Politecnico di Bari

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Image Processing and Artificial Vision

Deep Learning-Based Approach for Fovea Localization and Optic Disc Segmentation on Retinal Color Fundus Images

Docente:
Prof. Andrea Guerriero

Candidata:
Angela Di Fazio

Anno Accademico 2020–2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The inspection of retinal fundus images is one of the key steps in diagnosing a wide range of diseases such as diabetic retinopathy, glaucoma, or age related macular degeneration (AMD). Early diagnosing of such diseases is very important as they are some of the major cause of blindness.

Among the landmarks of interest in the human retina, the fovea and the optic disc (OD) are key for diagnostic purposes. The OD is the vertical oval region where the optic nerve and blood vessels leave the retina while the fovea is a small depression in the macula center. Accurate detection of these retinal landmarks can greatly improve diagnostic efficiency. This tasks are very timeconsuming and need to be performed only by professionals.

The aim of this project work is to automatize the detection of the fovea and the OD on retinal fundus images. This can be a first step towards automated mass screening and medical care.

Chapter 2

Related Works

Recently, deep learning techniques have been applied in many automated detection systems in medical images. There have been several works based on convolutional neural network (CNN) applied to fovea localization and OD segmentation. In this section we give an a briefly overview of some related studies.

For optic disc segmentation task, authors of [7] use a fully-convolutional neural network built upon U-Net [6]. Contrast Limited Adaptive Histogram Equalization (CLAHE) is used as a pre-processing. It equalizes contrast by changing color of image regions and interpolating the result across them.

U-net [6] is also used for fovea detection in [8], where the annotation of the fovea coordinates are directly used a mask.

In [2] a sequence of CNN is used for simultaneous detection of fovea and optic disc center. Detecting the centers of the fovea and OD is considered a regression task. Because the colors may just add extra complexity, authors decide to convert all of the images to grey scale and enhance the contrast of the images by applying the CLAHE technique to reduce uneven illumination in the images. A sequence of CNN is used to find the center of both fovea and OD. The first CNN is used to crop the region of interest (ROI) around fovea and OD, while the second CNN is used to detect the center coordinates.

Although fovea and OD are spatially correlated with each other, only a few studies have focused on joint fovea and OD segmentation. Works like [1] and [4] apply CNN based networks to detect both fovea and OD jointly.

Chapter 3

Description

3.1 Fovea Localization

The first task requires to predict the point coordinates of the location of the fovea, the center of the macula. This task is treated as landmark detection, which is a regression problem. The task output is the prediction of the coordinate (x, y) of the center of the fovea, that are then annotated on the correspondent image.

Figure 3.1: Example of Fovea Annotation

3.2 Optic disc segmentation

The second task is a semantic segmentation of the optic disk. The aim is to binary classify each pixel of the image, creating a mask where black pixel identify the optic disk while the white pixel identify the background, which is the output of the task. From the mask is then extracted the border of the optic disk used to annotate the correspondent image.

Figure 3.2: Example of Optic Disk Annotation

3.3 Dataset

The model is trained and evaluated on the ADAM dataset, which was released as part of a Grand Challenge. The dataset consist of 400 fundus images, 89 with eyes affected by AMD, and the rest from healthy eyes. The fundus images have either 2124x2056 or 1444x1444 resolution, so all pictures are preprocess to make them uniform in size to 256x256 for both training and testing. The ADAM dataset also includes the fovea coordinates and ground truth OD segmentation masks for each image.

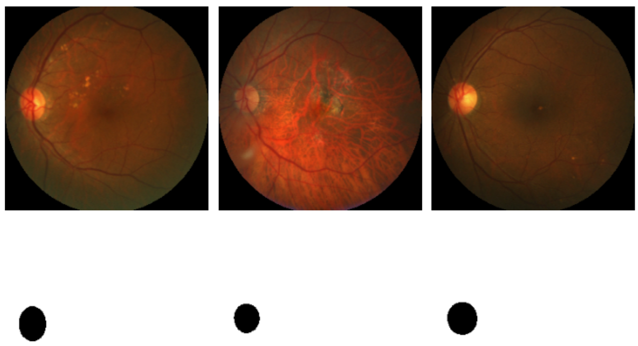


Figure 3.3: Sample Dataset Fundus Images and Masks

ID	imgName	Fovea_X	Fovea_Y
1	A0001.jpg	1182.26427759023	1022.01884158854
2	A0002.jpg	967.754045869676	1016.94665547749
3	A0003.jpg	1220.20671391458	989.944032771706
4	A0004.jpg	1141.14088834077	1000.59495451421

Table 3.1: First 4 rows of Fovea_location.xlsx

Chapter 4

Implementation Details

4.1 Data Preprocessing

Since the images have different size, every image in the dataset is resized to 256x256 using Nearest Neighbour Interpolation to avoid introducing artifacts in the resized image. Finally, the data is split 90-10 into train and test sets and 20% of the training images is held out for validation.

