ThreshNet: a novel machine learning technique to optimize sensitivity and specificity performance

Table of Contents

Contents

[Abstract 5](#_Toc65696012)

[Acknowledgements 6](#_Toc65696013)

[Introduction 7](#_Toc65696014)

[Project Goals 8](#_Toc65696015)

[Review of Literature 9](#_Toc65696016)

[1. Image classification 9](#_Toc65696017)

[2. Sensitivity and specificity 10](#_Toc65696018)

[3. Brain tumor 10](#_Toc65696019)

[4. Ensemble network 10](#_Toc65696020)

[5. Image segmentation 11](#_Toc65696021)

[6. Summary 12](#_Toc65696022)

[Statement of Problem 13](#_Toc65696023)

[Hypothesis 13](#_Toc65696024)

[Materials 13](#_Toc65696025)

[Procedure 14](#_Toc65696026)

[1. Steps 14](#_Toc65696027)

[2. Data preprocessing 15](#_Toc65696028)

[3. Individual networks for image classification 15](#_Toc65696029)

[1) Different baseline neural network structures 15](#_Toc65696030)

[2) Structure of added dense layers 15](#_Toc65696031)

[3) Pooling 15](#_Toc65696032)

[4) Activation functions 16](#_Toc65696033)

[5) Weights 16](#_Toc65696034)

[6) Loss/metrics function 16](#_Toc65696035)

[7) Optimizers 16](#_Toc65696036)

[8) EarlyStopping 16](#_Toc65696037)

[9) Drop out parameter 16](#_Toc65696038)

[10) Different random seeds 16](#_Toc65696039)

[4. ThreshNet systems 16](#_Toc65696040)

[1) Select individual networks for the ensembles 16](#_Toc65696041)

[2) Build ThreshNet systems 16](#_Toc65696042)

[3) Choice of ensemble members 16](#_Toc65696043)

[4) Generate results and plots 17](#_Toc65696044)

[5. image segmentation 18](#_Toc65696045)

[Results 20](#_Toc65696046)

[1. Networks ensemble 20](#_Toc65696047)

[6. ThreshNet system vs individual networks 21](#_Toc65696048)

[7. Image segmentation 23](#_Toc65696049)

[Analysis 24](#_Toc65696050)

[1. Individual Networks 24](#_Toc65696051)

[2. ROC 24](#_Toc65696052)

[Conclusion 25](#_Toc65696053)

[Recommendations 26](#_Toc65696054)

[Bibliography 27](#_Toc65696055)

[Appendices/Raw Data 29](#_Toc65696056)

[1. Data visualization (labelled Dataset) 29](#_Toc65696057)

[2. Individual Network Training and testing results 30](#_Toc65696058)

[1) Resnet101 (Only dense layers trainable) 30](#_Toc65696059)

[2) Resnet50 (Only dense layers trainable) 37](#_Toc65696060)

[3) Resnet50\_2 (Only dense layers trainable, pool\_size =(4,4)) 38](#_Toc65696061)

[4) Resnet50\_3 (Only dense layers trainable) 39](#_Toc65696062)

[5) Resnet101\_2 (Only dense layers trainable) 40](#_Toc65696063)

[6) Resnet152 (Only dense layers trainable) 41](#_Toc65696064)

[7) Resnet152\_2 (Only dense layers trainable) 42](#_Toc65696065)

[8) MobileNet (Only dense layers trainable) 43](#_Toc65696066)

[9) MobileNet-2 (Only dense layers trainable) 44](#_Toc65696067)

[10) MobileNet-3 (Only dense layers trainable) 45](#_Toc65696068)

[11) Resnet101 (All layers trainable) 46](#_Toc65696069)

[12) Resnet50 (All layers trainable) 49](#_Toc65696070)

[13) Resnet50\_2 (All layers trainable) 50](#_Toc65696071)

[14) Resnet50\_3 (All layers trainable) 51](#_Toc65696072)

[15) Resnet101\_2 (All layers trainable) 52](#_Toc65696073)

[16) Resnet152 (All layers trainable) 53](#_Toc65696074)

[17) Resnet152\_2 (All layers trainable) 54](#_Toc65696075)

[18) MobileNet (All layers trainable) 55](#_Toc65696076)

[19) MobileNet-2 (All layers trainable) 56](#_Toc65696077)

[20) MobileNet-3 (All layers trainable) 57](#_Toc65696078)

[21) Resnet101 (metrics: falsenegatives) 57](#_Toc65696079)

[22) Resnet50 (metrics: falsenegatives) 59](#_Toc65696080)

[23) Resnet50\_2 (metrics: falsenegatives) 61](#_Toc65696081)

[24) Resnet50\_3 (metrics: falsenegatives) 62](#_Toc65696082)

[25) Resnet101\_2 (metrics: falsenegatives) 63](#_Toc65696083)

[26) Resnet152 (metrics: falsenegatives) 64](#_Toc65696084)

[27) Resnet152\_2 (metrics: falsenegatives) 65](#_Toc65696085)

[28) MobileNet (metrics: falsenegatives) 66](#_Toc65696086)

[29) MobileNet-2 (metrics: falsenegatives) 67](#_Toc65696087)

[30) MobileNet-3 (metrics: falsenegatives) 68](#_Toc65696088)

[31) Resnet101\_2 (new seed) 68](#_Toc65696089)

[32) Resnet50 (poolsize: 2,2) 69](#_Toc65696090)

[33) Resnet50\_2 (different dropout) 71](#_Toc65696091)

[34) MobileNet (different dropout) 72](#_Toc65696092)

[35) MobileNet\_2 (pool\_size=(2,2)) 73](#_Toc65696093)

[36) MobileNetV2 74](#_Toc65696094)

[37) ResNet152 (optimizer='adagrad') 75](#_Toc65696095)

[38) ResNet101\_2 (dropout = 0.1) 76](#_Toc65696096)

[3. Image segmentation 76](#_Toc65696097)

# Abstract

A neural network with both high sensitivity and high specificity is not always achievable, and sensitivity and specificity are often inversely related. This project proposes “ThreshNet”, a novel method which utilizes ensembles of neural networks. It demonstrates better performance in sensitivity, specificity or both performance metrics.

An MRI dataset for brain tumor diagnosis was used as an example application. Transfer learning of many well-known neural networks were applied. Numerous networks were optimized with different dense layer architectures and network parameters. The networks with the highest accuracy were used as candidate networks for the ThreshNet ensembles. Each ThreshNet system contains multiple networks and each network outputs its diagnosis decision (0: no tumor; 1: tumor present). The sum of all network decisions are compared against the threshold parameter to yield ThreshNet’s decision. A ResUNet based segmentation model was also implemented to locate the brain tumor when the ThreshNet system predicts a present tumor.

Lower ThreshNet values yield the best sensitivity, outperforming best sensitivity individual networks with lowered specificity performance. Medium ThreshNet values achieve a balance between sensitivity and specificity. Higher ThreshNet values match the specificity performances of the best specificity individual networks while significantly improving sensitivity. Variance among ThreshNet systems is smaller than variance among individual networks, yielding more consistent performance.

In conclusion, ThreshNet systems are capable of outperforming individual ensemble member networks. In addition, it provides convenient means to achieving specific performance and sensitivity specificity trade-off goals through adjusting the threshold parameter.

# Acknowledgements

I would like to thank my teachers and mentors for their wonderful help and support. Thank you Dr. Jaiclin for your guidance throughout the project, and for giving me suggestions on organizing as well as presenting my results. Thank you Professor Guan for reviewing my project, providing me with critiques, and helping me improve. Thank you, Dr McCalla and Mr. David for inspiring me to launch my research journey last summer. I would also like to thank my parents who encouraged me to continue the project despite the many hardships and obstacles. Thank you Mom and Dad, for supporting me the entire way!

# Introduction

Binary classification is the task of dividing elements of a set into multiple groups on the basis of a classification rule. In the application of medical diagnoses based on images such as MRI Scans, we can determine if a patient has a certain disease or not.

Sensitivity and specificity are statistical measures of a binary classification test’s performance. They are widely used in medical diagnosis. Sensitivity is the ability of a test to correctly identify those with the disease (true positive rate), whereas specificity is the ability of the test to correctly identify those without the disease (true negative rate). While it is optimal to achieve both high sensitivity and high specificity, sensitivity and specificity in a test are often inversely related. Thus, selecting the optimal balance between sensitivity and specificity depends on the purpose for which the test is used or the severity of the potential disease. For example, in brain tumor detection applications, a false negative (missing the presence of a brain tumor) could mean delayed treatment and other detrimental consequences. So, a highly sensitive diagnosis is desired.

