks (RNNs): A model designed to process sequential data such as audio, time series and text, an example is long short-term memory models which use RNN to call certain pieces of information over a long timeframe.

Transformer Models: Transformer models use similar algorithms to RNNs but address some deficiencies such as handling increased lengths of input sequences, making them a more attractive commercial model for larger context window models and can process inputs in a parallel manner allowing for versatility of tasks to be carried out simultaneously.

Generative Adversarial Networks (GANs): A model which consists of 2 networks, a generator and a discriminator, the generator creates synthetic data like that of the input while the discriminator distinguishes between the original input and the synthetic data, this is particularly useful in AI safety to distinguish whether the data at hand is original or not. [9]

1.4 AI in Healthcare

The benefit of implementing AI in health care lies in several factors, namely, decreasing manual error of analyzing the patient's symptoms, a case study comparing an AI model's and a physician's accuracy of diagnosis has shown that the AI was accurate 89.69% of the time, while the physician was accurate 85.57% of the time. AI model's computational power allows it to assess multiple cases at the same time, the present business profitability due to the scalability of providing this analysis to multiple patients at once, but more importantly, it decreases the patient's waiting time, which in some cases, could be of critical effect on the patient's health, this also adds priority of treatment to the more severe cases. AI can further be integrated with the hospital's/clinic's pharmaceutical and surgery systems, this allows for customized treatment plan recommendations to be presented to physicians taking into consideration the patient's medical history and being able to detect potential diseases before they become persistent in a patient's body. AI systems can be implemented internally within the care center to act as an information tool for physicians & doctors, or can be implemented externally to push notifications reminding patients to come in for regular check-ups, through a UI, patients can interact with a general assistant and direct their repetitive or unurgent questions to it instead of having to come into the clinic, consuming both their and the medical professionals' time.

As useful as AI can potentially be in healthcare, we must account for potential limitations. Ethical considerations such as biases, data privacy, data quality which the model has been trained on are all factors that concern the end users. Some patients prefer a human touch to feel cared for from a psychological standpoint. The integration of these systems into the existing medical technological infrastructure at scale yet in varying personalization to a degree from one care center to the other proposes a challenge, the adoption of these systems by medical professionals and incorporating them into their daily workflows is an undermined aspect that takes training and time to get acquainted with. [11]

1.5 Lung Cancer Classification

Ensuring that AI models have the most minimal error rate and the highest accuracy is crucial to ensure that medical professionals have a full understanding of the patient's case and can choose the correct route of treatment to their specified case. Using convolutional neural networks, CT scans are transformed into grid-like images understandable by the model to correctly detect whether this image has a benign or malignant lung tumor and which type of subtype of cancer it is if any, ensuring the models have high sensitivity & specificity ratios are a rudimentary metric to the identification process, allowing the assessment of the severity of the case and determining how soon a treatment or surgical intervention is needed, in cases where this identification is made early on, the approach of prevent rather than treat is taken. Increased accuracy also saves the patient and health institution from unnecessary invasive procedures which in some cases may lead to complications and put in place a personalized plan catering to the unique attributes of the client's case. [12]

1.6 Current State of Research

Over the past 2 decades, several studies have been published demonstrating the power of infusing AI with lung cancer diagnosis, a constant curve of increased classification accuracy and treatment. Through deep learning methods AI can analyze tissue samples and identify cancer types, whether forming on the outer lining of internal organs (adenocarcinomas) or cancer that starts as a growth of cells on the skin (squamous cell carcinomas). AI is now being used as a lung nodule detector of CT scans with the use of algorithms such as random forests, support vector machines and convolutional neural networks, it is starting to establish itself as a second reader or CT scans, assisting radiologists in increasing their diagnosis accuracy. AI has also been used to predict the efficacy of immunotherapy and customize treatment plans to predict their outcomes. [13]

1.7 Challenges & Limitations

Various challenges face the implementation of AI systems in the field of lung cancer classification. Lung cancer in nature is a highly diverse disease, therefore high-quality models with a wide scope of specified clinical and demographic data are needed to ensure the different types of cancer can be detected and analyzed with the presence of versatile features. Widespread adoption is yet to happen in this field, no standardized methods of operation have been set, therefore by extension decreasing the trust in adopting it, partly also relating to the ethical concerns surrounding AI in general when it comes to sensitive aspects of user's data privacy, consent of information processing and compliance with existing medical and data regulations. [14]

To implement an AI model which can adequately asses whether a patient has lung cancer and dictate the severity (benign or malignant), we will create a CNN which uses a ResNet or VGG16 configuration applied to “The IQ-OTHNCCD lung cancer dataset” available in Kaggle, we will resize the CT scans to ensure compatibility and efficiency with our model, then we will label this dataset to connect the related labels together, finally, we will train, test and validate our model and store it’s results

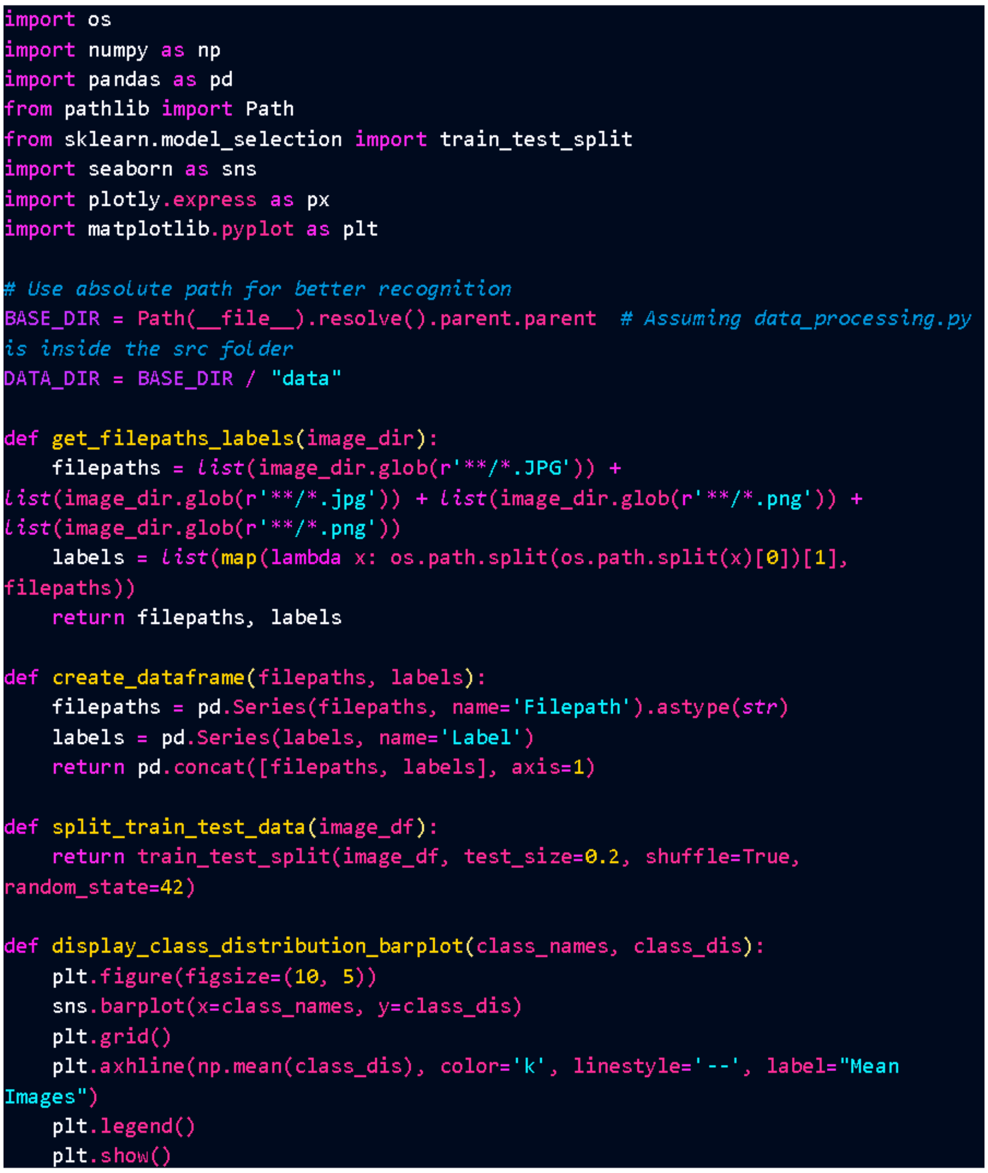
In conclusion, throughout this chapter, we have discussed the types of models & algorithms which constitute the rudimentary building blocks of AI models. The core focus was AI in healthcare, its current research state, its potential, as well as challenges & limitations and how throughout this thesis we’ll be able to address them to increase the efficiency of these systems, making them adoption ready.

2. Source Code & Theory

The main body of our program that gets executed as a script is written in the file “main.py”, this file uses 3 scripts that are stored in a single folder “src”, detailing the process of how each core part of the program is constructed to function, these 3 files are used to process data, build the model and visualize the results, we also have another file that contains our data set which is split into “Malignant”, “Benign” and “Normal” cases. In this chapter, we will discuss the 3 “src” files in detail.

