An Explainable Deep Learning Method for the Classification of Diabetic Retinopathy

Submitted May 2022, in partial fulfillment of
the conditions for the award of the degree: **MSci Computer Science with Artifical Intelligence.**

20160230

Supervised by Dr Xin Chen

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature THC

Date 25 / 04 / 2022

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Public access to this dissertation is restricted until: 25/04/2022

Abstract

Diabetic Retinopathy is a common complication of Diabetes that often leads to blindness if left untreated. Early detection is, therefore, paramount. It affects the eye, however is often hard to spot. To diagnosis, a ophthalmologist must spot small haemorrhages or exudates in 2D Fundus images. Deep learning can be utilised in order to assist the ophthalmologist and help reduce misdiagnosis.

This paper details a deep learning method that uses an efficient convolutional neural network, InceptionResnetV2 as well as an explainability approach called GradCAM++. These will be utilised to detect and highlight symptoms of differing severities of Diabetic Retinopathy in 2D fundus images. GradCAM++ has not previously been used for Diabetic Retinopathy, and due to improved detection of multiple occurrences of a class in an image, ophthalmologists will receive greater assistance in their diagnosis than other explainability approaches. The application itself has a fluid and simple design that quickly points out possible anomalies in an input image for use during the appointment with the patient.

Keywords: deep learning; diabetic retinopathy; explainability; gradcam; tensorflow; fundus imaging

Acknowledgements

I would like to thank my supervisor, Dr Xin Chen. His guidance was very important to the project and the knowledge I learnt was very valuable in bringing the project to life.

Contents

Abstract **i**

Acknowledgements **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 4

1.3 Aims and Objectives 4

1.4 Structure of Dissertation 4

2 Related Work **6**

2.1 Convolutional Neural Networks 6

2.2 Other Approaches to Explainability 7

3 Design **9**

3.1 Data Collection 9

3.2 Model Design 10

3.3 Application Design 13

3.4 Possible Extensions 13

4 Implementation **16**

4.1 Dataset 16

4.2 Data Pre-Processing 16

4.3 Neural Network Implementation 19

4.3.1 Training the Network 19

4.3.2 GradCAM++ Implementation 21

5 Evaluation **24**

5.1 Model Performance 24

5.2 Explainability Approach Evaluation 26

5.3 Application Evaluation 28

6 Summary and Reflections **30**

6.1 Project management 30

6.1.1 Resource Management 31

6.2 Contributions and reflections 32

6.2.1 Contributions 32

6.2.2 Reflections 32

Bibliography **33**

List of Tables

1.1 Features related to a severity level of Diabetic Retinopathy 2

3.1 Initial Dataset Changes 10

4.1 Training, Validation and Test dataset structures 16

4.2 Data Augmentation Values 18

4.3 Transfer Learning HyperParameter Results 20

4.4 Fine Tuning HyperParameter Results 20

4.5 Metrics of the model on the Test Dataset 21

5.1 The Hyper Parameters for the working GradCAM++ implementation 25

5.2 GradCAM++ vs GradCAM for differing severity levels of DR 27

List of Figures

1.1 A 2D Fundus image that illustrates MAs, HMs and HEs[1] 3

3.1 InceptionResNetV2 11
3.2 InceptionResNetV2: Inception-Resnet-C Module 12
3.3 GradCAM implementation 13
3.4 GradCAM++ Implementation 13
3.5 GradCAM++ vs GradCAM 14
3.6 UI Design 14
3.7 Patho-GAN 15
3.8 UI Design (with extensions) 15

4.1 An example of Otsu Thresholding Cropping 17
4.2 Data Augmentation Output 18
4.3 Overview of Modified Model for GradCAM++ 19
4.4 A step-by-step example of the bounding box algoirthm 22
4.5 The straightforward user interface for the project 23

5.1 Model Confusion Matrix 25
5.2 Two Examples of Applying GradCAM++ on a PDR Image 26
5.3 Graph Mode vs Eager Mode Execution for the GradCAM++ Algorithm 29

6.1 GANTT Chart showing proposed project tasks and schedule 31

Chapter 1

Introduction

1.1 Motivation

Diabetic Retinopathy (DR) is a common complication of diabetes, caused by high blood sugar levels damaging the retina. It can cause blindness if left undiagnosed and untreated[2]. DR is caused when glucose levels and blood pressure are consistently high, damaging the blood vessels[3]. As the eye has a large number of blood vessels supplying blood to the retina, any disruption of these vessels will likely result in vision loss and eventually blindness. According to the US National Eye Institute in 2010, 5.4% of Americans had DR[4]. DR can be treated by changing one's lifestyle; losing weight, cutting alcohol consumption, quitting smoking, exercising regularly, if discovered early enough. This will slow down the progression of the disease and help keep the patient healthier for longer. As DR progresses, however, it is necessary to treat with lasers, eye injections or invasive eye surgery. According to the NHS (UK National Health Service) website [2], all of these options could lead to other complications for the patient; further bleeding in the eye, cataracts, permanent blind spots and more surgery. It is important to detect DR early to avoid these unnecessary complications.

There are two main types of DR. Non-proliferative diabetic retinopathy (NPDR) and proliferative diabetic retinopathy (PDR). In the early stages of DR, it is referred to as NPDR. NPDR has three different stages depending on the presence and quantity of certain DR features. The first stage is mild NPDR - the patient has one micro-aneurysm (MA), which occurs from the weakening of capillary walls. Stage 2 is moderate NPDR - patients have haemorrhages (HM) or MAs in multiple retinal quadrants. HMs form when MAs rupture. The patient also might

have some hard exudates (HE), which are formed when the weakened capillaries leak fluid into the retina and hardens. Severe NPDR (Stage 3) is when there are multiple HMs in each quadrant (> 20). As DR becomes more serious, PDR can occur. During PDR, is when irregular blood vessels (neovascularization) grow at the rear of the eye [5]. Table 1.1 outlines the stages. Furthermore, Fig. 1.1, shows what some of these symptoms look like in a 2D fundus image.

Table 1.1: Features related to a severity level of Diabetic Retinopathy

Class Number	Severity Level	Features related to DR
0	No DR	• No features
1	Mild NPDR	• 1 MA, but no other findings
2	Moderate NPDR	• More than 1 MA • HEs • Venous Beading
3	Severe NPDR	• HMs (> 20) and MAs in each retinal quadrant • Venous Beading
4	PDR	• Neovascularization of the disc/elsewhere • Vitreous/Preretinal Haemorrhage - retinal blood vessels break down • Retinal Detachment

DR if left untreated will lead to blindness, therefore, early detection is paramount. Unfortunately, it is the case that sometimes ophthalmologists will diagnose the wrong severity level of DR. Often patients will only have their eyes evaluated once or twice a year (depending on their current severity of DR), meaning an inaccurate diagnosis could be extremely harmful. According to Krause et al (2018 [6]), US-board certified ophthalmologists will frequently miss MAs or misclassify HMs during diagnosis, due to the limited feature difference in severity levels. This misdiagnosis may result in blindness depending on the rate of advancement of the disease. This identifies the opportunity for a deep learning model that can not only classify an image into its DR severity level but also highlight MAs, HMs, and HEs. This can help ophthalmologists understand why the models classification differs from their initial diagnosis. Furthermore, if the model is utilised to highlight key areas of importance to the classification, this feedback will facilitate trust in the model results and foster a continued learning opportunity for ophthalmologists.

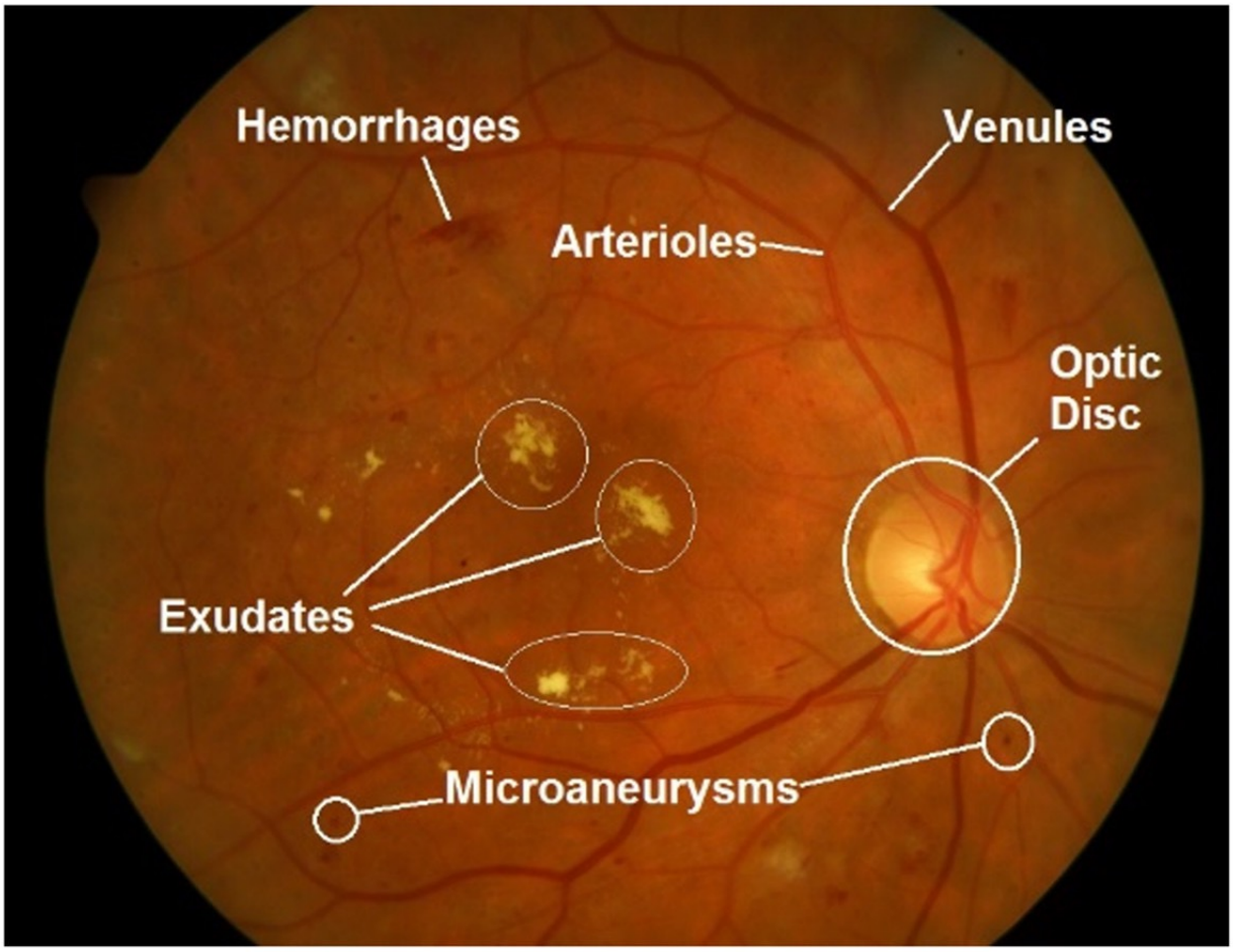


Figure 1.1: A 2D Fundus image that illustrates MAs, HMs and HEs[1]