Machine learning techniques are widely used for medical imaging diagnoses. Many publications investigate the development of a high performing neural network, although high performance is not always achievable with a single network and the balance between sensitivity and specificitiy may not be easily controllable / tunable.

This project proposes a “ThreshNet” system, which utilizes ensembles of neural networks to achieve specific goals of balancing sensitivity and specificity, as well as improving performance of both performance metrics. To help medical professionals visualize the data from our “ThreshNet” system, ResUNet (a popular segmentation model) is implemented.

# Project Goals

* Implement a system that can achieve high performance and specific trade-off goals between sensitivity and specificity
* Define a “threshold” (number of networks needed to “agree” with a positive diagnosis for the system to declare the case as positive). As this threshold increases, the sensitivity of the “system” will decrease, while specificity will increase until a certain threshold value
* Implement an image segmentation model to accurately and precisely determine the brain tumor’s location based off the neural network system’s outputted result

# Review of Literature

## Image classification

Image classification is the process of an input image being identified into different categories. In the machine learning field, deep neural networks are the most frequently used technique when it comes to image classification. Of the different deep neural networks, one of the most commonly used is the CNN (Convolutional Neural Network) architecture, which utilizes convolutions to surpass high image classification standards. These standards include exceeding performance of humans manually classifying the same images, and perfect image classification accuracy in cytopathology. Other classification techniques have also been researched, such as support vector machines, fuzzy logic, and genetic algorithm. However, it has been shown that a deep neural network model can classify images into multiple categories with more accuracy and in a shorter time. In addition, the paper Ye Tao, Ming Zhang, Mark Parsons “Deep Learning in Photovoltaic Penetration Classification” has found that deep learning CNNs outperform a fully connected model in terms of image classification accuracy.

Some widely used neural networks for image classification include ResNets, VGGs, and MobileNets.

Binary classification is the task of sorting a set of elements into multiple groups on the basis of a classification rule. In the application of medical diagnoses, we determine if a patient has a certain disease or not based on images such as MRI Scans. Binary image classification is used in this project.

## Sensitivity and specificity

Sensitivity and specificity are statistical measures of a binary classification test’s performance. They are widely used in medical diagnosis. Sensitivity is the ability of a test to correctly identify those with the disease (true positive rate), whereas specificity is the ability of the test to correctly identify those without the disease (true negative rate). While it is optimal to achieve both high sensitivity and high specificity, sensitivity and specificity in a test are often inversely related. Thus, selecting the optimal balance of sensitivity and specificity depends on the purpose for which the test is used or the severity of the potential disease. For example, in brain tumor detection applications, a false negative (missing the presence of a brain tumor) could mean delayed treatment and other detrimental consequences. So, a highly sensitive diagnosis is desired.

A common issue in reports (e.g. newspapers) discussing the performance of a neural network, medical diagnoses, etc. is the lack of information on statistics regarding sensitivity and specificity. For example, an accuracy of 90% detecting brain cancer may sound good on paper to an untrained eye. However, the report will likely make no mention of the potential mass of people that may have gone home with a false negative diagnose, which could lead to prolonged treatment and higher risk of death. It also does not bring up the potential group that were diagnosed with a false positive, generating large amounts of unnecessary panic, as well as unneeded invasive / expensive treatments.

## Brain tumors

Previous research have been done through machine learning algorithms in regards to areas of healthcare such as the brain tumor field. These include applications for image classification, evaluation, prediction, and segmentation. Brain tumor pattern classification techniques most commonly include Support Vector Machines and Random Forest (RF). Deep learning techniques have become popular in brain tumor diagnosis for their superior performance in segmentation, classification, detection, and analytical fields.

## Ensemble networks

Ensemble networks are sets of individually trained neural networks. The decisions of all individual neural networks are corroborated to a final decision. The combination of predictions from multiple neural network models can reduce the variance of predictions and generalization error.

Techniques for ensemble learning can be grouped by the element that is varied, such as training data, the model, and how predictions are combined. As pointed out in [5], many researchers have demonstrated that an effective combining scheme is to simply average the predictions of the member networks. In the application of this project, the averaging of predictions is equivalent to the ThreshNet threshold value set to 5. As shown in the results portion, it does not yield optimal returns in brain tumor diagnosis application, where high sensitivity is desired.

## Image segmentation

Ronneberger et al. [11] proposed the U-Net for segmenting biological microscopy images. The U-Net architecture is comprised of a contracting path and a symmetric expanding path that enables precise localization. The down-sampling (contracting path) has an FCN-like architecture that extracts features with 3 × 3 convolutions. The up-sampling (expanding path) uses up-convolution, which reduces the number of feature maps while increasing their dimensions. Feature maps from the down-sampling part of the network are copied to the up-sampling portion to avoid losing pattern information. Finally, a 1×1 convolution processes the feature maps to generate a segmentation map that categorizes each pixel of the input image.

This project implemented a similar architecture with 5 contracting (encoding) stages and 6 expanding (decoding stages). The bottleneck stage connected the contracting and expanding paths to segment the brain tumor MRI Scanning images.

## Summary

The most significant contribution of my project is the proposal of the novel ThreshNet system, which achieves better performance than individual networks. Although ensemble networks have been researched previously, my project aims to bring ingenuity with the threshold parameter.

I also built a ResUNet-based image segmentation utility, as demonstrated with the brain tumor dataset in [1]. This project provided a complete solution of detecting and locating (highlighting) the tumor binary tumor in MRI images.

# Statement of Problem

Sensitivity and specificity are important metrics to consider in medical diagnosis. One often takes priority depending on different conditions that may be diagnosed. For example, in diagnoses for more serious illnesses, a false negative could mean delayed treatment and may be fatal in some cases. Thus, for scenarios like these, sensitivity is prioritized. My project aims to create a tunable parameter such that medical professionals will be able to express the balance needed between sensitivity and specificity for a certain diagnosis. For conditions where sensitivity takes priority, ThreshNet can be tuned to a lower setting value. For conditions where specificity takes priority, ThreshNet can be tuned to a higher setting value.

# Hypothesis

It is hypothesized that:

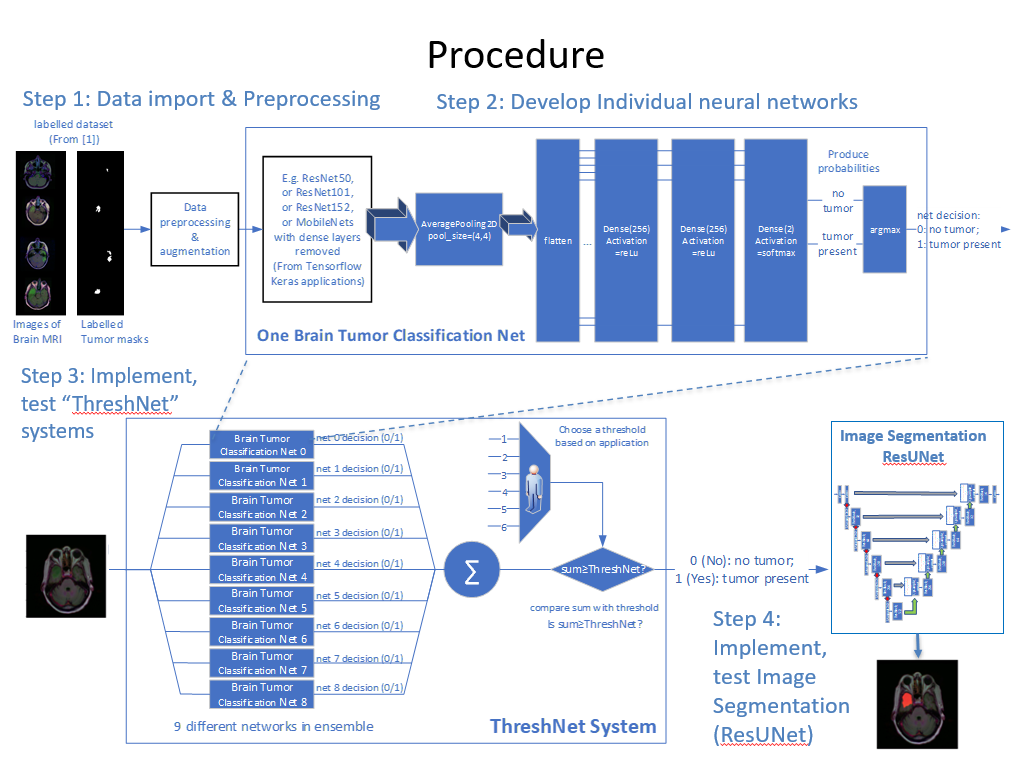
* Different neural networks will produce different results on the same image
* A lower performance network will make more mistakes, but its mistakes will be different from errors made by other well performing networks. Therefore, I hypothesize performance of the best performing network can be further improved by including lower performance networks in the ensemble as a “system” to jointly make decisions.