2.1 Data Processing

Code



```python
import os
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

# Use absolute path for better recognition
BASE_DIR = Path(__file__).resolve().parent.parent  # Assuming data_processing.py
is inside the src folder
DATA_DIR = BASE_DIR / "data"

def get_filepaths_labels(image_dir):
    filepaths = list(image_dir.glob(r'**/*.JPG')) +
list(image_dir.glob(r'**/*.jpg')) + list(image_dir.glob(r'**/*.png')) +
list(image_dir.glob(r'**/*.png'))
    labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],
filepaths))
    return filepaths, labels

def create_dataframe(filepaths, labels):
    filepaths = pd.Series(filepaths, name='Filepath').astype(str)
    labels = pd.Series(labels, name='Label')
    return pd.concat([filepaths, labels], axis=1)

def split_train_test_data(image_df):
    return train_test_split(image_df, test_size=0.2, shuffle=True,
random_state=42)

def display_class_distribution_barplot(class_names, class_dis):
    plt.figure(figsize=(10, 5))
    sns.barplot(x=class_names, y=class_dis)
    plt.grid()
    plt.axhline(np.mean(class_dis), color='k', linestyle='--', label="Mean
Images")
    plt.legend()
    plt.show()
```

Figure 1: Data Processing Code

2.1.1. Importing Libraries

We start off by importing the needed Python libraries and modules which will provide the functionality for data processing. We included imports such as "os" for file path operations, "numpy" and "pandas" for numerical operations and data manipulation, "Path" for handling file paths, and a few visualization libraries (seaborn, plotly.express, matplotlib.pyplot) that will later allow us to create informative plots that represent our training and testing results.

2.1.2. Defining Constants

We then define our constants that represent key directories in the project that will be used in our main script. “BASE_DIR” is set to the parent directory of the script “(__file__)” assuming it is inside the src folder. “DATA_DIR” represents the directory where our sample image data is stored.

2.1.3. Functions

get_filepaths_labels(image_dir):

This function takes an image directory as input and returns two lists
1. filepaths containing paths to image files
2. labels extracted from the parent directory of each image.

create_dataframe(filepaths, labels):

This function converts two lists of file paths and labels into a pandas data frame with the 2 columns 'Filepath' and 'Label'.

split_train_test_data(image_df):

Now that we have a DataFrame containing file paths and labels for our dataset, we will use the above function to split the data into training and testing sets using “train_test_split” from “sklearn.model_selection”

display_class_distribution_barplot(class_names, class_dis):

This function takes our lists of class names and generates a bar plot using “seaborn”. It also includes a horizontal line representing the mean number of images per class.

The key components we get from executing this code is the following

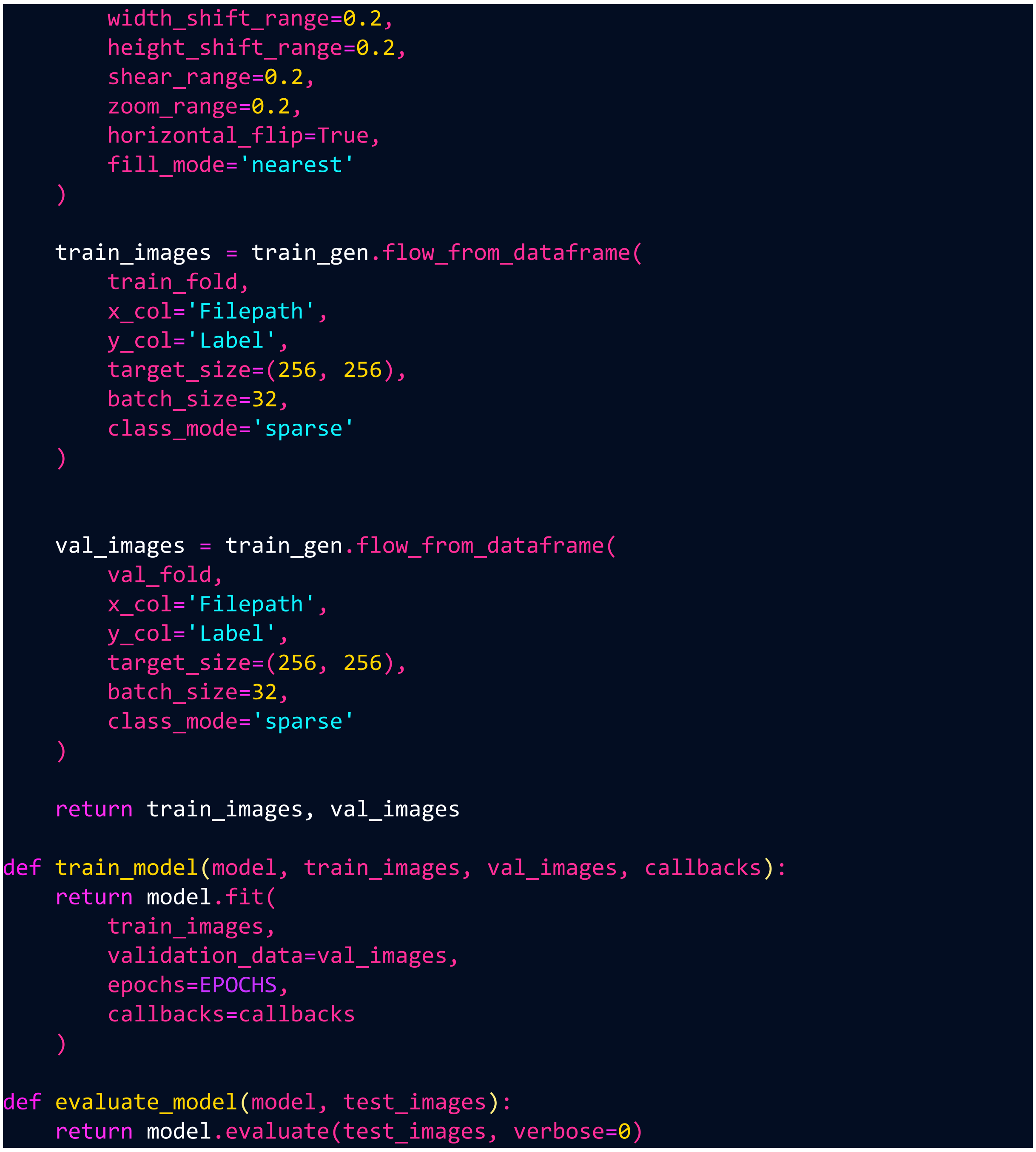
- ✔ Obtaining file paths and labels from a specified image directory.
- ✔ Converting the file paths and labels into a pandas data frame.
- ✔ Splitting the data into training and testing sets.

✔ Displaying a bar plot visualizing the number of images per class

2.2 Model Building

Code

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, GlobalAvgPool2D as GAP, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import KFold


EPOCHS = 1


def create_resnet152V2_model():
    name = "ResNet152V2"
    base_model = ResNet152V2(include_top=False, input_shape=(256, 256, 3),
weights='imagenet')
    base_model.trainable = False

    return Sequential([
        base_model,
        GAP(),
        Dense(256, activation='relu'),
        Dropout(0.2),
        Dense(3, activation='softmax')
    ], name=name)

def compile_model(model):
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
    )

def setup_callbacks(model_name):
    return [
        EarlyStopping(patience=5, restore_best_weights=True),
        ModelCheckpoint(model_name + ".h5", save_best_only=True)
    ]


def get_image_data_generators(train_fold, val_fold):
    train_gen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
```



```
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )

    train_images = train_gen.flow_from_dataframe(
        train_fold,
        x_col='Filepath',
        y_col='Label',
        target_size=(256, 256),
        batch_size=32,
        class_mode='sparse'
    )


    val_images = train_gen.flow_from_dataframe(
        val_fold,
        x_col='Filepath',
        y_col='Label',
        target_size=(256, 256),
        batch_size=32,
        class_mode='sparse'
    )

    return train_images, val_images

def train_model(model, train_images, val_images, callbacks):
    return model.fit(
        train_images,
        validation_data=val_images,
        epochs=EPOCHS,
        callbacks=callbacks
    )

