REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

DETECTION, CLASSIFICATION & SEGMENTATION OF HEMORRHAGIC AND ISCHEMIC STROKE FROM BRAIN CT IMAGES WITH DEEP LEARNING NETWORKS

16011059 — Ahmet Onur AKMAN
16011003 — Ahmet AYDIN
14011063 — Emine Betül ŞİNAR

SENIOR PROJECT

Advisor
Assist. Prof. Dr. Hamza Osman İLHAN

June, 2021

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to all Yıldız Technical University Computer Engineering Department members and assistants who contributed to the completion of this project and to our advisor Dr. Hamza Osman Ilhan, who guided us throughout the whole process and never stopped supporting us and answering our questions with all his patience and devotion. We would also like to thank our biggest motivation, our families and friends, for trying their best forth to provide a working environment where we can remain productive.

Ahmet Onur AKMAN
Ahmet AYDIN
Emine Betül ŞİNAR

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
CPU	Central Processing Unit
CT	Computed Tomography
DICOM	Digital Imaging and Communications in Medicine
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
IoU	Intersection Over Union
IoT	Internet of Things
JSON	JavaScript Object Notation
R-CNN	Region Based Convolutional Neural Network
RAM	Random Access Memory
ROI	Region Of Interest
VGG	Visual Geometry Group
YOLO	You Only Look Once

LIST OF FIGURES

LIST OF TABLES

ÖZET

DERİN ÖĞRENME AĞLARI İLE BEYİN BT GÖRÜNTÜLERİ ÜZERİNDEN HEMORAJİK VE İSKEMİK İNME TESPİT, SINIFLANDIRMA VE SEGMENTASYON

Ahmet Onur AKMAN
Ahmet AYDIN
Emine Betül ŞİNAR

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr. Ögr. Üyesi Hamza Osman İLHAN

Her yıl milyonlarca insanın tecrübe ettiği inme, hayati risk oluşturan bir tıbbi durumdur. Günümüzde bu durumun teşhisi yaygın olarak insan eliyle yapılmaktadır. Bu uygulama insan gücü ve zaman açısından oldukça maliyetlidir.

Bu çalışmada inmenin kanamalı ve iskemik türlerinin beynin bilgisayarlı tomografisi üzerinden tespiti, sınıflandırılması ve segmentasyonu işlevlerinin yüksek başarı oranları ile gerçekleştirilmesi için yöntem arayışı gerçekleştirilmiştir. Bu arayış esnasında mevcut problem için VGG-19, Inception, MobileNet, ResNet, DenseNet, EfficientNet, Mask R-CNN, YOLO, U-Net gibi sinir ağlarının performansları gözlenmiş, probleme adaptasyonlarını en üst seviyede tutmak için en uygun yaklaşımlar deney ve gözleme dayalı uygulamalar ile belirlenmiş ve alınabilecek en yüksek başarı oranları major voting gibi yöntemlerle daha da öteye götürülmeye çalışılmıştır. Teşhis, sınıflandırma ve lokalizasyon basamaklarının tümünü kapsayan bu çalışma sonucunda başarı oranı yüksek yöntemler ve modeller ortaya konmuştur. Bu arayış süreci boyunca denenmiş yöntemler, bu yöntemlerin başarı oranları, bu rapor içerisinde detaylarıyla gözler önüne serilmiştir.

Anahtar Kelimeler: İnme BT Görüntüleri, İnme Teşhisi, Segmentasyon, Tıbbi Veri Kümesi Sınıflandırma, Yapay Sinir Ağları

ABSTRACT

DETECTION, CLASSIFICATION & SEGMENTATION OF HEMORRHAGIC AND ISCHEMIC STROKE FROM BRAIN CT IMAGES WITH DEEP LEARNING NETWORKS

Ahmet Onur AKMAN
Ahmet AYDIN
Emine Betül ŞİNAR

Department of Computer Engineering
Senior Project

Advisor: Assist. Prof. Dr. Hamza Osman İLHAN

Stroke is a life-threatening medical condition that millions of people experience each year. Today, the diagnosis of this condition is commonly made manually by experts. This application is very costly in terms of manpower and time.

In this study, a search for a method has been carried out to detect, classify and segment the hemorrhagic and ischemic types of stroke from brain CT, with high success rates. During this search, the performances of neural networks such as VGG-19, Inception, MobileNet, ResNet, DenseNet, EfficientNet, Mask R-CNN, YOLO, U-Net were observed for the aforementioned problem, and in order to keep their adaptation to the problem at the highest level, the most appropriate approaches have been determined by experiments and observation-based applications. Also, in order to take the highest possible success rates even further, methods like major voting were adopted. As a result of this study, which covers all stages of diagnosis, classification, and localization, methods and models with high success rates have been presented. The methods tried during this search and the success rates of these methods are revealed in detail in this report.

Keywords: Artificial Neural Networks, Medical Dataset Classification, Segmentation, Stroke CT Images, Stroke Diagnosis

1
Introduction

It has become a necessity for every part of our lives to be digitalized and adapt to modern world standards. The digital transformation breakthroughs made in recent years have revealed that there is almost no field in human life that cannot be improved with digitalization. In this document, the steps and results of a study conducted for an industry that has made serious progress in its digitization journey in recent years will be presented.

The focus of our project is to make the process of diagnosing the disease made with human senses and knowledge faster, more precise, and easier by using the opportunities that deep learning models have given us, for the hemorrhagic and ischemic stroke diagnosis. The main goal of the study that we will explain to you in this document is to analyze the non-contrast computed tomography with the deep learning model to be created, to classify it for the presence or absence of stroke, and to identify this stroke area in the tomography detected as stroke in the following step. As the team that put forward this project, we aim to achieve high success rates with the model we will put forward, to contribute to the vision created for future digital tools that can give faster and more accurate results for such critical problems.

We would like to encourage you to read further sections of this report for technical details. To summarize, in this study, we aim to distinguish a brain tomography with a stroke from a normal one with the deep learning models we will present, to identify the type of stroke in the stroke tomography, to detect the area where the stroke is located, and to do all these by capturing high accuracy rates. We are hoping that the different methods, our creative ideas, and the final product that we will put forward in this development process will represent a work with high originality value for the literature.

In this process, we feel lucky that there are studies in different sources that enlighten us. We evaluated some of these studies that could inspire us and what they taught us in the "Preliminary Examination" section. In the next section, "Feasibility", our

feasibility studies for this project and the analysis we made as a result of these studies can be found. Technical information about the process of determining the technical details of the project and the methods we have decided on can be found in the "System Analysis" section. The design that was made in line with the decisions made and the details about this design are explained in the "Design" section. In the section of "Implementation", information about the methods we have adopted and their usage in this project are included. You can find the results of the adopted methods and how we analyzed these outcomes in the sections of "Experimental Results" and "Performance Analysis", respectively. Last but not least, THE "Result" section presents a conclusion and our opinions in regard TO the improvability and the future work.

In addition, we encourage you to have a deeper look at the referenced literature studies and a piece of brief information about us provided at the end of this document.

2
Preliminary Examination

2.1 Literature Review

During the literature review, various academic studies were examined and the methods used in seven of the most qualified among them were evaluated. The first study reviewed was published by Chawla-Sharma et al. with the title "A method for automatic detection and classification of stroke from brain CT images" [1]. The study was published abstractly, and the techniques applied and the success rates achieved were outlined. In the study, it was stated that with a machine learning model focused on symmetry, hemorrhagic stroke can be detected on the data set with a success rate of approximately 98%. Although the study has provided a perspective on data set processing, its contribution to the current project has not been observed algorithmically, since the features of the model are not detailed and only mentioned superficially.

Although the publication “Deep learning IoT system for online stroke detection in skull computed tomography images” has been published for a similar purpose to the subject we are working on, a system that enables remote diagnosis in health centers by taking diagnoses one step further with IoT support has been introduced [2]. The parts of the publication that can inspire our work are the machine learning process, the feature extraction made with a CNN network trained with a transfer learning approach, and then generating an estimate as a result of classifications made with Bayesian Classifier, k-Nearest Neighbor, ML Perceptron, Random Forest and SVM. In this publication, the transfer learning process of the CNN model used for feature extraction is described in detail, and the used CNN models; LeNet, AlexNet, VGG, Inception, ResNet, Xception, MobileNet and NasNet are described with detailed features. In this study, it was seen that the features extracted from the image with a CNN model can be used in estimation with the support of classification methods.

Another publication, “A fast and fully - automated deep - learning approach for accurate hemorrhage segmentation and volume quantification in non - contrast whole

- head CT” is about a study that succeeded in performing hemorrhagic segmentation with transfer learning on DS-CNN [3]. In the publication, comparisons were made with other methods and models, and the success level of the model obtained was analyzed. The created model has succeeded in segmentation with an approximately 0.84 dice score.

Another publication, "Deep Transfer Learning Application for Automated Ischemic Classification in Posterior Fossa CT Images", by Suberi-Zakaria et al. is about using a transfer learning approach for ischemic stroke detection [4]. In the process described in this publication, three frequently used models, VGG-16, GoogleNet and ResNet-50 were compared and it has been shown that in the applied conditions, the model of ResNet-50 was the model that can be used for such problems with a satisfactory performance.

The publication of "Classification of CT brain images based on deep learning networks" worked with a data set similar to the data type we will be working on, but studied a different detection problem [5]. In this publication, the advanced CNN model created by a fusion of different types of networks at the decision stage was compared according to the success rate revealed by the models in the structure alone in classifying the brain CT images as with Alzheimer’s, lesioned, normal image. With the success rate revealed, the impression was that such a fusion approach could have positive results for our problem.

Based on the publications in the literature, a candidate method was determined for the machine learning process within the scope of the presence/absence of stroke, which is described as the first step of the project. As a result of different information, inspiration and inferences from different studies, it was predicted that some networks, which can be used frequently for such problems, can be adapted to our problem with the transfer learning approach. With the adaptation of the last layers to our problem, it has been observed that a prediction made by preserving the structure of some models, whose success in separating features through visuals has been demonstrated in various studies, can reach very high accuracy rates. Some of the networks that are frequently encountered in the studies and that may have a high success rate for our study were determined as ResNet, EfficientNet, MobileNet, DenseNet, VGG-16 and Inception. In addition, the idea of combining different network architectures and creating a common decision mechanism will be tested by combining the selected networks to be applied in the presence/absence of stroke within the scope of the project. Using different classifiers or combining different models to be used with the fusion approach at the endpoint in order to make more accurate predictions with better evaluation of the features to be extracted with deep learning networks will be tested.

Segmentation studies were examined on similar CT images in the literature to determine the ischemic and hemorrhagic stroke regions on the stroke images, which constitute the second stage of the project.

The first example examined in the scans made specifically for these segmentation studies is Ramachandran-George et al. published by "Using YOLO based deep learning network for real time detection and localization of lung nodules from low dose CT scans" [6]. In this study, the use of the object identification model DetectNet with YOLO and as a result of this use, an output on the detection and localization of the masses in the lung via CT images has been presented.

With a similar purpose to the aforementioned study, “Segmentation of Lung Nodule in CT Images Based on Mask R-CNN”, published by Liu-Dong et al., describes a study made on CT data and in a way similar to our study [7]. In this publication, the success achieved by adapting another known deep learning model Mask R-CNN to the problem is presented.

As the final result of the literature review, CT images have been used in studies on segmentation and the methods, models and approaches that have achieved certain success have been analyzed. As a result of this analysis, models that could be useful for us were determined in our study. Considering the investigated studies, it was decided to use YOLOv5, which is the current version and whose performance is higher than the previous versions, and Mask R-CNN, which will enable us to locate the stroke in the CT image more precisely.

3
Feasibility

In this section, the economic, legal, technical and time feasibility studies of the project are presented.

3.1 Technical Feasibility

We examined the software and hardware features to be used for the project under this heading.

3.1.1 Software Feasibility

Necessary software requirements for the project were determined.

- Operating System: Linux- Ubuntu
- Programming Language: Due to the large number of resources and documents to be used, since there are many ready-made libraries for image processing and deep learning; Python 3.6
- Application Development Environment: Google Colab, PyQt(GUI)
- Framework and Python libraries to use:
  -TensorFlow
  -Keras
  -NumPy
  -OpenCV
  -PIL

3.1.2 Hardware Feasibility

The hardware features of our project development environment can be found at Table-3.1.

CPU	Intel Core i7 -7700hq CPU @ 2.80GHz(3.80 GHz max)
GPU	Nvidia GeForce GTX 1050 Ti
Ram	16 GB
Disk Capacity	NVMe SSD 240 GB

Table 3.1 Hardware Features Of The Project Development Environment

It is predicted that there is a need for a GPU to run deep learning algorithms and train models faster, and this problem is planned to be solved using Google Colaboratory. With Google Colab, an application will be developed on a free Tesla K8 GPU.

The disk capacity required for the processes we perform in the preprocessing stage is set to 1 GB. During the design of the data set, Google Drive cloud service was also required for the storage and sharing of our data. During the works in local, the training codes will be written in Python programming language and will be run on the Anaconda distribution. Necessary DICOM format viewing tools (RadiAnt DICOM Viewer etc.) will be used to view the training data set. In addition, some of the most basic libraries to use are Keras, TensorFlow, OpenCV, and NumPy.

3.1.3 Communication Feasibility

Under this heading, communication technologies and the appropriate device/software features have been examined. Zoom application was preferred to be used as a communication channel for parts that need to be carried out remotely throughout the project. In addition, using the Zoom application for remote desktop access was deemed sufficient for the continuity of the project.

3.2 Workforce and Time Planning

At this stage, the important dates and timeline of the project are determined. The important dates and the Gantt diagram showing the extent of the planned process are added in Image-3.2.

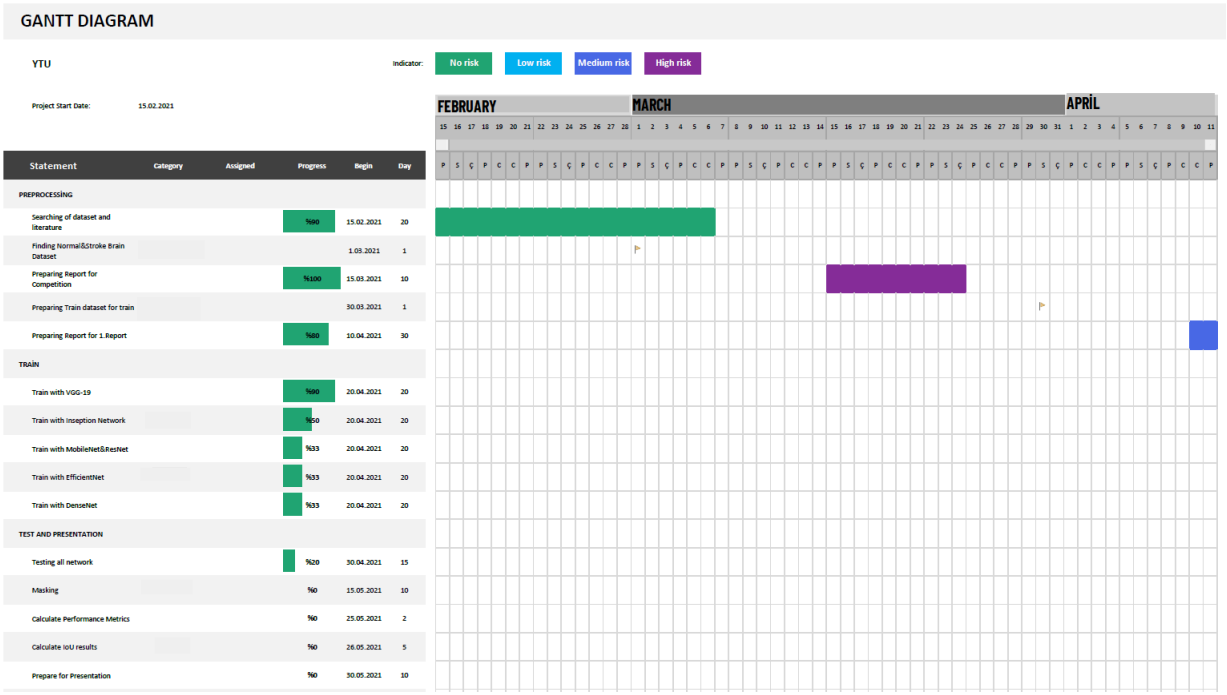

Figure 3.1 Gantt Diagram Part-1

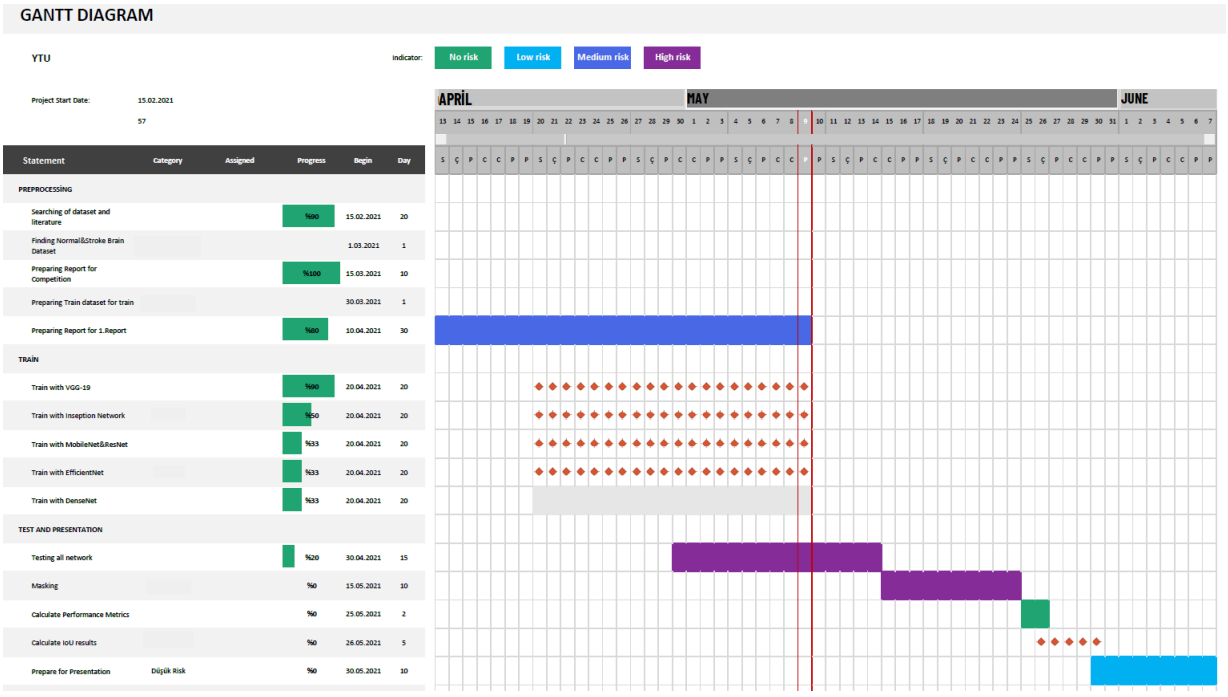

Figure 3.2 Gantt Diagram Part-2

The organization chart of the activities, which are also subject to the report of the project, is as in Figure-3.3.