4.2 Models Architecture

Both models were implemented using TensorFlow 2.8.0 (Python 3.7.) and run on a NVIDIA Tesla T4 GPU provided by the free version of Google Colab.

4.2.1 Localization Model

For the localization task is used a pre-trained convolutional neural network (CNN). The network architecture is based on the *ResNet* architecture [3]. In particular, the ResNet-50 model, a 50 layers deep CNN, from the keras library is used. The model is pre-trained on the ImageNet dataset.

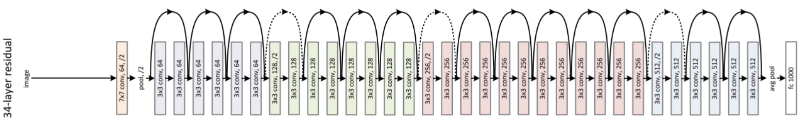


Figure 4.1: Original ResNet Architecture [3]

The realized architecture is illustrated in figure 4.2

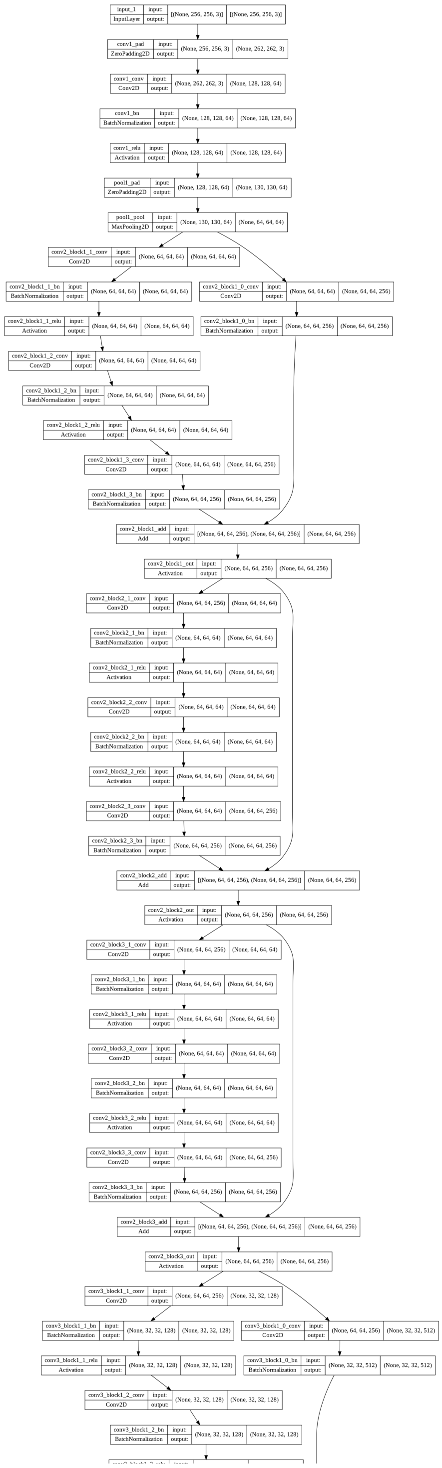
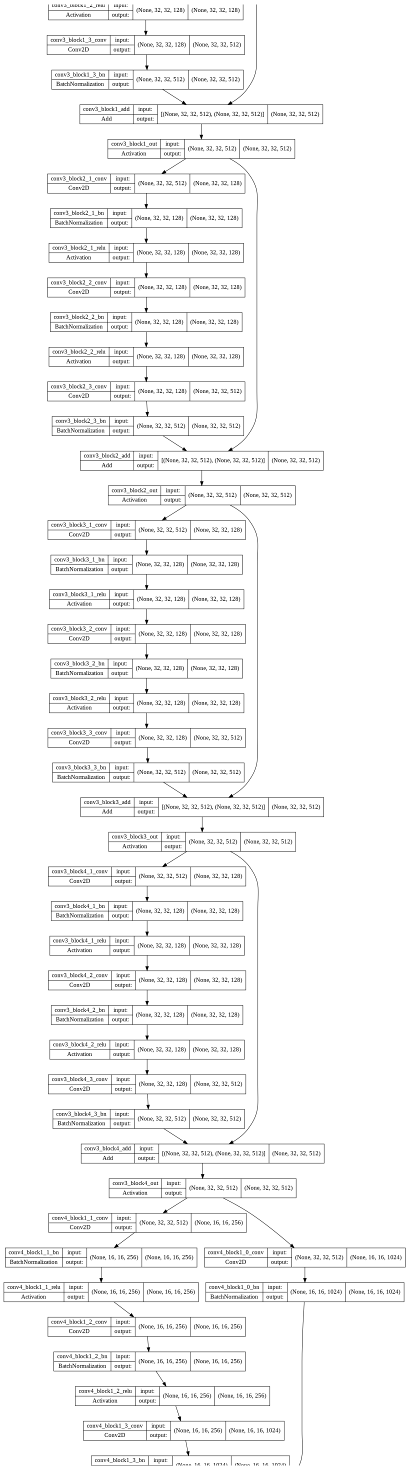
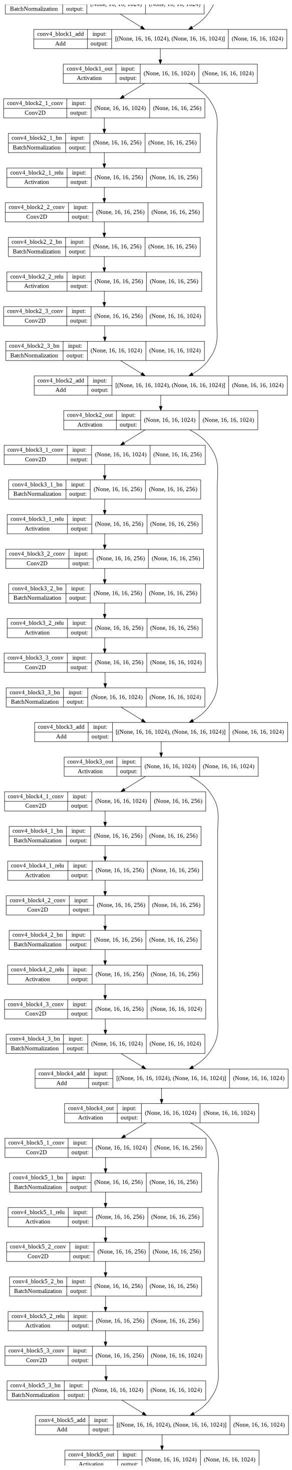