This project focuses on building that exact model. A fine-tuned InceptionResNetV2 model with a custom output layer will be used to determine the severity level of an input image. The data used to train this model is collected from the Kaggle Dataset (a publicly available dataset), and trimmed to make for a more balanced dataset. An explainability approach will have to be applied in order to point out of the clinical features of DR. There are a lot of explainabilty approaches used to work out the reasoning behind a CNNs decision. A very common approach used is GradCAM by Selvaraju et al. (2017 [7]), which uses the gradients of any target concept, flowing into the final convolutional layer to produce a coarse localization map highlighting important regions in the image for predicting the concept. However, GradCAM struggles to detect multiple instances of a class within the images and as such, Chattopadhay et al (2019, [8]) developed GradCAM++. For this reason, an implementation of GradCAM++ will be designed for the InceptionResNetV2 which will produce a heatmap for ophthalmologists to easily recognise the problem areas in the Fundus image.

1.2 Description of Work

The goal for this project is to build a application which can run on an ophthalmologist's computer. During an appointment with the patient, a doctor can feed the Fundus images to this application. The computer should quickly compute the result of running the image through both the model and the explainability approach, then display a probable DR severity level, with a heatmap over the Fundus image highlighting the symptoms of DR within the image. Ophthalmologists can then quickly make a decision to decide how to diagnose the patient.
This project will add a more optimal explainability approach to deep learning for diabetic retinopathy detection and classification, as well as a usable application that applies the model to new images. I have detailed some objectives to show exactly how this will be achieved

1.3 Aims and Objectives

Aim: **To build a system capable of determining the severity level of DR in a 2D fundus image and to locate the abnormalities in the image that led to this decision with greater precision than previous work.**
This aim can be completed by breaking it down into this objectives:

- Design and develop a model that can take a 2D fundus image as an input and output a severity level.
- Evaluate model to confirm that it makes the correct decision.
- Add the explainability approach to the application that utilises the previously defined model.
- Test the explainability method against previous methods to prove or disprove it's optimality.
- Build a user interface for doctors to use that applies the model to an image, and displays the output of the explainability method.

1.4 Structure of Dissertation

The dissertation structure is as follows, Chapter 2 is dedicated to reviewing existing work in the field of explainability approaches and Diabetic Retinopathy. Chapter 3 will detail the design choices behind all areas of my application, including the data collection, model design and ex-

plainability approach. Chapter 4, will discuss how the designs have been implemented including the problems encountered during implementation. The performance evaluation of the software will then be laid out in Chapter 5. Finally, Chapter 6 will contain the Summary and Reflections on the project as a whole.

Chapter 2

Related Work

This section provides a review of other research papers and studies that are in the same field as this paper - to improve DR detection and classification using deep learning, whilst also explaining how the model produced a decision.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are very common in computer vision tasks. This section discusses the uses of CNNs in DR detection.

Chetoui and Akhloufi (2020 [9]) conducted a study on DR detection using CNNs with Grad-CAM. This study used an InceptionResNetV2 model which yielded high quality metrics - Area Under Curve (AUC) (0.986), sensitivity (0.958), and specificity (0.971). This paper uses an interesting technique, transfer learning and fine tuning - the fully connected layer of Inception-ResNetV2 is replaced with a bespoke fully connected layer tailored to the needs of the dataset and then the base model is fine tuned. Chetoui et al., studied the difference between different levels of fine-tuning. These results showed that fine-tuning is essential in a task such as DR detection,the non fine-tuned model had inferior performance. The optimal fine-tuning algorithm was performing fine tuning with cosine learning rate decay with warm up.

A notable usage of a CNN is Shorfuzzaman et al. (2021 [10]). In this study, an ensemble CNN was used to produce the deep learning model. Using 4 'backbone' CNNs, Shorfuzzaman et al. performed the ensemble of average model weights taken from each 'backbone' CNN. The resultant average weights were applied to a new CNN to produce an ensemble CNN. This

produced better results than a regular CNN due to the different model architectures of the 'backbone' CNNs. The results had a high degree of precision (0.970), sensitivity (0.980), and AUC (0.978) on a DR dataset. Furthermore, the explainability approach used was built using GradCAM and SHapley Additive exPlanations (SHAP) to highlight the areas of the fundus images that are most important to a DR classification.

To improve the adoption of Deep Learning in the medical field, Singh et al. (2021 [11]) sought to evaluate several explainability approaches for CNNs. The DeepTaylor approach (a method based on Taylor expansions which decomposes the output of a deep neural network in terms of input variables) received the highest ranking from their group of explainability approaches, with a median grade of 3.8/5. Furthermore, SHAP was also rated highly in this paper; SHAP combines with GradCAM well, as shown by the previously discusssed study by Shorfuzzaman et al. [10], making this potentially useful when thinking about extensions for GradCAM++.

Wang et al. (2019, [12]) used a different approach in their study. Wang et al designed their own CNN, and used a novel explainability approach called Regression Activation Mapping (RAM). With one exception, RAM is extremely similar to Class Activation Mapping (CAM) (a more particular form of GradCAM that only applies to a few specified model architectures). RAM displays all aspects that contributed to the classification decision, not just those connected to a class, whereas CAM uses the class score and only emphasises instances of that specific class.

2.2 Other Approaches to Explainability

Other techniques to explainability do not attempt to deduce the reasoning behind a CNN's output. Although these have not been utilised in the current project, they may add more perspectives to the ophthalmologist in order to construct an application with greater detail and functionality. Generative Adversarial Networks (GANs) are used in unsupervised learning and consist of a generator and a discriminator training each other to enable the networks to produce almost identical images to the dataset they were trained on. Due to the expertise of the personnel required to label photos in certain fields of medicine, unsupervised learning is beneficial. This section details some ways GANs have been used as explainability approaches in the detection of DR.

GANs have been used in different, interesting ways to provide some explainability behind the decisions of a neural network. Firstly, Niu, Gu, Zhao, and Lu (2022 [13]), built a novel architecture called 'patho-GAN' (using a technique inspired by Koch's Postulates [14]). They used an activation network based on Guided Back-Propagation, which encodes the impact of each neuron on the final prediction, allowing the network to locate lesions (MAs, HEs, and HMs, etc.) with a descriptor set D(x). This descriptor of lesions can be passed into a Generator Network (trained with adversarial learning), to produce images with a specific number of lesions. These synthesized images can be fed back through the DR detection network and the predictions can be studied to explain how different numbers of lesions affect the final prediction. Schlegl et al (2019 [15]) used GANS in a novel way. In this case, the architecture was called 'f-anoGAN'. The process behind this was to train a GAN with healthy OCT images and produce an encoder that could map new images to the GANs latent space. If a new unhealthy image was passed to the model, an anomaly detection technique can be used for pixel-level anomaly detection, therefore pointing out the lesions in the unhealthy image. The 'f' in 'f-anoGAN' stands for fast and is intended to show that the encoder can map data quickly to latent space and therefore this model can be used in real-time, which would be useful to ophthalmologists. Using a GAN-based explainability method could also be useful to suport this project after having applied GradCAM++, such as using 'patho-GAN'.

Another way in which explainability in deep learning has been approached is through fuzzy inference. Bonanno et al. (2017 [16]) proposed a possible approach. They want to embed fuzzy inference systems into deep learning networks. Fuzzy-inference systems allow for complex non-linear problems to be approximated using if-then statements. They are trying to extend deep learning by allowing it to operate unsupervised and to select the desirable features as defined by fuzzy-inference systems. These features can then be interpreted by a fuzzy-inference system offering explanations for why a classification label has been selected by the system. Although this is an interesting approach, it is currently only hypothetical, and therefore not useful for this project. However, future work could include incorporating this into the ophthalmologists application.

Chapter 3

Design

3.1 Data Collection

The dataset used for this project is the Kaggle dataset [17]. This dataset contains approximately 35,000 2D Fundus images. With this number of images, it is important to confirm that most of the images are good quality and viable for use in training a neural network. Zhou et al. (2019 [18]) show proof that most of the images are good quality, with the ratio of poor to good quality images being 0.026. A large dataset like this would be useful to reduce overfitting on my model, however, in its current state this dataset is greatly unbalanced. Therefore, this data is unusable because the model would struggle to pick classes other than the majority class. As shown by Table 3.1, 25,809 images were of class 0 in the original dataset. This greatly outnumbers the number of images in classes 1,2,3 and 4 combined. Some images were randomly removed from class 0 to make the dataset more balanced. Furthermore, upon further testing, the model struggled to differentiate between the classes 1 and 2. Therefore, classes 1,2 and 3 were combined into one class, labelled "Non-Proliferative Diabetic Retinopathy". An approximately equal number of images were randomly taken from each class to combine them into the single class. The model struggled because it could not tell the exact number of HMs and MAs that corresponded to each class, as it is quite a fine margin. The quantity and type of features in in class only differs slightly. It is likely that the more subtle changes between the NPDR classes will be shown via the explainability method with GradCAM++ being improved at detecting multiple occurrences of the class. The ophthalmologist will be able to predict the severity level easily in this case, due to how anomalies will be shown on the application. The structure of the new dataset can be seen in Table 3.1.

Table 3.1: Initial Dataset Changes

Dataset	Class ID	Class	Number Of Images
Original	0	No DR	25,809
	1	Mild NPDR	2,443
	2	Moderate NPDR	5,292
	3	Severe NPDR	873
	4	PDR	708
	-	Total	**35,125**
New	0	No DR	990
	1	NPDR	954
	2	PDR	708
	-	Total	**2,652**

3.2 Model Design

Convolutional Neural Networks (CNNs) are very useful in the field of Computer Vision. These models require 3 basic components to define them: the convolutional layer, the pooling layer and the output layer. The convolutional layer applies the mathematical operation of a convolution to the input image and a filter. This produces a feature map, the most basic of which will be information about edges and corners in the image and the most advanced will be features specifically related to the dataset the model has been trained on. Most of the time the convolutional layer will be followed by a pooling layer which reduces the dimensions of the feature map by combining elements. For example, Max Pooling takes the largest element from a set of elements. The output layer consists of weights and biases and is used for classification usually forming the last few layers of a CNN architecture. The input image from the last layer is flattened and fed to the output layer [19].