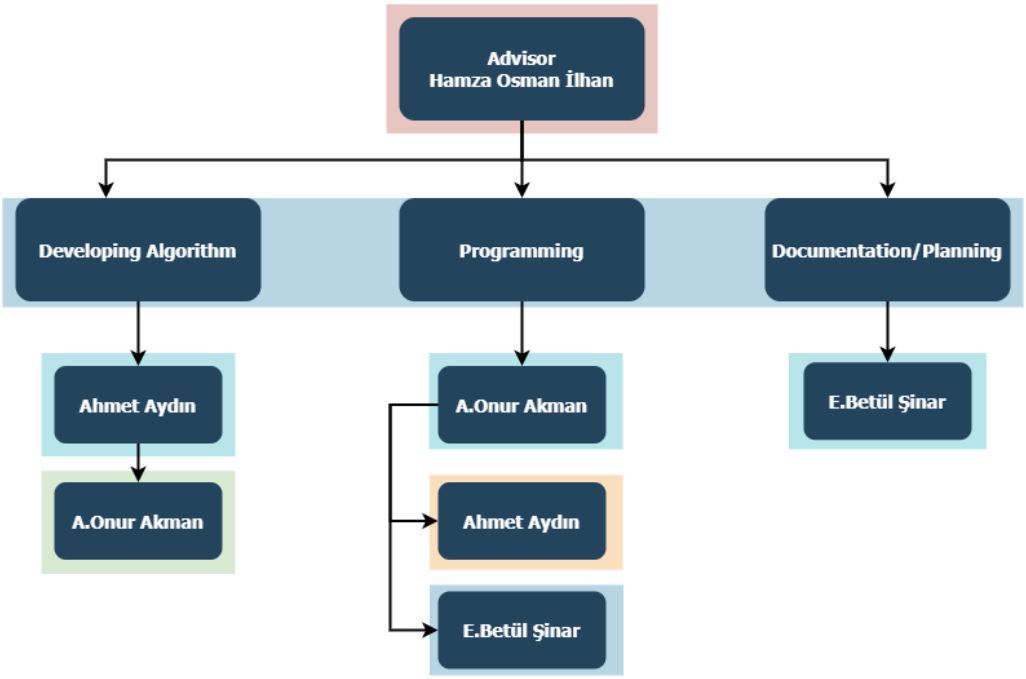

Figure 3.3 Organization Schema

3.3 Economic Feasibility

When we evaluate the project economically, it was determined that this problem will be solved free of charge by using Google's Colab environment or our hardware, and it is determined that there is no need to make a separate expenditure for our hardware and software needs for the project.

3.4 Legal Feasibility

This project is within the scope of scientific and literary work according to the 2nd article of the "Fikir ve Sanat Eserleri Kanunu", numbered 5846.

Where deemed necessary, citations were made in accordance with Article 35 of the same law.

4
System Analysis

In this section, the functional and performance requirements, design constraints and quality features of the project have been taken into consideration and analyzed. The diagrams of the system we designed are presented.

4.1 Usage Use-Case Diagram

The usage use-case diagram can be found in Figure-4.1.

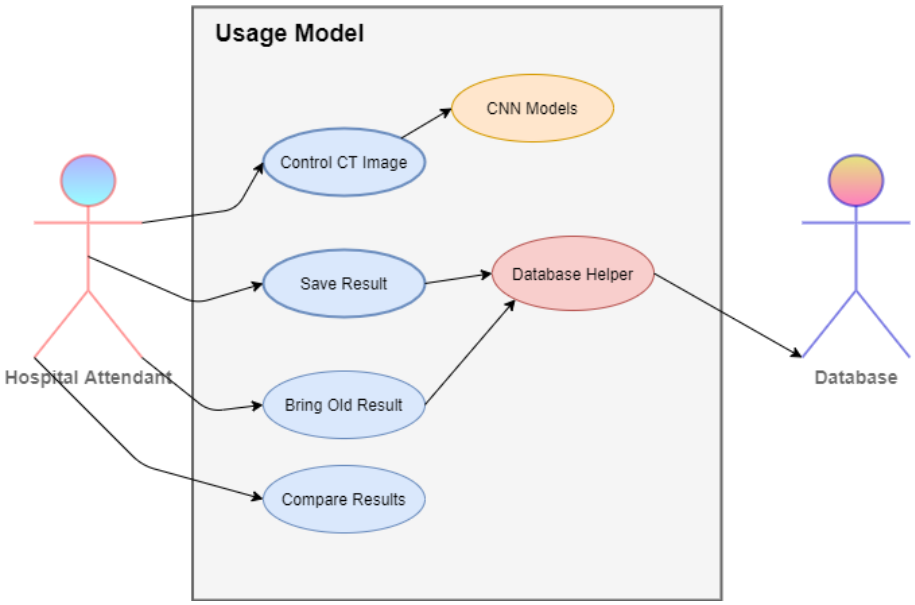

Figure 4.1 Usage Use-Case Diagram

4.2 Train Use-Case Diagram

The most essential part of this project's development process is the training part, which the success rate of the final product almost merely depends on. The general view of this process is presented as a use-case diagram in Figure-4.2.

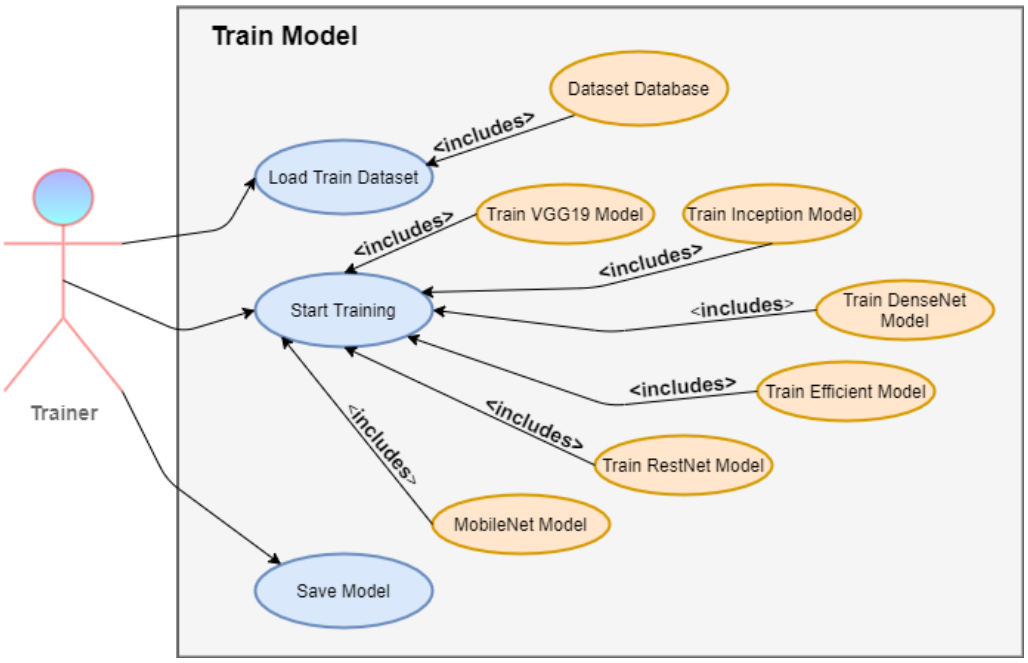

Figure 4.2 Train Use-Case Diagram

We, the trainers of the models, followed the steps of data preparation, training decided models, and saving the trained models for later use and testing.

4.3 Test Use-Case Diagram

In the following sections, you will be shown the success rate of the models we put forward. Under this heading, the process of success measurement is briefly described.

Under the previous heading, the plan of the process of model training was described. The trained models need to be saved for later comparison with their success rates and statistical scores. To do that, after the training phase we moved to the next step; the testing phase.

This process is shown as a use-case diagram as well in Figure-4.3. This time, with the "Tester" role, we followed the steps of loading the testing data, loading the saved trained models, testing the models, and finally showing and saving their scores. These obtained scores will be presented to you in the form of charts and tables.

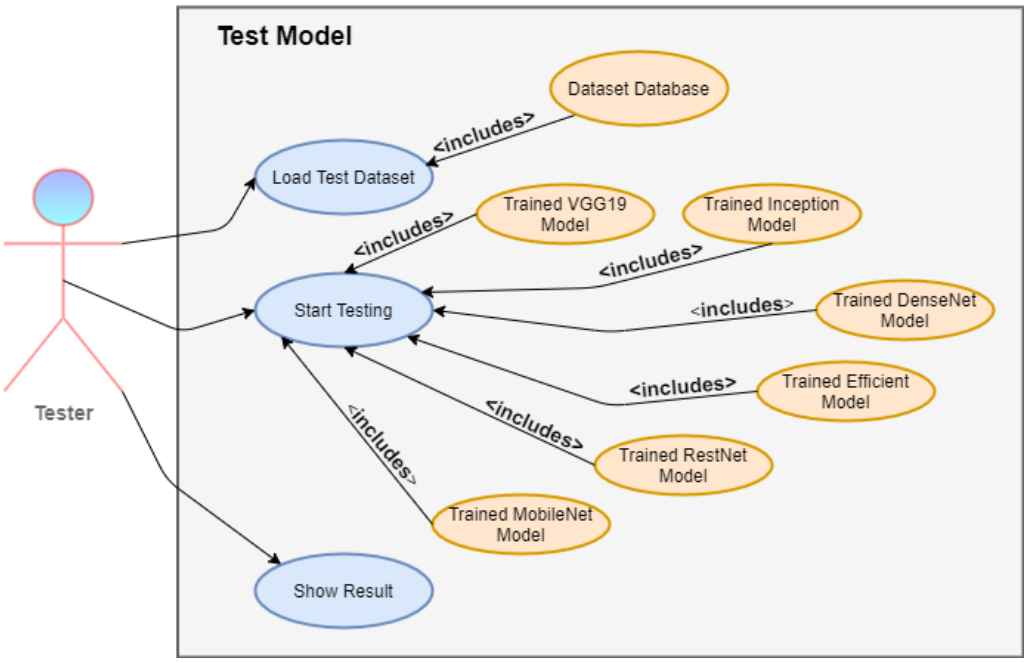

Figure 4.3 Test Use-Case Diagram

4.3.1 Success Measurement

Accuracy, Recall, Precision and F1 Score metrics are the performance metrics used for the detection of stroke present / absent, which was the 1st stage of the project. In addition, accuracy score that is obtained from the model performance over the test data is another metric to measure the success rate of a trained model.

Calculation of Intersection Over Union (IoU) criteria is used as a success measurement method in the segmenting stage of the stroke regions, which is described as the second stage of the project. Also, predicted masks are provided in the scope of this report.

5
System Design

5.1 Dataset Information

When the literature is reviewed, the two data sets that we came across are planned to be used to meet the data need for the models to learn better.

- https://lapisco.ifce.edu.br/producao-academica/private-datasets/database-skull-from-cnn-stroke-classification/

In the dataset created for this study where stroke lesions were detected with CNN, normal CT (174 images), ischemic (IS) CT (157 images) and hemorrhagic (ICH) images (142 images) are available in 3 formats, divided into ROI, Grayscale and DICOM. Since the class types are the same as our project and we have their consent to use the data set in our study, we will use the images here in the stages of stroke presence / absence and stroke type determination.

- https://www.kaggle.com/vbookshelf/computed-tomography-ct-images

This shared data set contains 2500 brain CT images for 82 patients. In addition to approximately 30 images (30 sections) for each patient, there are 318 masked images of the hemorrhage site. Masking in this data set will be used in model trainings to identify bleeding regions and mask them.

5.2 1st Level Diagram

The program consists of many sub-programs. These sub-programs form an integrity with the pipeline structure by connecting to each other from input and output points. The process between the user's providing the input and the stroke detection request, and the output to be given according to the user's demand and input, is given in Figure-5.1.

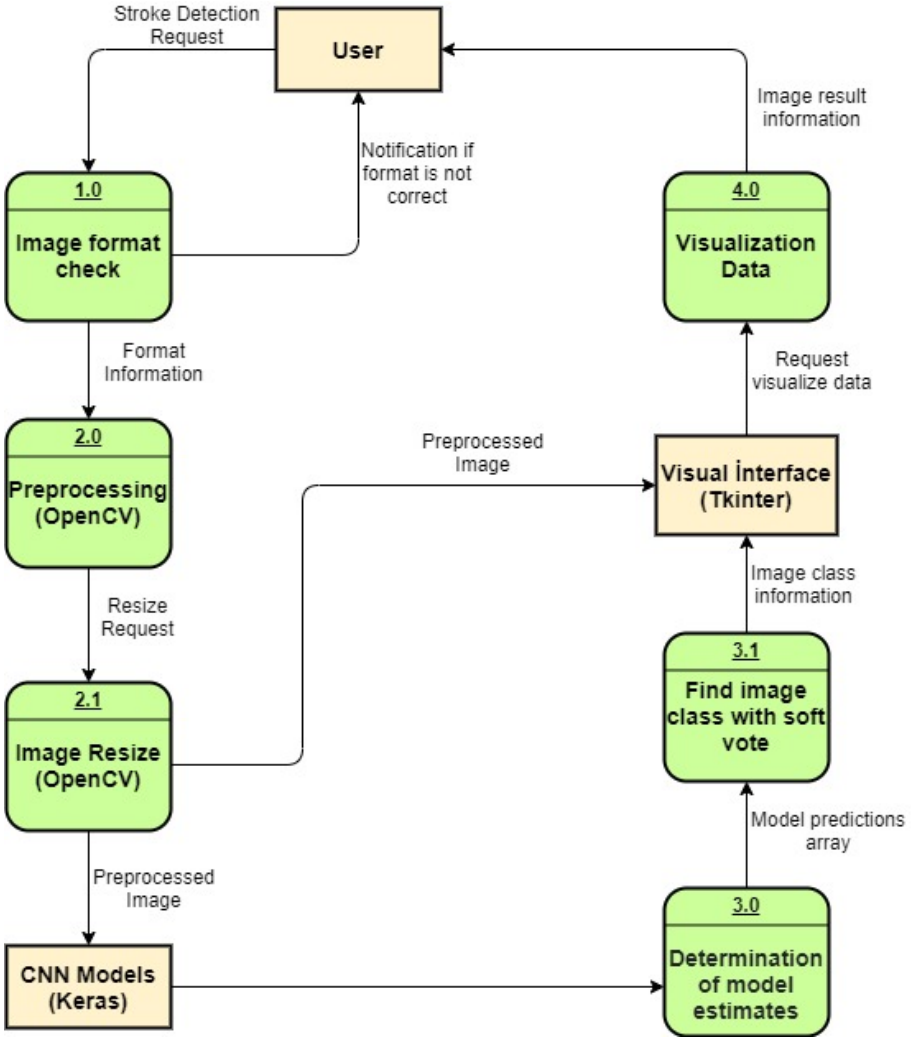

Figure 5.1 1st Level Diagram

6
Implementation

6.1 Stroke/No Stroke Classification

6.1.1 Preprocessing

The brain with DICOM extension, taken from the patients in the datasets we used, was shown by taking horizontal sections from top to bottom. Each of these layers is kept as a separate image, some of the layers belonging to an example patient are shown in Image-6.1

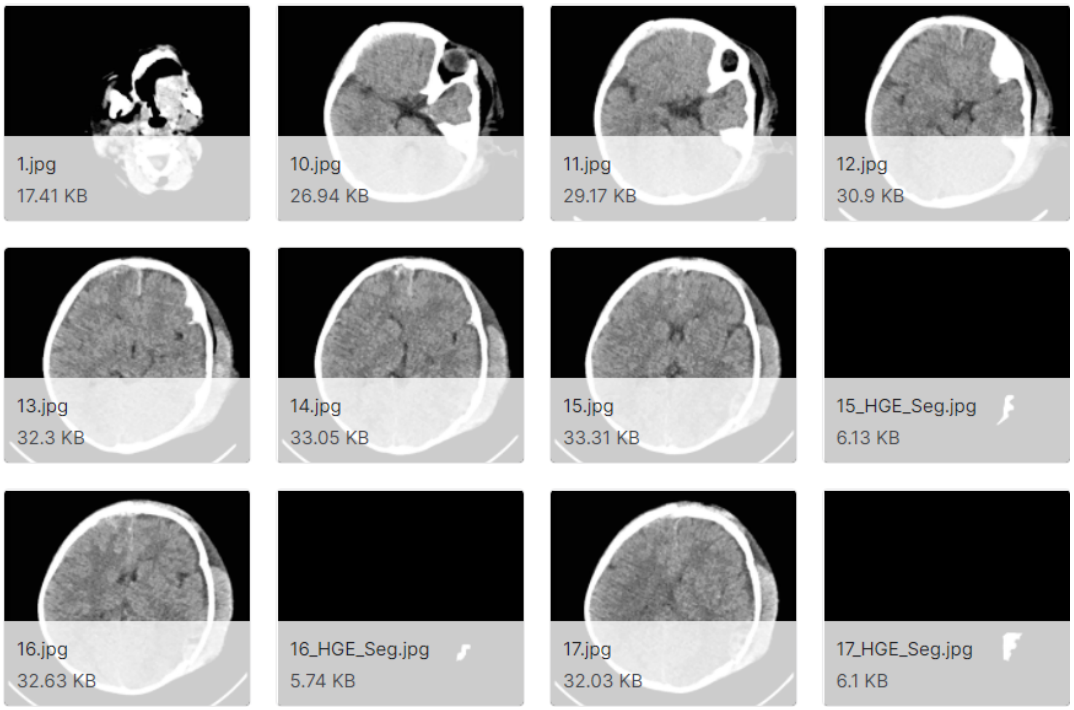

Figure 6.1 Some brain layer images

Since the upper and lower points are narrower in surface area due to the oval structure of the brain, the preprocessing problem involving this narrow area was eliminated in our training set. Since our dataset without stroke image is more dominant, 30% of

the dataset is reserved for training with the help of the function added in Image-6.2 in order to balance it with images containing stroke images.

```
list = np.random.random_integers(1340,size=(308))
select_no_hemoraj = []
for i in range(308):
    select_no_hemoraj.append(no_hemoraj[list[i]])

train_x = np.concatenate((np.array(select_no_hemoraj),np.array(hemoraj)),0)
train_y = np.concatenate((np.zeros(308),np.ones(308)))

train_x,test_x,train_y,test_y = train_test_split(train_x,train_y,test_size = 0.15,random_state=43)

train_yy = []
for i in range(len(train_y)):
    if(train_y[i] == 0):
        train_yy.append([1,0])
    else:
        train_yy.append([0,1])
train_y = np.array(train_yy)

test_yy = []
for i in range(len(test_y)):
    if(test_y[i] == 0):
        test_yy.append([1,0])
    else:
        test_yy.append([0,1])
test_y = np.array(test_yy)
```

Figure 6.2 Balancing of dataset

After this stage, because the images in the shared dataset are of different sizes, the images were brought to a fixed common size.

6.1.2 Splitting of Train and Test

In the code snippet given in Figure-6.2, the steps required to separate training and test data are also shown. While performing this stage, the training and testing parts are divided into 85-15.

6.1.3 VGG-19 Network

With the help of Keras, we created a new model by modifying the input layer of the VGG-19 model in accordance with our own data set. Since we will learn transfer, the pre-trained VGG-19 layers are closed to training. And since there are 2 classes in the first stage of the project, an output layer with 2 classes has been added.

Since we are classifying, categorical_crossentropy is given as the loss parameter.