def evaluate_model(model, test_images):
    return model.evaluate(test_images, verbose=0)
```

Figure 2: Model Building Code

2.2.1. Importing Libraries

- **"TensorFlow"** provides us with the core functionalities for creating and training our CNN.

- **"Sequential"** is a Keras class that allows the layer-by-layer building in a sequential manner of our CNN.

  The sequential model is used in single input single output tasks and when the architecture of layer stacks is linear, adding one layer at a time, this is the configuration of our program. There are 3-layer types in this configuration, the input layer, hidden layers and the output layer. The input layer does not have any trainable parameters, it is where we specify the shape of our input. [15]

- **"Dense"**, **"GlobalAvgPool2D"**, and **"Dropout"** are layer classes from Keras used to define the architecture of the neural network.

  Within our second layer (hidden layer) we have the above-mentioned layers

- The **"Dense"** layer is a fully connected layer, meaning that each neuron in each layer, received input from every single neuron in the previous layer, mathematically this can be expressed with the following equation,

$$\left(\sum_{n=1}^{n} (wi * xi) + b\right)$$

  Where

  Wi – Weights , Xi – Inputs , b – Bias

  To be put simply, $output = activation(inputs * weights + bias)$ [16]

- The **"GlobalAvgPool2D"** computes the output value of each feature map by computing the average values across its spatial dimensions, this reduces the spatial dimensions and captures the global average information, this layer and it's operation can be represented in the following mathematical notation

$$i, j, k = \frac{1}{h^*w} \sum_{m=1}^{h} \sum_{n=1}^{w} input\,(i + m - 1,\ j + n - 1, k)$$

  Where

  h & w – height & width of feature map

  $\sum_{m=1}^{h} \sum_{n=1}^{w} input\,(i + m - 1,\ j + n - 1, k)$ – Input at a specific point [16]

- The **"Dropout"** layer is a randomized regularization technique where during the training, we set a randomly selected set of neurons activations to 0, meaning they are ignored (dropped out) for both forward and backward pass, this allows us to prevent co-adaptation of neurons and rewards the program to explore different methodologies of obtaining the desired result or keeping an open minded approach towards the solution. The output of the dropout layers during training can be expressed as

$$Output = input * mask * \frac{1}{1-rate}$$

Where

Input – is the input to the layer
Mask – a binary mask where dropped out neurons = 0 and active neurons = 1
Rate – dropout rate [16]

- **"ImageDataGenerator"** is a Keras utility for real-time data augmentation during model training. It allows us to enhance the diversity of our training dataset and improve the model's generalization capabilities, meaning it widens the spectrum of data variations which our model can learn from, this will later come in handy when we show our model images it hasn't seen before.

- **"ResNet152V2"** is a pre-trained deep learning model architecture available in Keras, and it will be used as a base in the custom model. It's one of the Residual Network's variants and is a proven method of training CNNs commonly in computer vision, object detection and classification tasks. The features of this ResNet variant and mathematical approach of this architecture are as follows.

  A common issue when training a CNN is the "vanishing gradient" effect, this happens during the training phase and means that the gradients being used to update the network are so small that it plateaus the model's learning rate. A solution to mitigate this effect is to equip the blocks with a "shortcut connection" that skips one or more layers. The shortcut output is achieved by adding the transformational output happening in the block to the initial input of that block.

  To reduce the computational cost of this approach, the convolutional layers are organized in the following manner (1x1—3x3—1x1), the end of this configuration often applies global average pooling where the spatial dimensions are reduced to 1x1, counting for only the average computed value across each feature map as mentioned earlier. [17]

- **"ModelCheckpoint"** and **"EarlyStopping"** are callback classes from Keras used throughout the training process to decrease the chances of overfitting.

- The **“ModelCheckpoint”** allows us to save the best model/version of our program observed during training at different epochs specifically for validation sets and allows us to call on that model when needed.
- The **“EarlyStopping”** allows us to stop the training process if a specific condition is met such as when a specific metric we’re optimizing for stops improving in a set period of tries, this is facilitated through the *‘Patience’* parameter. This prevents overfitting from occurring during the model’s training. [18]

● **“KFold”** from scikit-learn is used for cross-validation, estimating the skill of a model on unseen data, this is where we obtain our *‘val_accuracy’* parameter from. It splits our dataset into subsets of equal size known as folds and trains the model on the amount of those splits *‘k’,* using a single subset as the validation set while the model trains on every other dataset, this process is repeated for each subset i.e. the number of folds. The model’s average performance across all k-folds is used an estimate of it’s acquired skill level. [19]

2.2.2. Functions

- **create_resnet152V2_model():** This function creates a deep learning model based on the ResNet152V2 architecture consisting of a pre-trained ResNet152V2 base, followed by global average pooling, a dense layer with ReLU activation, dropout for regularization, and a final dense layer with softmax activation for classification.

- **compile_model(model):** This function configures and prepares the neural network model for training. It specifies the loss function, optimizer, and evaluation metrics. In our case, it uses the ‘sparse_categorical_cross-entropy’ loss, the Adam optimizer, and accuracy as the evaluation metrics.

- **setup_callbacks(model_name):** This function sets up callbacks for the training process. ‘EarlyStopping’ monitors the validation loss and stops training if no improvement is observed after a certain number of epochs. ‘ModelCheckpoint’ saves the best model weights based on the value of our validation loss.

- **get_image_data_generators(train_fold, val_fold):** This function prepares data generators for image processing during training and validation. It uses the ‘ImageDataGenerator’ to apply various data augmentation techniques such as zoom, rescaling and rotation , ensuring the model generalizes for more versatile data.

- **train_model(model, train_images, val_images, callbacks):** This function trains our neural network using the provided training and validation data generators. It returns the training history, which includes information about the loss and accuracy during each epoch.

- **evaluate_model(model, test_images):** This function evaluates the trained model on a separate set of test images. It returns metrics such as loss and accuracy, providing insights into the model's performance on previously unseen data and provides us our validation accuracy score.

The key components we get from executing this code is the following

- ✔ Create a neural network architecture based on the ResNet152V2 pre-trained base.
- ✔ Compile the model, specifying the loss function, optimizer, and evaluation metric.
- ✔ Set up callbacks for early stopping and model checkpointing during training.
- ✔ Prepare data generators with data augmentation for both training and validation.
- ✔ Train the model on the training data, monitor it using the validation data, save the best model weights and evaluate the trained model on a separate set of test images.

2.3 Data Visualization

Code

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_accuracy_loss(epochs, best_train_acc, best_val_acc, best_train_loss,
best_val_loss):
    plt.plot(epochs, best_train_acc, 'g', label='Best training accuracy',
linestyle='--', marker='o')
    plt.plot(epochs, best_val_acc, 'b', label='Best validation accuracy',
linestyle='--', marker='o')

    plt.title('Best Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

    plt.figure()
    plt.plot(epochs, best_train_loss, 'g', label='Best training loss',
linestyle='--', marker='o')
    plt.plot(epochs, best_val_loss, 'b', label='Best validation loss',
linestyle='--', marker='o')

    plt.title('Best Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def plot_accuracy_epochs(train_accs, val_accs):
    plt.figure(figsize=(8, 6))
    for fold, acc_values in enumerate(train_accs, start=1):
        plt.plot(range(1, len(acc_values) + 1), acc_values, label=f'Fold
{fold}', linestyle='--', marker='o')

    plt.title('Training Accuracy Across Epochs for Each Fold')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()


def plot_loss_epochs(train_losses):
    plt.figure(figsize=(8, 6))
    for fold, acc_values in enumerate(train_losses, start=1):
        plt.plot(range(1, len(acc_values) + 1), acc_values, label=f'Fold
{fold}', linestyle='--', marker='o')

    plt.title('Training Losses Across Epochs for Each Fold')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def plot_validation_accuracy_epochs(val_accs):
    plt.figure(figsize=(8, 6))
    for fold, acc_values in enumerate(val_accs, start=1):
        plt.plot(range(1, len(acc_values) + 1), acc_values, label=f'Fold
{fold}', linestyle='--', marker='o')

    plt.title('Validation Accuracy Across Epochs for Each Fold')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def plot_validation_loss_epochs(val_losses):
    plt.figure(figsize=(8, 6))
    for fold, acc_values in enumerate(val_losses, start=1):
        plt.plot(range(1, len(acc_values) + 1), acc_values, label=f'Fold
{fold}', linestyle='--', marker='o')

    plt.title('Validation Losses Across Epochs for Each Fold')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def plot_confusion_matrix(y_true, y_pred, class_names):
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=class_names)
    disp.plot(cmap='viridis', values_format='d')
    plt.title('Confusion Matrix')
    plt.show()
```
Figure 3: Data Visualization Code

2.3.1. Importing Libraries

- matplotlib.pyplot is a library for creating static, animated, and interactive visualizations in Python.

- sklearn.metrics provides functions for calculating various metrics, and confusion_matrix is used to compute the confusion matrix and detect data mislabeling.

2.3.2. Functions

- **plot_accuracy_loss(epochs, best_train_acc, best_val_acc, best_train_loss, best_val_loss):** This function creates two subplots, the first is to visualize the best training and validation accuracy and the second is to visualize the best training and validation loss across epochs.

- **plot_accuracy_epochs(train_accs, val_accs):** This function creates a plot to show the training accuracy across epochs for each fold when when we're validating across the data subsets of K-fold cross-validation.

- **plot_loss_epochs(train_losses):** This function generates a plot to illustrate and visualize the training losses across epochs for each fold in a K-fold cross-validation setup.

- **plot_validation_accuracy_epochs(val_accs):**
  Similar to the previous function that measures across epochs for each fold in K-fold cross-validation, this function visualizes the validation accuracy

- **plot_validation_loss_epochs(val_losses):** This function produces a plot showing the validation losses across epochs for each fold in a K-fold cross-validation setup.