Different CNNs have a different number and combinations of convolutional layers, pooling and output layers. Inception [20] and ResNet [21] show incredible efficieny in computer vision tasks. The Inception architecture utilises Inception modules, which allow the network to choose between multiple convolutional filter sizes in each block. Benefits include the ability to extract feature data at varying scales through to utilisation of varying convolutional filter sizes [22]. ResNet introduced a novel architecture with 'skip (residual) connections' and features heavy batch normalization. This allows ResNets to have a lower complexity than basic CNNs whilst having the same number of layers. These two architectures were combined in 2016, to create Inception-ResNet V2. According to the paper that introduced them by Szegedy et al [23] training an Inception network with residual connections accelarates the training of Inception

networks significantly. Furthermore, there is also evidence that InceptionResnetV2 slightly outpeforms expensive Inception networks without residual conncetions. The architecture of InceptionResNetV2 is shown in Fig. 3.1

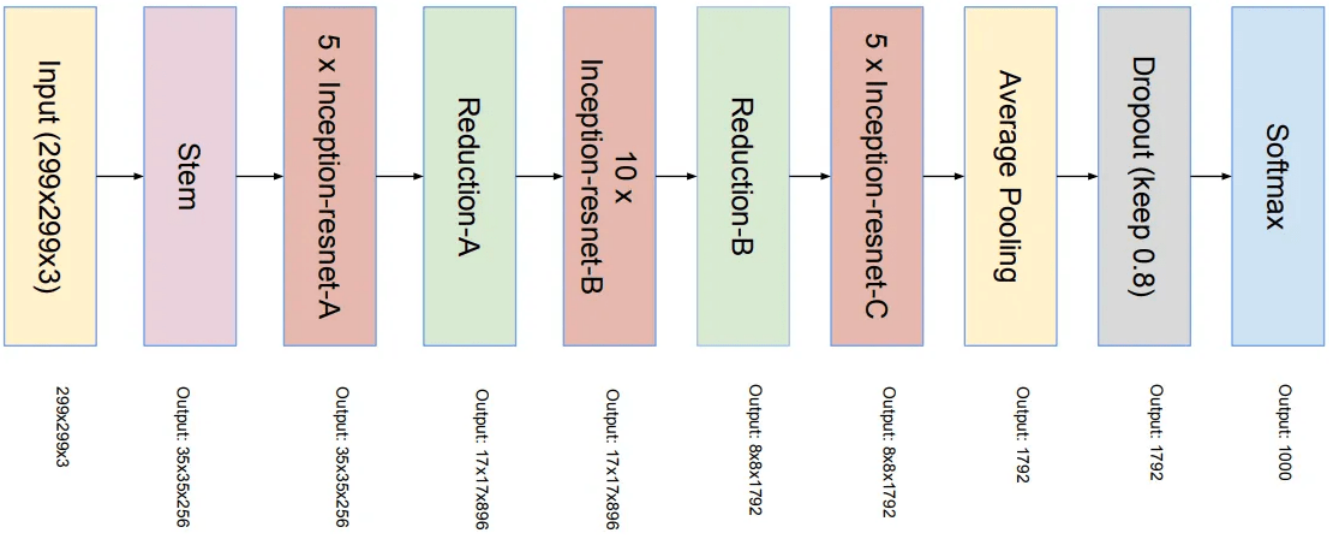


Figure 3.1: InceptionResNetV2

An example of the Inception-Resnet modules is Fig. 3.2, they have both skip connections and multiple convolutional filter sizes.

The proposed approach for this model is therefore to use InceptionResNetV2 as the base model. This base model will be imported pre-trained with imagenet [24] weights. The top of InceptionResNetV2, specified by the Average Pooling, Dropout and Softmax layers in Fig. 3.1 will need to be removed, and a custom output layer that suits the 3 class dataset will be added. This output layer will include at least the following layers:

- GlobalAveragePooling2D Layer - will be used instead of a fully connected layer.
- Dense Layer - to add L1 regularization.
- Dropout Layer - to reduce overfitting.
- Dense Layer - to reduce to a 3 neuron output.

A GaussianNoise layer might be included after the first Dense Layer to reduce overfitting if needed. Once the output layers have been trained to convergence, the network can be fine tuned. As the 2D fundus images are very different from the images in imagenet, a large number of layer from the base IneceptionResNetV2 model will need to be fine-tuned. In order to preserve the edge detection capabilities obtained from imagenet weights, some of the layers at the beginning of the model will need to stay frozen. Hyper-parameter optimization can be used to determine

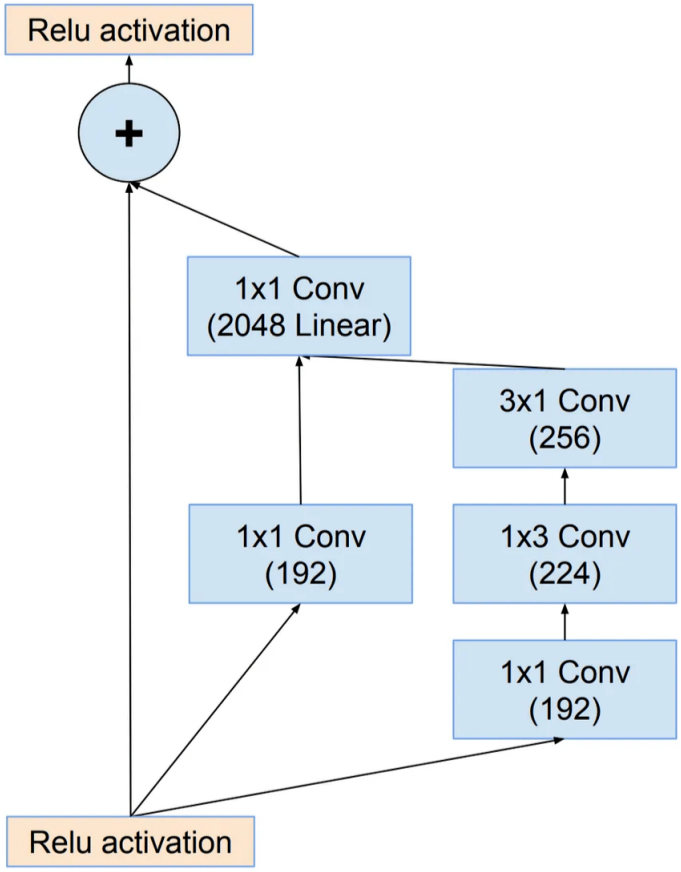


Figure 3.2: InceptionResNetV2: Inception-Resnet-C Module

the exact number.

GradCAM++ [8] will be used in this project. It will built as defined in the paper. The major difference between GradCAM and GradCAM++ is the improvement in detecting multiple occurences of a class in an image compared to GradCAM. The reason for this is that GradCAM++ can be loosely considered as a generalization of GradCAM. In GradCAM, the saliency map is defined as L_c in Fig. 3.3 where A is the output of the last convolutional layer, Z is the number of pixels in the feature map and w is defined. Y^c is final classification score for a particular class c. In GradCAM++, the weights are defined in Fig. 3.4, where α_{ij}^{kc} is equal to the square-bracketed part of Fig. 3.4 (b). If $\forall_{i,j}\ \alpha_{ij}^{kc}$ is equal to $\frac{1}{Z}$ then GradCAM++ reduces to the formulation for GradCAM. By defining GradCAM++ in the way defined by Fig. 3.4, it means that the presence of all objects in the feature map will be highlighted with equal importance as the objects are no longer proportional to the number of pixels in the image.

This improved algorithm can be explained further by Fig. 3.5 An input image is fed into a model and GradCAM++ will evenly highlight all of the objects related to a class in the image.

$$L_{ij}^{c} = \sum_{k} w_{k}^{c}.A_{ij}^{k}$$

(a) GradCAM Saliency Map definition

$$w_{k}^{c} = \frac{1}{Z}\sum_{i}\sum_{j}\frac{\partial Y^{c}}{\partial A_{ij}^{k}}$$

(b) GradCAM weights (w) definition

Figure 3.3: GradCAM implementation

$$L_{ij}^{c} = relu(\sum_{k} w_{k}^{c}.A_{ij}^{k})$$

(a) GradCAM++ Saliency Map

$$w_{k}^{c} = \sum_{i}\sum_{j}[\frac{\frac{\partial^{2}Y^{c}}{(\partial A_{ij}^{k})^{2}}}{2\frac{\partial^{2}Y^{c}}{(\partial A_{ij}^{k})^{2}}+\sum_{a}\sum_{b}A_{ab}^{k}\frac{\partial^{3}Y^{c}}{(\partial A_{ij}^{k})^{3}}}].relu(\frac{\partial Y^{c}}{\partial A_{ij}^{k}})$$

(b) GradCAM++ Weights

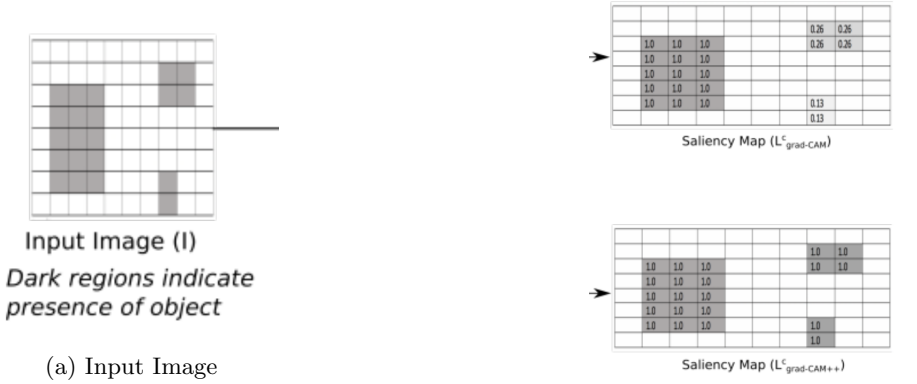
Figure 3.4: GradCAM++ Implementation

3.3 Application Design

The application needs to be simple and fast, as ophthalmologists will use the application during appointments with the patient for quick diagnosis. However, there is no need for great complexity as the application is only needed for one specific task. Tkinter is a commonly used front-end framework for Python, which enables easy integration with Tensorflow model processing. The UI will be simple but efficient. It will contain the option for users to upload an image and save the output of running the image through the neural network. A mock-up UI can be seen at Fig. 3.6.