The codes written while performing these steps are shown in Figure-6.8.

```
model = VGG19(include_top=False, input_shape=(650, 650, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
#class1 = Dense(1024, activation='relu')(flat1)
output = Dense(2, activation='softmax')(flat1)

model = Model(inputs=model.inputs, outputs=output)
model.summary()

opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)

model.fit(train_x,train_y,epochs = 4, batch_size = 64 , verbose = 1)

model.save("/content/drive/MyDrive/model_epochs4batch64")
```

Figure 6.3 Codes of train with VGG-19

6.1.4 Inception Network

```
model = InceptionResNetV2(include_top=False, input_shape=(512, 512, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
```

Figure 6.4 Codes of train with Inception

6.1.5 MobileNet Network

```
model = MobileNet(include_top=False, input_shape=(512, 512, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
#class1 = Dense(1024, activation='relu')(flat1)
output = Dense(2, activation='softmax')(flat1)

model = Model(inputs=model.inputs, outputs=output)
model.summary()
```

```
conv_pw_9_relu (ReLU)        (None, 32, 32, 512)       0
_________________________________________________________________
conv_dw_10 (DepthwiseConv2D) (None, 32, 32, 512)       4608
_________________________________________________________________
conv_dw_10_bn (BatchNormaliz (None, 32, 32, 512)       2048
_________________________________________________________________
conv_dw_10_relu (ReLU)       (None, 32, 32, 512)       0
_________________________________________________________________
conv_pw_10 (Conv2D)          (None, 32, 32, 512)       262144
_________________________________________________________________
```

Figure 6.5 Codes of train with MobileNet

6.1.6 ResNet-50 Network

```
model = ResNet50(include_top=False, input_shape=(512, 512, 3))
model2 = ResNet101V2(include_top=False, input_shape=(512, 512, 3))
model3 = ResNet152V2(include_top=False, input_shape=(512, 512, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
```

Figure 6.6 Codes of train with ResNet-50,ResNet101v2,ResNet152V2

6.1.7 DenseNet-121 Network

```
model = DenseNet121(include_top=False, input_shape=(512, 512, 3))
model2 = DenseNet169(include_top=False, input_shape=(512, 512, 3))
model3 = DenseNet201(include_top=False, input_shape=(512, 512, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
```

Figure 6.7 Codes of train with DenseNet121, DenseNet169, DenseNet201

6.1.8 EfficientNet Network

```
model = EfficientNetB0(include_top=False, input_shape=(512, 512, 3))
model2 = EfficientNetB3(include_top=False, input_shape=(512, 512, 3))
model3 = EfficientNetB7(include_top=False, input_shape=(512, 512, 3))
for layer in model.layers:
    layer.trainable = False
flat1 = Flatten()(model.layers[-1].output)
```

Figure 6.8 Codes of train with EfficientNetB0, EfficientNetB3, EfficientNetB7

6.1.9 Principle of Fusion & Major Voting

It is planned to use 6 different networks in the project. After the normal model training processes of these networks, instead of continuing to identify with the most successful one, which is the method generally preferred, a decision mechanism method different from the majority of the studies in the literature review will be applied. If this major voting method does not unanimously determine the presence/absence of stroke for 1 image, it will be decided that there is too much game. In this way, it will been sured that the total number of features extracted from the input data and evaluated in decision making will be increased and the decision-making phase will be made as a result of the joint work of networks with different feature inferences, and more precise results will be obtained. As can be seen in the related image, Image-6.9 where we explain this mechanism, stroke label can be given to the image in cases where 2 out of 6 networks detect noStroke and 4 of them detect Stroke. In addition, when the system remains in the 3-3 state, the decision of MobileNet, which is observed to have the lowest success rate, is ignored and the result is decided. The accuracy of the decision made at the last stage will not depend on a single model, the different vulnerabilities of each network will thus be more compensated in each forecast process.

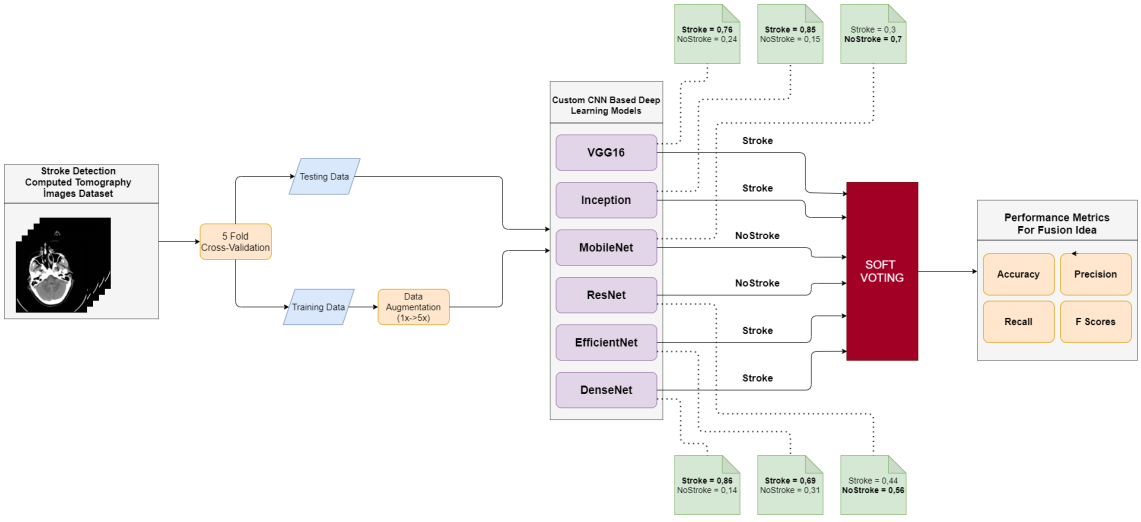

Figure 6.9 Fusion algorithm of the project

6.2 Detection of Ischemic-Hemorrhage and Masking

6.2.1 Mask R-CNN

In the segmentation part of the second phase of the project, Mask R-CNN network was chosen to have a better IoU score by approaching the localization problem in a more precise manner. It is aimed to mask the entire stroke region. This network uses the Feature Pyramid Network (FPN) and ResNet101 networks as backbones.

Mask R-CNN is an extension on Faster R-CNN which adds the functionality of predicting segmentation masks on each RoI. This functionality is achieved with a small Fully Convolutional Network applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner [8].

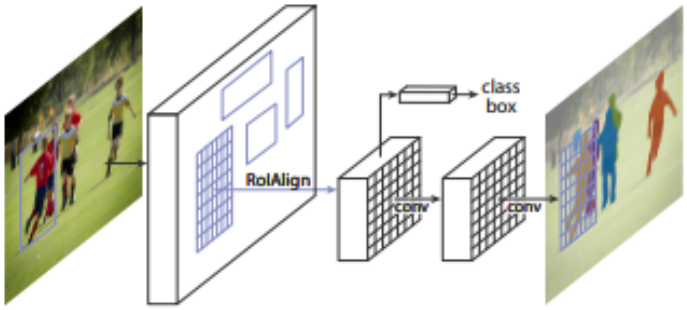

Figure 6.10 The Mask R-CNN framework for instance segmentation. [8]

6.2.1.1 Anotation & JSON Format

To prepare the dataset for masking, the first thing to do is to apply annotation to the dataset and as a result get a JSON file. In this process, using the VGG Anotator tool, the areas to be masked in all images are selected polygonally and the information of the x and y coordinates around each mask region is kept. However, since our masked ROI images are ready in our dataset, we calculated the coordinate information of the resulting edges after first applying edge detection to the masked image using edge detection and OpenCV functions, instead of manually annotating nearly 2000 images one by one. The codes written to obtain JSON format from all images to be used for training are given in Figure-6.11 and Figure-6.12.

JSON format content of a demo image is shown in the Figure-6.13.

```
end_string =""
for i,path in enumerate (images_paths_list):

    c = contours_list[i]

    name = "\"polygon\""
    name_string = "\"name\":" + name

    ## filename
    filename = path
    filename_string = "\"filename\":" + "\"" + filename + "\""

    ## size
    size = "1032"              # size bilgisini öğren, değiştir
    size_string = "\"size\":" + size

    for each in c:
        x = each[:,:,0]
        y = each[:,:,1]

        x = x.reshape(x.shape[0])
        y = y.reshape(y.shape[0])

        all_points_x = "["
        all_points_y = "["
        for j in range(len(x)):
            all_points_x = all_points_x  + str(x[j]) + ","
            all_points_y = all_points_y  + str(y[j]) + ","

        all_points_x = all_points_x[:len(all_points_x) -1] + "]"
        all_points_y = all_points_y[:len(all_points_y) -1] + "]"
    ## regions
        all_points_x_string = "\"all_points_x\":" + all_points_x

        all_points_y_string = "\"all_points_y\":" + all_points_y
```

Figure 6.11 Converting JSON Code Part-1

```
    all_points_x_string = "\"all_points_x\":" + all_points_x

    all_points_y_string = "\"all_points_y\":" + all_points_y

    shape_attributes = "{" + name_string + "," + all_points_x_string + ","
    + all_points_y_string + "}"
    shape_attributes_string = "\"shape_attributes\":" + shape_attributes

    names = "\"iskemik\""
    names_string = "\"names\":" + names

    region_attributes = "{" + names_string + "}"
    region_attributes_string = "\"region_attributes\":" + region_attributes

    regions = "["
    regions_element = "{" + shape_attributes_string +","
    + region_attributes_string + "}"
    regions = regions + regions_element + "," + regions_element + "]"
    regions_string = "\"regions\":" + regions

##file_attributes
file_attributes = "{}"
file_attributes_string = "\"file_attributes\":" + file_attributes

##image
image = "{" + filename_string +"," + size_string + "," + regions_string
+ "," + file_attributes_string + "}"
image_string = "\"" + filename + size +"\":" + image

end = image_string + ","
end_string = end_string + end
```

Figure 6.12 Converting JSON Code Part-2

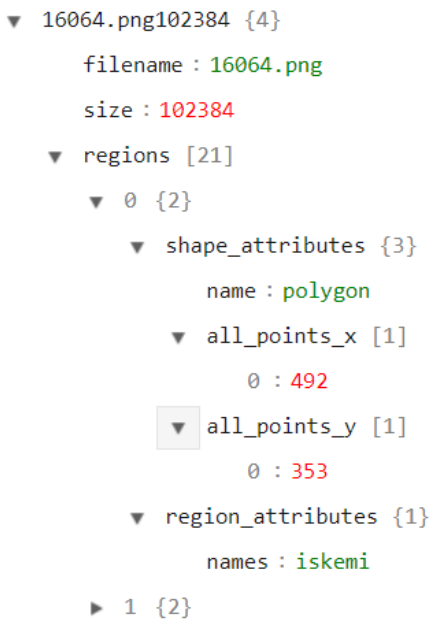

Figure 6.13 JSON Format

- size -> Image file byte size
- regions -> mask regions
  - shape_attributes: mask drawing tool type(rectangular etc.)

    all_points_x: x coordinate

    all_points_y: y coordinate
  - region_attributes: class name

Since each image has more than one mask, we used the findContours function of the OpenCV library while applying edge detection to the our images. With this function, the x and coordinate information of all ROIs in the image were also found.

```
with os.scandir("C:/Users/emine/MASK/MASK/") as file:
    for img_size in file:
        print(img_size.name,"isimli görüntü size(Byte): ",os.stat(img_size).st_size)
        a = os.stat(img_size).st_size
        images_size_list.append(a)

for f1 in data_path:
    images_paths_list.append(f1)
    img = cv2.imread(img_dir+f1)
    img_grey = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(img_grey, 25, 255, 0)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours_list.append(contours)
    data.append(img)
```

Figure 6.14 Code of find contours and image size

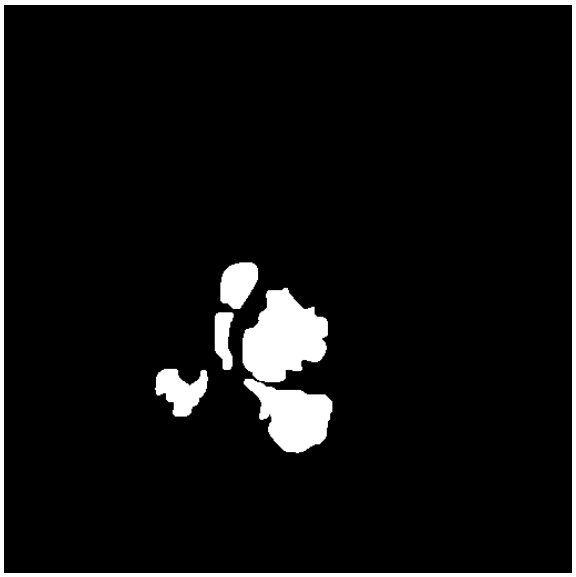

Figure 6.15 After edge detection and find contours

6.2.1.2 Prepare Validation and Train Data

Train and test data are placed in different directories, and their information is stored in one single JSON file each. These JSON files are placed in the same directory as the data. For the Mask R-CNN model to work correctly, it is necessary to prepare the data and read the JSON file, by applying the methods specified in the original Mask R-CNN GitHub Repository. These methods are not suitable for implementing directly in this project, so some functions and variables need to be adapted to the problem. For this purpose, a CustomDataset class that extends the Dataset class under mrcnn/utils.py has been created and the load_custom, load_mask, image_reference functions in the parent class have been overridden according to the numbers and names of the classes in the data set. Most of the training code given in the appendices for Mask R-CNN consists of these functions.

The created CustomDataset class and its functions are used in the "train" function in the code. The lines written for the purpose of reading the data from the data set in the train function in the code are given in the Figure-6.16.

```
# Train set.
train_data = CustomDataset()
train_data.load_custom(os.path.join(ROOT_DIR, "Dataset"), "train")
train_data.prepare()

# Validation set.
val_data = CustomDataset()
val_data.load_custom(os.path.join(ROOT_DIR, "Dataset"), "val")
val_data.prepare()
```

Figure 6.16 Reading From Dataset for the Training of Mask R-CNN

Although most of the images in the data set adopted a size standard, some sizes do not fit with the others. For this reason, while reading the data set, the dimensions of each image were also measured one by one, which influenced the performance drastically.

Since training a complex network structure like Mask R-CNN from scratch would require high processing power and time, weights taken from Mask R-CNN's training already done with the famous Microsoft Common Objects in Context dataset were loaded into the model before the train process started [9]. The next step after this stage is to start the train process.

6.2.1.3 Train Phase

In different experiments, the number of epochs and the selection of layers to be trained were kept as variables and the effects of these changes on the resulting model were observed.

By overriding the parameters of the Config class under mrcnn/config.py, it is aimed that the training process will progress at an acceptable level accordingly with the available processing power, but that the data set will be recognized successfully by the model. Among these configuration parameters, STEPS_PER_EPOCH directly affects how often the model will be validated and logged, the DETECTION_MIN_CONFIDENCE parameter directly affects the probability threshold of prediction which below will be ignored, and VALIDATION_STEPS directly affects how many steps the current weights will be validated after each epoch. Besides layer selection and epoch count, these parameters were also updated with different values during the training process.

As mentioned, the change of configuration parameters directly affected the training speed. With this effect, the time required for a single epoch in the training process varied between 1 minute and 3 minutes. Although it is known that keeping it high has a positive effect on the validation accuracy, the VALIDATION_STEPS parameter

has the biggest share in this performance change.

As a result of the validation process performed at the end of each epoch, different types of loss values were obtained. These values are taken as the main success measure of the training process. Measurements expressed by different loss values are given in Table-6.1. Loss functions are provided for both train data and validation data (with the val_ prefix) during training.

rpn_class_loss	Indicator of the success of RPN separating background and objects.
rpn_bbox_loss	Indicator of the success of RPN localizing objects.
mrcnn_bbox_loss	Indicator of the success of Mask R-CNN localizing objects.
mrcnn_class_loss	Indicator of the success of Mask R-CNN recognizing classes of objects.
mrcnn_mask_loss	Indicator of the success of Mask R-CNN segmenting objects.
loss	Combination of all other losses.

Table 6.1 Loss Functions for Mask R-CNN Training

During the training, the resulting weights at the end of each epoch were recorded as logs. Each recorded model with the ".h5" extension has a size of 250 megabytes approximately.

During the training process, validation loss values were followed closely for each epoch. The most important parameter for the success criterion of a model is the loss values, which allow us to measure its success from different perspectives. Among the models obtained after the training process, the successful ones were measured with both these loss values and the tests applied on the tomography images with the selected CT images outside of the training data. After these tests, a mean IoU score for each selected model was obtained.

Validation loss change graphs observed for four different scenarios will be shared in this section.

Parameters for the training scenario #1 are shown at Table-6.2. For these parameters, validation loss change graph is shown at the Figure-6.17

IMAGES_PER_GPU	1
STEPS_PER_EPOCH	100
DETECTION_MIN_CONFIDENCE	0.9
VALIDATION_STEPS	10
Epoch Count	20
Layers to Train	The RPN, classifier and mask heads of the network

Table 6.2 Parameters for Mask R-CNN Training - Scenario 1

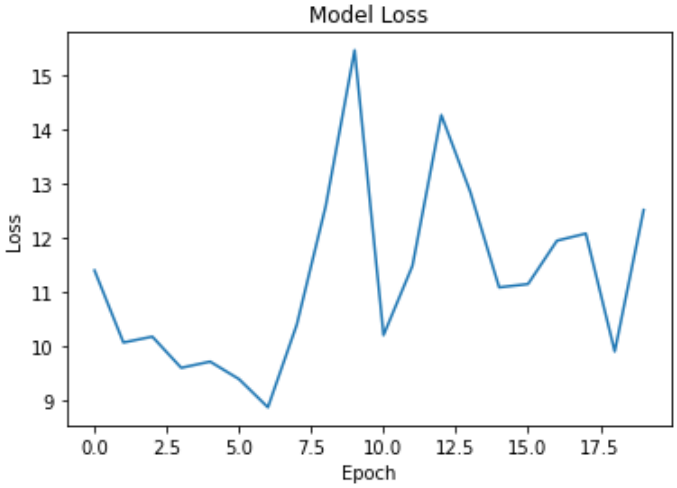

Figure 6.17 Mask R-CNN val_loss Change Graph For Training Scenario 1

Parameters for the training scenario #2 are shown at Table-6.3. For these parameters, validation loss change graph is shown at the Figure-6.18

IMAGES_PER_GPU	2
STEPS_PER_EPOCH	100
DETECTION_MIN_CONFIDENCE	0.8
VALIDATION_STEPS	20
Epoch Count	50
Layers to Train	The RPN, classifier and mask heads of the network

Table 6.3 Parameters for Mask R-CNN Training - Scenario 2

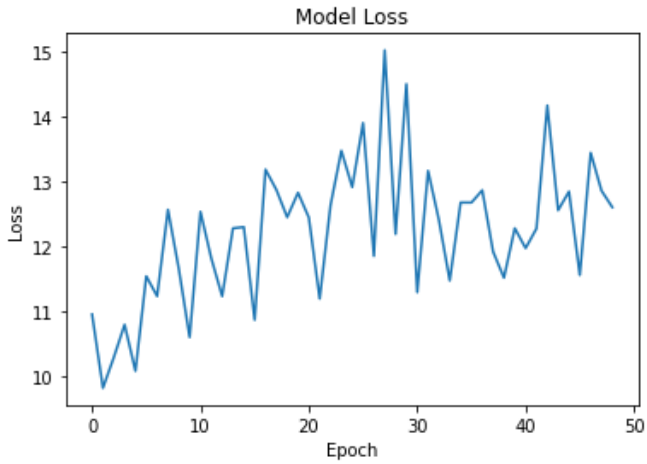

Figure 6.18 Mask R-CNN val_loss Change Graph For Training Scenario 2

Parameters for the training scenario #3 are shown at Table-6.4. For these parameters, validation loss change graph is shown at the Figure-6.19

IMAGES_PER_GPU	2
STEPS_PER_EPOCH	100
DETECTION_MIN_CONFIDENCE	0.8
VALIDATION_STEPS	10
Epoch Count	75
Layers to Train	ResNet stage 3 and up

Table 6.4 Parameters for Mask R-CNN Training - Scenario 3

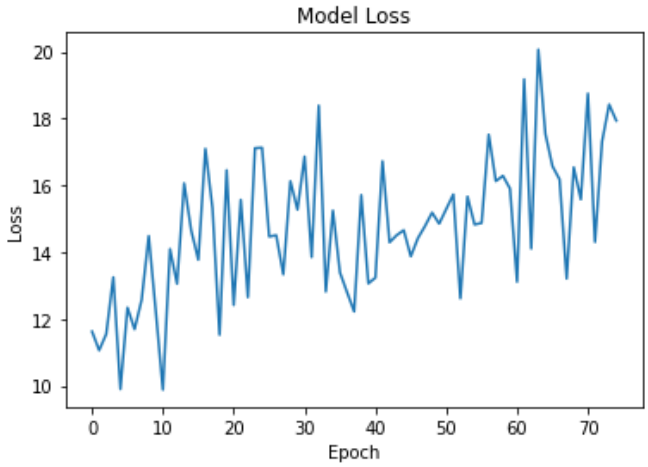

Figure 6.19 Mask R-CNN val_loss Change Graph For Training Scenario 3

Parameters for the training scenario #4 are shown at Table-6.5. For these parameters, validation loss change graph is shown at the Figure-6.20

IMAGES_PER_GPU	2
STEPS_PER_EPOCH	50
DETECTION_MIN_CONFIDENCE	0.8
VALIDATION_STEPS	10
Epoch Count	50
Layers to Train	All the layers

Table 6.5 Parameters for Mask R-CNN Training - Scenario 4

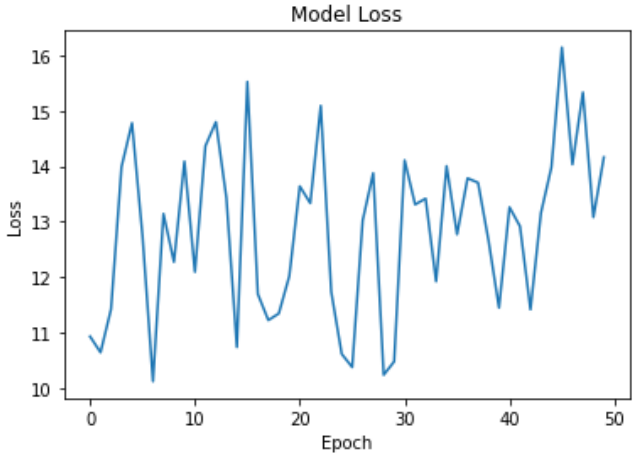

Figure 6.20 Mask R-CNN val_loss Change Graph For Training Scenario 4

6.2.1.4 Testing & IoU Phase

Validation data was used to measure success without affecting the weights during model training. For this reason, the Loss scores described in the previous section are also a factor for measuring the success of the model, so they can also be included in the testing process.