# Materials

* Computer
* TensorFlow 2.3.1 Framework
* Keras API
* Kaggle Notebook for Computation

# Procedure

## Illustration



## Steps

Step 1. Import Data and Data Preprocessing

* 3929 labelled images, 85% / 15% training / testing split
* Data augmentation

Step 2. Develop Individual Neural Networks

* Built and tuned variety of individual neural networks
* Use transfer learning techniques on variety of well-known networks with different dense layer architectures and different network configurations/parameters.

1. Different neural network base structures tried in the project include:
   * SVC
   * Logistic Regression.
   * ResNet50
   * ResNet101
   * ResNet152
   * MobileNet
   * MobileNetV2
   * VGG16
   * VGG19

Different networks have different parameters

1. Structure of added dense layers

* Variations of Pooling 🡪 Flatten 🡪 Dense🡪 X Dropout 🡪 Dense 🡪 Dropout 🡪 Dense layer patterning
* Variations of dimension of dense layers
* Variations of number of dense layers

1. Pooling
   * Average pooling
   * Max pooling
   * Pool size: (2,2), (4,4)
2. Activation Functions

* Relu
* Softmax

1. Weights
   * Transfer learning of imagenet weights (Only dense layers trainable with our dataset)
   * All layers trainable with our dataset
2. Loss/metrics function
   * Loss = ‘categorical\_crossentropy’,
   * Metrics
     + ‘accuracy’
     + ‘tf.keras.metrics.FalseNegatives’
3. Optimizers
   * Adam
   * Adagrad
4. I used EarlyStopping
   * Maximum of 50 epochs
5. Drop out parameters
   * 0.1
   * 0.2
   * 0.3

Also, to train these networks, data augmentation was performed to expand the dataset (rotation angle, horizontal flip, vertical flip, rescaling, brightness)

1. Variance of random seeds

* Select 33 networks (>91% accuracy, each pretty good by themselves)) as candidate networks for the next steps
* Each network generates a decision of either there is a tumor (1) or there is not a tumor (0)

Step 3. Implement “ThreshNet” systems

* 1. Randomly select 9 networks from Step 2 to form a ThreshNet system
  2. The 1 or 0 decisions of all individual networks in the system are added together and compared against “ThreshNet” parameter (I have studied system performance between 1-6 depending on application) to generate system’s decision
  3. number of networks needed to “agree” with a positive diagnosis for the system to declare the case as positive
  4. For example, if the parameter is set as 1, it would intuitively be very sensitive (as long as 1 out of 9 networks says positive, it’s positive. I will discuss more in results)

More Details on “ThreshNet”:

* Calculate sum of the decisions of each network that makes up the system
* Set ThreshNet threshold value
  + - 1 to 6
  + If “sum”> Threshnet threshold, the system generates a tumor present (labelled with 1). Otherwise, the system predicts there is no tumor and (labelled with 0).

Choice of ensemble members

To be able to have a fair comparison individual networks to ThreshNet systems, I first examined on slide 8 the performance of the individual networks that make up an ensemble, each being randomly selected out of our group of 33. Each x coordinate displays a new ensemble, 10 per graph. By the way, the lines pointing towards dotted lines are the individual networks with either the highest senstiivtiy or highest specificity (this silde and next). On slide 9, I moved a step further to examine how the individual nets performed side-by-side with the threshnet system they are a part of.

The highest sensitivity net typically has lowest specificity, and vice versa. We can see improvements on the performance of each of the 10 ThreshNet Systems compared to the individual nets.

Figure 10 small variance, consistent results

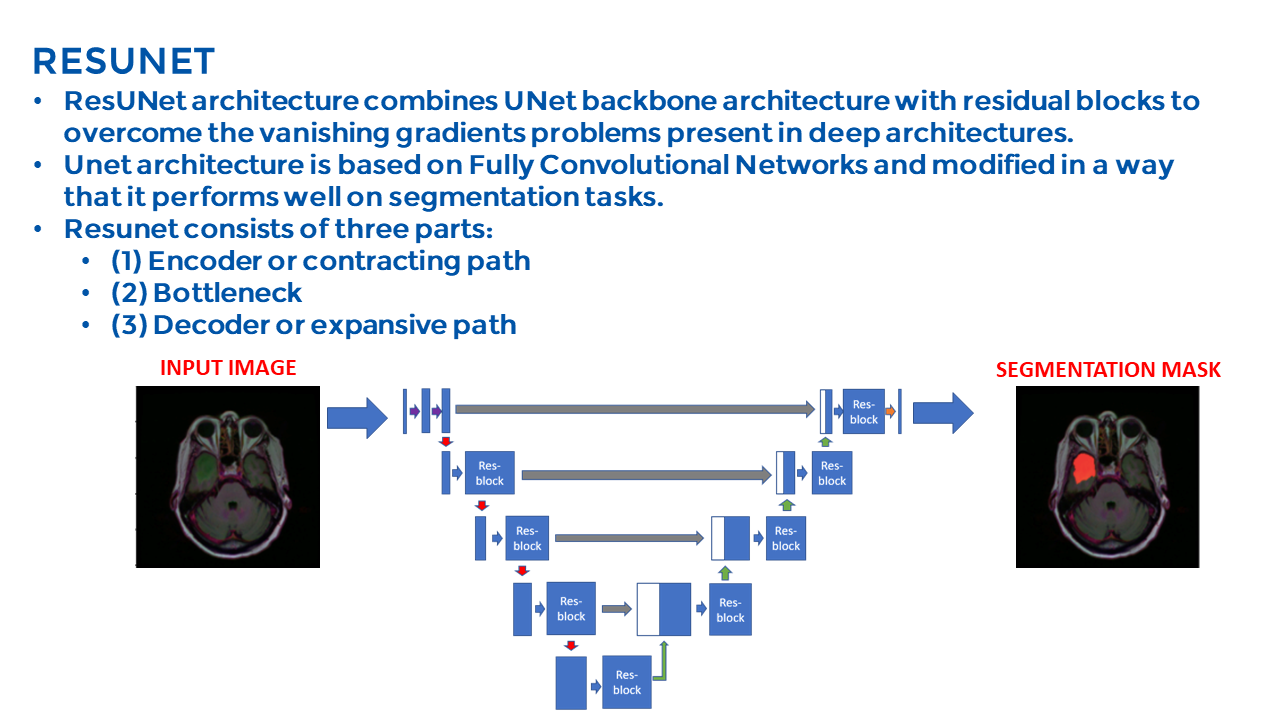
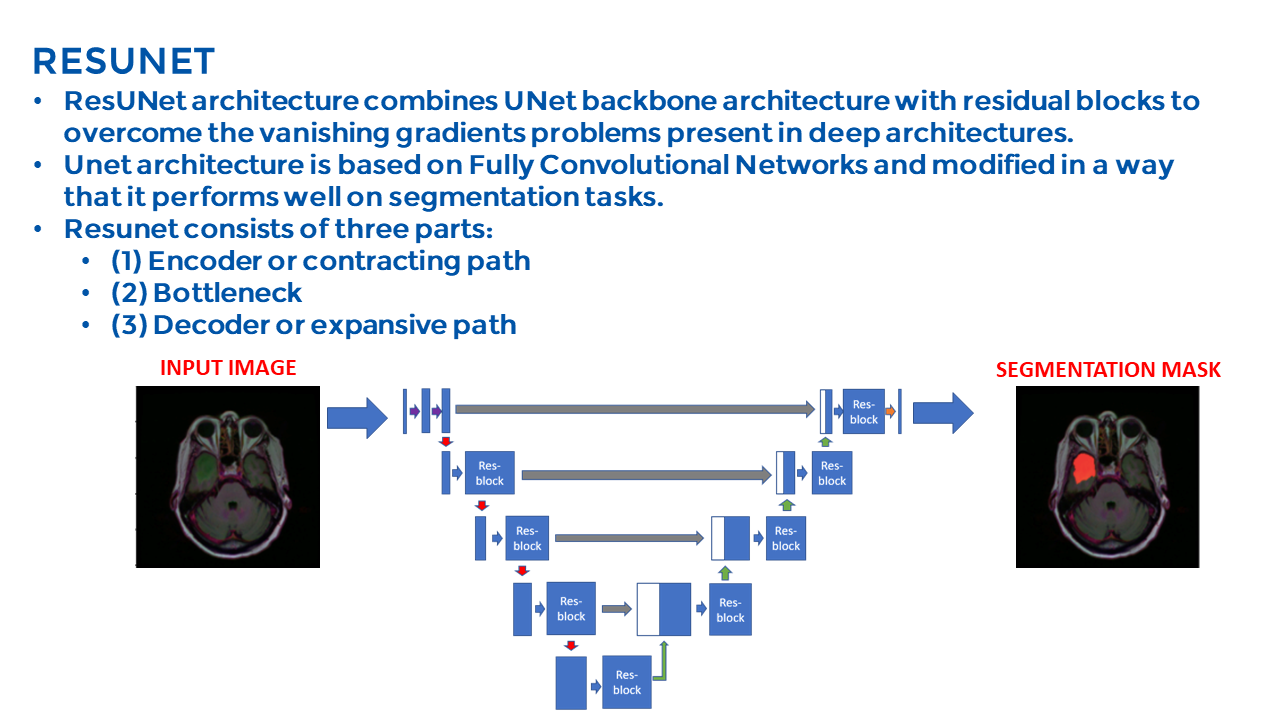
Throughout slide 8 and 9, trend is consistent no matter which network selected