3.4 Possible Extensions

The possible extensions for the project would be adding a time frame to let the ophthalmologist know how long until the patient progressed to the next stage of DR, however this would require more data that is not publicly available to non-researchers. Furthermore, Patho-GAN could also be used to allow the ophthalmologist to see what the patients eye would look like with a progressed stage of DR. Patho-GAN can synthesize images with a specific level of DR severity based on the number of lesions in the image. It is trained with the InceptionResNetV2 I defined previously to calculate the severity loss. The architecture of patho-GAN is defined in Fig. 3.7, where the blue networks are trainable and the green networks are pre-trained and fixed. Perceptual loss is the divergence of the real image and synthesized image in the space of the perceptual net. patho-GAN would be used in the final application. Ophthalmologists could use a 'slider' to see how the patient's eye would progress as they move through the severity levels of DR. Producing images of what the eye could look like if it progressed, or what patho-GAN



(a) Input Image

(b) Output of both Algorithms

Figure 3.5: GradCAM++ vs GradCAM

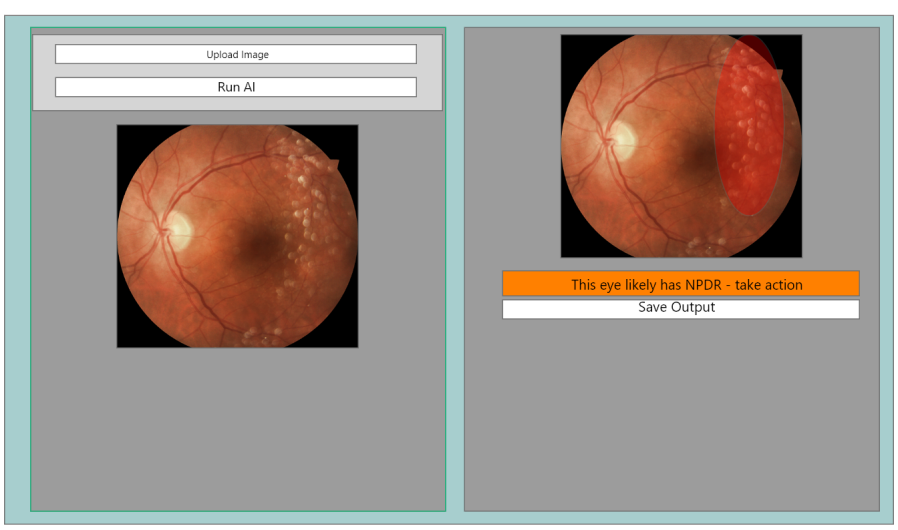


Figure 3.6: UI Design

believed the eye previously looked like to compare to the real previously taken images would be useful. A question could be posed to ophthalmologists to see if this would improve the trust between medical professionals and deep learning, due to how well the machine knows the symptoms of DR. A mock-up for an application that used patho-GAN can be seen as Fig. 3.8.

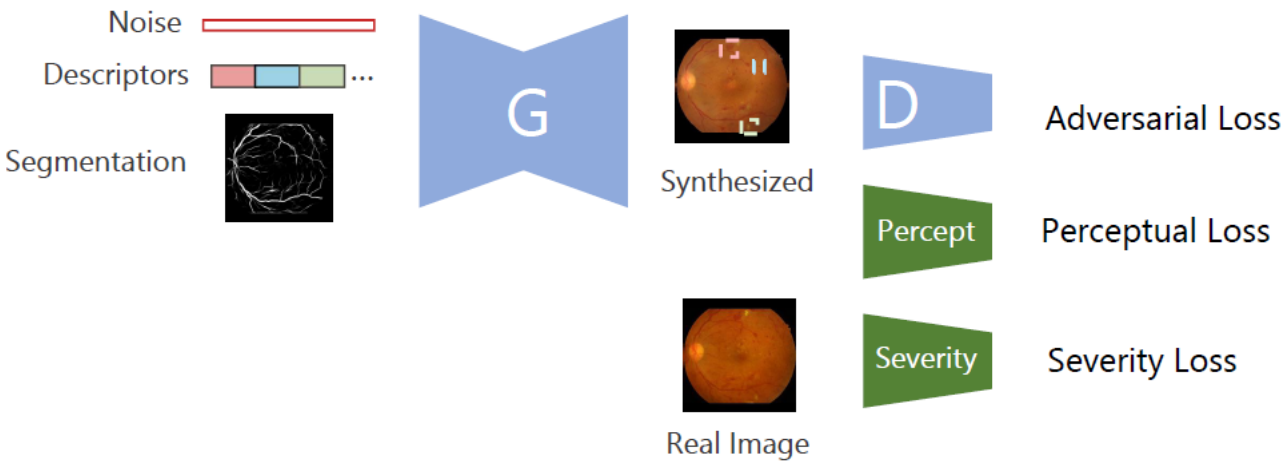


Figure 3.7: Patho-GAN

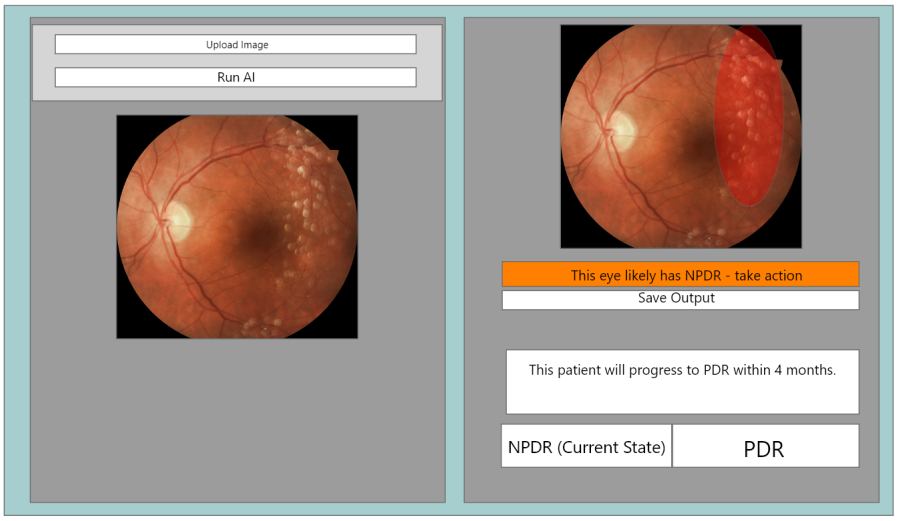


Figure 3.8: UI Design (with extensions)

Chapter 4

Implementation

4.1 Dataset

Initially, the 2,652 images had to be divided into the training, validation and test datasets. Table 4.1 shows the number of images in each of these datasets. In future work, a larger dataset should be collected, but unlike the Kaggle dataset utilised for this, it ideally should be more balanced.

Dataset	Number of Images
Training	1,856
Validation	636
Testing	160

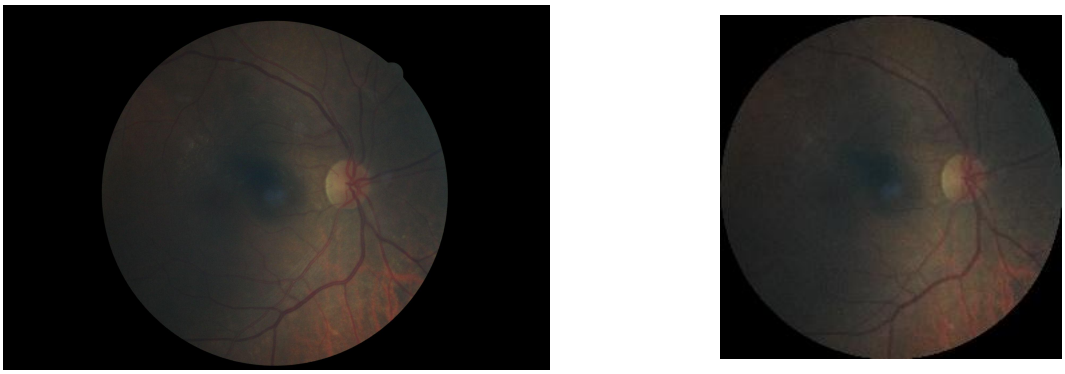
Table 4.1: Training, Validation and Test dataset structures

4.2 Data Pre-Processing

All the data preprocessing and image handling throughout this project was performed using OpenCV [25], a commonly used image processing library for Python. The preprocessing step is comprised of 3 stages.

Firstly, in the Kaggle dataset, the images contain unnecessary black space around the edge of the eyeball which needed to be removed to save valuable processing time during training as the images would be much smaller. To automate this, the Otsu thresholding algorithm was used [26]. The area which needed cropping was black, so a greyscale image and a threshold (a colour value at which to crop at) can be passed to the algorithm to obtain the foreground (the eye)

and the background (the black space). Using a built-in OpenCV function, a bounding box can be created from the foreground. The box can be used to remove the maximum amount of black space possible, whilst still keeping the image square. An example of this can be seen in Fig. 4.1.



(a) Pre-Otsu Cropping (b) Post-Otsu Cropping

Figure 4.1: An example of Otsu Thresholding Cropping

Secondly, the images in the dataset needed to be resized to reduce processing time while training the model. InceptionResnetV2 was designed to use images that have been resized using Linear Interpolation, so all images in the dataset were resized to 512x512 using OpenCVs Linear Interpolation. I selected this size after experimentation to obtain a good trade-off between training time (and available GPU/RAM resources) and the amount of detail available in each image.

Finally, once the data was loaded into a TensorFlow dataset, data augmentation techniques can be mapped to the dataset in order to reduce overfitting. Data augmentation is commonly used in small datasets, and because of the reduced the dataset size that balances the dataset due to the small number of PDR images, it should be especially useful. This technique will diversify the training set, and all of these augmentations are realistic as they will not be severely different from the normal images. The type of data augmentation used is called data warping, which means there are additional samples generated in the data space. According to this paper by Wong et al. [27], creating additional samples in the data space is a good way of performing data augmentation and helps to increase generalisation. Table 4.2 shows the techniques and value used in data augmentation.