For having a better understanding about the success of a trained model, the mask predictions and thier IoU scores are significant. Among more than one hundred models stored as log records, those with high potential were tested with selected CT images, again, outside the train data set. Some examples of these tests will be shared in detail in the next section. The section after that, the mean IoU score of each selected model obtained from a prediction test held with 510 non-training CT images will be presented. In this section, some parts of the test code used in the manual test (shown in the next chapter) will be given.

The code snippets of the prediction and IoU score calculation can be found at Figures 6.21 and 6.22, respectively.

```
# Predicting images which are not in the training set.

img_name = "15704.png"

# Getting the image to be predicted
image_path = ROOT_DIR + "test_imgs/" + img_name
test_image = skimage.io.imread(image_path)

# Fitting the image into processable form by Mask R-CNN Network
if test_image.ndim != 3:
    test_image = skimage.color.gray2rgb(image1)
# Remove alpha channel for consistency, if it has one
if test_image.shape[-1] == 4:
    test_image = test_image[..., :3]

results = model.detect([test_image], verbose=1) # Returns the class prediction, mask and the box
results = results[0]

# Display prediction results
ax = get_ax(1)
visualize.display_instances(test_image, results['rois'], results['masks'], results['class_ids'],
["bg", "İskemik", "Kanamalı"], results['scores'], ax=ax, title="Prediction")
```

Figure 6.21 The Code Snippet In Which Testing Predictions Are Made

```
# Getting the mask image provided for the test image
path_to_mask = ROOT_DIR+"test_masks/"+img_name
mask = skimage.io.imread(path_to_mask)

# Extracting mask pixels in bool manner
mask_extracted = []
for x in mask:
  line = []
  for y in x:
    if y.any() != 0:
      line.append([True])
    else:
      line.append([False])
  mask_extracted.append(line)
mask_extracted = np.array(mask_extracted)

# Prediction results include prediction mask data.
# Forming this data into the same way as we have.
extracted_pred_mask = []
for x in results['masks']:
  line = []
  for y in x:
    if y.any() != 0:
      line.append([True])
    else:
      line.append([False])
  extracted_pred_mask.append(line)
extracted_pred_mask = np.array(extracted_pred_mask)

# IoU Score
utils.compute_overlaps_masks(mask_extracted, extracted_pred_mask)
```

Figure 6.22 The Code Snippet In Which IoU Scores For Mask Predictions Are Obtained

6.2.2 YOLOv3

6.2.2.1 Anotation & YOLO Format

When we create our images in YOLO format, if we take a look at what information is kept in the ".txt" file with the same name next to our image, we see Table-6.6 as <image-class-id> <x-coordinate> <y-coordinate> <width> <height> It turns out a n*5 matrix as in .

image-class-id	x-coordinate	y-coordinate	width	height height

Table 6.6 Content of A File As YOLO Format

The first of these parameters, <object-class-id> : represents the appropriate class id from 0 to class number(N)-1 in the "train/classes.names" file. While keeping the height and width information, the <x-coordinate> and <y-coordinate> parameters give us the x and y coordinates of the center of the Bounding box, respectively.

0	0.697265625	0.6044921875	0.0078125	0.009765625
0	0.2939453125	0.5791015625	0.103515625	0.072265625
0	0.6953125	0.5927734375	0.0859375	0.111328125
0	0.4892578125	0.4052734375	0.033203125	0.044921875

Table 6.7 Content of Hemorrhage Class .txt File

1	0.544921875	0.708984375	0.07421875	0.08984375
1	0.4208984375	0.7216796875	0.162109375	0.189453125
1	0.599609375	0.4892578125	0.10546875	0.177734375

Table 6.8 Content of Ischemic Class .txt File

6.2.2.2 Prepare Validation and Train Data

After the images are separated and the .txt files are ready, the first thing to do is to open the train.data file. The content of this file is as in Image-6.23.

```
classes = 2
train = /content/drive/MyDrive/data_set/train.txt
valid = /content/drive/MyDrive/data_set/validation.txt
names = /content/drive/MyDrive/data_set/data.names
backup = /content/drive/MyDrive/data_set/train/
```

Figure 6.23 Content of train.data File

After the pre-processing for our YOLOv3-based training is finished, we copy the contents of the cfg/yolov3.cfg configuration file, which contains all the settings, layers,

filters and parameters that the model needs, into the stroke.cfg file we just created under the dataset folder.

The batch parameter is then set to 64, which refers to dividing the images into small subsets to update the weights iteratively.

Third, we set the maximum number of iterations required to train our network as max_batches=4000. At the same time, we update %80 of the maximum number of batches and %90 3200 and 3600, which are 3200 and 3600, to the steps section.

Then 610,696,783 in the [yolo] and [convolutional] layers that precede the [yolo] layer in the file. The classes sections in the rows are updated with our class number of 2.

The number of filters to be used varies depending on the number of classes in our dataset and is calculated with the formula filters=(number of classes + 5) * 3. So finally 603,689,776 in the train.cfg file. The filters values in the lines are updated to (2+5)*3="21" for our model.

6.2.2.3 Train Phase

After all changes which mentioned in subsection 6.2.2.2 are saved in the dataset/train.cfg file, the necessary settings for the training are completed. Since the training is now ready, the command is written to start the training with the help of darknet detector by giving the paths of the .data and train cfg files to start the training.

- !./darknet detector train dataset/train.data dataset/train.cfg darknet53.conv.74

6.2.2.4 Testing & IoU Phase

During the training, the weigths obtained every 1000 epochs were recorded. These weights have been tested starting from the 2000th epoch. In the performance analysis section, the loss values in the 2000, 3000 and 4000 epochs are shown by comparing them in a table.

As a result of the training, we benefited from the ".weight" and ".cfg" files we obtained for testing. Images one by one can be tested on Colab with the command:

- !./darknet detector test /dataset/train.data /dataset/testing.cfg /dataset/train/training_2000.weights /dataset/images/test.jpg

- !./darknet detector test /dataset/train.data /dataset/testing.cfg /dataset/train/training_3000.weights /dataset/images/test.jpg
- !./darknet detector test /dataset/train.data /dataset/testing.cfg /dataset/train/training_4000.weights /dataset/images/test.jpg

and "predictions.jpg" with detection object is obtained as output.

6.2.3 U-Net

6.2.3.1 Anotation

Since U-Net is a segmentation network that can be imported directly through Keras, the annotation part was not as troublesome as other networks. The format that U-Net expects from us for training is only png versions of masks and training images. For this stage, the images we will be interested in are divided into folders and the file paths are given in the relevant part of the code.

6.2.3.2 Prepare Validation and Train Data

After the annotation steps were performed, the dataset was separated into validation and train and made ready for training.

6.2.3.3 Train Phase

The U-Net algorithm produces an output containing the image on which the object will be detected as input and mask information in image size as output. For example, if the size of our input image (128,128.3) is the size of our output image (128,128.1). Our input image is in RGB and our output image is in gray image format. In our output image, the pixel value of the regions containing objects is 1 and the remaining regions are 0. Therefore, when we perform a test using the model, it gives us a matrix output with pixel values varying between 0-1 (128,128.1). We, on the other hand, decide that if the pixel value is greater than 0.5, this pixel belongs to the object, and if it is less than 0.5, it does not belong to any object. In the dataset we use, the images are in (512,512,3) format. We change the size of our input images to (128,128,3) because the model works effectively in (128,128,3) size. In the dataset we use, the masks that indicate the object are in a different picture (512,512,3) in size, and pixels containing objects (255,0,0) are blue and pixels that do not contain objects are (0,0,0). In order for the size and color format of our mask dataset to be suitable for our model, we make the size from (512,512,3) to (128,128,1) and pixels containing objects (255) become white and pixels without objects (0) are black. Then we divide the picture by

255, since our object class value is 1. Thus, the object regions with a pixel value of 255 have a value of 1. Finally, this dataset is given to U-Net and the training of the model is started.

6.2.3.4 Testing & IoU Phase

The function written to test all images with the model.h5 and calculate total IoU values obtained after the training is completed is shown in Figure-6.24 and Figure-6.25. In the testing phase of our model, our input image is given to the model and a matrix varying between 0-1 values is obtained as output. Pixels with a value greater than 0.5 are considered to belong to the object and their values are 255. Values less than 0.5 are considered not to belong to the object and their values are 0. Then, the generated output mask and the real mask are compared to find the IoU. Pixels with the value of 255 in the mask we produce are compared with the pixels with the value of 255 in our real mask. Pixels that are common to both masks are considered as intersection pixels. The intersection pixels and the sum of the values in different pixels in both masks are also considered as union pixels. IoU is calculated with the basic calculation formula of IoU metric = (intersection/(intersection+union)). In order to provide visual convenience to the user, we paint the common points of both masks in blue, only the points of the mask we produce are red, and only the points that our real mask finds in green.

One of the images seen as output is shown in Image-6.26 as an example. As explained above, the blue areas represent the intersection, the red areas represent the predictor, and the green areas represent the real mask.

```
y = model.predict(image)
y = y>0.5
y = y*255
y = np.reshape(y,(128,128))

y512 = cv2.resize(y, (512, 512))

for i in range(512):
    for j in range(512):
        if(mask512[i][j]>0):
            if(y512[i][j]>0):
                image_s[i][j][0] = 255
            else:
                image_s[i][j][1] = 255
        elif(y512[i][j]>0):
            image_s[i][j][2] = 255
```

Figure 6.24 Code of Coloring Mask with U-Net

```
result = model.predict(test_image)
result = result > 0.2
result = result*255

total_union = 0
total_intersection = 0

percentages = []

for k in range(test_mask.shape[0]):
  union = 0
 intersection = 0
  for i in range(test_mask.shape[1]):
    for j in range(test_mask.shape[2]):
      if(test_mask[k][i][j] == 255):
        if(result[k][i][j] == 255):
          total_union +=1
          total_intersection +=1
          union+=1
          intersection+=1
        else:
          total_union +=1
          union+=1
      elif(result[k][i][j] == 255):
        total_union += 1
        union+=1
  percentages.append(intersection/union*100)

percentage_total = 0
for i in range(len(percentages)):
  percentage_total += percentages[i]

print("Iterative = " + str(total_intersection/total_union*100))
print("Cumulative = " + str(percentage_total/len(percentages)))
```

Figure 6.25 Code of Calculate U-Net IoU Values

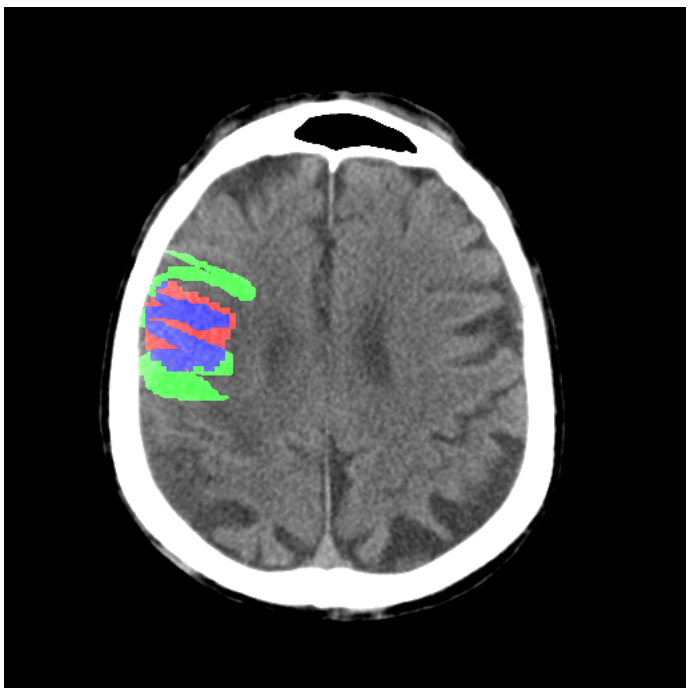

Figure 6.26 Mask Coloring Output

7
EXPERIMENTAL RESULTS

7.1 Fusion Results

After fusion using with 1 model, values of accuracy is seen at Table-7.1 As a result of numerous trainings, the success of the most successful models in 6 networks is as follows. As can be seen from this table, the highest accuracy value among the 6 networks was obtained with ResNet-50 and the lowest accuracy was obtained with MobileNet. The accuracy values of the models with 30 epochs of training generally ranged from 90 to 94.26 at the end of 30 epochs.

Model Name	Accuracy
MobileNet	%93.29805996
ResNet50	%94.26807760
VGG19	%93.38624338
Inception	%93.73897707
DesNet	%94.09171075
Efficient	%93.47442680
General	%96.47266313

Table 7.1 Accuracy Values Before Fusion

Fusion values with 2 model from each of accuracy is seen at Table-7.2

7.2 Mask R-CNN Results

In this section, the segmentation success of different models selected from different scenarios for the randomly selected images outside of the train data will be shown. This will be an introduction and an illustration of the IoU test applied to the selected models in order to have a better understanding of their success rates. The results of this IoU test are presented in the next chapter, at Table-8.1.

In the training scenario #4 (see Table-6.5), the model that obtained in the end of 30th epoch and its segmentation performance for a random CT image is given in Figure-7.1

Model Name	Accuracy
MobileNet1	%92.857142
MobileNet2	%93.386243
ResNet50_1	%94.003527
ResNet50_2	%92.945326
VGG19_1	%93.298059
VGG19_2	%93.915343
Inception_1	%93.033509
Inception_2	%92.768959
DesNet_1	%92.328042
DesNet_2	%92.328042
Efficient_1	%94.444444
Efficient_2	%93.650793

Table 7.2 Accuracy Values With 2 Model Each Network

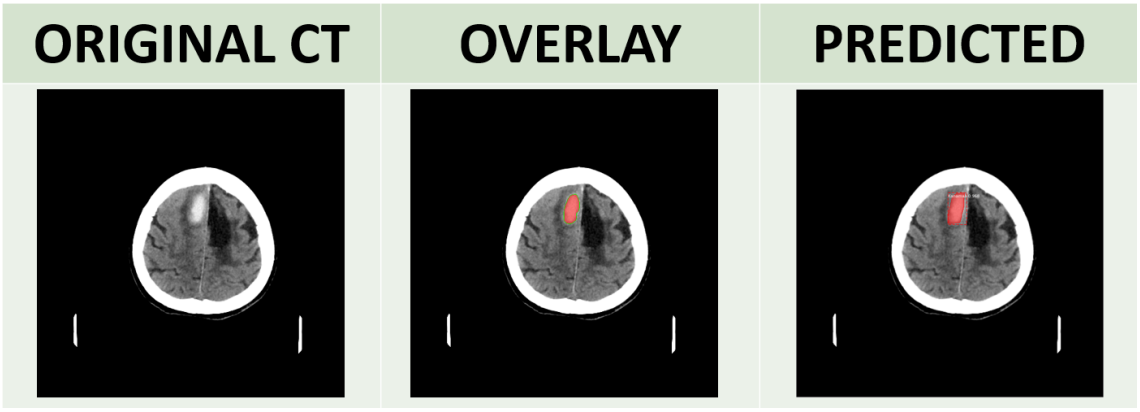

Figure 7.1 Segmentation Performance of the 30th Model From Scenario #4

In the training scenario #4 (see Table-6.5), the model that obtained in the end of 53th epoch and its segmentation performance for a random CT image is given in Figure-7.2

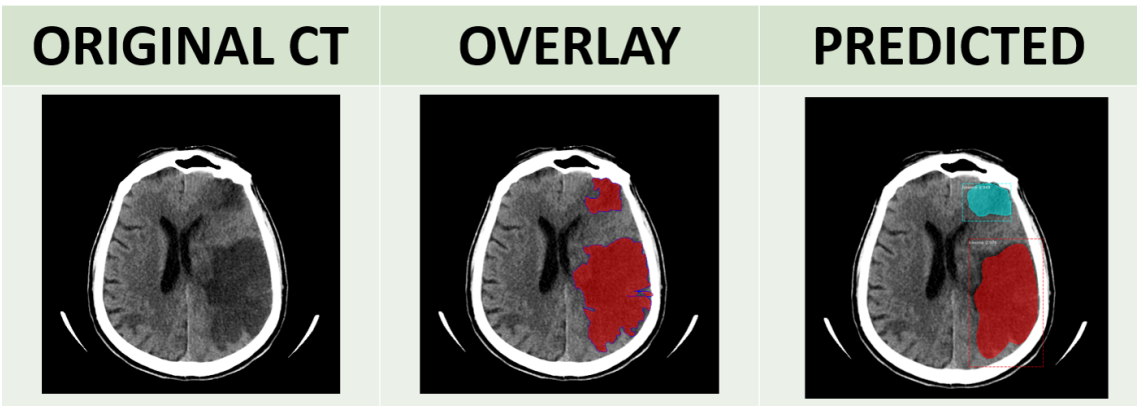

Figure 7.2 Segmentation Performance of the 53th Model From Scenario #4

In the training scenario #3 (see Table-6.4), the model that obtained in the end of 11th epoch and its segmentation performance for a random CT image is given in Figure-7.3