On slide 11, we see the comparison of individual networks with varying thresholds vs threshNet systems. The closer a line is to the upper left corner of the graph, the more good it is performing. Another experimentation I did in my project was with choosing specific groups of networks rather than random selection. the 9 best sensitivity nets as ensemble members, as well as 9 best specificity nets as ensemble members.

Step 4. Generate results and plots

* Generate results and plots to evaluate overall performance
* Generate ROC plots to compare individual networks and the ThreshNet system

Step 5. Image Segmentation



Segmentation mask

(overlay on input image)

Input image

Build and train ResUNet Model:

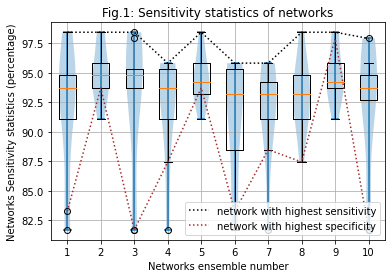
* ResUNet is based off of UNet, a U-Shaped fully Convolutional Network, and residual blocks
* It contains three parts:
  + Encoder Path 🡪 res-blocks & maxPooling ( 5 stages)
  + Decoder Path 🡪 res-blocks & upSampling (5 stages)
  + Bottleneck 🡪 Connection between Encoder and Decoder (1 stage)
* ResBlock unit within ResUNet
* Main path: two conv2D layers
* Shortcut path: one conv2D layer

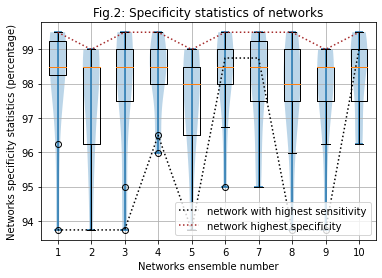
Test ResUNet with images that the image classification step returned as “1” (tumor present)

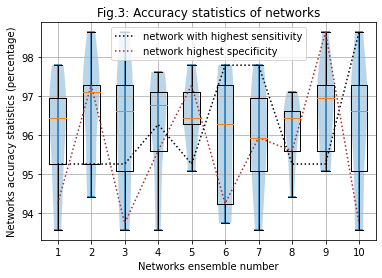
# Results

## Network Ensemble

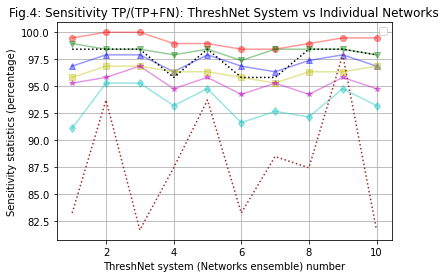
* 10 network ensembles, with 9 networks in each ensemble
  + Violin plots & box and whisker plots: network performance in each ensemble
  + Dotted lines = performance of individual network in ensemble:

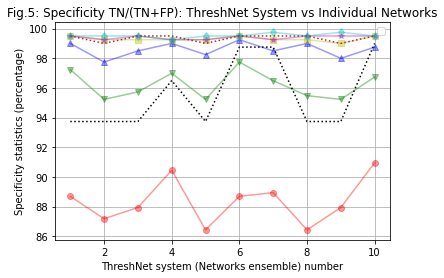


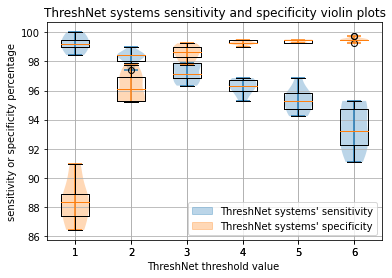




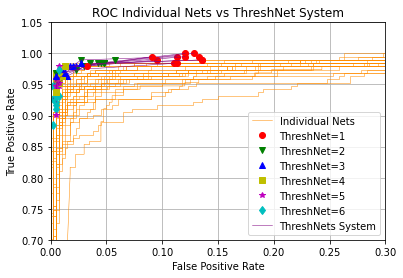
## ThreshNet system vs individual networks

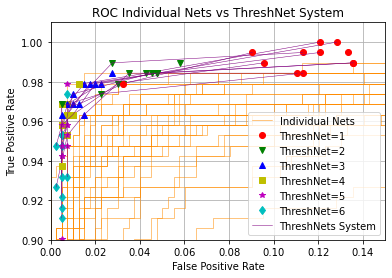


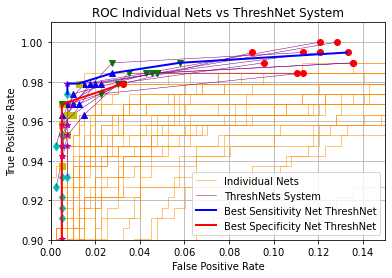




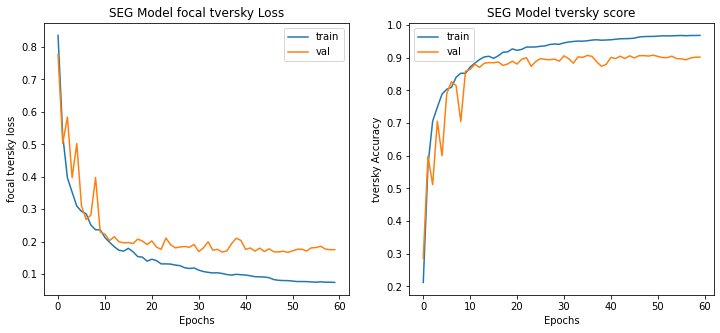
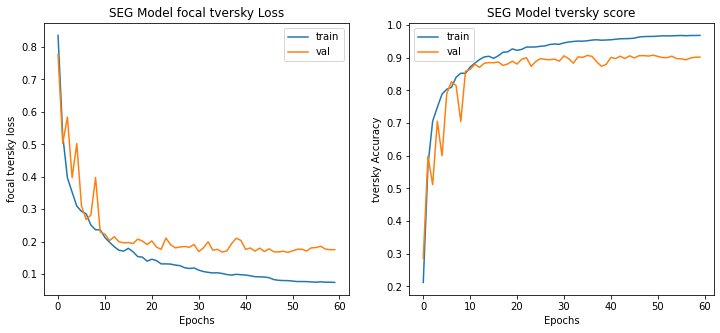
## ROC