- **plot_confusion_matrix(y_true, y_pred, class_names):** By utilizing sklearn.metrics.confusion_matrix and ConfusionMatrixDisplay, this function computes the confusion matrix

The key components we get from executing this code are the following

✔ Visualize the best training and validation accuracy, as well as the best training and validation loss across all epochs.

✔ Present the training losses, training accuracy, validation loss and validation accuracy across epochs for each fold in K-fold cross-validation.

- ✔ Compute and display the confusion matrix based on correct and predicted class labels.

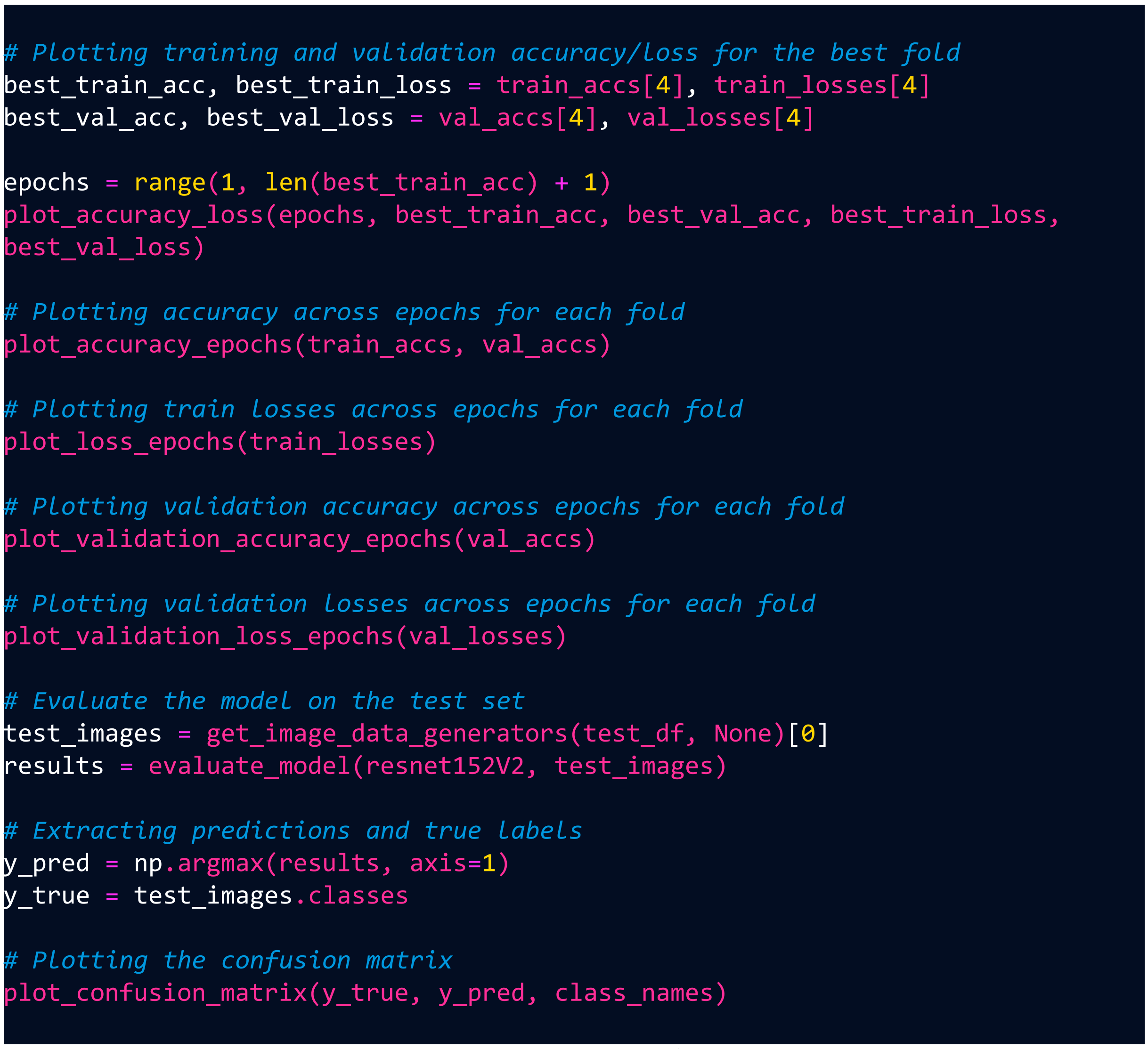
3. Execution & Results

In this chapter, we will run our main code (main.py), which entails the components of the 3 src files mentioned in the previous chapter in a functional manner to obtain our results. Throughout this chapter, we will be breaking down the compiled code that our model runs based on, as well as evaluating the results we have obtained after our model has been trained.

3.1 Main Code Breakdown

```
from src.data_processing import *
from src.model_building import *
from src.visualization import *

# Check if data directory exists
if not os.path.exists(DATA_DIR):
    print(f"Error: Directory '{DATA_DIR}' not found.")
    exit()

# Load dataset using TensorFlow's image dataset utility
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    DATA_DIR,
    seed=123,
    shuffle=True,
    image_size=(512, 512),
    batch_size=32
)

# Set image directory
image_dir = DATA_DIR

# Get filepaths and labels
filepaths, labels = get_filepaths_labels(image_dir)
# Create DataFrame with filepaths and labels
image_df = create_dataframe(filepaths, labels)

# Split the dataset into training and test sets
train_df, test_df = split_train_test_data(image_df)
```

```python
# Display class distribution
class_names = dataset.class_names
class_dis = [len(os.listdir(DATA_DIR / name)) for name in class_names]


# Display a bar plot of class distribution
display_class_distribution_barplot(class_names, class_dis)

# Create a transfer learning model (ResNet152V2)
resnet152V2 = create_resnet152V2_model()

# Compile the model
compile_model(resnet152V2)

# Set up callbacks for early stopping and model checkpoint
cbs = setup_callbacks("ResNet152V2")

# Initialize lists to store training/validation accuracy and loss for each
fold
train_accs, val_accs, train_losses, val_losses = [], [], [], []

# Initialize list to store fold-wise accuracy
fold_accuracy = []

# Set up KFold cross-validation
num_folds = 20
kf = KFold(n_splits=num_folds, shuffle=True)

# Loop through folds
for fold, (train_idx, val_idx) in enumerate(kf.split(train_df)):
    train_fold, val_fold = train_df.iloc[train_idx], train_df.iloc[val_idx]

    # Set up image data generators for training and validation
    train_images, val_images = get_image_data_generators(train_fold, val_fold)

    # Train the model
    history = train_model(resnet152V2, train_images, val_images, cbs)

    # Store accuracy and loss history for this fold
    train_accs.append(history.history['accuracy'])
    val_accs.append(history.history['val_accuracy'])
    train_losses.append(history.history['loss'])
    val_losses.append(history.history['val_loss'])

    # Evaluate the model on the current fold
    _, accuracy = evaluate_model(resnet152V2, val_images)
    fold_accuracy.append(accuracy)
    print(f"Accuracy for Fold {fold + 1}: {accuracy}")

# Calculate and print average cross-validation accuracy
avg_accuracy = np.mean(fold_accuracy)
print(f"Average Cross-Validation Accuracy: {avg_accuracy}")
```



```python
# Plotting training and validation accuracy/loss for the best fold
best_train_acc, best_train_loss = train_accs[4], train_losses[4]
best_val_acc, best_val_loss = val_accs[4], val_losses[4]

epochs = range(1, len(best_train_acc) + 1)
plot_accuracy_loss(epochs, best_train_acc, best_val_acc, best_train_loss,
best_val_loss)

# Plotting accuracy across epochs for each fold
plot_accuracy_epochs(train_accs, val_accs)

# Plotting train losses across epochs for each fold
plot_loss_epochs(train_losses)

# Plotting validation accuracy across epochs for each fold
plot_validation_accuracy_epochs(val_accs)

# Plotting validation losses across epochs for each fold
plot_validation_loss_epochs(val_losses)

# Evaluate the model on the test set
test_images = get_image_data_generators(test_df, None)[0]
results = evaluate_model(resnet152V2, test_images)

# Extracting predictions and true labels
y_pred = np.argmax(results, axis=1)
y_true = test_images.classes

# Plotting the confusion matrix
plot_confusion_matrix(y_true, y_pred, class_names)
```

Figure 4: Main Code

3.1.1. Importing Libraries

We will start of by importing modules from our src files, this methodology was followed to ensure that we have a modular and organized structure throughout the whole project. Each module (src.data_processing, src.model_building, and src.visualization) each contain functions and classes specific to data processing, model building, and visualization as discussed in the previous chapter.

3.1.2. Data Loading & Parameterization

We then move onto the data loading and parameterization, we will check if our specified data directory exists to ensure the availability of our data before moving to the next steps, if the data directory does not exist, our program will an error message and exit. Once the program verifies that the data exists, we load our data by using the image dataset utility provided from TensorFlow.