Figure 4.2: Localization CNN Architecture. Continues in subsequent pages.


Activation output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block1_3_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 512)
conv3_block1_3_bn BatchNormalization input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block1_add Add input: output: [(None, 32, 32, 512), (None, 32, 32, 512)] (None, 32, 32, 512)
conv3_block1_out Activation input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block2_1_conv Conv2D input: output: (None, 32, 32, 512) (None, 32, 32, 128)
conv3_block2_1_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block2_1_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block2_2_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block2_2_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block2_2_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block2_3_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 512)
conv3_block2_3_bn BatchNormalization input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block2_add Add input: output: [(None, 32, 32, 512), (None, 32, 32, 512)] (None, 32, 32, 512)
conv3_block2_out Activation input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block3_1_conv Conv2D input: output: (None, 32, 32, 512) (None, 32, 32, 128)
conv3_block3_1_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block3_1_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block3_2_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block3_2_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block3_2_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block3_3_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 512)
conv3_block3_3_bn BatchNormalization input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block3_add Add input: output: [(None, 32, 32, 512), (None, 32, 32, 512)] (None, 32, 32, 512)
conv3_block3_out Activation input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block4_1_conv Conv2D input: output: (None, 32, 32, 512) (None, 32, 32, 128)
conv3_block4_1_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block4_1_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block4_2_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block4_2_bn BatchNormalization input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block4_2_relu Activation input: output: (None, 32, 32, 128) (None, 32, 32, 128)
conv3_block4_3_conv Conv2D input: output: (None, 32, 32, 128) (None, 32, 32, 512)
conv3_block4_3_bn BatchNormalization input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv3_block4_add Add input: output: [(None, 32, 32, 512), (None, 32, 32, 512)] (None, 32, 32, 512)
conv3_block4_out Activation input: output: (None, 32, 32, 512) (None, 32, 32, 512)
conv4_block1_1_conv Conv2D input: output: (None, 32, 32, 512) (None, 16, 16, 256)
conv4_block1_1_bn BatchNormalization input: output: (None, 16, 16, 256) (None, 16, 16, 256)
conv4_block1_0_conv Conv2D input: output: (None, 32, 32, 512) (None, 16, 16, 1024)
conv4_block1_1_relu Activation input: output: (None, 16, 16, 256) (None, 16, 16, 256)
conv4_block1_0_bn BatchNormalization input: output: (None, 16, 16, 1024) (None, 16, 16, 1024)
conv4_block1_2_conv Conv2D input: output: (None, 16, 16, 256) (None, 16, 16, 256)
conv4_block1_2_bn BatchNormalization input: output: (None, 16, 16, 256) (None, 16, 16, 256)
conv4_block1_2_relu Activation input: output: (None, 16, 16, 256) (None, 16, 16, 256)
conv4_block1_3_conv Conv2D input: output: (None, 16, 16, 256) (None, 16, 16, 1024)
conv4_block1_3_bn input:


BatchNormalization | output:
conv4_block1_add | Add | input: | output: | [(None, 16, 16, 1024), (None, 16, 16, 1024)] | (None, 16, 16, 1024)
conv4_block1_out | Activation | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block2_1_conv | Conv2D | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 256)
conv4_block2_1_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block2_1_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block2_2_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block2_2_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block2_2_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block2_3_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 1024)
conv4_block2_3_bn | BatchNormalization | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block2_add | Add | input: | output: | [(None, 16, 16, 1024), (None, 16, 16, 1024)] | (None, 16, 16, 1024)
conv4_block2_out | Activation | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block3_1_conv | Conv2D | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 256)
conv4_block3_1_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block3_1_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block3_2_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block3_2_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block3_2_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block3_3_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 1024)
conv4_block3_3_bn | BatchNormalization | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block3_add | Add | input: | output: | [(None, 16, 16, 1024), (None, 16, 16, 1024)] | (None, 16, 16, 1024)
conv4_block3_out | Activation | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block4_1_conv | Conv2D | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 256)
conv4_block4_1_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block4_1_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block4_2_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block4_2_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block4_2_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block4_3_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 1024)
conv4_block4_3_bn | BatchNormalization | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block4_add | Add | input: | output: | [(None, 16, 16, 1024), (None, 16, 16, 1024)] | (None, 16, 16, 1024)
conv4_block4_out | Activation | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block5_1_conv | Conv2D | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 256)
conv4_block5_1_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block5_1_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block5_2_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block5_2_bn | BatchNormalization | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block5_2_relu | Activation | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 256)
conv4_block5_3_conv | Conv2D | input: | output: | (None, 16, 16, 256) | (None, 16, 16, 1024)
conv4_block5_3_bn | BatchNormalization | input: | output: | (None, 16, 16, 1024) | (None, 16, 16, 1024)
conv4_block5_add | Add | input: | output: | [(None, 16, 16, 1024), (None, 16, 16, 1024)] | (None, 16, 16, 1024)
conv4_block5_out | input: | (None, 16, 16, 1024) | (None, 16, 16, 1024)

4.2.2 Segmentation Model

The architecture of the CNN used for the segmentation task is base on the *U-Net* [6] architecture developed by Olaf Ronneberger et al. for Bio Medical Image Segmentation. The architecture is divided in two parts: the contraction path, also called encoder, which downsamples the image to create a lower-dimensional encoding and an expansion path, also called decoder, that takes and encoding and create an higher dimensional representation of the image. The job of the encoder is to learn the features that represent the image while the job of the decoder it to reconstruct a version of the image using the features obtained from the decoder.

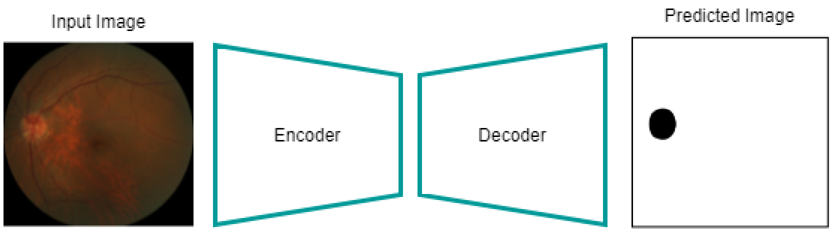


Figure 4.3: Encoder-Decoder Architecture for Semantic Segmentation

An important feature of the U-Net architecture are the skip connection, similar to the ResNet skip connection. They are used to resolve the problem of vanishing gradient as the depth of the model increase.