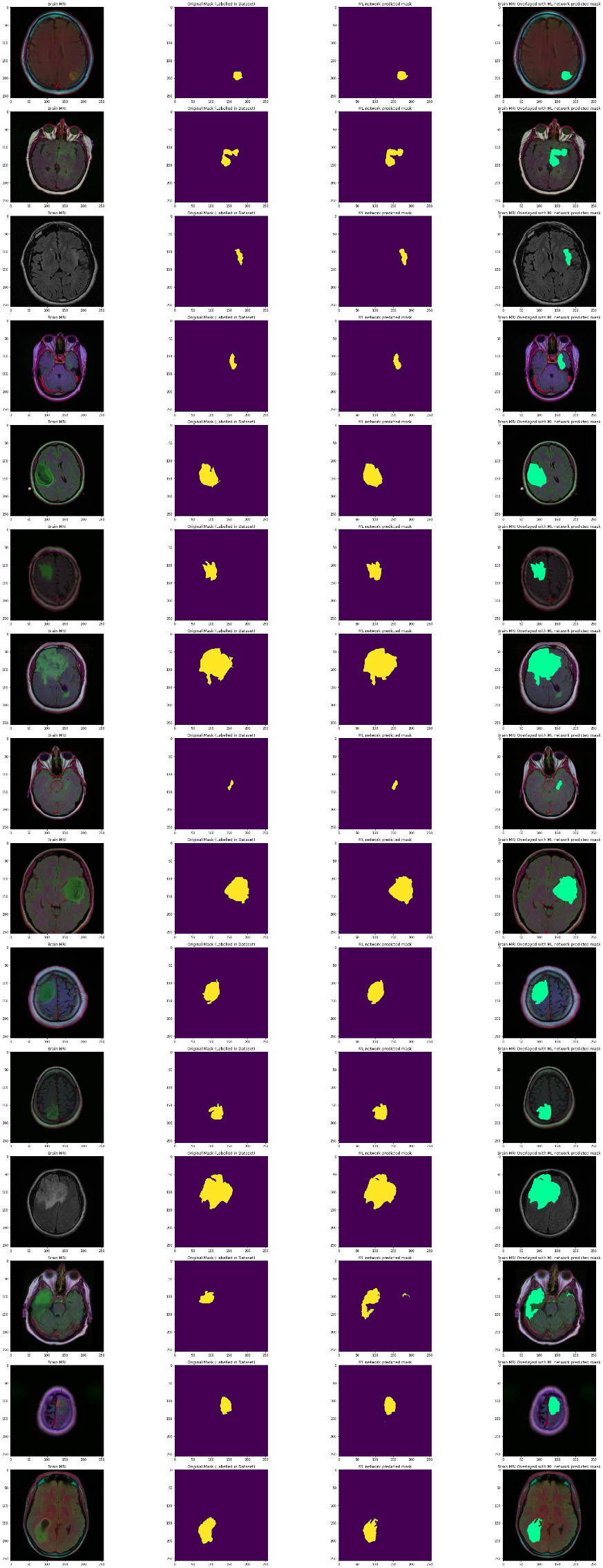




## Image segmentation







MRI Image Labelled Mask ResUNet Predicted Overlay

# Analysis

## Individual Networks

* Sensitivity TP/(TP+FN) : 81.7%-98.4%
* Specificity TN/(TN+FP): 93.7%-99.5%
* Network sensitivity & specificity may be inversely related

## ThreshNet systems vs. individual networks

* Performance of ThreshNet systems vs individual networks in ensemble
  + Lower ThreshNet (= 1 – 2):
    - Very high sensitivity; always outperforms the best sensitivity network (BSN) by approximately 1-3%
    - Specificity is consistently lower than BSN when ThreshNet = 1, but is higher than BSN 7/10 times when ThreshNet = 2
  + Higher ThreshNet (= 4 - 6):
    - Specificity similar to BSN (difference <1%), while sensitivity significantly improved (up to 15.2%)
  + Medium ThreshNet (= 3 - 5) :
    - Well balanced performance metrics among sensitivity, specificity and accuracy. All of these metrics are in the high 90%s.
* Sensitivity and specificity violin & box-and-whisker plots
  + Variance among ThreshNet systems < variance among individual networks
  + With different ThreshNet configurations, different sensitivity and specificity goals can be achieved

## ROC Curves

* ThreshNet systems consistently perform better than individual networks (closer to upper left corner).
* Performance of best ThreshNet selections are very good- better than individual nets
* But, not significantly better from ThreshNets of randomly chosen networks
* This means the ThreshNet System does not need best networks to perform well

## Image segmentation

ResUNet produced accurate results. ResUNet’s predicted tumor locations match closely with the masks from the labelled datset visually.

* Tversky index = 90.49%

# Conclusion

* ThreshNet systems proposed throughout the project achieved better performance than individual networks in the ensemble with regards to particular performance metrics and/or across all performance metrics
* Variance among ThreshNet systems are smaller than variance among individual networks, yielding more consistent performance.
* Individual networks in the ensemble do not need to have high performance for the ThreshNet system to yield high performance
* ThreshNet provided convenient means to achieve specific performance and trade-off goals between sensitivity and specificity through adjusting the ThreshNet parameter
* ResUNet's prediction mask closely matched the labelled mask in the dataset.

This project’s ThreshNet system achieves better performance compared to individual networks that make up the ensemble in particular performance metrics and/or across all performance metrics. Each individual network in the ensemble does not need to have high performance for the ThreshNet system to yield high performance. Different performance optimization goals can be achieved by adjusting ThreshNet parameter values-- this matches with my hypothesis and achieves the engineering design goals of this project. ResUNet's prediction mask also closely matches the labelled mask in the dataset.

# Recommendations

Future Work

* Apply ThreshNet technique to other types of medical diagnoses
* Extend project outside of the medical field (ex, to the self-driving automobile industry, where objects in images need to be detected very quickly to prevent accidents)
* Use individual neural network decisions as inputs to train other neural networks

I would continue the project for another year and do the project again if given the chance. Through my research, I have learned a lot about neural networks and data science, and I am excited to extend my findings into the future.

# Bibliography

[1] Buda, Mateusz, et al. Association of Genomic Subtypes of Lower-Grade Gliomas with Shape Features Automatically Extracted by a Deep Learning Algorithm. Computers in Biology and Medicine, 2019.

[2] Mazurowski, Maciej A., et al. “Radiogenomics of Lower-Grade Glioma: Algorithmically-Assessed Tumor Shape Is Associated with Tumor Genomic Subtypes and Patient Outcomes in a Multi-Institutional Study with The Cancer Genome Atlas Data.” Journal of Neuro-Oncology, U.S. National Library of Medicine, 2017, pubmed.ncbi.nlm.nih.gov/28470431/.

[3] Grote, Thomas, and Philipp Berens. “On the Ethics of Algorithmic Decision-Making in Healthcare.” Journal of Medical Ethics, Institute of Medical Ethics, 1 Mar. 2020, jme.bmj.com/content/46/3/205.citation-tools.

[4] LeCun, Yann, et al. “Convolutional Networks and Applications in Vision.” ResearchGate, May 2010, [www.researchgate.net/publication/221376179\_Convolutional\_Networks\_and\_Applications\_in\_Vision](http://www.researchgate.net/publication/221376179_Convolutional_Networks_and_Applications_in_Vision).

[5] Opitz, D.; Maclin, R. (1999). ["Popular ensemble methods: An empirical study"](https://doi.org/10.1613%2Fjair.614). [Journal of Artificial Intelligence Research](https://en.wikipedia.org/wiki/Journal_of_Artificial_Intelligence_Research). 11: 169–198. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1613/jair.614](https://doi.org/10.1613%2Fjair.614)

[6] Abdulla, Waleed. Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow, 19 Mar. 2018, <http://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46>.

[7] Sharma, Pulkit. “Image Segmentation Python: Implementation of Mask R-CNN.” Analytics Vidhya, 22 July 2019, [www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/](http://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation).

[8] Brownlee, Jason. “Best Practices for Preparing and Augmenting Image Data for CNNs.” Machine Learning Mastery, 5 July 2019, <http://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/>

[9] “Sensitivity and Specificity.” Wikipedia, Wikimedia Foundation, 28 Jan. 2021, <http://en.wikipedia.org/wiki/Sensitivity_and_specificity>.

[10] J. Li et al., “Multi-stage object detection with group recursive learning,” Arxiv.org. [Online]. Available: <http://arxiv.org/abs/1608.05159v1>. [Accessed: 24-Jul-2020].