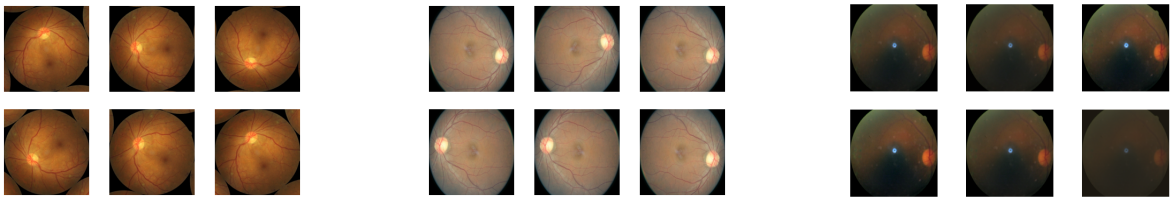
Table 4.2: Data Augmentation Values

Data Augmentation Technique	Value
RandomFlip	horizontal_and_vertical
RandomRotation	0.3
RandomContrast	0.15

These techniques do three things to the dataset.

- The RandomFlip layer will horizontally and vertically flip images to add some noise to the dataset.
- The RandomRotation will rotate an image by a random amount in the range $[-0.3 * 2\pi, 0.3 * 2\pi]$ radians
- The RandomContrast layer will, for each channel, compute the mean of the image pixels in the channel and then adjusts each component x of each pixel to $(x - mean) * 0.15 + mean$.

Fig. 4.2 shows some examples of performing the data augmentation techniques repeatedly to the same Fundus image. The RandomContrast is this figure is at 0.9, so it's easier to see what the layer will do.



(a) RandomRotation (b) RandomFlip (c) RandomContrast

Figure 4.2: Data Augmentation Output

The benefit of this is that the model will have more data for where the anomalies are in slightly different places, so the model can recognise anomalies in more areas therefore improving generalization. The data augmentation techniques were only applied to the training dataset to reduce overfitting, and not the validation or testing datasets. The validation and testing datasets both evaluate the model so require non-augmented images.

Also, as InceptionResNetV2 expects pixels in the range [-1, 1] rather than [0, 255], an function to do this was mapped to the training, validation and test datasets This normalizes the inputs, which improves the performance of the model; training can happen faster and the model is less likely to get stuck at a local minima.

4.3 Neural Network Implementation

To build the neural network I used Google's Tensorflow library [28]. Using this, we can simply import an InceptionResNetV2 model and use the built in functions to train it. However, we also want to apply our explainability approach, GradCAM++ to this. GradCAM++ requires the output of the last convolutional layer of the model, this means that how we add our custom fully connected layer to the top of the CNN differs slightly to how it is defined in the documentation of Tensorflow [29]. Using a lambda function we can get the last convolutional layer ('conv_7b'), and then pass the output of that to the GlobalAveragePooling2D layer, rather than passing the output of the entire model. This means that later, when we apply GradCAM++, we can easily access this output. When we want to perform GradCAM++, we can define a new model like so:

```
gradModel = tf.keras.Model([model.inputs], [base_model.conv_layer.output, model.
    output])
```

Fig. 4.3 shows an simple overview of what this would look like.

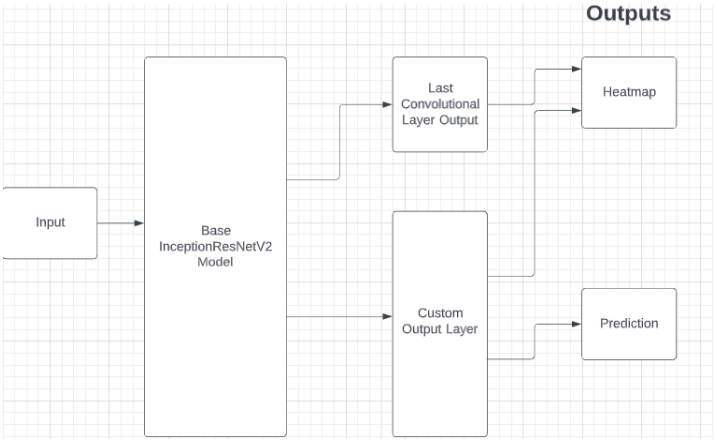


Figure 4.3: Overview of Modified Model for GradCAM++

4.3.1 Training the Network

The network had hyperparameters that needed to be tuned to their optimal values, and for this KerasTuner [30] was the best approach due to the number of available algorithms: Grid Search, Random Search and Bayesian Optimization. With KerasTuner, the hyperparameters were tuned with Bayesian Optimization (BO). Initially, I used a Grid Search, but due to the number of hyperparameters that needed to be tuned, this was too costly in terms of compute

power and the time it took to do so. The other two options are Random Search and Bayesian Optimization, and according this to paper by Turner et al. [31] and this Jupyter notebook [32], BO is slightly better than random search for hyper-parameter tuning and tends to concentrate on higher scoring values more than Random Search. Therefore, Bayesian optimization was ran for 30 trials, and the results can be seen in Table 4.3.

Table 4.3: Transfer Learning HyperParameter Results

HyperParameter	Value
Gaussian Noise Standard Deviation	0.4
Number of Units in First Dense Layer	128
Dropout Rate	0.6
Initial Learning Rate	0.00061291

The validation accuracy produced with these hyper-parameters was 0.4882. The accuracy is low because this tuning was just for the custom output layer of the model. A second hyperparameter tuning run was performed in order to tune the hyperparameters for the fine tuning of the base InceptionResnetV2. There are two more hyperparameters that need to be tuned, but due to KerasTuner's limitations they could not be trained at the same time as the output layer hyperparameters. Fine tuning requires a second compilation of the model, as more of the layers in the model are set to trainable after the transfer learning step. This cannot be done in one step using the KerasTuner API. However, we can use the optimal hyperparameters from the transfer learning step as these have converged to allow the output layer to classify an output of a convolution layer into a severity of DR. This time BO was run for 15 trials because the number of epochs per trial (15 epochs) was greater as we are training much more layers in this case. The results can be seen in Table 4.4. The resulting validation accuracy from using these parameters was 0.68031.

Table 4.4: Fine Tuning HyperParameter Results

HyperParameter	Value
Number of Layers to keep untrainable	250
Initial Learning rate	0.0002106
Number of steps to decay over	2000

The model can then be trained. The model was trained for 10 epochs before unfreezing some of the base model layers, only keeping 250 untrainable (frozen). The model was then fine tuned for 25 epochs. Finally, the model was feed the test dataset. The output metrics of this can be seen in Table 4.5.

Table 4.5: Metrics of the model on the Test Dataset

Metrics	Value
Accuracy	0.6887
Loss	1.0113
AUC	0.8507
Precision	0.6855
Recall	0.6812

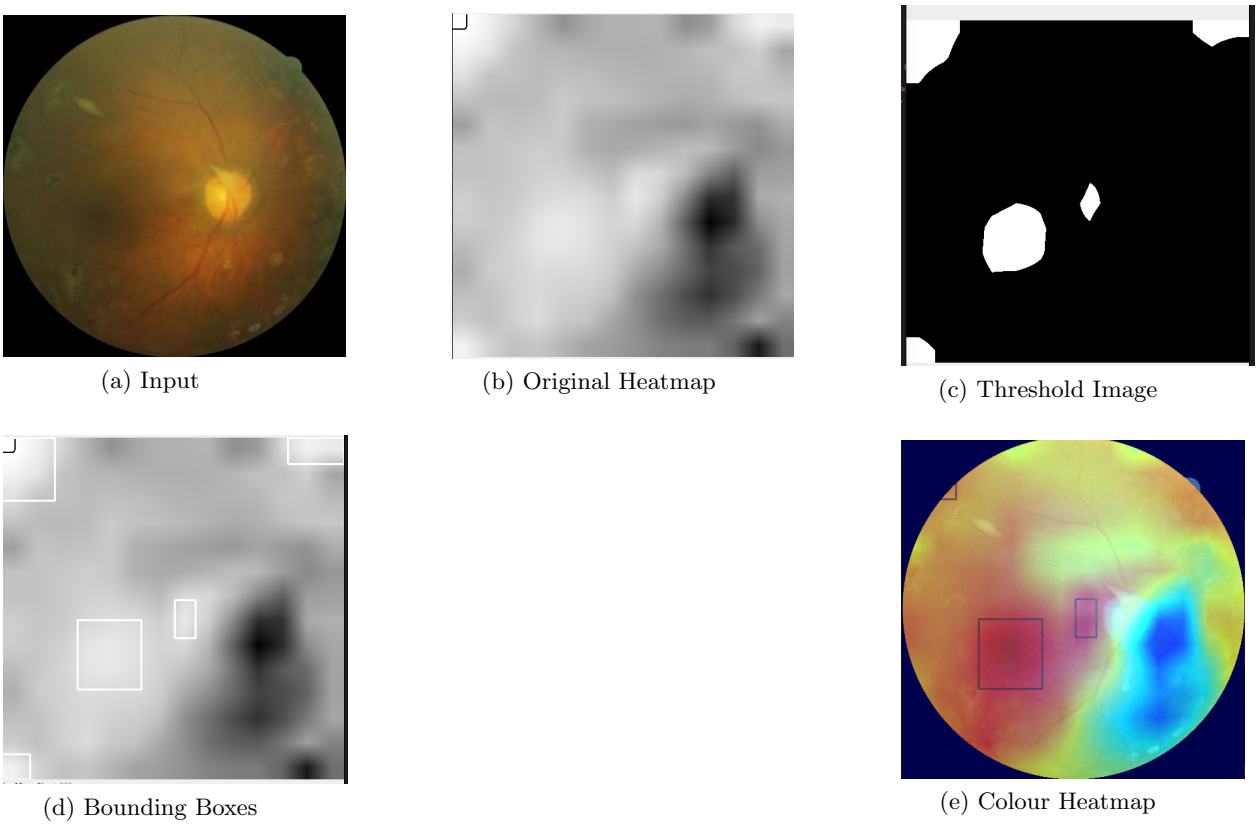
4.3.2 GradCAM++ Implementation

GradCAM++ is implemented as it is defined in Chattopadhay et al. [8], with help from a GitHub repository [1]. The time it took to produce a heatmap (also known as a saliency map) output was on average 10.8 seconds on an IntelCore i7 CPU. Initially, the first, second and third partial derivatives of the convolutional layer output and the final classification score were calculated using three GradientTapes (an TensorFlow API used for calculating gradients). However, using a persistent GradientTape improved the speed of the algorithm to 8.6 seconds on a Intel Core i7 CPU. This is a 21% improvement. α_{ij}^{kc} is then calculated using the second and third derivates and this is multiplied by the output of passing the first derivative through a ReLU function on across axis' 0 (i) and 1 (j). Finally, the weights are multiplied by the final convolution layer output across axis 2 (k). This is then passed through a ReLU function to derive the saliency map for the image.