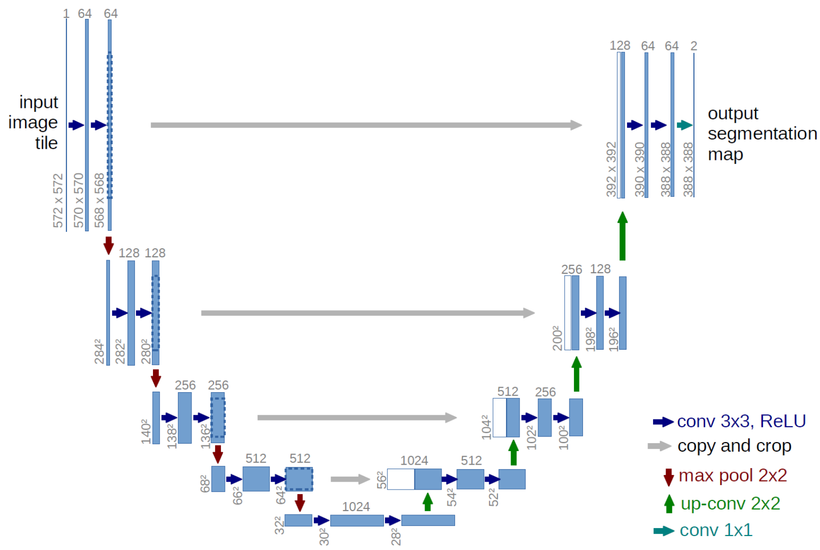


Figure 4.4: Original U-Net Architecture [6]

The realized architecture is illustrated in figure 4.5

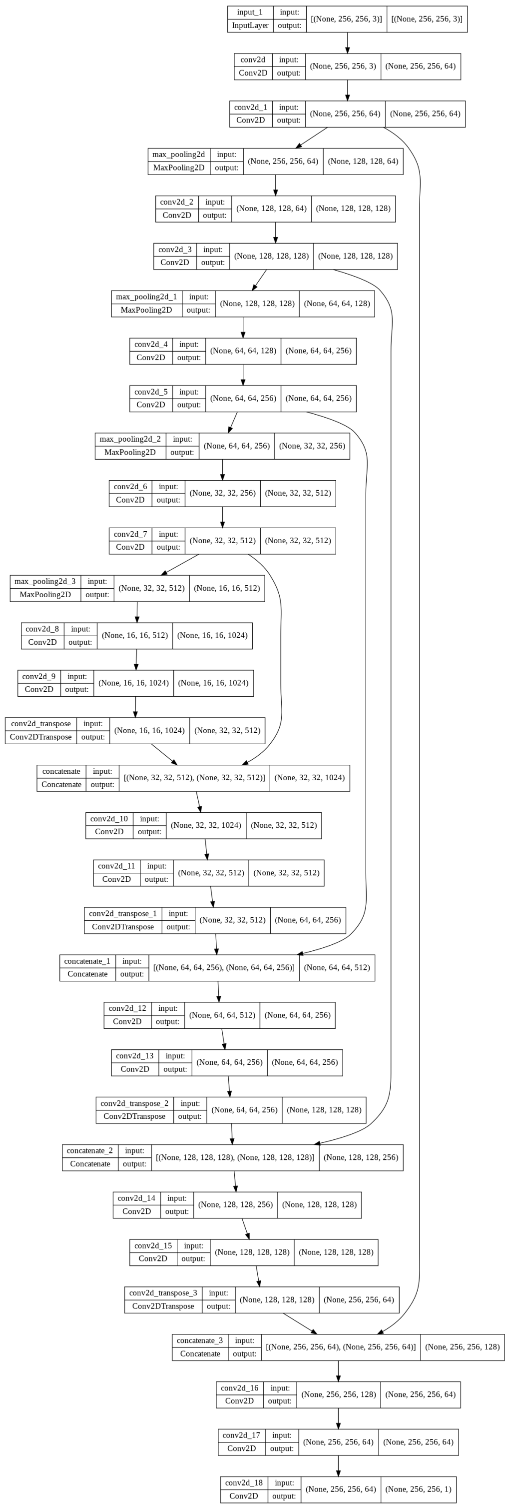


Figure 4.5: Segmentation CNN Architecture

4.3 Training Procedure

After defining the model we need to define the loss function we want to minimize during the training phase, so that the model can determine good values for fall the parameters. We have different loss function based on the task.