In order to have control over the loading process of the dataset, we will define the parameters **"seed", "shuffle", "image_size"** and **"batch_size"** The **"seed"** parameter allows our algorithm to produce the same results in each run using the random processes It encompasses by initializing the random number generator. The **"shuffle"** parameter is a parameter that determines whether the dataset should be shuffled before each epoch during each training session, this helps us avoid having the model train on the sequence of input samples instead of diversifying the set of examples in each batch to have a better sense of generalization. The **"image size"** parameter allows us to dictate the width and height dimensions to have a uniform input format that's suitable for our CNN, this also assists us in defining the input layer of the neural network. The **"batch size"** represents the number of utilized training in our examples in a single iteration (each forward & backward pass), this parameter influences our model's learning process. The advantage large batch sizes can sometimes lead to faster training but the downside is that is may require more memory, on the other hand, smaller batch sizes can increase our model's generalization capabilities but introduce noise into our training process.

3.1.3. Data Processing

During data processing we will use **"filepaths, labels = get_filepaths_labels(image_dir)"** to prepare the data by extracting file paths and labels from the directory. We will then create a Pandas data frame using the extracted filepaths and labels to handle our data in a structured format and make it easier for us to analyze later using **"image_df = create_dataframe(filepaths, labels)"**. We will then split our data into training sets and validation sets for our model to have a point of reference dictating how good our model's generalization capabilities are, this helps us in detecting whether the model is overfit or not, as well as make accurate predictions when exposed to unseen data. This process is carried out using **"train_df, test_df = split_train_test_data(image_df)".** The next step for us is to display the class distribution information which is obtained from the loaded dataset, it provides us insights into how balanced or imbalanced the class labels are and how many samples exist in each labeled class. This is performed using the code shown below

```
# Display class distribution
class_names = dataset.class_names
```

```python
class_dis = [len(os.listdir(DATA_DIR / name)) for name in class_names]
# Display a bar plot of class distribution
display_class_distribution_barplot(class_names, class_dis)
```

Figure 5: Class Distribution Code

3.1.4. Model Building & Training

Using **“resnet152V2 = create_resnet152V2_model()”** we will create a pre trained CNN model and then compile it using **“compile_model(resnet152V2)”.** We then set up our callbacks to control the training process and save the best model weights that got us the best results. The next step is to define our desired metrics lists, this will allow us to analyze the model’s performance during each fold and store the training and validation accuracy and loss, this is achieved by using **“train_accs, val_accs, train_losses, val_losses = [], [], [], []”.** To be able to set up K-Fold Cross Validation to give the model enough runs to achieve better results we will use **“num_folds = 20, kf = KFold(n_splits=num_folds, shuffle=True)”.**

Since our code loops through multiple folds, we have to ensure that each fold has all the necessary components to run as we desire, therefore, we instruct the program on how to set up data generators, what its history is composed of, what it should store as the metrics we are optimizing for that matter to us, how it is evaluated on its performance during this fold, how to present us that evaluation and lastly what is the average validation accuracy throughout the training process. The code used to execute this is as follows.

```python
# Loop through folds
for fold, (train_idx, val_idx) in enumerate(kf.split(train_df)):
    train_fold, val_fold = train_df.iloc[train_idx], train_df.iloc[val_idx]

    # Set up image data generators for training and validation
    train_images, val_images = get_image_data_generators(train_fold, val_fold)

    # Train the model
    history = train_model(resnet152V2, train_images, val_images, cbs)

    # Store accuracy and loss history for this fold
    train_accs.append(history.history['accuracy'])
    val_accs.append(history.history['val_accuracy'])
    train_losses.append(history.history['loss'])
    val_losses.append(history.history['val_loss'])

    # Evaluate the model on the current fold
    _, accuracy = evaluate_model(resnet152V2, val_images)
    fold_accuracy.append(accuracy)
    print(f"Accuracy for Fold {fold + 1}: {accuracy}")

# Calculate and print average cross-validation accuracy
```

```
avg_accuracy = np.mean(fold_accuracy)
print(f"Average Cross-Validation Accuracy: {avg_accuracy}")
```
Figure 6: Building & Training Code

3.1.5. Results Visualization

After the training of our model is complete, we want to obtain a visualized format of the metrics we have optimized for throughout the whole training process to help us get a better understanding of what the internal process of the model has looked like. We start off by generating plots of the best training accuracy and loss, as well as the best validation accuracy and loss for the single best fold using

```
# Plotting training and validation accuracy/loss for the best fold
best_train_acc, best_train_loss = train_accs[4], train_losses[4]
best_val_acc, best_val_loss = val_accs[4], val_losses[4]

epochs = range(1, len(best_train_acc) + 1)
plot_accuracy_loss(epochs, best_train_acc, best_val_acc, best_train_loss,
best_val_loss)
```

We then advanced onto plotting the training accuracy and loss as well as the validation accuracy and loss for each single fold using the following code

```
# Plotting accuracy across epochs for each fold
plot_accuracy_epochs(train_accs, val_accs)

# Plotting train losses across epochs for each fold
plot_loss_epochs(train_losses)

# Plotting validation accuracy across epochs for each fold
plot_validation_accuracy_epochs(val_accs)

# Plotting validation losses across epochs for each fold
plot_validation_loss_epochs(val_losses)
```
We also evaluated the model on the validation set as a reference point using

```
test_images = get_image_data_generators(test_df, None)[0]
results = evaluate_model(resnet152V2, test_images)
```
Then we extracted the predictions and true labels using

```
y_pred = np.argmax(results, axis=1)
y_true = test_images.classes
```
and finally, we plotted our confusion matrix using the extracted data to visualize the information using

```
plot_confusion_matrix(y_true, y_pred, class_names)
```

Our code follows a systematic and organized approach allowing us to load data, preprocess it, build our model, training it and evaluate the results.

3.2 Results

After executing our code, we have obtained the following results

3.2.1. Numerical Results

```
53/53 [==============================] - 125s 2s/step - loss: 0.6095 - accuracy: 0.7936 -
va36 - val_loss: 0.3107 - val_accuracy: 0.9091

Accuracy for Fold 1: 0.875                                   l_loss: 0.31

53/53 [==============================] - 120s 2s/step - loss: 0.3589 - accuracy: 0.8602 -
va02 - val_loss: 0.3600 - val_accuracy: 0.8864                       l_loss: 0.36

Accuracy for Fold 2: 0.875

53/53 [==============================] - 124s 2s/step - loss: 0.3340 - accuracy: 0.8728 -
val_loss: 0.2669 - val_accuracy: 0.8409

Accuracy for Fold 3: 0.8863636255264282

53/53 [==============================] - 121s 2s/step - loss: 0.2926 - accuracy: 0.8740 -
val_loss: 0.2598 - val_accuracy: 0.8977

Accuracy for Fold 4: 0.9431818127632141

53/53 [==============================] - 123s 2s/step - loss: 0.2646 - accuracy: 0.8950 -
val_loss: 0.2066 - val_accuracy: 0.9205

Accuracy for Fold 5: 0.9545454382896423

53/53 [==============================] - 121s 2s/step - loss: 0.2549 - accuracy: 0.8950 -
val_loss: 0.1888 - val_accuracy: 0.9318

Accuracy for Fold 6: 0.8863636255264282

53/53 [==============================] - 122s 2s/step - loss: 0.2450 - accuracy: 0.9022 -
val_loss: 0.39

Accuracy for Fold 7: 0.8972549152632175

53/53 [==============================] - 119s 2s/step - loss: 0.2239 - accuracy: 0.9154 -
val_loss: 0.2044 - val_accuracy: 0.9091

Accuracy for Fold 8: 0.9318181872367859                    54 - val_loss: 0.20

53/53 [==============================] - 125s 2s/step - loss: 0.2137 - accuracy: 0.9112 -
val_loss: 0.2503 - val_accuracy: 0.9205                    12 - val_loss: 0.25
```

```
Accuracy for Fold 9: 0.8977272510528564

53/53 [==============================] - 124s 2s/step - loss: 0.2331 - accuracy: 0.9010 -
val_loss: 0.2715 - val_accuracy: 0.8523

Accuracy for Fold 10: 0.8977272510528564

53/53 [==============================] - 123s 2s/step - loss: 0.2002 - accuracy: 0.9214 -
val_loss: 0.1839 - val_accuracy: 0.9205

Accuracy for Fold 11: 0.9318181872367859

53/53 [==============================] - 118s 2s/step - loss: 0.1979 - accuracy: 0.9220 -
val_loss: 0.1534 - val_accuracy: 0.8864

Accuracy for Fold 12: 0.9204545617103577

53/53 [==============================] - 119s 2s/step - loss: 0.2108 - accuracy: 0.9172 -
val_loss: 0.1217 - val_accuracy: 0.9659

Accuracy for Fold 13: 0.9431818127632141

53/53 [==============================] - 120s 2s/step - loss: 0.1965 - accuracy: 0.9178 -
val_loss: 0.2072 - val_accuracy: 0.9205

Accuracy for Fold 14: 0.9090909361839294

53/53 [==============================] - 125s 2s/step - loss: 0.1970 - accuracy: 0.9166 -
val_loss: 0.1900 - val_accuracy: 0.9091

Accuracy for Fold 15: 0.9545454382896423

53/53 [==============================] - 122s 2s/step - loss: 0.1613 - accuracy: 0.9394 -
val_loss: 0.1729 - val_accuracy: 0.9310

Accuracy for Fold 16: 0.9425287246704102

53/53 [==============================] - 122s 2s/step - loss: 0.1815 - accuracy: 0.9287 -
val_loss: 0.1580 - val_accuracy: 0.9195

Accuracy for Fold 17: 0.954023003578186

53/53 [==============================] - 122s 2s/step - loss: 0.1614 - accuracy: 0.9305 -
val_loss: 0.1802 - val_accuracy: 0.9310

Accuracy for Fold 18: 0.9195402264595032

53/53 [==============================] - 122s 2s/step - loss: 0.1720 - accuracy: 0.9329 -
val_loss: 0.1466 - val_accuracy: 0.9195

Accuracy for Fold 19: 0.954023003578186

53/53 [==============================] - 126s 2s/step - loss: 0.1595 - accuracy: 0.9311 -
val_loss: 0.2385 - val_accuracy: 0.8851
```