The saliency map is obtained and passed to another function to overlay it over the original image. This function also generates bounding boxes around the the highlighted areas for easier observation. This is also done using OpenCV. This proved difficult at first, as GradCAM++ generates pixels that are brightest when they affected the classification the most. It proved easier to invert the GradCAM++ output and then apply a thresholding algorithm to generate contours which now encircled the darkest parts of the heatmap (the brightest pixels in the original output). The bounding boxes can then be drawn, pixels inverted again and a jet (which sets dark pixels to blue and bright pixels to red) colour map applied. This can then be overlayed back over the original image.

An example output of this can be seen in section (e) of Fig. 4.4. This image is an PDR (Class 2) image, and the model correctly predicted that. GradCAM++ also is highlighted a lot of Hard Exudates on the left hand side of image. Furthermore, sometimes GradCAM++

[1] https://github.com/samson6460/tf_keras_gradcamplusplus



(a) Input (b) Original Heatmap (c) Threshold Image

(d) Bounding Boxes (e) Colour Heatmap

Figure 4.4: A step-by-step example of the bounding box algoirthm

highlights pixels in the corners on the image. These are clearly not relevant to the decision making so these have been cropped from the output.

Tkinter was used to develop the UI. The UI was built as closely as possible to the design defined in Fig. 3.6. The buttons in the program are part of the sub-module of Tkinter - tkk. This allows for a slightly more modern look to the application, which makes it look more professional. When building the UI, I had to balance usability and aesthetics. The final application is more on the usability side of the scale. The reason behind this is that I thought ophthalmologists would prefer a simple interface with no confusing buttons which would prevent them from doing their job efficiently - which is the most important aspect of this project. The UI for the project can be seen in Fig. 4.5. The demo video in the git repository will show the application working [2], but the main process consists of the ophthalmologist uploading an image, clicking "Run AI" and the resulting classification and heatmap will be displayed on the right hand side.

[2]https://github.com/tomcotter7/explainable-dr-detection

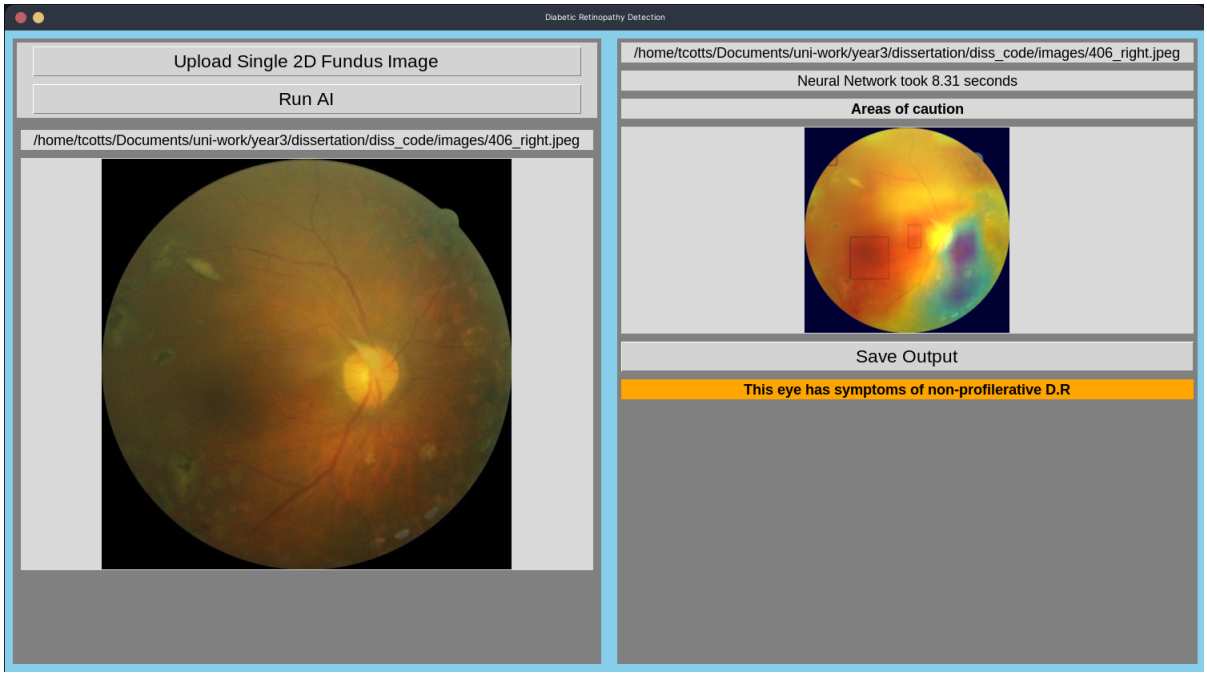


Figure 4.5: The straightforward user interface for the project

Chapter 5

Evaluation

5.1 Model Performance

The 3-class model defined in the previous chapter performed well in comparison to other models in the field. Most other work in the field has produced models that perform binary classification (PDR vs Not PDR) - such as the work by Chetoui et al (2020, [9]) which produced an AUC metric of 0.986 or Voets et al (2019, [33]) who produced an AUC of 0.951 with binary classification. This explains the difference in accuracy between my model and other papers in the field. There have been limited papers on 3-class classification but a paper by Lam et al (2018, [34]) produced a 3-class classification model which produced an accuracy of 68.8%. Furthermore, Voets et al also noted that the Kaggle dataset contains some ungradable images, and thus it is possible that the model will "learn" features for ungradable images, thus affecting performance.

The metrics on the test dataset can be seen in Table 4.5. The accuracy was 68.87% which compares to Lam et al's paper. These could have been improved therefore by a higher quantity of data and a higher quality of dataset. Moreover, perhaps the model could have also been improved with access to better hardware, which would allow the model to be trained on 1024x1024 images without running out of RAM. It could have also been improved by a larger number of PDR fundus images, which would have allowed for a larger balanced dataset. A confusion matrix for the test data can be seen in Fig. 5.1. Confusion matrices are another tool in order to show the performance of the model This shows the model performs relatively well on the dataset as the results are concentrated on the diagonal where the input class is the same as the predicted class. Furthermore, we can see that the model performs the best in differentiating between

class 2 (PDR) and classes 0 and 1 (No DR and NPDR). The model only performs bad in the differentiation of NPDR and No DR. This is likely due to the very small number of features in the fundus images of NPDR symptoms. This also confirms that the model would perform well if binary classification was required.

Figure 5.1: Model Confusion Matrix

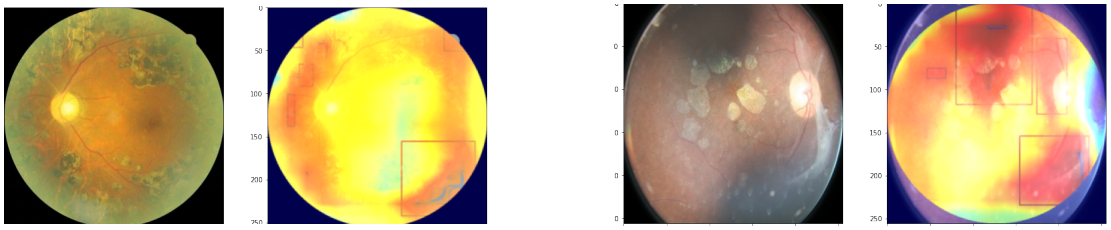
However, whilst this was the best model in terms of performance, this combination of hyper-parameters defined in Table 4.3 and Table 4.4 made it impossible to calculate the gradient between the classification output and the output of the final convolutional layer. There is no information in the Tensorflow documentation or in papers about the reason behind this, and this leads me to suspect that it might possible be a TensorFlow bug, however further research could prove this. The combination of hyper parameters that allowed for GradCAM++ to be used is defined in Table 5.1 and these produced a validation accuracy of 0.67874%. Without time constraints this could have been investigated further, and will continued to be researched in the next stage of the project.

HyperParameter	Value
Gaussian Noise Standard Deviation	0.5
Number of Units in Dense Layer	512
Dropout Rate	0.3
Initial Learning Rate for Transfer Learning	0.001
Initial Learning Rate for Fine Tuning	0.001
Number of Steps to Decay over	1500
Number of Layers to keep frozen	100

Table 5.1: The Hyper Parameters for the working GradCAM++ implementation

5.2 Explainability Approach Evaluation

GradCAM++ performed well, managing to detect most of the symptoms of DR within the Fundus images. GradCAM++ performed the best on PDR images, when it was clear to see where exactly the HEs, HMs and MAs were due to the sheer number of them. However, it usually managed to show all of the symptoms, as shown by two examples in Fig. 5.2. In both of these images, the model correctly predicted that both of these images were of class 2 (PDR). (a) shows that the model correctly predicts that this image is a PDR Fundus image because of the hard exudates around the center of the eye. GradCAM++ highlights most of them. Image (b) is more interesting, as while GradCAM++ does try to highlight some of the hard exudates seen in the center of the image, the model seems to get confused by the dark spots in the image. This testifies to the need for high quality datasets.



(a) Correctly Identifying Hard Exudates

(b) Confused by dark spots in the original image

Figure 5.2: Two Examples of Applying GradCAM++ on a PDR Image

Table 5.2 shows Fundus images from both the PDR and NPDR classes, with the GradCAM++ output displayed next to them. Furthermore, since one of the jey objectives of this project is to improve the explainability methods for Diabetic Retinopathy detection, Table 5.2 also shows the GradCAM (the predecessor of GradCAM++) output for each image. GradCAM++ is better than GradCAM in detecting all the areas of the image that contain symptoms of DR. GradCAM often struggled to produce any heatmap at all, even when the model correctly predicted the class. GradCAM++ clearly performed better than GradCAM in all areas. The GradCAM algorithm was built according to the Keras implementation in the documentation [35]. An interesting output was Image D, where GradCAM focuses on the damaged retina, but GradCAM++ highlights the hard exudates around the edges of the eye. GradCAM++ positively detected symptoms of DR in NPDR images too. In Images E and F of Table 5.2, the model highlighted some areas of abnormality in the images. These might not be specifically symptoms of DR, but they are different from the images in the No DR class.