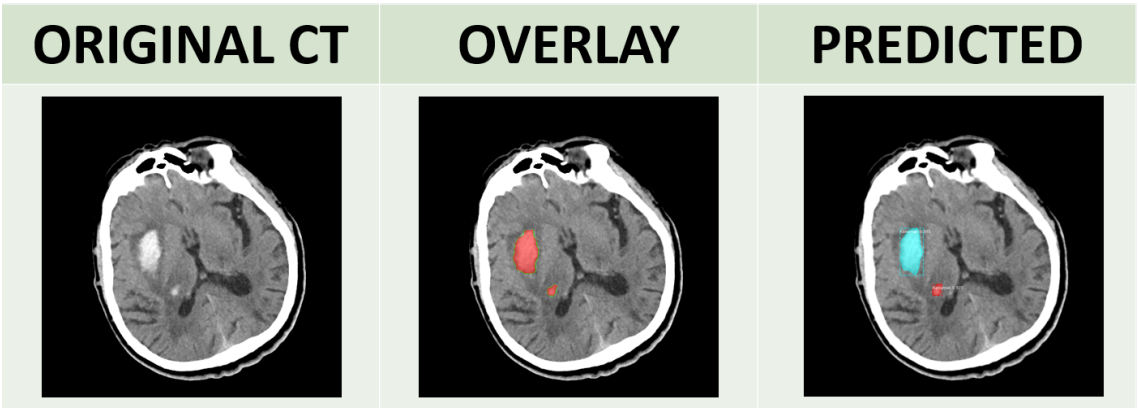

Figure 7.3 Segmentation Performance of the 11th Model From Scenario #3

In the training scenario #3 (see Table-6.4), the model that obtained in the end of 11th epoch and its segmentation performance for a random CT image is given in Figure-7.4

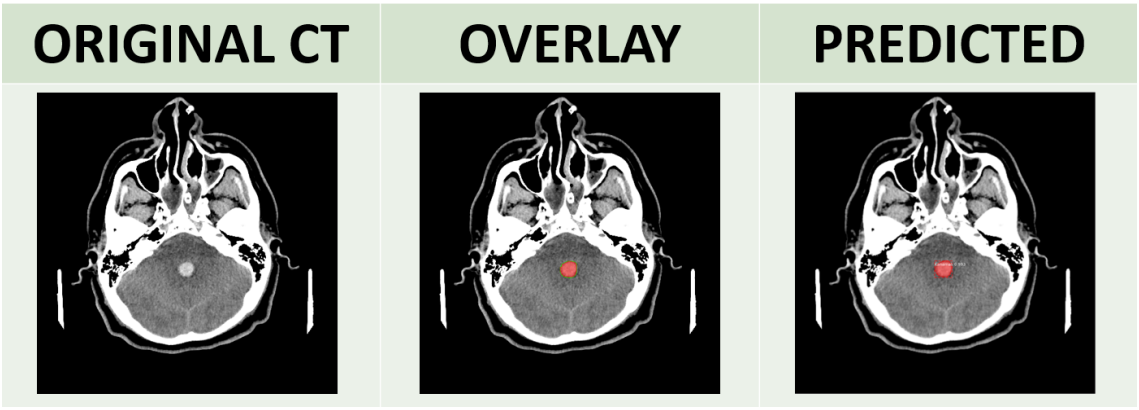

Figure 7.4 Segmentation Performance of the 11th Model From Scenario #3

In the training scenario #3 (see Table-6.4), the model that obtained in the end of 53th epoch and its segmentation performance for a random CT image is given in Figure-7.5

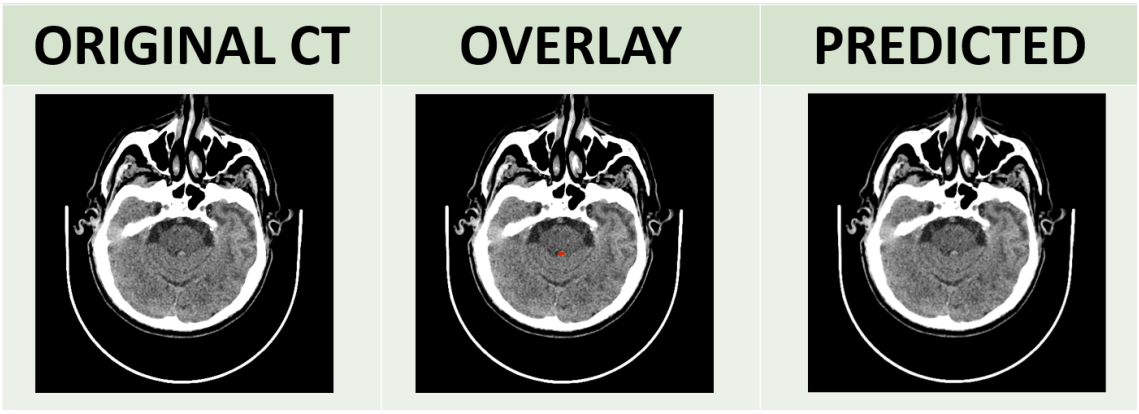

Figure 7.5 Segmentation Performance of the 53th Model From Scenario #3

In the training scenario #3 (see Table-6.4), the model that obtained in the end of 53th epoch and its segmentation performance for a random CT image is given in Figure-7.6

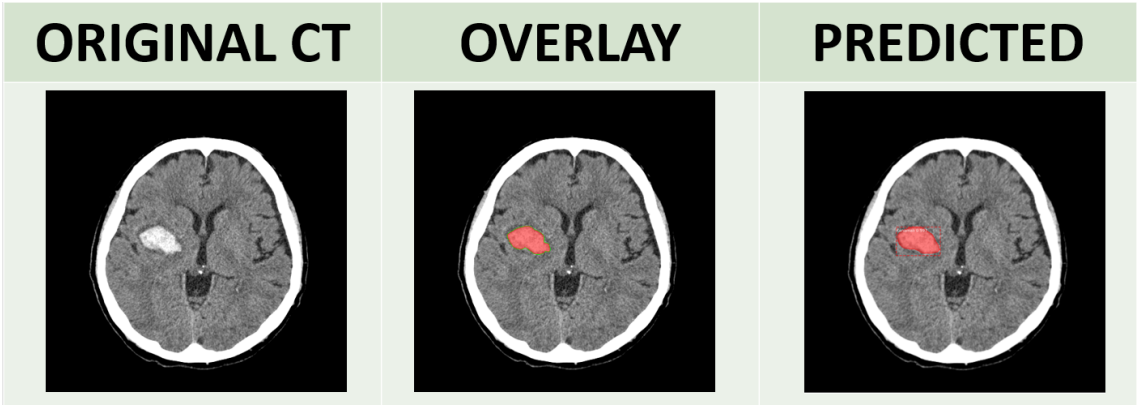

Figure 7.6 Segmentation Performance of the 53th Model From Scenario #3

The success rate of the test samples shown can be better understood by obtaining the IoU scores. The IoU scores for each prediction are given in the Table-7.3.

Test #	Shown In	IoU Score	Class Prediction
1	Figure-7.1	0.8262787	Correct
2	Figure-7.2	0.7623966	Correct
3	Figure-7.3	0.82919914	Correct
4	Figure-7.4	0.8605634	Correct
5	Figure-7.5	0	Correct
6	Figure-7.6	0.9194741	Correct

Table 7.3 IoU Scores For Different Predictions Made Using Trained Mask R-CNN Models

7.3 U-Net Results

In this section, after the training with the U-Net network, the outputs of the test images made with randomly selected images are given. The blue color used in the images represents the intersection of prediction and real bleeding, that is, the mask area that can be detected successfully. The prediction output of our model is shown in red, and the actual bleeding mask in green. Thus, the intersection and missed areas on the CT image can be easily distinguished from the outside.

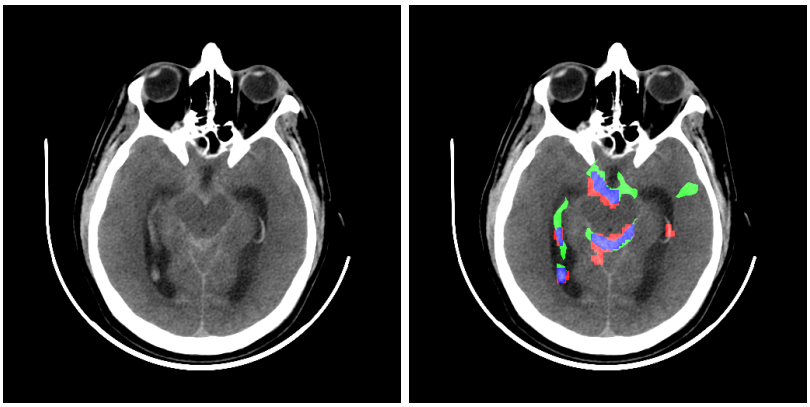

Figure 7.7 Ground Truth (Left), Prediction Mask(Right)

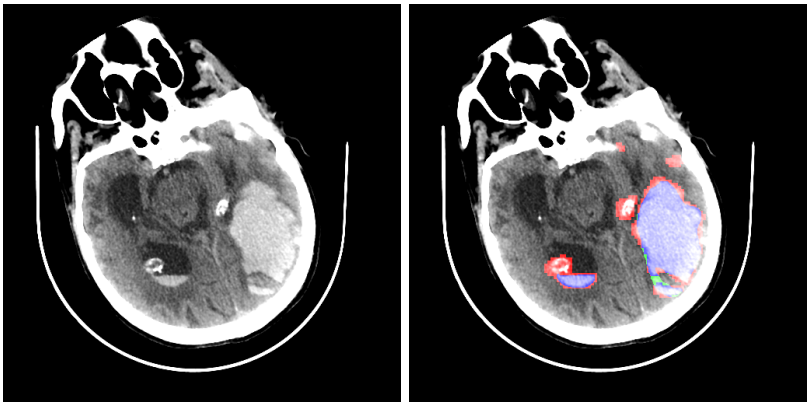

Figure 7.8 Ground Truth (Left), Prediction Mask(Right)

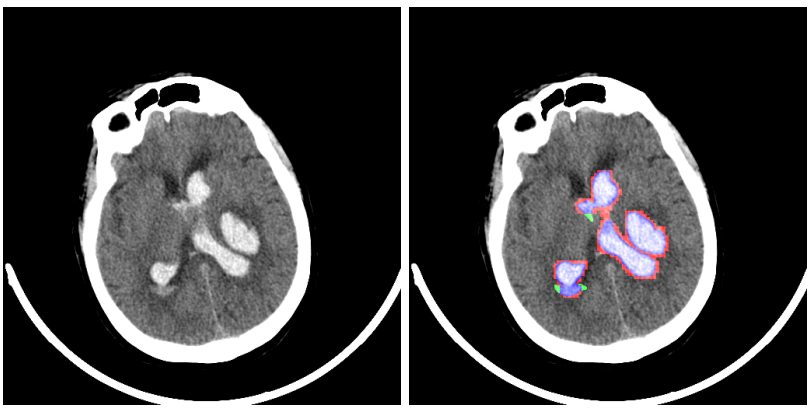

Figure 7.9 Ground Truth (Left), Prediction Mask(Right)

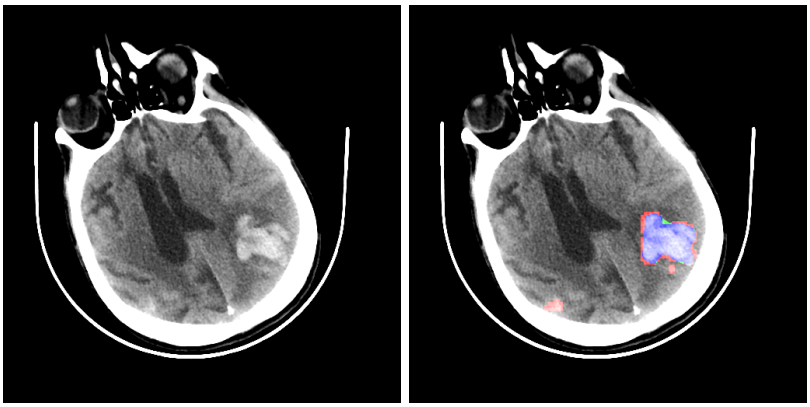

Figure 7.10 Ground Truth (Left), Prediction Mask(Right)

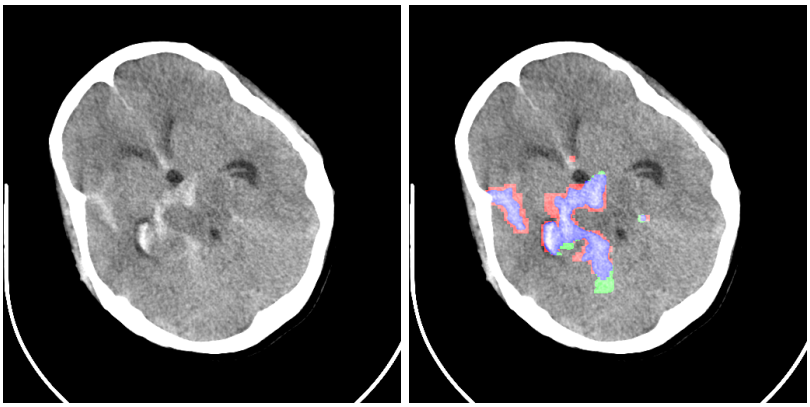

Figure 7.11 Ground Truth (Left), Prediction Mask(Right)

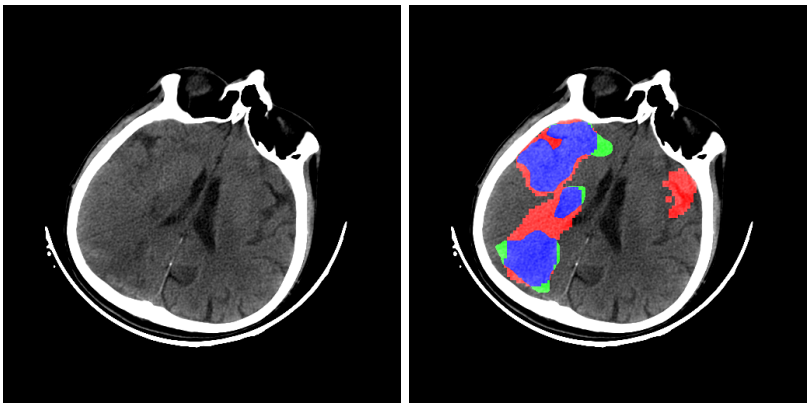

Figure 7.12 Ground Truth (Left), Prediction Mask(Right)

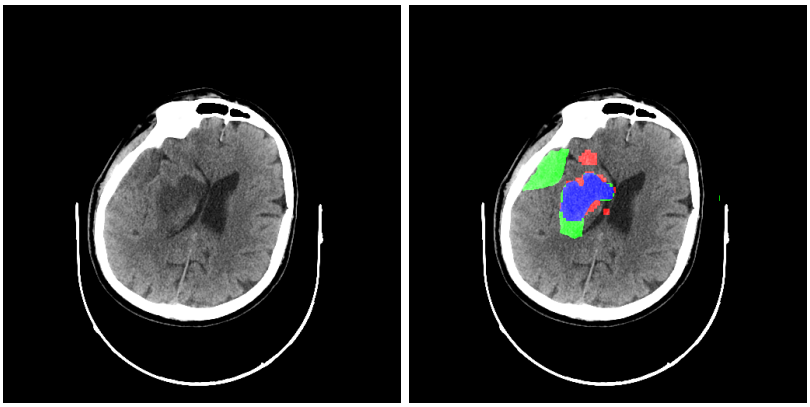

Figure 7.13 Ground Truth (Left), Prediction Mask(Right)

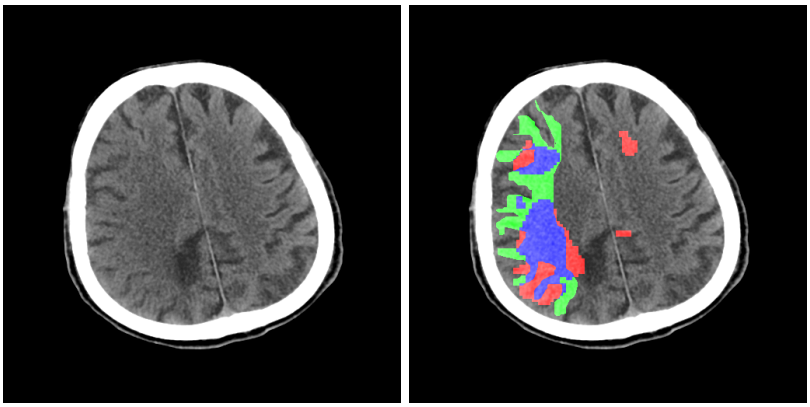

Figure 7.14 Ground Truth (Left), Prediction Mask(Right)

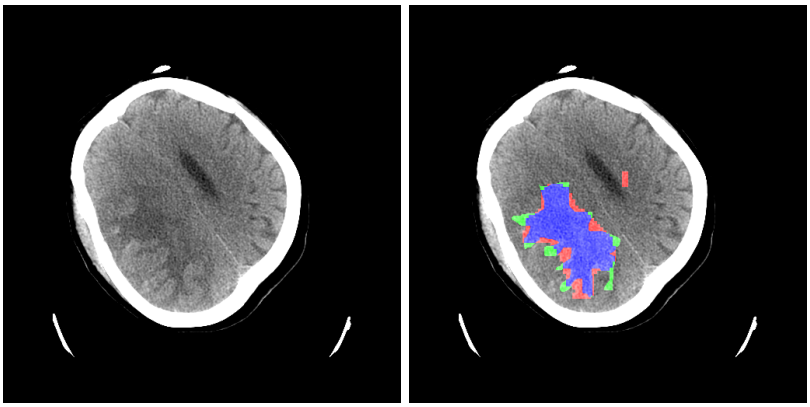

Figure 7.15 Ground Truth (Left), Prediction Mask(Right)

7.4 YOLOv3 Results

In this section, the ones encountered after testing the randomly tested images with darknet are added. The first 8 of the images given (4 are hemorrhage, 4 are ischemic) using 4000.weights 0.25 and 0.5 threshold, the next 8 with 3000.weights and the last 8 with 2000. weights give us information about the real mask regions of the related image in the dataset.

In order to better distinguish the difference in 2k, 3k and 4k weigths, it was preferred to include the tests performed on the same images in the report. For example, in the 4th of the images below, only 1 of the 2k areas was found, but when tested with 3k and 4k, the other missed bleeding area could also be masked. Although it is not distinguished in these images, the knowledge that Yolo correctly classifies all of these images was another of YOLO's motivating outputs.

In addition, as indicated in the captions of the images, while the classification and ground truth version in the CT image, the images in the middle show the prediction with 0.25 threshold, and the images on the far right show the prediction with 0.5 threshold.