Accuracy for Fold 20: 0.9195402264595032

Average Cross-Validation Accuracy: 0.9185736656188965

3.2.2. Mean Class

Using the “class_dis” variable, we are able to calculate the mean image per class by counting the files in each specified subdirectory in the directory “DATA_DIR”, the distribution is visualized using the below presented bar plot where we have the class names displayed on the x-axis and the corresponding image count of that directory on the y-axis. We have a low image count in the benign cases class causing an imbalance that potentially can impact the performance of our model, we have used techniques such as rescaling, rotating, flipping and width and height shift to mitigate the effect of imbalance and bias during our training and help the model generalize better.

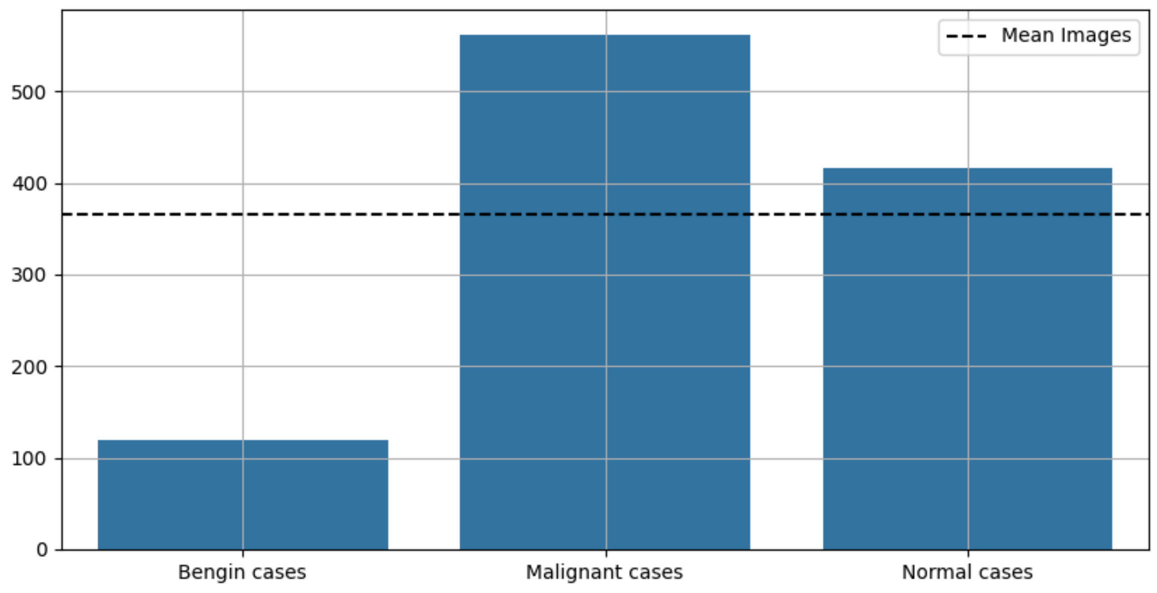


Figure 7 : Class Distribution Result

3.2.3. Best Training & Validation Accuracy & Loss

Throughout all of our folds, the best training accuracy we have achieved was just under 90% and the best validation accuracy achieved was just over 92%. The training accuracy is an indication that our model is able to learn & classify examples of images correctly while the validation accuracy signals that our model’s is able to generalize on data it has not seen before. Both of our training and validation accuracies closeness in value prove that our model was not subject to overfitting throughout the process. This is visualized in the illustration down below

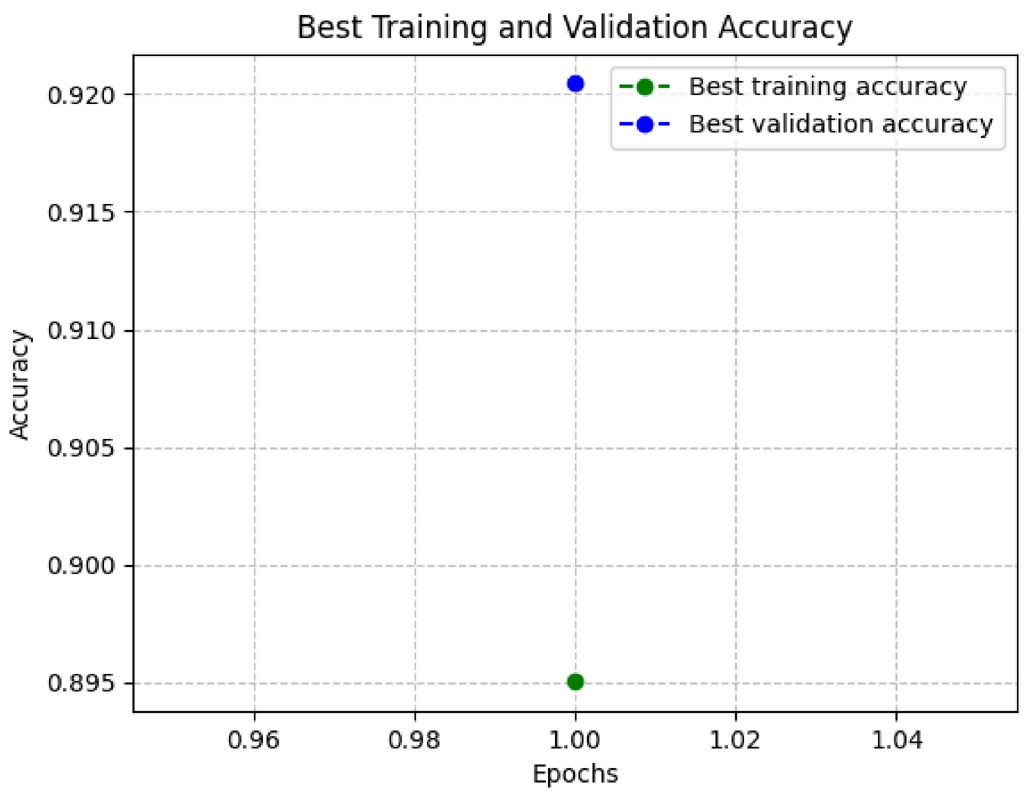


Figure 8 : Best Training and Validation Accuracy

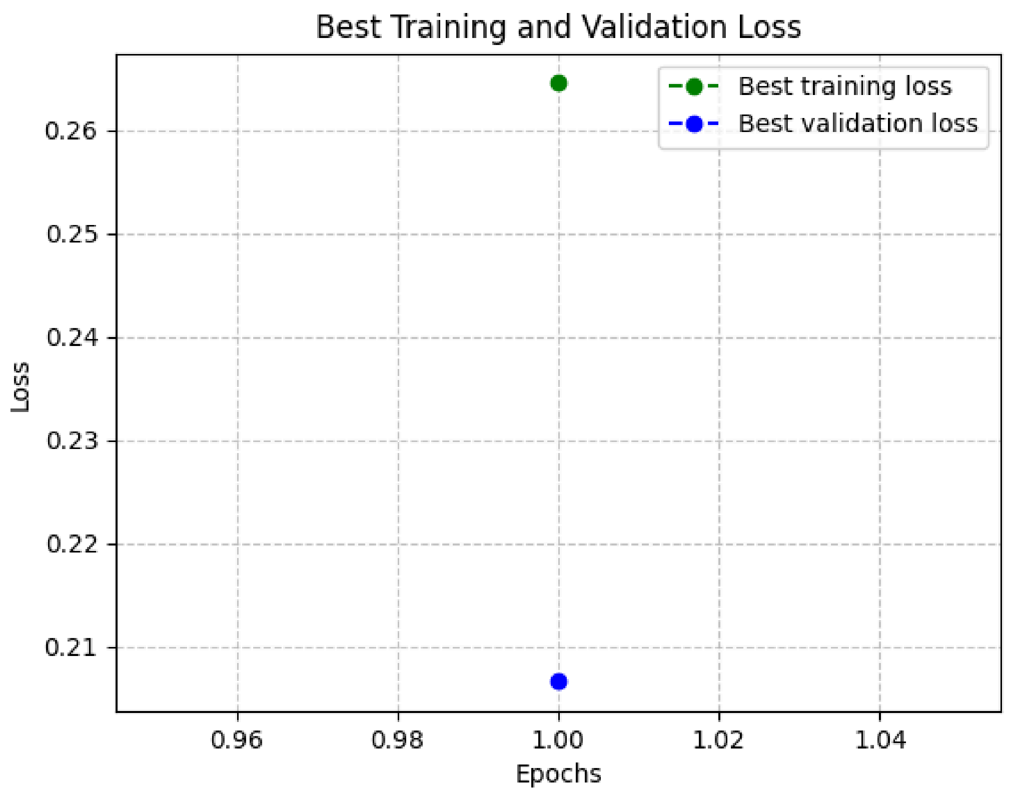


Figure 9 : Best Training and Validation Loss

From the image shown above, we can see that the best training loss reached in for a single fold was around 0.265, and the best validation loss achieved was just above 20.5. In terms of training loss, these results directly correlate the difference between the predicted result and the actual results during that training run, the lower the training loss is, the better the fit of the model. As for the validation loss, it indicates how well our model has performed on unseen data within that run. If the values of both the training and validation loss have a large difference between them, this would indicate that the possibility of overfitting, this is not the case in our model.