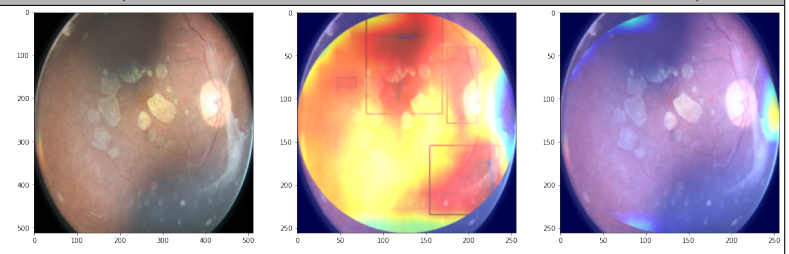
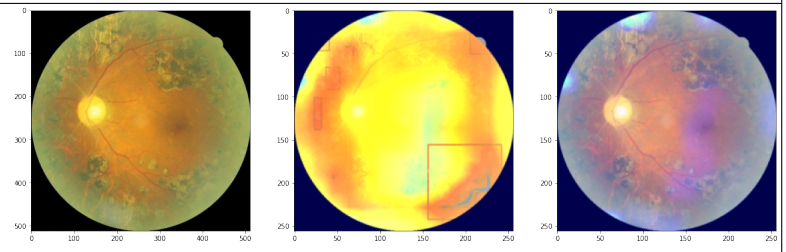
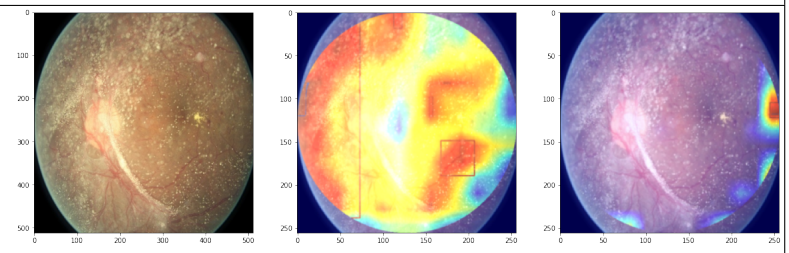
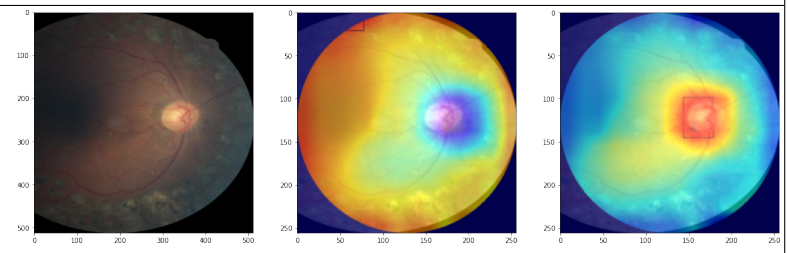
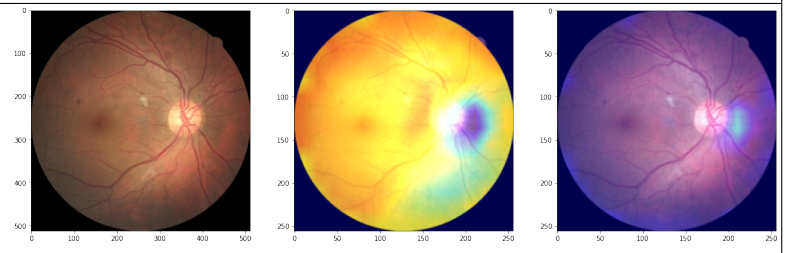
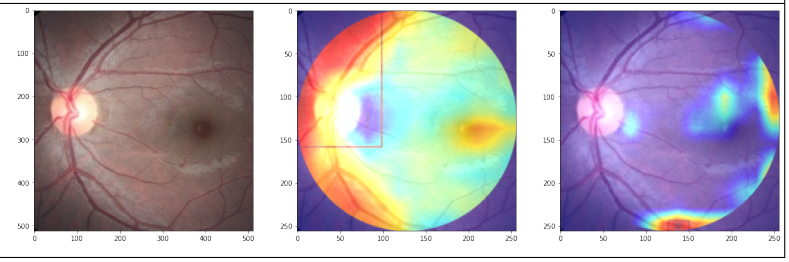
Image	Class	Output (Input Image → GradCAM++ → GradCAM)
A	PDR Fundus Image	
B	PDR Fundus Image	
C	PDR Fundus Image	
D	PDR Fundus Image	
E	NPDR Fundus Image	
F	NPDR Fundus Image	

Table 5.2: GradCAM++ vs GradCAM for differing severity levels of DR

This improved heatmap map could be used for other tasks that improve the detection of diabetic retinopathy, perhaps using the data on where lesions and anomalies usually appear to train even more accurate CNN or using areas within the bounding boxes to train GANs to synthesize images with anomalies in.

5.3 Application Evaluation

The final application and the initial design are almost identical. This functionality means that I delivered on keeping the app straightforward for use by ophthalmologists. Tkinter was new to me, so this is a positive. For reference, Fig. 3.6 is the original design and Fig. 4.5 is the final implementation.

As spoken about in the implementation section, the applications takes on average 8.6 seconds to execute an image. Whilst this is good enough, it is not optimal and there is are ways to improve the level of performance. One approached researched included executing the GradCAM++ function in graph mode. Graph mode is part of TensorFlow that allows TensorFlow specific code to execute much more efficiently than running under normal Python. As stated by the tensorflow documentation, Graphs are data structures that contain a set of `tf.Operation` objects, which represent units of computation; and `tf.Tensor` objects, which represent the units of data that flow between operations [36]. There are a great deal of benefits of graphs as they can be easily optimized allowing the compiler to do optimizations like:

- Statically infer the value of tensors by folding constant nodes in your computation
- Separate sub-parts of a computation that are independent and split them between threads or devices.
- Simplify arithmetic operations by eliminating common sub-expressions.

In short, utilising graphs mean that tensorflow code can run extremely quickly. Fig. 5.3, shows the GradCAM++ algorithm run 15 times and the code in run in graph mode is much faster - once the graph has been created it is a 91% speed increase (8.6 seconds $\rightarrow$ 0.8 seconds).

However, calling the GradCAM++ function from the Tkinter requires that it runs in eager mode - i.e not optimized. The way to fix this would be Ahead Of Time (AOT) Compilation. AOT compilation allows the graph to compiled into a binary and then directly called as a tensorflow graph from the application [37]. However, due to the complex nature of calculating gradients, the InceptionResnetV2 model and the image to pass through it must be passed into the graph

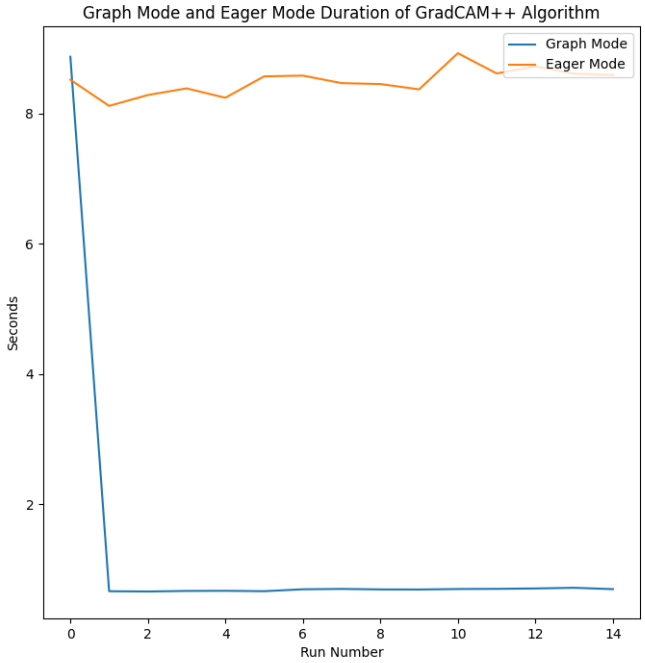


Figure 5.3: Graph Mode vs Eager Mode Execution for the GradCAM++ Algorithm

as inputs. Otherwise, the gradients cannot be calculated which are the most costly functions in terms of time taken. In order to build a binary of a graph, a tensorflow.tf2xla.Config proto must be defined in terms of the inputs and outputs of the graph. Unfortunately, the InceptionResnetV2 model cannot be defined as 1 input, it is instead defined as 1004 inputs all with varying shapes and configurations. This would take to much time to manually figure out all of these inputs and type them into a proto, and there is no automatic tool for this at all, let alone for an input this complex. So, whilst this is not technically within the scope of the project, a future extension could be forking the TensorFlow library and writing some Python / C++ code to do this. The manual steps for a very basic graph are defined here [37], so it would hopefully follow this but automatically. The reason to do this would be if complex Deep Learning models can be defined in terms of TensorFlow graphs and then compiled into binary code, these models could be ported to any type of application that requires them. If this was the case, the speed at which Deep Learning is being accepted into the medical field would increase as applications could be enabled on the desktop with Deep Learning models without much cost to the doctors using them.

Chapter 6

Summary and Reflections

In this chapter, I discuss my contributions to the field of DR detection, and what I learned throughout the process of completing this project. I also detail my thoughts on the management of the project and the approaches taken to do this.

6.1 Project management

The original aim of this project was: **To build a system capable of determining the severity level of DR in a 2D fundus image and to locate the abnormalities in the image that led to this decision with greater precision than previous work.** This goal was reached as shown by the improvements of GradCAM++ in detecting DR anomalies compared to GradCAM. There is a lot more progress to be made in terms of the detection of the smallest anomalies in the Fundus images but GradCAM++ does well in this case.

The project itself was managed well, however I think it would have helped to have a more defined aim. The initial subject matter expertise required to get to a more detailed aim was substantial. As this project is attempting to push existing boundaries on explainability approaches for DR detection, it meant a lot of investigating and experimenting before settling on the chosen approach. There is a number of frameworks that could have been applied to the DR detection problem and I struggled to decide on one. This early discover and design stage was time-constrained so I was not able to fully understand the vast supply of resources such as papers and journals which could have provided much more insight into the field, and I will definitely spend much more time researching and studying before deciding on how to implement a project goal for my next project.

I stayed on track with consistent progress throughout my time working on this project. Following on from the GANTT chart supplied in the interim report shown in Fig. 6.1, whilst the UK BioBank data was never used all other tasks were completed. The two evaluation steps however were more continuous than I'd originally thought. I took more of an agile approach on these steps, continually updating the model and GradCAM++ algorithm in unison. This improved the overall quality of both of these algorithms.