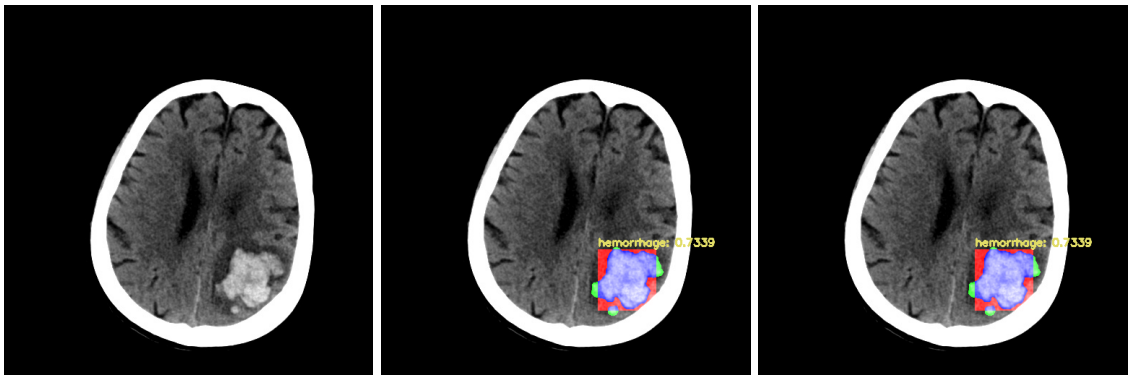

Figure 7.16 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

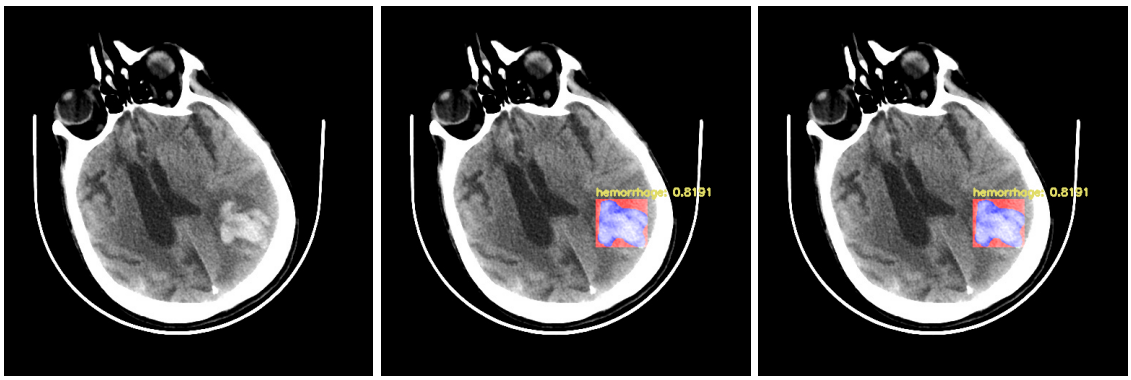

Figure 7.17 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

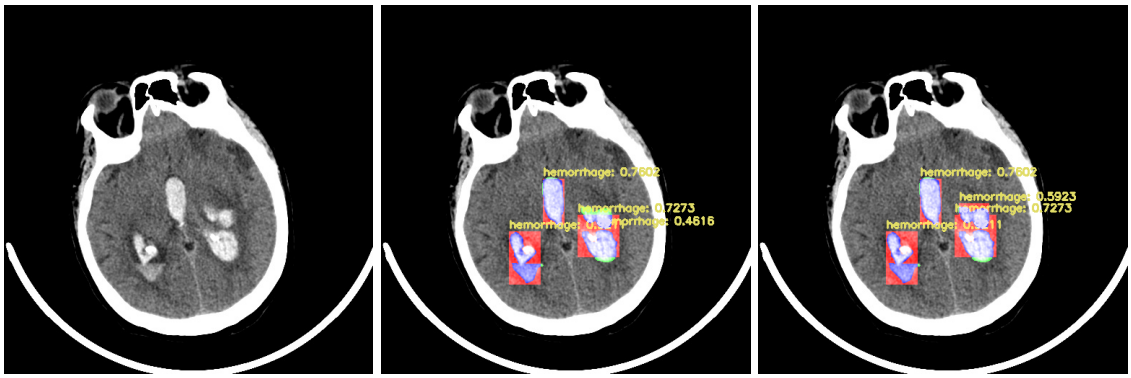

Figure 7.18 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

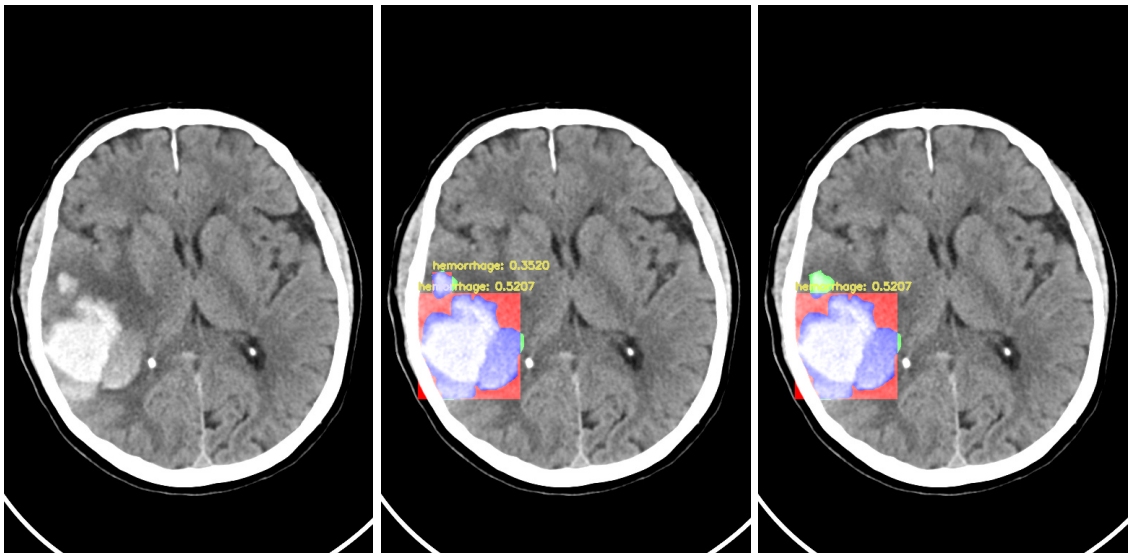

Figure 7.19 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

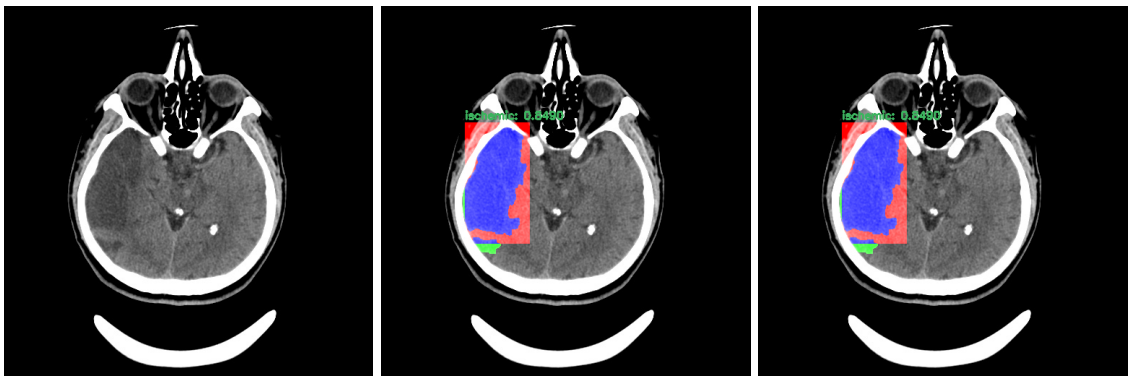

Figure 7.20 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

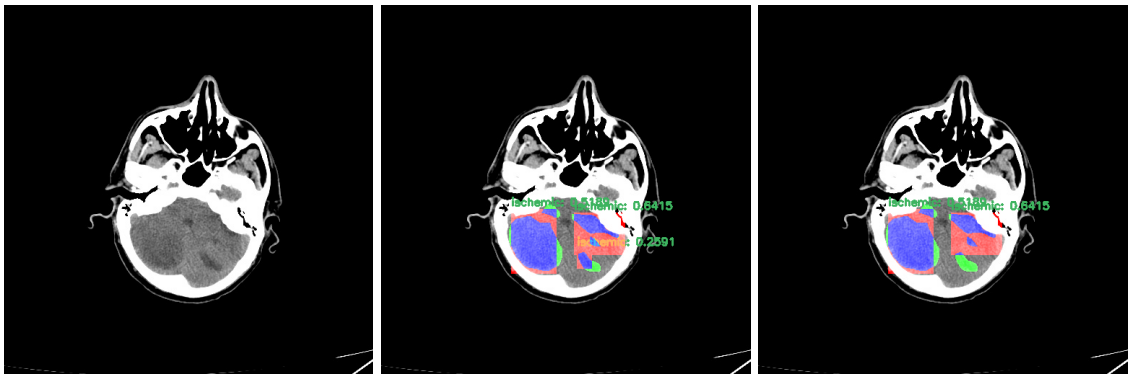

Figure 7.21 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

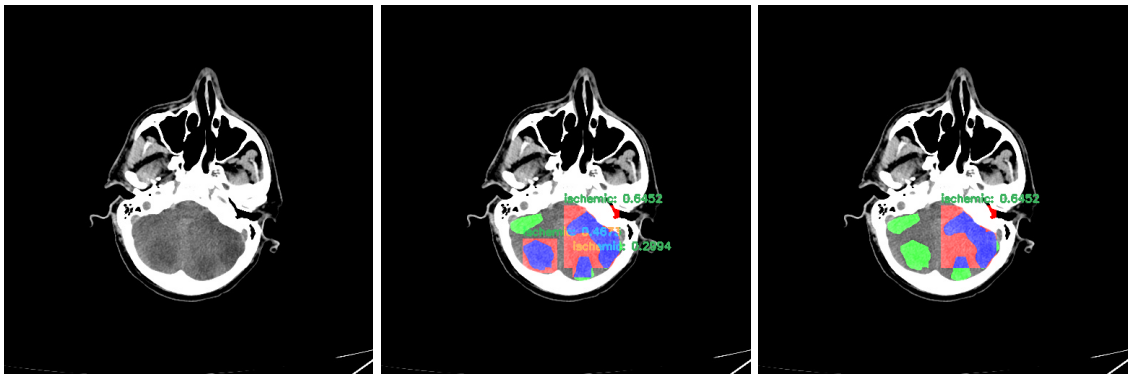

Figure 7.22 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

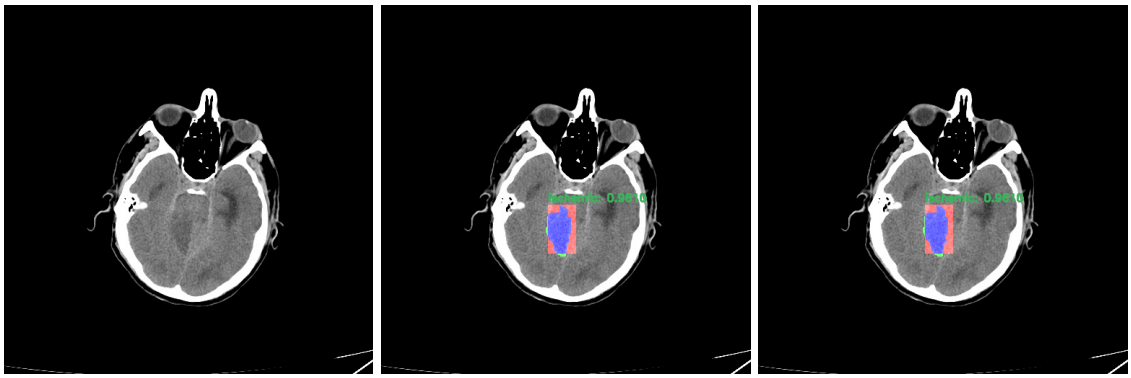

Figure 7.23 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

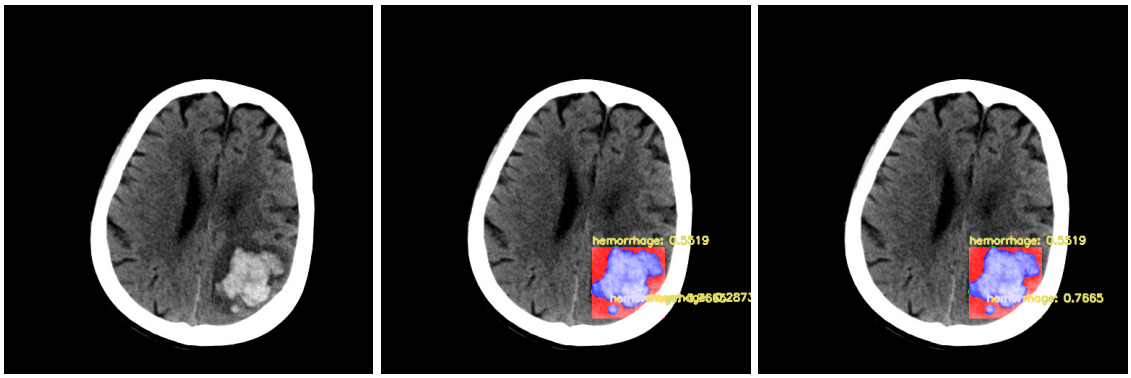

Figure 7.24 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

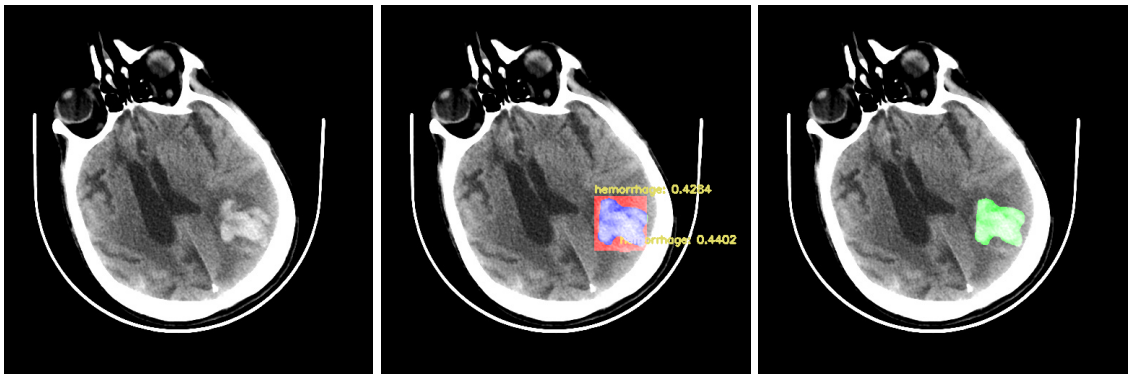

Figure 7.25 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

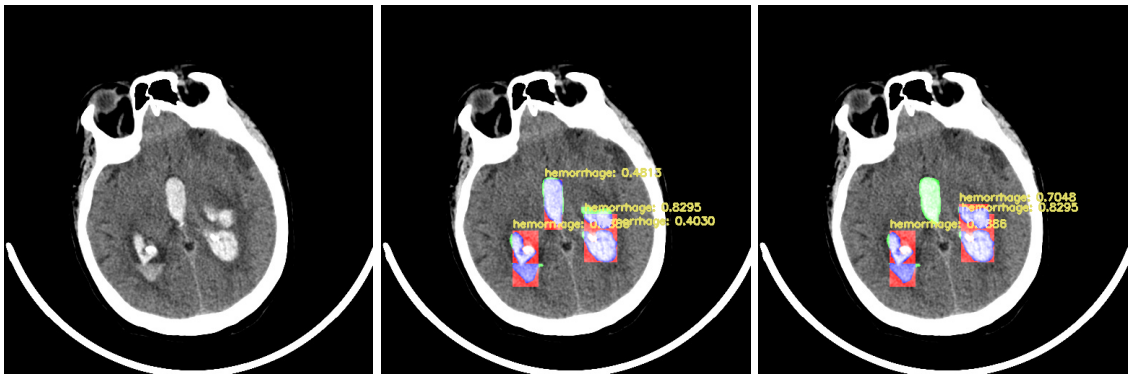

Figure 7.26 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

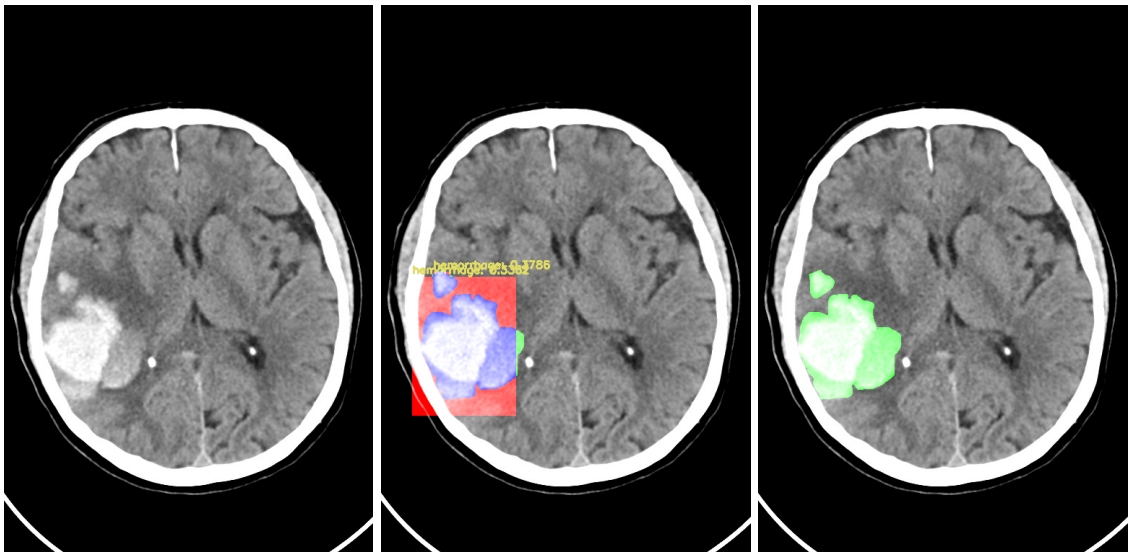

Figure 7.27 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

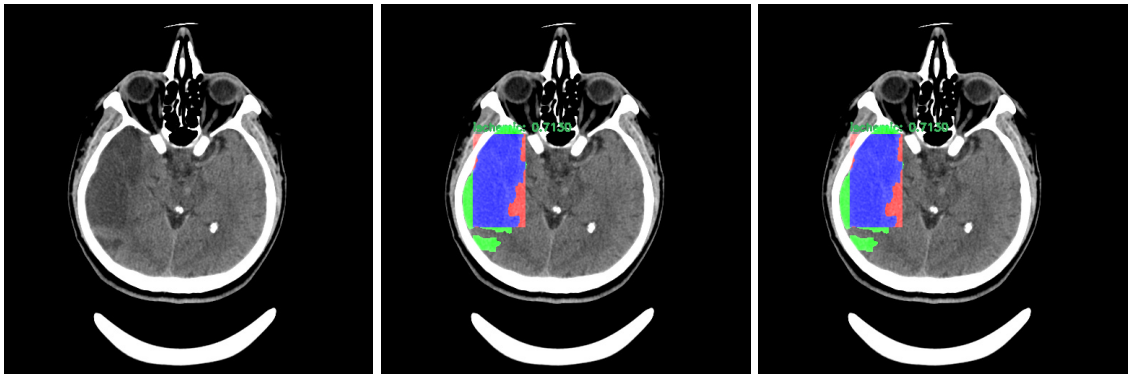

Figure 7.28 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

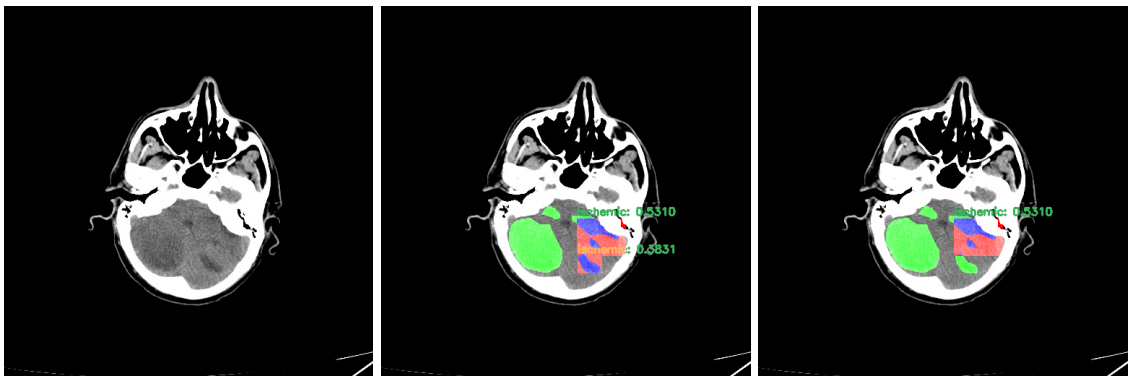

Figure 7.29 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

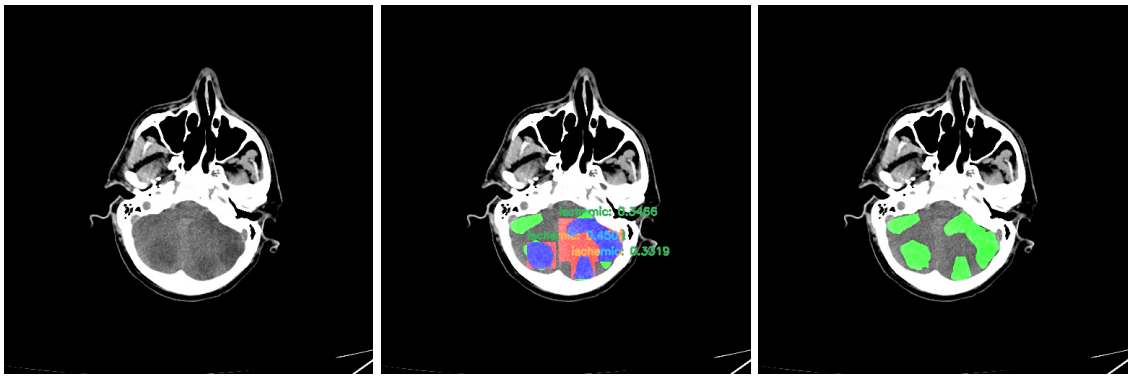

Figure 7.30 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

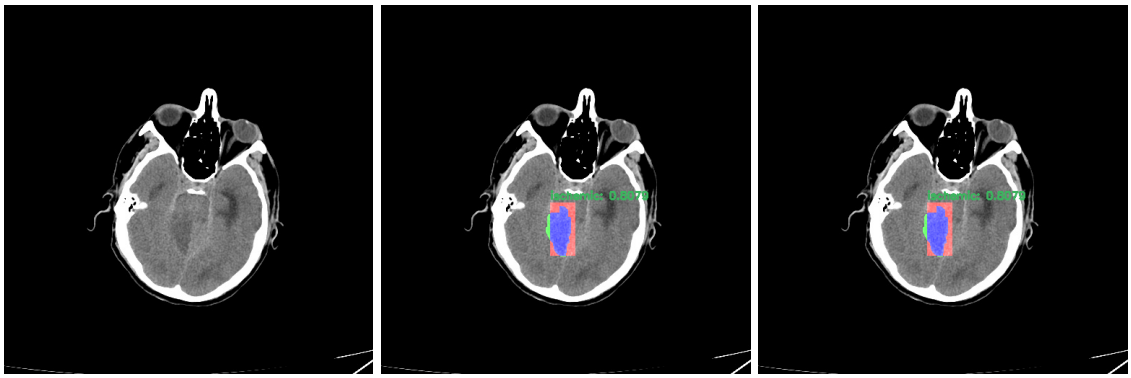

Figure 7.31 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

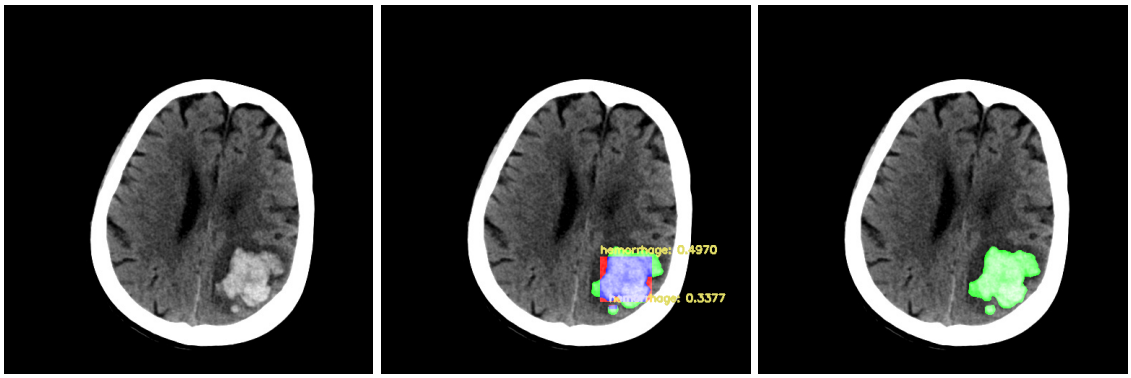