[11] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

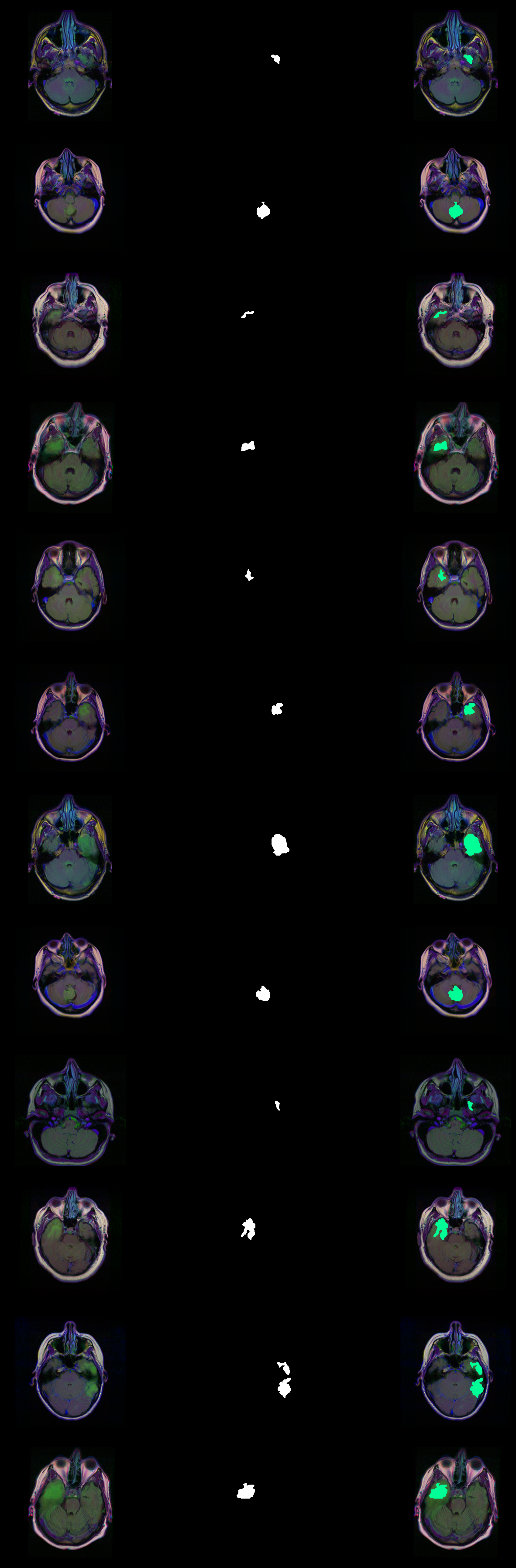
[12] Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, 30(11), 3212-3232.

[13] F. Amato, A. Lopez, E. M. Pena-Mendez, P. Vanhara, A. Hampl, and J. Havel, Eds., Artificial neural networks in medical diagnosis, vol. 11, no. 47–58. Journal of Applied Biomedicine, 2013. Available: doi: 10.2478/v10136-012-0031-x

# Appendices/Raw Data

## Data visualization (labelled Dataset)

MRI Scan | Labelled Mask | Overlay



## Individual Network Training and testing results

### Resnet101 (Only dense layers trainable)

Only dense layers trainable, rest layers using imagenet weights

*# Train resnet101*

*# run a different model using the same dataset and data split*

import tensorflow.keras.applications as applications

from keras import applications

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications import ResNet101

from tensorflow.keras.applications import VGG16

clf\_model = ResNet101(weights='imagenet', include\_top=False, input\_tensor=Input(shape=(256,256,3)))

*# clf\_model.summary()*

for layer **in** clf\_model.layers:

layers.trainable = False

head = clf\_model.output

head = AveragePooling2D(pool\_size=(4,4))(head)

head = Flatten(name='Flatten')(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(2, activation='softmax')(head)

model = Model(clf\_model.input, head)

model.compile(loss = 'categorical\_crossentropy',

optimizer='adam',

metrics= ["accuracy"]

)

*# model.summary()*

earlystopping = EarlyStopping(monitor='val\_loss',

mode='min',

verbose=1,

patience=15

)

checkpointer = ModelCheckpoint(filepath="clf-resnet101-weights.hdf5",

verbose=1,

save\_best\_only=True

)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',

mode='min',

verbose=1,

patience=10,

min\_delta=0.0001,

factor=0.2

)

callbacks = [checkpointer, earlystopping, reduce\_lr]

h = model.fit(train\_generator,

steps\_per\_epoch= train\_generator.n // train\_generator.batch\_size,

epochs = 50,

validation\_data= valid\_generator,

validation\_steps= valid\_generator.n // valid\_generator.batch\_size,

callbacks=[checkpointer, earlystopping])

Training and results log:

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet101\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

171450368/171446536 [==============================] - 5s 0us/step

Epoch 1/50

187/187 [==============================] - ETA: 0s - loss: 0.8586 - accuracy: 0.6719

Epoch 00001: val\_loss improved from inf to 1.92600, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 90s 482ms/step - loss: 0.8586 - accuracy: 0.6719 - val\_loss: 1.9260 - val\_accuracy: 0.6313

Epoch 2/50

187/187 [==============================] - ETA: 0s - loss: 0.5197 - accuracy: 0.7378

Epoch 00002: val\_loss improved from 1.92600 to 1.69215, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 68s 364ms/step - loss: 0.5197 - accuracy: 0.7378 - val\_loss: 1.6921 - val\_accuracy: 0.3688

Epoch 3/50

187/187 [==============================] - ETA: 0s - loss: 0.5026 - accuracy: 0.7425

Epoch 00003: val\_loss did not improve from 1.69215

187/187 [==============================] - 66s 352ms/step - loss: 0.5026 - accuracy: 0.7425 - val\_loss: 2.3174 - val\_accuracy: 0.3688

Epoch 4/50

187/187 [==============================] - ETA: 0s - loss: 0.4610 - accuracy: 0.7756

Epoch 00004: val\_loss improved from 1.69215 to 0.73206, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 69s 367ms/step - loss: 0.4610 - accuracy: 0.7756 - val\_loss: 0.7321 - val\_accuracy: 0.3750

Epoch 5/50

187/187 [==============================] - ETA: 0s - loss: 0.4061 - accuracy: 0.8157

Epoch 00005: val\_loss improved from 0.73206 to 0.66169, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 68s 362ms/step - loss: 0.4061 - accuracy: 0.8157 - val\_loss: 0.6617 - val\_accuracy: 0.6500

Epoch 6/50

187/187 [==============================] - ETA: 0s - loss: 0.3603 - accuracy: 0.8492

Epoch 00006: val\_loss improved from 0.66169 to 0.37812, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 68s 362ms/step - loss: 0.3603 - accuracy: 0.8492 - val\_loss: 0.3781 - val\_accuracy: 0.7937

Epoch 7/50

187/187 [==============================] - ETA: 0s - loss: 0.3405 - accuracy: 0.8682

Epoch 00007: val\_loss improved from 0.37812 to 0.25635, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 68s 366ms/step - loss: 0.3405 - accuracy: 0.8682 - val\_loss: 0.2564 - val\_accuracy: 0.8844

Epoch 8/50

187/187 [==============================] - ETA: 0s - loss: 0.3030 - accuracy: 0.8829

Epoch 00008: val\_loss did not improve from 0.25635

187/187 [==============================] - 66s 351ms/step - loss: 0.3030 - accuracy: 0.8829 - val\_loss: 0.3218 - val\_accuracy: 0.8969

Epoch 9/50

187/187 [==============================] - ETA: 0s - loss: 0.2889 - accuracy: 0.8923

Epoch 00009: val\_loss did not improve from 0.25635

187/187 [==============================] - 66s 352ms/step - loss: 0.2889 - accuracy: 0.8923 - val\_loss: 0.3279 - val\_accuracy: 0.8438

Epoch 10/50

187/187 [==============================] - ETA: 0s - loss: 0.2588 - accuracy: 0.9040

Epoch 00010: val\_loss did not improve from 0.25635

187/187 [==============================] - 66s 352ms/step - loss: 0.2588 - accuracy: 0.9040 - val\_loss: 0.3034 - val\_accuracy: 0.8531

Epoch 11/50

187/187 [==============================] - ETA: 0s - loss: 0.2459 - accuracy: 0.9057

Epoch 00011: val\_loss did not improve from 0.25635

187/187 [==============================] - 66s 355ms/step - loss: 0.2459 - accuracy: 0.9057 - val\_loss: 0.4431 - val\_accuracy: 0.8375