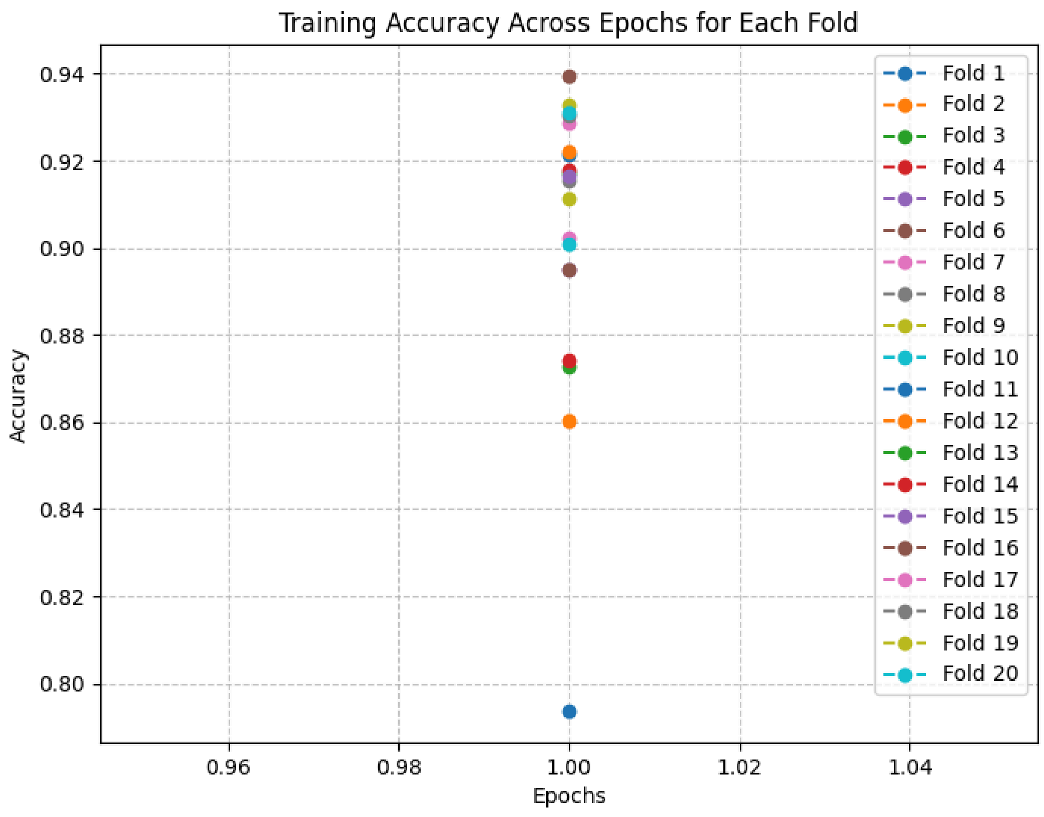


Figure 10 : Best Training Accuracy for Each Fold

The result presented above shows how our training accuracy has acted over the course of all folds, we can see that the first fold had a significantly lower accuracy compared to all that came after it due to the fact that its initial weights are not optimized, the optimization process takes place as the folds increase. The average range of our training accuracy was around the 92% mark, with the highest reaching up to 94% and the lowest being 86% excluding the first fold.

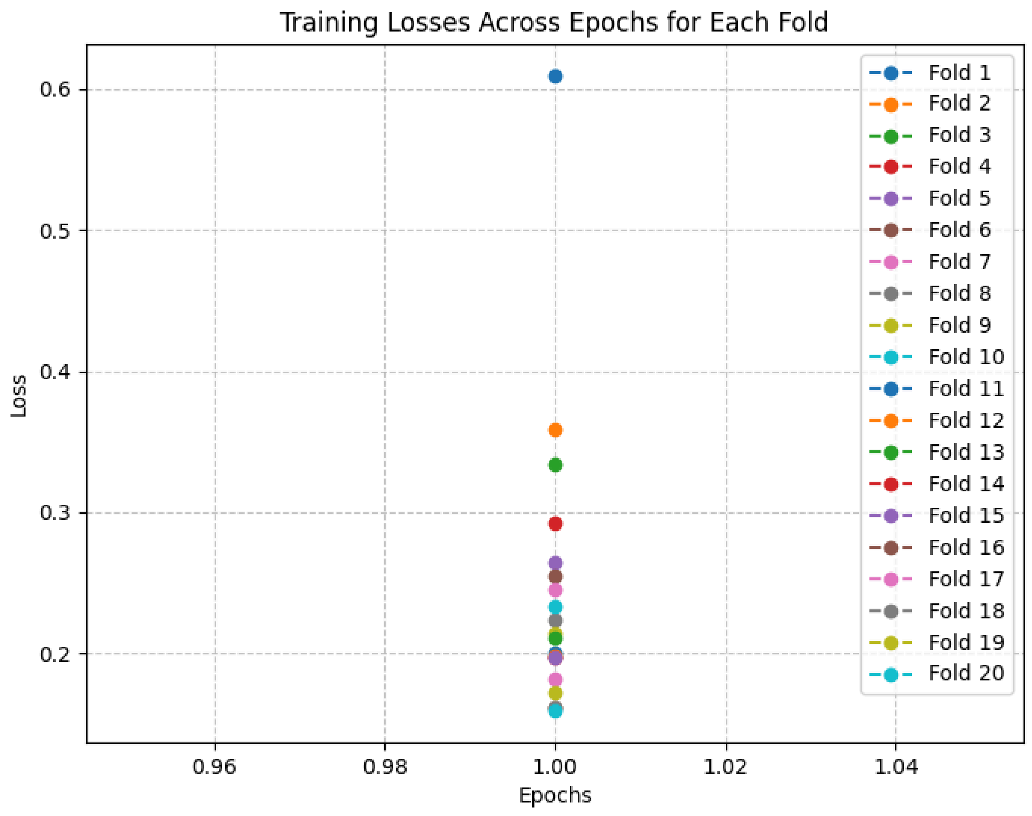


Figure 11 : Best Training Loss for Each Fold

As for our training loss, we can observe a similar pattern where the first fold's training loss was significantly higher than that of the rest. This can be attributed to the randomness introduced in the model from the initial data splitting which may be not fully representative of the entire dataset leading to a less effective learning process for that specific fold. The best and observed training loss was close to 0.1 and took place in the last fold, while the highest and worst training loss was observed in the 2nd fold excluding the 1st. This is an indication that throughout the training process, the model has gradually progressed in a positive manner towards our desired results, ensuring that it performs better on the provided dataset.