Figure 7.32 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

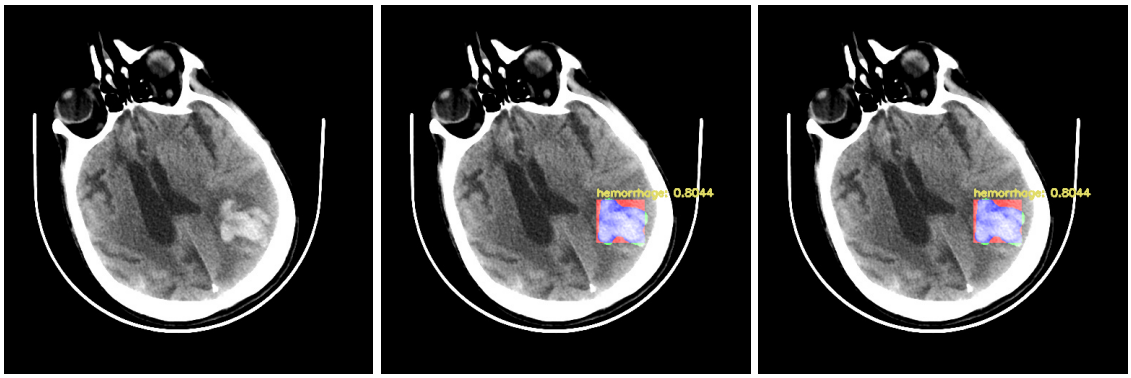

Figure 7.33 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

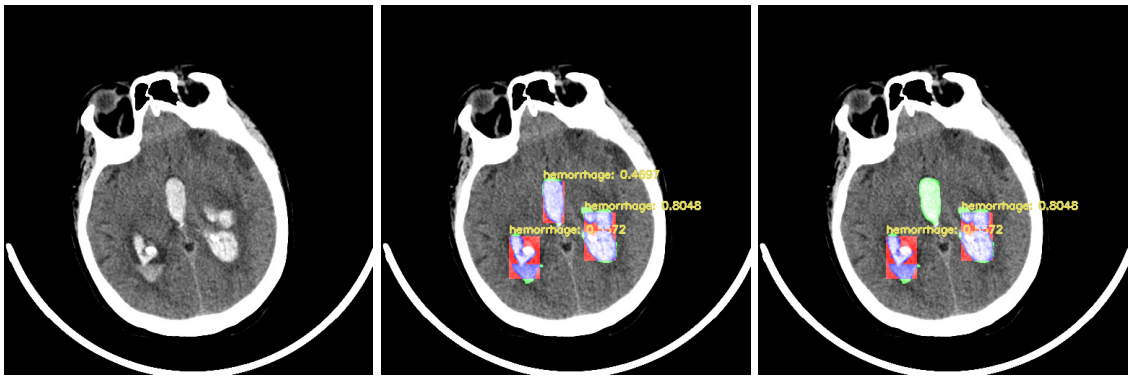

Figure 7.34 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

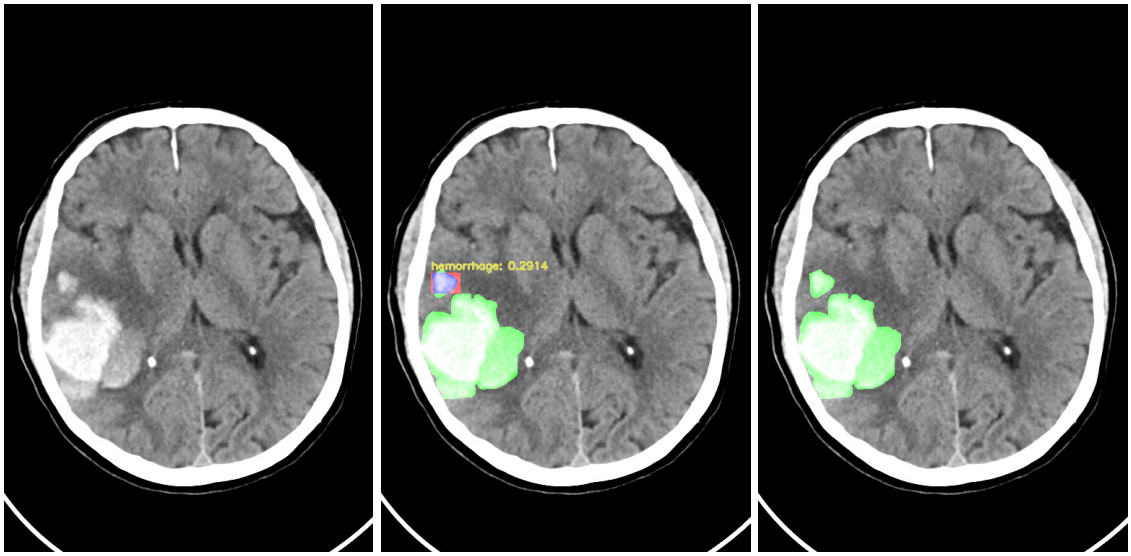

Figure 7.35 Ground Truth(Left), Hemorrhage Prediction th=0.25(Middle), Hemorrhage Prediction th=0.5(Right)

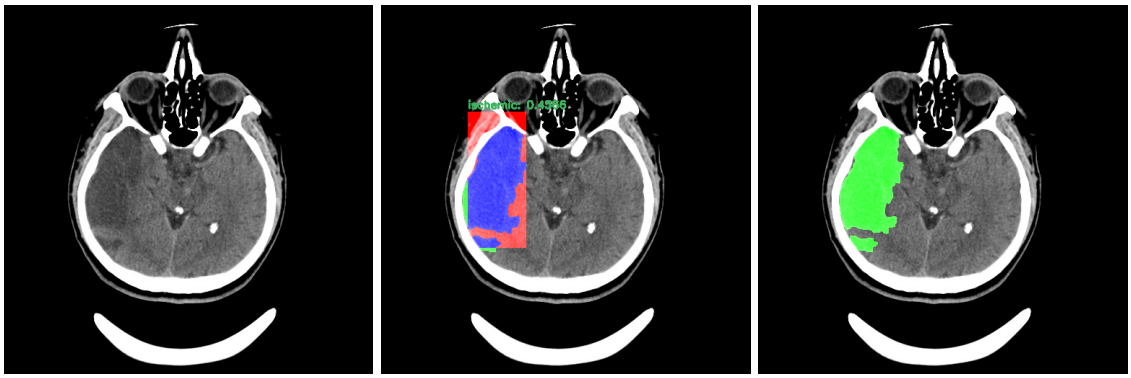

Figure 7.36 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

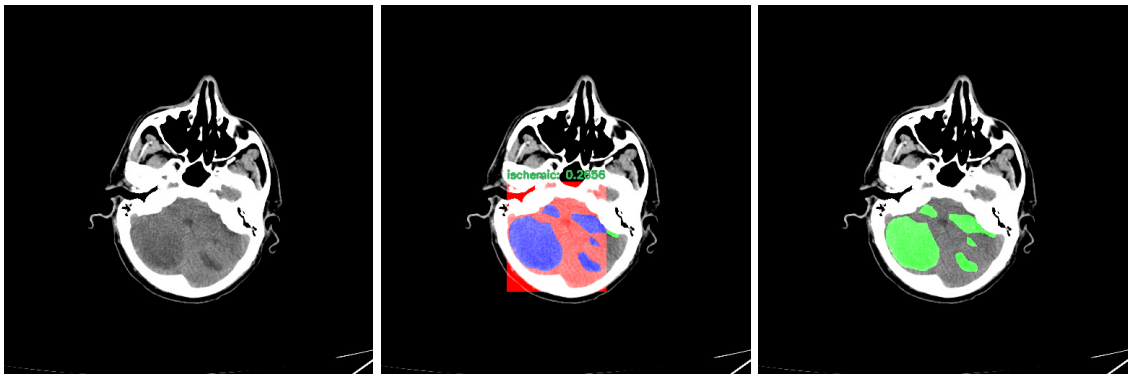

Figure 7.37 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

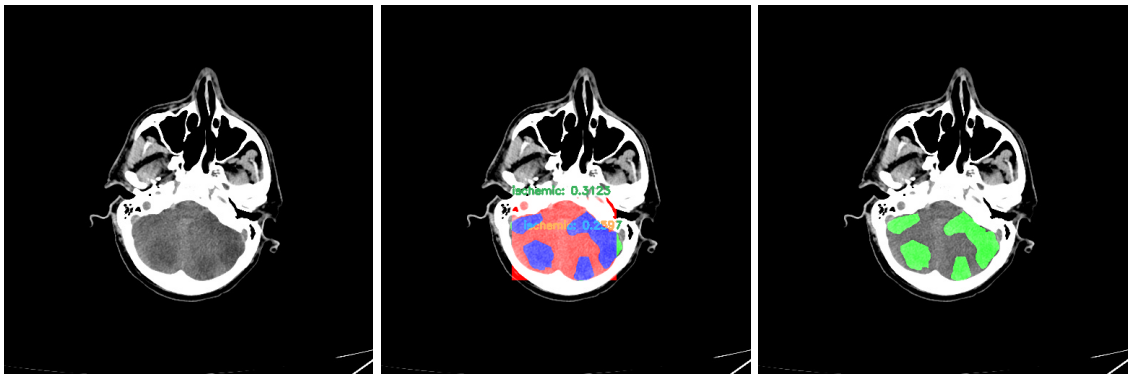

Figure 7.38 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

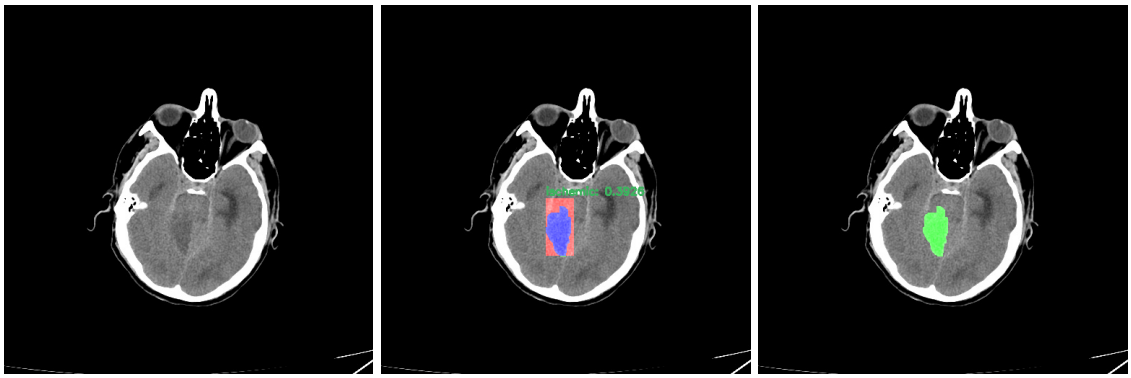

Figure 7.39 Ground Truth(Left), Ischemic Prediction th=0.25(Middle), Ischemic Prediction th=0.5(Right)

8
PERFORMANCE ANALYSIS

8.1 Comparison of Using Models

In the given table at Figure-8.1, all networks used in the project are compared in terms of size and proportionally speed, accuracy and depth parameters. We can observe that the lightest network is MobileNet, the deepest and largest network is Inception with VGG-19. In addition, we can see that the highest accuracy values mentioned in the experimental findings chapter are EfficientNet and ResNet50.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-

Figure 8.1 Table of comparing all networks used in the project

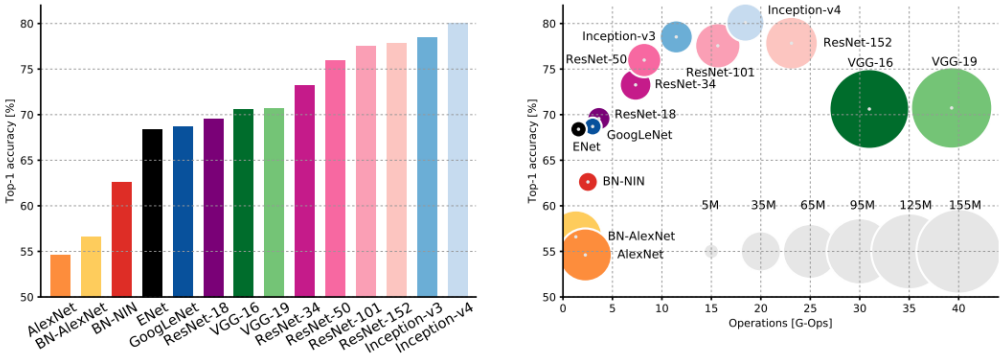

Figure 8.2 Compare of Models (Left), Processing load of models (Right)

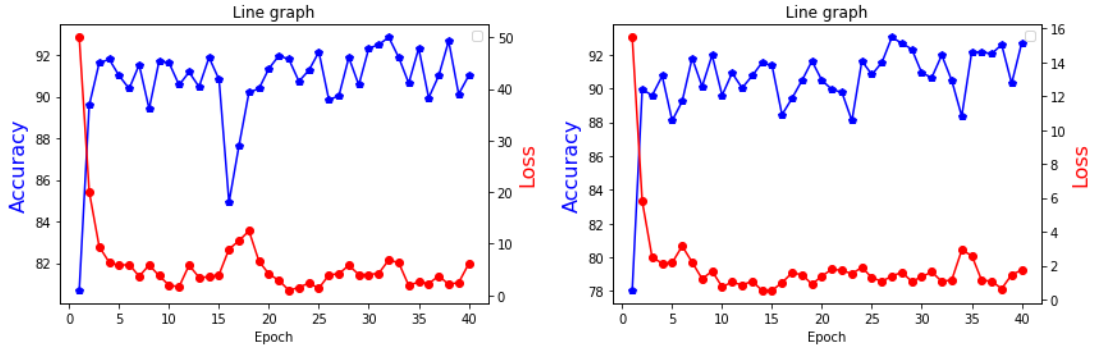

Figure 8.3 Accuracy for VGG-19 Model-1 (Left),Loss for VGG-19 Model-2 (Right)

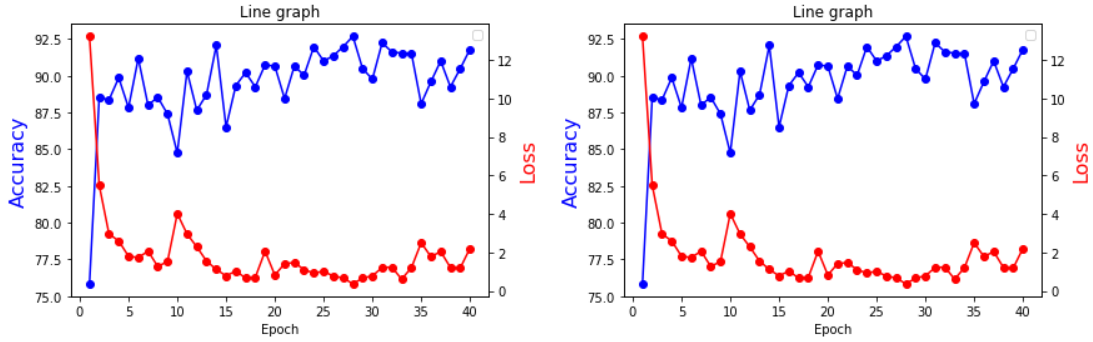

Figure 8.4 Acc/Loss for Inception Model-1 (Left),Acc/Loss for Inception Model-2 (Right)