Epoch 12/50

187/187 [==============================] - ETA: 0s - loss: 0.2245 - accuracy: 0.9254

Epoch 00012: val\_loss improved from 0.25635 to 0.24533, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 69s 368ms/step - loss: 0.2245 - accuracy: 0.9254 - val\_loss: 0.2453 - val\_accuracy: 0.9031

Epoch 13/50

187/187 [==============================] - ETA: 0s - loss: 0.2339 - accuracy: 0.9157

Epoch 00013: val\_loss improved from 0.24533 to 0.15392, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 69s 371ms/step - loss: 0.2339 - accuracy: 0.9157 - val\_loss: 0.1539 - val\_accuracy: 0.9469

Epoch 14/50

187/187 [==============================] - ETA: 0s - loss: 0.2173 - accuracy: 0.9187

Epoch 00014: val\_loss did not improve from 0.15392

187/187 [==============================] - 67s 356ms/step - loss: 0.2173 - accuracy: 0.9187 - val\_loss: 0.2045 - val\_accuracy: 0.9250

Epoch 15/50

187/187 [==============================] - ETA: 0s - loss: 0.2049 - accuracy: 0.9284

Epoch 00015: val\_loss did not improve from 0.15392

187/187 [==============================] - 66s 355ms/step - loss: 0.2049 - accuracy: 0.9284 - val\_loss: 0.2804 - val\_accuracy: 0.8844

Epoch 16/50

187/187 [==============================] - ETA: 0s - loss: 0.1873 - accuracy: 0.9348

Epoch 00016: val\_loss improved from 0.15392 to 0.13024, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 69s 367ms/step - loss: 0.1873 - accuracy: 0.9348 - val\_loss: 0.1302 - val\_accuracy: 0.9531

Epoch 17/50

187/187 [==============================] - ETA: 0s - loss: 0.1930 - accuracy: 0.9324

Epoch 00017: val\_loss did not improve from 0.13024

187/187 [==============================] - 67s 359ms/step - loss: 0.1930 - accuracy: 0.9324 - val\_loss: 0.1699 - val\_accuracy: 0.9500

Epoch 18/50

187/187 [==============================] - ETA: 0s - loss: 0.2301 - accuracy: 0.9221

Epoch 00018: val\_loss did not improve from 0.13024

187/187 [==============================] - 68s 362ms/step - loss: 0.2301 - accuracy: 0.9221 - val\_loss: 0.3893 - val\_accuracy: 0.9344

Epoch 19/50

187/187 [==============================] - ETA: 0s - loss: 0.1851 - accuracy: 0.9361

Epoch 00019: val\_loss did not improve from 0.13024

187/187 [==============================] - 67s 359ms/step - loss: 0.1851 - accuracy: 0.9361 - val\_loss: 0.1475 - val\_accuracy: 0.9688

Epoch 20/50

187/187 [==============================] - ETA: 0s - loss: 0.1827 - accuracy: 0.9318

Epoch 00020: val\_loss did not improve from 0.13024

187/187 [==============================] - 67s 360ms/step - loss: 0.1827 - accuracy: 0.9318 - val\_loss: 0.2025 - val\_accuracy: 0.9344

Epoch 21/50

187/187 [==============================] - ETA: 0s - loss: 0.2002 - accuracy: 0.9251

Epoch 00021: val\_loss did not improve from 0.13024

187/187 [==============================] - 68s 364ms/step - loss: 0.2002 - accuracy: 0.9251 - val\_loss: 0.2004 - val\_accuracy: 0.9094

Epoch 22/50

187/187 [==============================] - ETA: 0s - loss: 0.2270 - accuracy: 0.9214

Epoch 00022: val\_loss did not improve from 0.13024

187/187 [==============================] - 69s 370ms/step - loss: 0.2270 - accuracy: 0.9214 - val\_loss: 0.2995 - val\_accuracy: 0.8719

Epoch 23/50

187/187 [==============================] - ETA: 0s - loss: 0.1750 - accuracy: 0.9408

Epoch 00023: val\_loss did not improve from 0.13024

187/187 [==============================] - 67s 360ms/step - loss: 0.1750 - accuracy: 0.9408 - val\_loss: 0.3871 - val\_accuracy: 0.8531

Epoch 24/50

187/187 [==============================] - ETA: 0s - loss: 0.1974 - accuracy: 0.9321

Epoch 00024: val\_loss improved from 0.13024 to 0.10726, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 69s 367ms/step - loss: 0.1974 - accuracy: 0.9321 - val\_loss: 0.1073 - val\_accuracy: 0.9656

Epoch 25/50

187/187 [==============================] - ETA: 0s - loss: 0.1769 - accuracy: 0.9418

Epoch 00025: val\_loss did not improve from 0.10726

187/187 [==============================] - 68s 363ms/step - loss: 0.1769 - accuracy: 0.9418 - val\_loss: 0.2750 - val\_accuracy: 0.9187

Epoch 26/50

187/187 [==============================] - ETA: 0s - loss: 0.1735 - accuracy: 0.9391

Epoch 00026: val\_loss improved from 0.10726 to 0.08880, saving model to clf-resnet101-weights.hdf5

187/187 [==============================] - 70s 377ms/step - loss: 0.1735 - accuracy: 0.9391 - val\_loss: 0.0888 - val\_accuracy: 0.9812

Epoch 27/50

187/187 [==============================] - ETA: 0s - loss: 0.1837 - accuracy: 0.9405

Epoch 00027: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 358ms/step - loss: 0.1837 - accuracy: 0.9405 - val\_loss: 0.1496 - val\_accuracy: 0.9500

Epoch 28/50

187/187 [==============================] - ETA: 0s - loss: 0.1671 - accuracy: 0.9505

Epoch 00028: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 359ms/step - loss: 0.1671 - accuracy: 0.9505 - val\_loss: 0.1093 - val\_accuracy: 0.9656

Epoch 29/50

187/187 [==============================] - ETA: 0s - loss: 0.1824 - accuracy: 0.9435

Epoch 00029: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 356ms/step - loss: 0.1824 - accuracy: 0.9435 - val\_loss: 0.2152 - val\_accuracy: 0.9375

Epoch 30/50

187/187 [==============================] - ETA: 0s - loss: 0.1685 - accuracy: 0.9431

Epoch 00030: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 359ms/step - loss: 0.1685 - accuracy: 0.9431 - val\_loss: 0.1585 - val\_accuracy: 0.9500

Epoch 31/50

187/187 [==============================] - ETA: 0s - loss: 0.1564 - accuracy: 0.9492

Epoch 00031: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 357ms/step - loss: 0.1564 - accuracy: 0.9492 - val\_loss: 0.1744 - val\_accuracy: 0.9219

Epoch 32/50

187/187 [==============================] - ETA: 0s - loss: 0.1753 - accuracy: 0.9388

Epoch 00032: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 360ms/step - loss: 0.1753 - accuracy: 0.9388 - val\_loss: 0.1211 - val\_accuracy: 0.9594

Epoch 33/50

187/187 [==============================] - ETA: 0s - loss: 0.1781 - accuracy: 0.9421

Epoch 00033: val\_loss did not improve from 0.08880

187/187 [==============================] - 69s 367ms/step - loss: 0.1781 - accuracy: 0.9421 - val\_loss: 0.0974 - val\_accuracy: 0.9781

Epoch 34/50

187/187 [==============================] - ETA: 0s - loss: 0.1490 - accuracy: 0.9492

Epoch 00034: val\_loss did not improve from 0.08880

187/187 [==============================] - 68s 362ms/step - loss: 0.1490 - accuracy: 0.9492 - val\_loss: 0.1873 - val\_accuracy: 0.9187

Epoch 35/50

187/187 [==============================] - ETA: 0s - loss: 0.1664 - accuracy: 0.9462

Epoch 00035: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 360ms/step - loss: 0.1664 - accuracy: 0.9462 - val\_loss: 0.1204 - val\_accuracy: 0.9625

Epoch 36/50

187/187 [==============================] - ETA: 0s - loss: 0.1409 - accuracy: 0.9545

Epoch 00036: val\_loss did not improve from 0.08880

187/187 [==============================] - 68s 363ms/step - loss: 0.1409 - accuracy: 0.9545 - val\_loss: 0.1157 - val\_accuracy: 0.9625