4.3.1 Fovea Localization

The fovea localization is interpreted as a landmark detection problem, which is a regression problem. Mean squared error is the most commonly used loss function for regression.

$$Loss(y, \hat{y}) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{4.1}$$

where $\hat{y}$ is the predicted value and y is the ground-truth.

The model is trained using the adam optimizer, with a learning rate of 0.001, and is trained for 100 epochs.

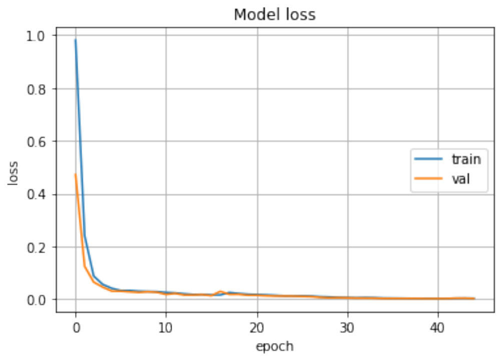


Figure 4.6: Loss Curves for Fovea Localization Task

4.3.2 Optic Disc Segmentation

In semantic segmentation the goal is to classify each pixel in an image. Since we have only two class (background and optic disc) the problem is a binary classification problem, so to train the model is used the *binary cross entropy loss*, that can be calculated as:

$$\text{Loss}(y, \hat{y}) = -\frac{1}{m} \sum_{i=1}^{m} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \tag{4.2}$$

where $\hat{y}$ is the output of the network and y is the ground-truth. The network is optimized using the *adam* optimizer with a learning rate of 0.001 for 200 epochs.

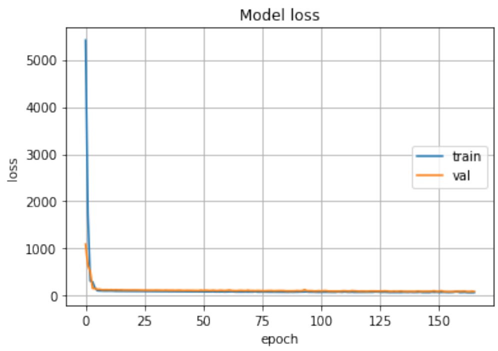


Figure 4.7: Loss Curves for OD Segmentation Task

4.4 Evaluation Metrics

The performance of the fovea localization model is evaluated with *Euclidean distance* while *Dice coefficient* and *Accuracy* are used as a metrics for the segmentation of the optic disc.

4.4.1 Euclidean Distance

The Euclidean Distance or Root Mean Squared Error measure the distance between the each actual point and the correspondent predicted point.

The formula for calculate the euclidean distance is:

$$\text{Euclidean distance}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{m}(y_i - \hat{y}_i)^2}{m}} \tag{4.3}$$

where y is the actual point and $\hat{y}$ is the predicted point.

4.4.2 Accuracy

Accuracy is the simplest metric to evaluate how well an image segmentation model performs. It is the percent of pixel classified correctly in an image.

The formula for calculate accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

Where, for a given class y:

- True Positive (TP): pixel correctly classified as y
- False Positive (FP): pixel incorrectly classified as y
- True Negative (TN): pixel correctly classified as not y
- False Negative (FN): pixel incorrectly classified as not y

The problem is that accuracy is not a suitable metrics when there is class imbalance, when a class make up only a small portion of the image.

4.4.3 Dice Coefficient

The Dice score is a commonly used metric for the evaluation of segmentation tasks. It measures the similarity between two sets.

The formula for calculate dice coefficient is:

$$\text{Dice coefficient}(y, \hat{y}) = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|} \tag{4.5}$$

where y is the ground truth set and $\hat{y}$ is the prediction set. This formula equals twice the number of elements common to both sets divided by the sum of the number of elements in each set. A value of 0 indicates no overlap between the sets while a value of 1 indicates perfect agreement.

Chapter 5

Experiments and Results

5.1 Localization Task

Table 5.1 shows the test results for fovea localization.