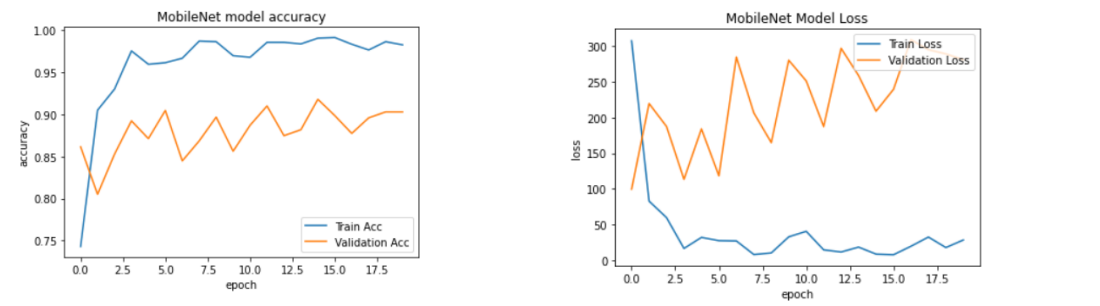

Figure 8.5 Accuracy for MobileNet Model-1 (Left),Loss for MobileNet (Right)

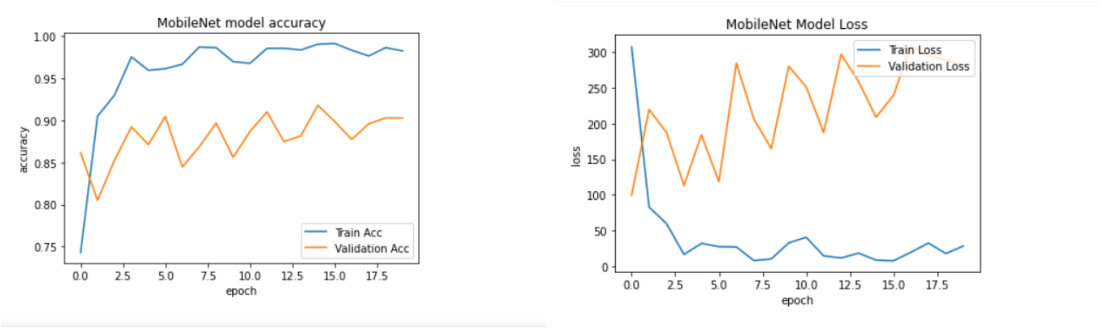

Figure 8.6 Accuracy for MobileNet Model-2 (Left),Loss for MobileNet (Right)

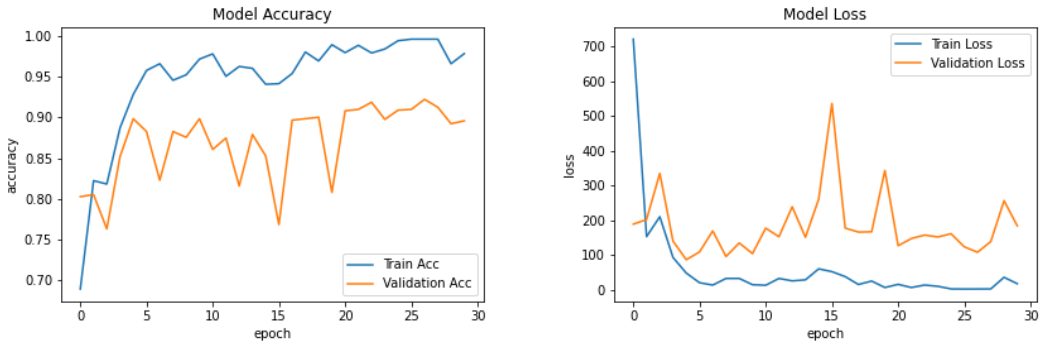

Figure 8.7 Accuracy for DenseNet Model-1 (Left),Loss for DenseNet (Right)

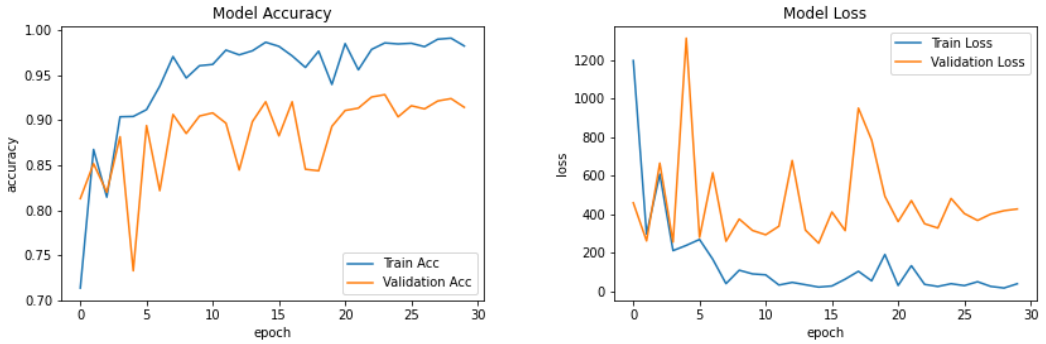

Figure 8.8 Accuracy for DenseNet Model-2 (Left),Loss for DenseNet (Right)

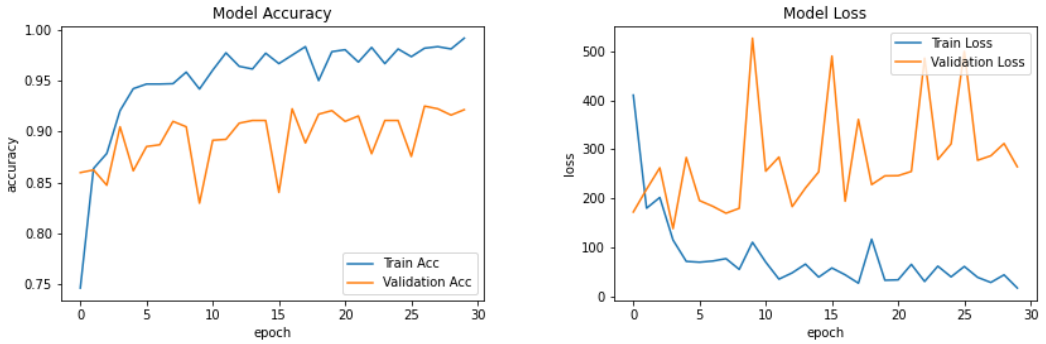

Figure 8.9 Accuracy for EfficientNet Model-1 (Left),Loss for EfficientNet (Right)

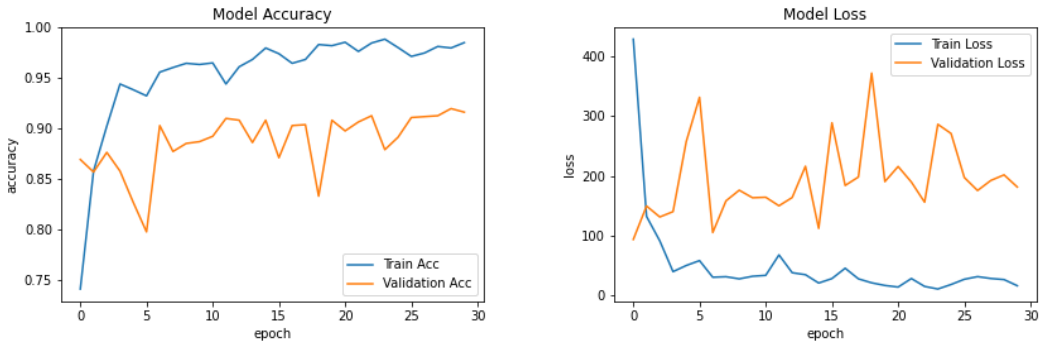

Figure 8.10 Accuracy for EfficientNet Model-2 (Left),Loss for EfficientNet (Right)

8.2 Mask R-CNN

8.2.1 IoU Scores

From selected models, a general IoU rating test has been applied. This test was held with 510 mixed hemorrhagic and ischemic stroke CT images outside of the training data. Results of this test are shown at Table-8.1.

Parameters (Scenerio #)	Model (Epoch #)	Number of Test Samples	Average IoU Score	Max IoU Score
2	23	510	0.3042	0.9297
2	47	510	0.3601	0.9146
3	53	510	0.4317	0.9349
3	75	510	0.4681	0.9349
4	30	510	0.3812	0.9200
4	50	510	0.3875	0.9221

Table 8.1 Average IoU Scores For Test Data From Different Mask R-CNN Models

8.2.2 Analysis

The scenarios during the training process, the meanings and values of the Loss values obtained in these scenarios, code snippets from the program prepared for manual testing of some selected models, manual test examples made with some selected models and the IoU scores calculated as a result of these examples were shared in detail in the previous sections. In addition, results of a test application for selected models held with 510 test samples are shown in this chapter, at Table-8.1.

It is observed that loss values change erratically during the training process, but it should be noted that the loss charts presented within the scope of this report reflect the val_loss values that are a combination of all Validation Loss types. This value undoubtedly gives us clues about the success of the model, so it has been evaluated in this direction and ways to minimize it have been sought. However, since this value shows the success of the model in many areas collectively, it will be much more

efficient to examine the archived log records in order to understand the success rate in a selected area. These log records will be shared within the project file.

Even though a rapid change trend has been observed for each training scenario, results in Table-8.1 clearly illustrate that it almost always resulted with better results when the epoch count was kept as high as possible. This indicates that for future improvements, the results shown in this report should be taken into the next level with larger epoch counts and see how many more epochs would it take to adapt the Mask R-CNN model to our problem. This, undoubtedly, will be costly in terms of processing power.

In order to better explain the situation, in the shared test samples a test experiment in which stroke detection could not be performed was also shared. In this example and some of other examples that are not shared here, there are cases where models fail to detect stroke, particularly in challenging cases. In addition to this, some models may perceive some color changes that do not indicate a stroke in tomography images as a stroke. By keeping the training going as long as the processing power allows, and by keeping the number of models at hand as high as possible, it is enabled to search for situations where all these negative situations occur the least. At the end of the various test experiments, it was seen that successful models were obtained with an epoch count between 30 and 50, depending mainly on the mentioned parameters. These models have shown improvement in stroke detection and classification, and their IoU scores are satisfactory for given parameters.

Another positive point that should be mentioned is that the models produced in the training are quite successful in classifying the stroke in the given testing tomography images.

8.3 U-Net

Iterative IoU which mentioned in Figure-8.11, while expressing the score we obtained by proportioning the intersection of each image to the combination of the prediction and the real mask image; Cumulative IoU which mentioned in Figure-8.11, it also shows the ratio of the total intersection value obtained from all images to the combination of prediction and mask images obtained from all images. Although both metrics are close to each other, the general iterative score is calculated as almost %4 higher than the cumulative. The overall IoU results obtained after testing all test images with UNet and obtaining the masks are given in Table-8.2

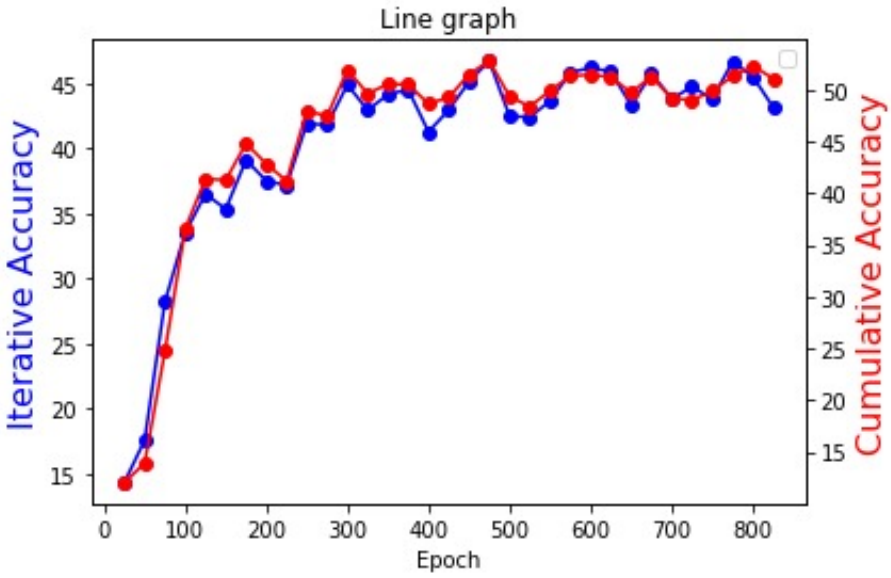

Figure 8.11 Comparison of IoU in U-Net

IoU Type	Accuracy
Iterative IoU	%52.798128
Cumulative IoU	%46.773961

Table 8.2 Result of General U-Net IoU

8.4 YOLOv3

Table-8.3,the table of Iterative and Cumulative IoU scores obtained as a result of calculations made with 6 different parameters is given. The W parameter specified in the table shows the weights information, and the Th parameter represents the threshold ratio of the labels to be given in the output. All test images have been tested by giving 0.5, 0.25 and 0.15 thresholds separately in 2K, 3K and 4K weights.

IoU Type/Parameter	Iterative IoU	Cumulative IoU
W=4k Th=0.5	%27.085335	%21.673630
W=3k Th=0.5	%20.570341	%13.983812
W=2k Th=0.5	%13.970838	%9.315312
W=4k Th=0.25	%31.245511	%25.403607
W=3k Th=0.25	%23.726957	%21.294386
W=2k Th=0.25	%25.732590	%18.26678

Table 8.3 Iterative and Cumulative IoU for YOLOv3

9
RESULT

9.1 Summary

Within the scope of this study, models that can perform detection, classification and localization with high accuracy rates for hemorrhagic and ischemic strokes in brain have been presented. The testing of these models, the results obtained in these testing processes, the selection of the models and the technical approaches applied in all these processes are presented in detail. The results of each experimental study are embodied by presenting them in image, table and graphic formats. In the final state of the project, it has reached the targets set at the very beginning.

Under the following headings, our opinions on the methods that can be applied if we or another team aim to carry this kind of work further will be explained with reasons.

9.2 Areas for Improvement and Recommendations

9.2.1 K-Fold Splitting

The parts of the project that we see as missing from our point of view are that the train and validation data are not separated by the K-fold philosophy. The K-fold philosophy is important in terms of seeing the accuracy value more reliable. The shortcoming here is the GPU problem experienced during the training of the models. Due to the GPU constraint in Colab, less time could be allocated for training the models than expected. And since no distinction was made with the K-fold during the separation of the initial training test data, it was not possible to go back and make an update until the project delivery.

9.2.2 Using NasnNet Instead of MobileNet

MobileNet, one of the preferred networks for classification in the first phase of the project, showed the lowest success compared to the other 5 networks. This may affect

the success in the fusion part. In order to increase the success of Fusion, a network that can produce more successful results can be used. This may be the NasnNet network, which the Keras library qualifies as the most successful in the accuracy table on its site.

9.2.3 Fusion with UNet and YoloV3

The YOLO algorithm is more successful in detecting the positions of objects than other algorithms, but instead of detecting the location of the objects from the exact border, it takes the object into a rectangular box that covers it. U-Net, on the other hand, can detect objects from the exact border as we want, but this algorithm can also show the object in unrelated places. Therefore, if we restrict the object we are looking for to a certain area with YOLO and search in this restricted area with U-Net, we can achieve a much higher success.

9.2.4 Data Augmentation

In order to ensure that the data is balanced in the training processes of the project, the unbalanced part of the train data was eliminated. In order to avoid this situation and further increase the success rate achieved, the application of the data augmentation approach may be applied. The sensitivity of the available data and the general quality standard in CT images should be considered when applying this approach. Where possible, the validity of the derived data resulting from this process should be confirmed with the help of an expert.

9.2.5 Detection on images including 2 classes

All of the currently trained models are trained to detect only 1 class label on 1 image. However, in the ongoing process of the project, this deficiency will be tried to be eliminated and efforts will be made to bring it to a level that can make multiple detections on images containing ischemic/hemorrhage at the same time.

9.2.6 Deeper Experimental Observations with A Greater Computational Power

As we, the project team, were quite motivated to obtain the best results possible for our problem with the tools in hand, we had limitations during this process. The biggest limitation, undoubtedly, was at the computational power in hand to be used in the training phases. In the future improvements, the results demonstrated in this report should be considered and the methods adopted in scope of this project should be taken into a deeper level with a better hardware solutions in order to get the best possible versions of the selected tools.

References

[1] M. Chawla, S. Sharma, J. Sivaswamy, and L. Kishore, "A method for automatic detection and classification of stroke from brain ct images," in *2009 Annual international conference of the IEEE engineering in medicine and biology society*, IEEE, 2009, pp. 3581–3584.

[2] C. M. Dourado Jr, S. P. P. da Silva, R. V. M. da Nobrega, A. C. d. S. Barros, P. P. Reboucas Filho, and V. H. C. de Albuquerque, "Deep learning ıot system for online stroke detection in skull computed tomography images," *Computer Networks*, vol. 152, pp. 25–39, 2019.

[3] A. Arab, B. Chinda, G. Medvedev, W. Siu, H. Guo, T. Gu, S. Moreno, G. Hamarneh, M. Ester, and X. Song, "A fast and fully-automated deep-learning approach for accurate hemorrhage segmentation and volume quantification in non-contrast whole-head ct," *Scientific Reports*, vol. 10, no. 1, pp. 1–12, 2020.

[4] A. M. Suberi, W. W. Zakaria, R. Tomari, A. Nazari, M. N. H. Mohd, and N. F. N. Fuad, "Deep transfer learning application for automated ıschemic classification in posterior fossa ct ımages," *vol*, vol. 10, pp. 459–465, 2019.

[5] X. W. Gao, R. Hui, and Z. Tian, "Classification of ct brain images based on deep learning networks," *Computer methods and programs in biomedicine*, vol. 138, pp. 49–56, 2017.

[6] J. George, S. Skaria, V. Varun, *et al.*, "Using yolo based deep learning network for real time detection and localization of lung nodules from low dose ct scans," in *Medical Imaging 2018: Computer-Aided Diagnosis*, International Society for Optics and Photonics, vol. 10575, 2018, p. 105751I.

[7] M. Liu, J. Dong, X. Dong, H. Yu, and L. Qi, "Segmentation of lung nodule in ct images based on mask r-cnn," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*, IEEE, 2018, pp. 1–6.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[9] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Ahmet Onur AKMAN
Birthdate and Place of Birth: 12.11.1997, Kayseri
E-mail: l1116059@std.yildiz.edu.tr
Phone: 0507 280 57 53
Practical Training:
Bilişim Destek Hizmetleri A.Ş.
PROTEL A.Ş.
Mey Diageo

SECOND MEMBER

Name-Surname: Ahmet AYDIN
Birthdate and Place of Birth: 26.09.1998, Bursa
E-mail: l1116003@std.yildiz.edu.tr
Phone: 0536 269 35 68
Practical Training: Visiomex

THIRD MEMBER

Name-Surname: Emine Betül ŞİNAR
Birthdate and Place of Birth: 11.12.1995, İstanbul
E-mail: l1114063@std.yildiz.edu.tr
Phone: 0554 135 84 95
Practical Training: Phexum Artificial Intelligence Software Company

Project System Informations

System and Software: Microsoft Windows Operating System, Python
Required RAM: 2 GB
Required Disk: 256 MB