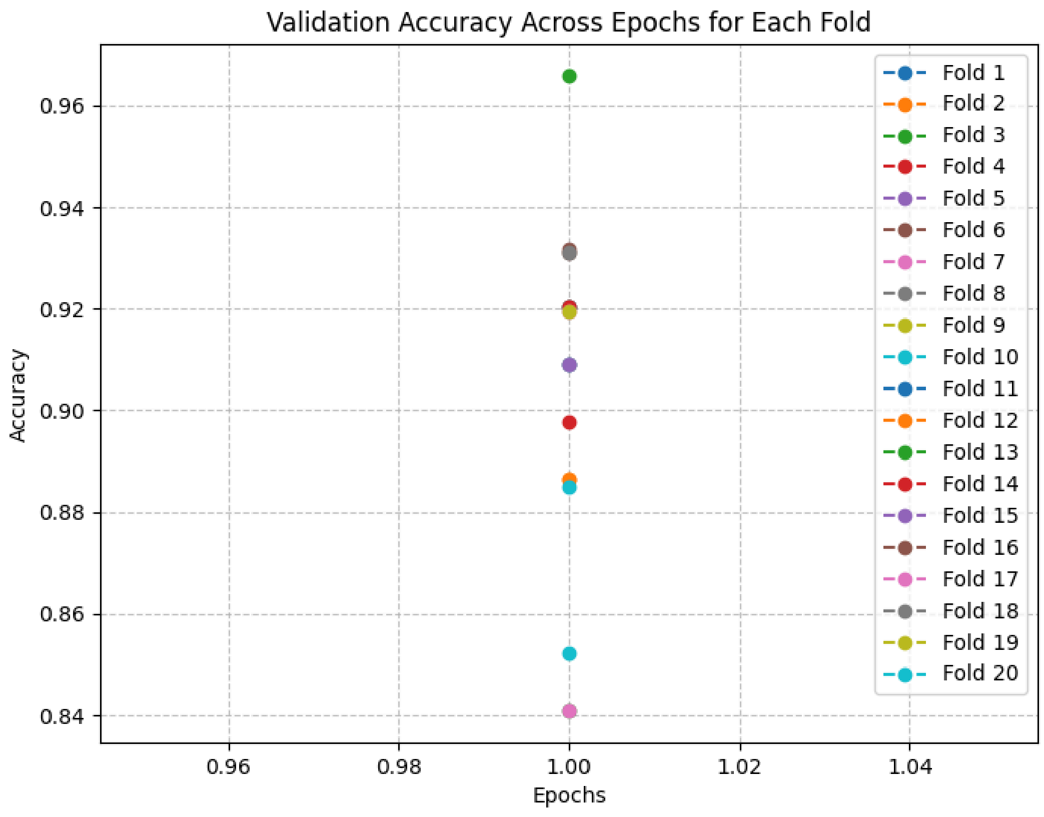


Figure 12 : Best Validation Accuracy for Each Fold

Moving onto the validation accuracy for our of our folds, the patter is observed to be a little different and spread out over the Y-axis in comparison to the training accuracy. This suggests that our model generalizes well to new data and that the patterns it has learned are relevant to analyzing images it previously hadn't seen. This also means that during the training process, our model was exposed to different patterns each time which leads to it not being highly dependent on a specific subset of data to be able to recognize images. The highest validation accuracy achieved was as high as above 96%, this took place at the 13th fold, while the lowest achieved accuracy was 84% and that took place at the 3rd fold.

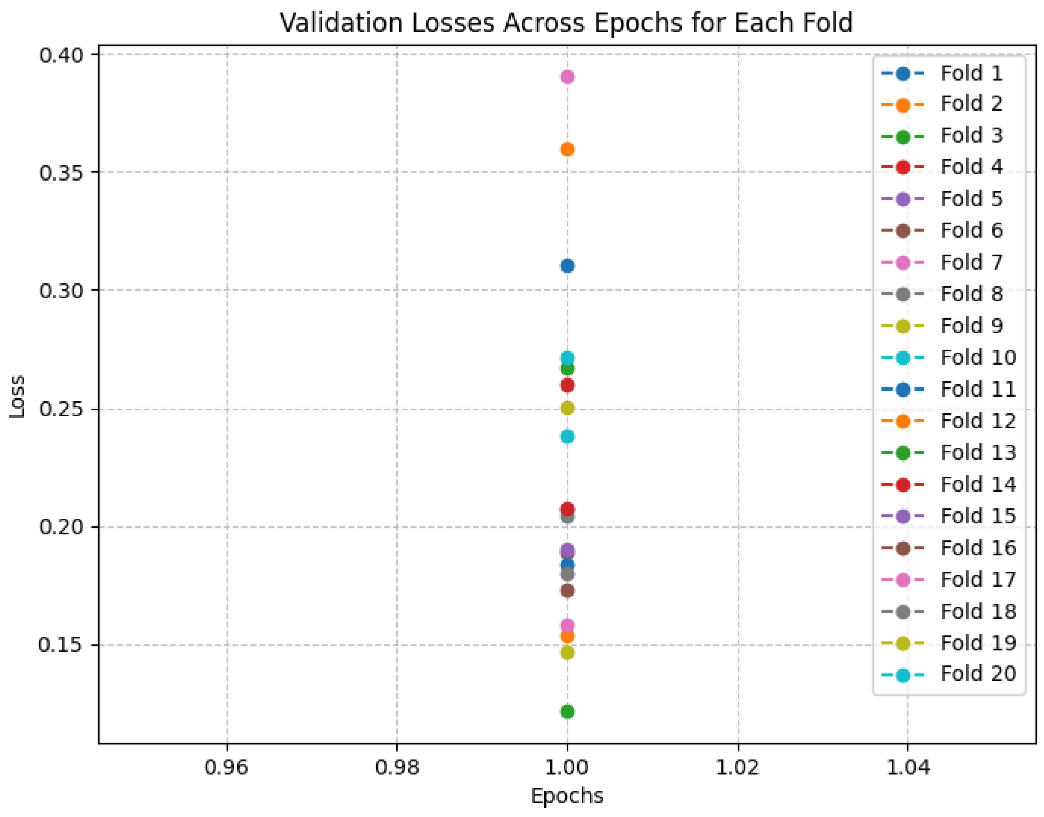


Figure 13 : Best Validation Loss for Each Fold

Similar to the results of the validation accuracy, our validation loss is fairly spread out over the course of the 20 folds, with the highest density and most recurring losses being on the lower end of the y-axis, this is an indication that our model has been encountering different variations of data across each fold and has successfully generalized to new data not keeping a strict criteria to a specific subset. The lowest and best validation loss was observed in the 13th fold at a value of 0.12, while the highest and worst validation loss was observed in the 7th fold at a value of 0.39.

All in all, throughout this segment, we have reaped the fruits of our model's training, we have displayed the desired metrics such as training accuracy and loss as well as validation accuracy and loss in a visualized format. We have also analyzed each fold on run-by-run basis and obtained the average validation accuracy of these runs.

4. Summary & Conclusion

In conclusion, this thesis has been centered around the process of developing and evaluating a convolutional neural network with the purpose of processing images of CT lung scans allowing us to distinguish between three classes of data “normal” ”benign” and ”malignant”. We started off by defining some of the main concepts of AI & machine learning as well as going into details of the ones that apply to our model. When it came to developing our model and execution, we designed our architecture by leveraging the ResNet152V2 pre-trained model, this choice of the model allowed us to harness the power of a well-established CNN and mold it to fit our image processing use case. The ResNet152V2’s structure was further enhanced by the modular approach we took to build our code by organizing it into data processing, model compiling and visualization segments. The loading and preprocessing phases of our dataset was a crucial step to ensure the data is handled in the appropriate desired manner, we utilized TensorFlow’s image dataset utility to facilitate this process, we specified out image’s parameters such as seed, image size, shuffle, and batch size to control the loading mechanism of images into the model. We also created a structured Padas DataFrame by splitting the data into training and validation sets for each run.

Throughout our training, we incorporated the K-Fold Cross-Validation technique to evaluate our model’s generalization capabilities when faced with different data subsets, this approach involved setting up image data generators, the training instructions themselves and per epoch, as well as the storing of our results for each fold, paired with call backs for early stopping purposes, we were able to ensure the model did not encounter overfitting and was only getting better throughout the training procedure, we also saved the best folds to later be used in the visualization of our results. After executing this whole process, we obtained promising results, with the average cross validation across all folds reaching almost 92% with the highest fold reaching up to 96%. This suggests that our approach to resolve the task at hand is correct but still has room for further iterations and improvements. For the visualization part of our results, we have successfully plotted and delivered results of our best training and validation accuracy and loss results in a single fold, as well as results of every single fold throughout the training. The close alignment of our results indicates that our model has good generalization capabilities and was able to mitigate overfitting risks.

In conclusion, despite the model reaching good results, there is still room for future enhancements and improvements, this would start by increasing the dataset size specifically in the benign class, we can also fine-tune hyper parameters and incorporate more data manipulation techniques to increase the generalization capabilities of our model. Our study has laid the groundwork for an effective image processing model in the medical field, as time goes on and the machine learning technology evolves, time will tell how much of an impact it will have on our daily lives in crucial fields of work like medicine.

5. References

5.1 Literature References

[1]https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence.......................

[2]https://www.spiceworks.com/tech/artificial-intelligence/articles/narrow-general-super-ai-difference/amp/...

[3]https://www.coursera.org/articles/types-of-ai..

[4]https://builtin.com/artificial-intelligence/examples-ai-in-industry...

[5]https://en.wikipedia.org/wiki/Artificial_intelligence...

[6]https://www.coursera.org/articles/machine-learning-algorithms...

[7]https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b...

[8]https://arxiv.org/abs/2309.00267v2...

[9]https://www.javatpoint.com/deep-learning-algorithms..

[10]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8970870/..

[11]https://www.foreseemed.com/artificial-intelligence-in-healthcare..

[12]https://www.sciencedirect.com/science/article/pii/S2666379123000253.....................................

[13]https://jhoonline.biomedcentral.com/articles/10.1186/s13045-023-01456-y................................

[14]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10650618/..

[15]https://keras.io/guides/sequential_model/...

[16]https://www.sciencedirect.com/science/article/pii/S0925231218310610.....................................

[17]https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8...

[18]https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/...

[19]https://machinelearningmastery.com/k-fold-cross-validation/..

5.2 Figure References