Epoch 37/50

187/187 [==============================] - ETA: 0s - loss: 0.1478 - accuracy: 0.9502

Epoch 00037: val\_loss did not improve from 0.08880

187/187 [==============================] - 69s 370ms/step - loss: 0.1478 - accuracy: 0.9502 - val\_loss: 0.1147 - val\_accuracy: 0.9563

Epoch 38/50

187/187 [==============================] - ETA: 0s - loss: 0.1594 - accuracy: 0.9508

Epoch 00038: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 359ms/step - loss: 0.1594 - accuracy: 0.9508 - val\_loss: 0.1067 - val\_accuracy: 0.9563

Epoch 39/50

187/187 [==============================] - ETA: 0s - loss: 0.1322 - accuracy: 0.9555

Epoch 00039: val\_loss did not improve from 0.08880

187/187 [==============================] - 67s 358ms/step - loss: 0.1322 - accuracy: 0.9555 - val\_loss: 0.2366 - val\_accuracy: 0.8844

Epoch 40/50

187/187 [==============================] - ETA: 0s - loss: 0.1789 - accuracy: 0.9438

Epoch 00040: val\_loss did not improve from 0.08880

187/187 [==============================] - 68s 362ms/step - loss: 0.1789 - accuracy: 0.9438 - val\_loss: 0.1170 - val\_accuracy: 0.9656

Epoch 41/50

187/187 [==============================] - ETA: 0s - loss: 0.1501 - accuracy: 0.9512

Epoch 00041: val\_loss did not improve from 0.08880

187/187 [==============================] - 69s 367ms/step - loss: 0.1501 - accuracy: 0.9512 - val\_loss: 0.2498 - val\_accuracy: 0.9438

Epoch 00041: early stopping

37/37 [==============================] - 6s 159ms/step - loss: 1.5252 - accuracy: 0.9390

Test accuracy : 93.89830231666565 %

0.9389830508474576

precision recall f1-score support

0 0.96 0.95 0.95 399

1 0.90 0.92 0.91 191

accuracy 0.94 590

macro avg 0.93 0.93 0.93 590

weighted avg 0.94 0.94 0.94 590

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50 (Only dense layers trainable)

Only dense layers trainable, rest layers using imagenet weights

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_2 (Only dense layers trainable, pool\_size =(4,4))

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_3 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet101\_2 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet152 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet152\_2 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

### MobileNet-2 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a cell phone

Description automatically generated with low confidence

### MobileNet-3 (Only dense layers trainable)

Only dense layers trainable, rest layers with imagenet weights

Graphical user interface

Description automatically generatedA picture containing text, electronics, screenshot

Description automatically generated

### Resnet101 (All layers trainable)

*Train resnet101*

*# run a different model using the same dataset and data split*

import tensorflow.keras.applications as applications

from keras import applications

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications import ResNet101

from tensorflow.keras.applications import VGG16

clf\_model = ResNet101(weights='imagenet', include\_top=False, input\_tensor=Input(shape=(256,256,3)))

*# clf\_model.summary()*

for layer **in** clf\_model.layers:

All layers trainable

head = clf\_model.output

head = AveragePooling2D(pool\_size=(4,4))(head)

head = Flatten(name='Flatten')(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(2, activation='softmax')(head)

model = Model(clf\_model.input, head)

model.compile(loss = 'categorical\_crossentropy',

optimizer='adam',

metrics= ["accuracy"]

)

*# model.summary()*

earlystopping = EarlyStopping(monitor='val\_loss',

mode='min',

verbose=1,

patience=15

)

checkpointer = ModelCheckpoint(filepath="clf-resnet101-weights.hdf5",

verbose=1,

save\_best\_only=True

)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',

mode='min',

verbose=1,

patience=10,

min\_delta=0.0001,

factor=0.2

)

callbacks = [checkpointer, earlystopping, reduce\_lr]

h = model.fit(train\_generator,

steps\_per\_epoch= train\_generator.n // train\_generator.batch\_size,

epochs = 50,

validation\_data= valid\_generator,

validation\_steps= valid\_generator.n // valid\_generator.batch\_size,

callbacks=[checkpointer, earlystopping])

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50 (All layers trainable)

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

### Resnet50\_2 (All layers trainable)

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_3 (All layers trainable)

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

### Resnet101\_2 (All layers trainable)

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

### Resnet152 (All layers trainable)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet152\_2 (All layers trainable)

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a cell phone

Description automatically generated with low confidence

### MobileNet (All layers trainable)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet-2 (All layers trainable)

Graphical user interface, application

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet-3 (All layers trainable)

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

### Resnet101 (metrics: falsenegatives)

*# Train resnet101*

*# run a different model using the same dataset and data split*

import tensorflow.keras.applications as applications

from keras import applications

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications import ResNet101

from tensorflow.keras.applications import VGG16

clf\_model = ResNet101(weights='imagenet', include\_top=False, input\_tensor=Input(shape=(256,256,3)))

*# clf\_model.summary()*

for layer **in** clf\_model.layers:

layers.trainable = False

head = clf\_model.output

head = AveragePooling2D(pool\_size=(4,4))(head)

head = Flatten(name='Flatten')(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(2, activation='softmax')(head)

model = Model(clf\_model.input, head)

model.compile(loss = 'categorical\_crossentropy',

optimizer='adam',

metrics=[tf.keras.metrics.FalseNegatives()]

)

*# model.summary()*

earlystopping = EarlyStopping(monitor='val\_loss',

mode='min',

verbose=1,

patience=15

)

checkpointer = ModelCheckpoint(filepath="clf-resnet101-weights.hdf5",

verbose=1,

save\_best\_only=True

)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',

mode='min',

verbose=1,

patience=10,

min\_delta=0.0001,

factor=0.2

)

callbacks = [checkpointer, earlystopping, reduce\_lr]

h = model.fit(train\_generator,

steps\_per\_epoch= train\_generator.n // train\_generator.batch\_size,

epochs = 50,

validation\_data= valid\_generator,

validation\_steps= valid\_generator.n // valid\_generator.batch\_size,

callbacks=[checkpointer, earlystopping])

Graphical user interface

Description automatically generated with medium confidence

Chart, treemap chart

Description automatically generated

### Resnet50 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_2 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_3 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet101\_2 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet152 (metrics: falsenegatives)

Graphical user interface

Description automatically generated with medium confidence

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet152\_2 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet-2 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet-3 (metrics: falsenegatives)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet101\_2 (new seed)

### Resnet50 (poolsize: 2,2)

*# run a different model using the same dataset and data split*

import tensorflow.keras.applications as applications

from keras import applications

from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.keras.applications import ResNet101

from tensorflow.keras.applications import VGG16

clf\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=Input(shape=(256,256,3)))

*# clf\_model.summary()*

head = clf\_model.output

head = AveragePooling2D(pool\_size=(2,2))(head)

head = Flatten(name='Flatten')(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(256, activation='relu')(head)

head = Dropout(0.3)(head)

head = Dense(2, activation='softmax')(head)

model = Model(clf\_model.input, head)

model.compile(loss = 'categorical\_crossentropy',

optimizer='adam',

metrics= ["accuracy"]

)

*# model.summary()*

earlystopping = EarlyStopping(monitor='val\_loss',

mode='min',

verbose=1,

patience=15

)

checkpointer = ModelCheckpoint(filepath="clf-resnet50-weights.hdf5",

verbose=1,

save\_best\_only=True

)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',

mode='min',

verbose=1,

patience=10,

min\_delta=0.0001,

factor=0.2

)

callbacks = [checkpointer, earlystopping, reduce\_lr]

h = model.fit(train\_generator,

steps\_per\_epoch= train\_generator.n // train\_generator.batch\_size,

epochs = 50,

validation\_data= valid\_generator,

validation\_steps= valid\_generator.n // valid\_generator.batch\_size,

callbacks=[checkpointer, earlystopping])

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### Resnet50\_2 (different dropout)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### MobileNet (different dropout)

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

### MobileNet\_2 (pool\_size=(2,2))

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

### MobileNetV2

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### ResNet152 (optimizer='adagrad')

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

### ResNet101\_2 (dropout = 0.1)

Graphical user interface

Description automatically generated

A picture containing text, electronics, screenshot

Description automatically generated

## Image segmentation

Columns left to right:

MRI image | Labelled Mask | ResUNet predicted mask | labelled mask Overlay| Predicted Overlay

A picture containing light, traffic, outdoor, lit

Description automatically generated