Metric	**Value**
Root Mean Squared Error	15.1398

Table 5.1: Metrics Value for Fovea Localization

While figure 5.1 shows some example of fovea prediction.

Figure 5.1: Sample Fovea Prediction

5.2 Segmentation Task

Table 5.2 shows the test results of optic disc segmentation.

Metric	**Value**
Accuracy	0.9978
Dice	0.9986

Table 5.2: Metrics Value for OD Segmentation

While figure 5.2 shows some example of OD prediction.

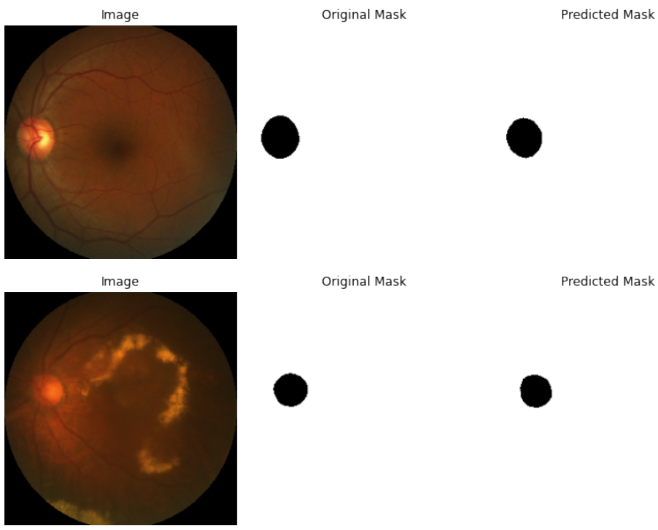


Figure 5.2: Sample OD Prediction

Finally, figure 5.3 shows the corresponding OD edge, obtained with morphological gradient.

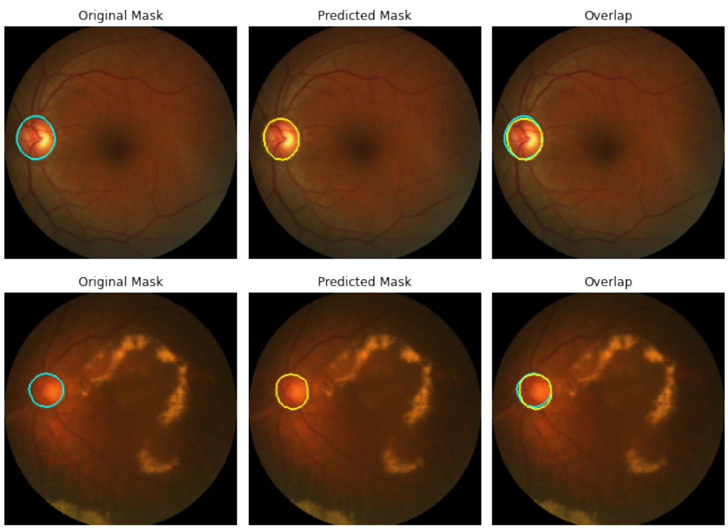


Figure 5.3: Resulting OD Border

Bibliography

[1] Baidaa Al-Bander, Waleed Al-Nuaimy, Majid A. Al-Taee, Ali Al-Ataby, and Yalin Zheng. Automatic feature learning method for detection of retinal landmarks. In *2016 9th International Conference on Developments in eSystems Engineering (DeSE)*, pages 13–18, 2016.

[2] Baidaa Al-Bander, Waleed Al-Nuaimy, Bryan M. Williams, and Yalin Zheng. Multiscale sequential convolutional neural networks for simultaneous detection of fovea and optic disc. *Biomedical Signal Processing and Control*, 40:91–101, 2018.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Shishira Maiya and Puneet Mathur. Rethinking retinal landmark localization as pose estimation: Naïve single stacked network for optic disk and fovea detection. pages 1125–1129, 05 2020.

[5] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning with applications to medical imaging. *CoRR*, abs/1902.07208, 2019.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[7] Artem Sevastopolsky. Optic disc and cup segmentation methods for glaucoma detection with modification of u-net convolutional neural network. *CoRR*, abs/1704.00979, 2017.

[8] Wu Y. Du W. Fong S. Duan Y. Yao X. Zhou X. Li Q. Lin C. Liu J. Huang L. Wu F. Wang H., Yang J. A fast lightweight based deep fusion learning for detecting macula fovea using ultra-widefield fundus images. In *Preprints 2021, 2021080469*.