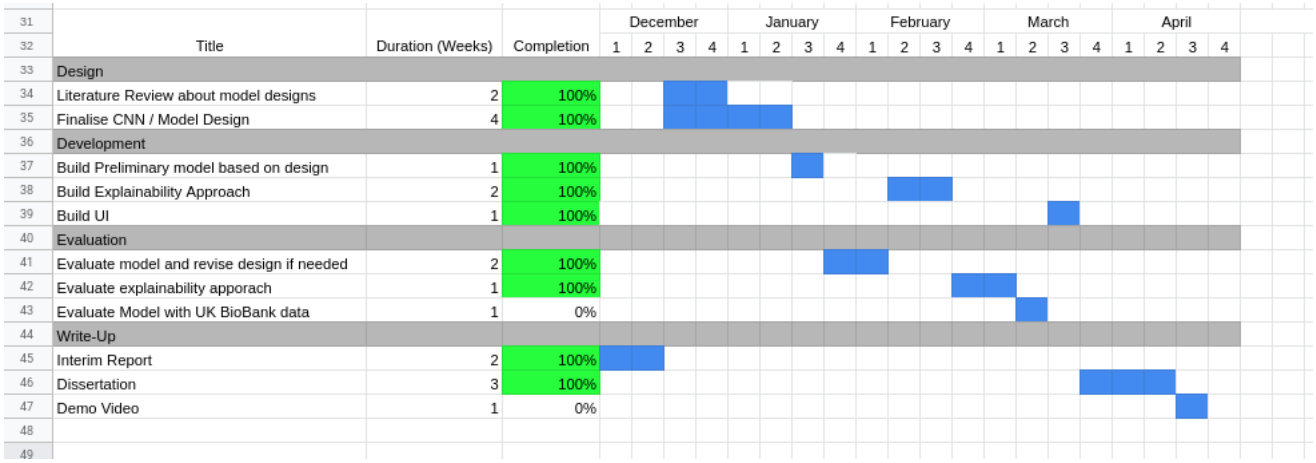


Figure 6.1: GANTT Chart showing proposed project tasks and schedule

The discovery and design focus and the iterative approach to the development and evaluation cycles would have benefited significantly by seeking more feedback from my supervisor. The knowledge and expertise of my supervisor were not taken advantage of and I often found myself stuck at times. This knowledge could have provided clarity sooner and avoided some prolonged moments of uncertainty and 'down-the-rabbit-hole' research. If I undertake an individual project again, that would be my main take-away from this process. However, I believe I managed my time well, especially from January. I consistently broke down the required tasks into smaller and smaller ideas, which helped me stay on top of what needed to be done.

6.1.1 Resource Management

The resources were especially hard to manage, considering the time it takes to train a model. I had to plan for when I was unable to use my Google Colab GPU for long periods whilst I was training the model. This was especially difficult when optimizing hyper-parameters as this took a long time, even with the improved Bayesian Optimization algorithm. Googles runtime limits were a consistent problem throughout this project as tuning and training were often have to be broken down into multiple sections. The amount of RAM required also had to be managed, as

I had to resized the images smaller than I would have liked in order to produce the model.

6.2 Contributions and reflections

6.2.1 Contributions

The first and main contribution is that GradCAM++ has been demonstrated to improve detection of multiple anomalies in an image, and therefore should be used across more medical image classification deep learning models. A lot of diseases could have many symptoms that benefit from this, such as detecting multiple ulcers in Crohn's disease. In order to diagnose Crohn's disease, Gastroenterologists perform an MRI scan of the entire large and small intestine. If there were multiple ulcers and inflammation across the intestines this explainability approach could help to highlight most of them.

Secondly, the ability to put this learning capability onto the desktop of the ophthalmologist - delivering powerful and timely results, but in a simply to understand manner is something that is not often done. The idea of porting deep learning models to usable applications should be more feasible considering the small time it is possible to get them to compile and run in. Embedding deep learning into usable applications in the medical field is still quite novel considering the low acceptance rate of deep learning among doctors. During this project, I had some ideas about the speed at which deep learning models could be applied on regular low-grade hardware and would this increase the acceptance of deep learning within doctoral communities. This is something I'd like to pursue in future projects.

6.2.2 Reflections

Deep Learning is a massive, constantly evolving field these days and I found it difficult to stay on tasks whilst doing this project. I became distracted by the sheer quantity of algorithms and optimizations available in this field that it was difficult to keep the project as guided as I would have liked. Given my curious nature and desire to learn, I would often spend days studying ideas that did not need to be studied to in order to test GradCAM++'s efficiency in explaining an models classification decision for Diabetic Retinopathy. This has been a massive learning curve for me and whilst learning about other ideas in the field is interesting it does not help me solve the task.

The time constrained nature of this project resulted in a stressful experience, however, I have

definitely learned a lot in this process. I believe I have developed a much deeper understanding of deep learning as a whole and found that it's something I really enjoy learning about and using in projects. There are additional goals I would like to attain, which I have specified but all are outside of the scope of the project given the time constraints. I would really like to improve on more ways to have fast explainability methods integrated into medical applications. The improvement to health and well-being across people with all diseases could be very impactful. If the project had an immediate next stage, it would definitely be speeding up the explainability approaches and using a higher-fidelity balanced dataset.

Bibliography

[1] Dadareye, "Diabetic retinopathy." http://www.dadareye.com/diabetic-retinopathy, 2021. [Online; Accessed 27-11-2021].

[2] NHS, "Diabetic retinopathy." https://www.nhs.uk/conditions/diabetic-retinopathy, 2021. [Online; Accessed 26-11-2021].

[3] DiabetesUK, "Diabetes and eye problems." https://www.diabetes.org.uk/guide-to-diabetes/complications/retinopathy, 2021. [Online; Accessed 28-11-2021].

[4] NEI, "Diabetic retinopathy tables — national eye institute." https://www.nei.nih.gov/learn-about-eye-health/outreach-campaigns-and-resources/eye-health-data-and-statistics/diabetic-retinopathy-data-and-statistics/diabetic-retinopathy-tables, 2021. [Online; Accessed 28-11-2021].

[5] M. Optometry, "The four stages of diabetic retinopathy - modern optometry." https://modernod.com/articles/2019-june/the-four-stages-of-diabeticretinopath, 2021. [Online; Accessed 26-11-2021].

[6] J. Krause, V. Gulshan, E. Rahimy, P. Karth, K. Widner, G. S. Corrado, L. Peng, and D. R. Webster, "Grader variability and the importance of reference standards for evaluating machine learning models for diabetic retinopathy," *Ophthalmology*, vol. 125, no. 8, p. 1264–1272, 2018.

[7] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[8] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.

[9] M. Chetoui and M. A. Akhloufi, "Explainable end-to-end deep learning for diabetic

retinopathy detection across multiple datasets," *Journal of Medical Imaging*, vol. 7, no. 04, 2020.

[10] M. Shorfuzzaman, M. S. Hossain, and A. El Saddik, "An explainable deep learning ensemble model for robust diagnosis of diabetic retinopathy grading," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 17, no. 3s, p. 1–24, 2021.

[11] A. Singh, J. Jothi Balaji, M. A. Rasheed, V. Jayakumar, R. Raman, and V. Lakshminarayanan, "Evaluation of explainable deep learning methods for ophthalmic diagnosis," *Clinical Ophthalmology*, vol. Volume 15, p. 2573–2581, 2021.

[12] Z. Wang and J. Yang, "Diabetic retinopathy detection via deep convolutional networks for discriminative localization and visual explanation," Dec 2019.

[13] Y. Niu, L. Gu, Y. Zhao, and F. Lu, "Explainable diabetic retinopathy detection and retinal image generation," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 1, p. 44–55, 2022.

[14] Medicinnet, "Koch's postulates." `https://www.medicinenet.com/kochs_postulates/definition.htm`, 2021. [Online; Accessed 10-04-2022].

[15] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "F-anogan: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, p. 30–44, 2019.

[16] D. Bonanno, K. Nock, L. Smith, P. Elmore, and F. Petry, "An approach to explainable deep learning using fuzzy inference," *SPIE Proceedings*, 2017.

[17] Kaggle, "Diabetic retinopathy detection — kaggle." `https://www.kaggle.com/c/diabetic-retinopathy-detection/data`. [Online; Accessed 01-12-2021].

[18] K. Zhou, Z. Gu, A. Li, J. Cheng, S. Gao, and J. Liu, "Fundus image quality-guided diabetic retinopathy grading," *Computational Pathology and Ophthalmic Medical Image Analysis*, p. 245–252, 2018.

[19] upGrad, "Basic cnn architecture: Explaining 5 layers of convolutional neural network — upgrad blog." `https://www.upgrad.com/blog/basic-cnn-architecture/`, 2021. [Online; Accessed 06-12-2021].

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[22] TowardsDataScience, "Understanding the inception module." `https://towardsdatascience.com/deep-learning-understand-the-inception-module-56146866e652`, 2020. [Online; Accessed 10-04-2022].

[23] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," Aug 2016.

[24] S. University, "imagenet." `https://image-net.org/`. [Online; Accessed 10-04-2022].

[25] OpenCV, "Home - opencv." `https://opencv.org/`, 2021. [Online; Accessed 02-12-2021].

[26] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, p. 62–66, 1979.

[27] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: When to warp?," *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2016.

[28] Google, "Tensorflow." `https://www.tensorflow.org/`, 2022. [Online; Accessed 05-04-2022].

[29] T. Docs, "Transfer learning and fine tuning." `https://www.tensorflow.org/tutorials/images/transfer_learning`, 2022. [Online; Accessed 07-04-2022].

[30] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, "Kerastuner." `https://github.com/keras-team/keras-tuner`, 2019. [Online; Accessed 15-04-2022].

[31] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," 04 2021.

[32] W. Koehsen, "Model tuning results: Random search vs bayesian opt.," 2018.

[33] M. Voets, K. Møllersen, and L. Ailo Bongo, "Reproduction study using public data of: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," 2019.

[34] C. Lam, D. Yi, M. Guo, and T. Lindsey, "Automated detection of diabetic retinopathy using deep learning," 2018.

[35] Keras, "Gradcam class activation visualization." `https://keras.io/examples/vision/grad_cam/`. [Online; Accessed 22-04-2022].

[36] Google, "Graphs and functions." `https://www.tensorflow.org/guide/intro_to_graphs`, 2022. [Online; Accessed 12-04-2022].

[37] Tensorflow, "Aot." `https://www.tensorflow.org/xla/tfcompile`. [Online; Accessed 12-04-2022].