Data Science Capstone

A Case Study on Deep Learning and Object Detection: Stroke Object Detection and Classification with Convolutional Neural Networks

University of Wisconsin Data Science Program

Author: Patrick Hearin Ph.D.

Date: February 2023

Table Of Contents

Contents

Chapter I

Introduction

Table Of Contents

Artificial Intelligence (AI) has been revolutionizing the medical field. Many different applications based on AI have been helping medical professionals make more accurate diagnoses and overall improve the quality of patient care. Frost & Sullivan, a research firm, estimates that in the future patient outcomes will improve by thirty to forty percent because of AI. Furthermore, AI could also reduce the cost of treatment by fifty percent [1] Frost et al. If these predictions are accurate AI is going drastically change healthcare, save many people's lives, and change society in profound ways.

The topic of all AI in the medical field is broad and complicated. There are many notable projects using AI in different medical fields. Most pertinent to this Capstone was when Deep Learning Algorithms were used to diagnose Tuberculosis in chest X-ray images [2] Lakhani et al. This research provides a great example of how to obtain high accuracy. In [2] untrained and pre-trained networks are used with augmentation optimization to achieve one hundred percent specificity and ninety-six percent accuracy. This level of accuracy is comparable with human radiologists.

For this Capstone project, Deep Learning will be applied to high-resolution stroke images using computer vision methods. This project was inspired by the Kaggle competition [3] and many of the other articles referenced in this chapter previously. The subfield of medicine that studies strokes are vascular neurology. An interesting article that discusses how AI can compete with human neurologists [4] Vishnu et al. Also, at the end of the article, it provides a great list of references to AI being used in outcome predictions and prognostic evaluation after strokes.

This approach taken in the pedagogy of this Capstone is similar to a thesis style of the document and is in line with the parameters of the Capstone course. First Computer Vision and Deep Learning basics will be covered in the Literature Review, chapter 2. Then the stroke images will be preprocessed and many different algorithms will be trained using TensorFlow and PyTorch. The methodology of these calculations will be covered in the Methodology chapter Currently, the Kaggle notebook [5] is being used to reduce the size of the high-resolution images and this is a fundamental calculation for this Capstone. Furthermore, after obtaining a reasonable image set that won't overload the memory the algorithms will be calculated and then they will be used to make an object detection system. I intend to start with simpler neural networks and then keep building the complexity and optimization to achieve competitive accuracy. Then different pre-trained algorithms like VGG16 and ResNet will be used to try to find the optimal accuracy. Finally, an object detection model will be trained to give a useful product to stakeholders. This product could be used to give physicians the ability to detect strokes and classify them. Object detection literature will be reviewed in Chapter 2, then in Chapter 3 the methodology will be presented. Chapter 4 will present the results from Chapter 3. Finally, Chapter 5 will elaborate on the conclusion of this Capstone.

This capstone's goal is to give stakeholders like Mayo Clinic and other similar institutions accurate algorithms that approximate human neurologists. Although there is not as much time to produce such an algorithm as in [2], this project will try to give a competitive algorithm for the time constraints. Such that the stakeholders have a useful algorithm that can be used in production. Finally, the references [6], [7] are also great articles on artificial intelligence in

medical images.

Chapter II

Literature Review

Table Of Contents

1 Computer Vision Basics

Images are data made up of pixels. There are two basic types of images: grayscale and color RGB. Grayscale images can have values between 0 and 255. With 0 being black and 255 being white. As an example, the pixels from a simple image of a circle shown in Figure II.1 can be loaded into a list and then printed out. If the raw numbers are printed out the image is hard to see, so characters are used instead of the raw numbers and the image of the circle is easier to see. The characters of the circle in Figure II.1 can be found in Figure II.2. The code to produce Figure II.2 can be found in this capstones code addendum [8].

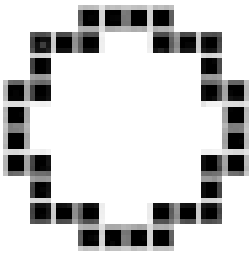


Figure II.1: Greyscale Example

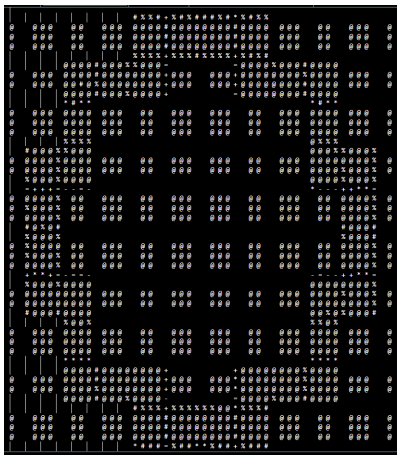


Figure II.2: Pixel values replaced with characters.

So greyscale images are just two-dimensional arrays of data. This data is not structured and is just an array of values from zero to two-hundred-fifty-five. These values can be printed out such that the columns are just a single value and the rows start on the left at zero and go to two-hundred fifty-five. This creates a gray gradient that is shown in Figure II.3

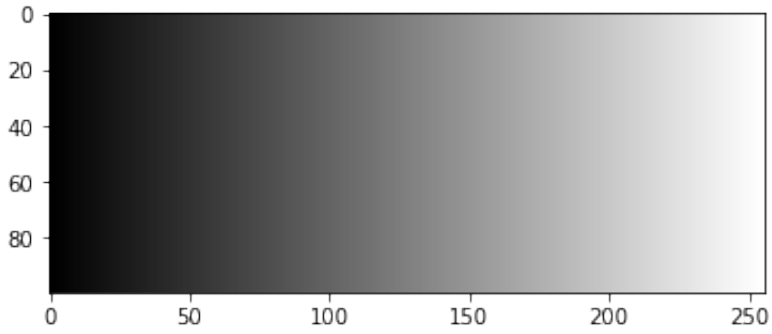


Figure II.3: Greyscale Example

For color images, there is three channels: red, green, and blue. These colors are abbreviated by RGB. Although in OpenCV the colors are BGR. This fact has to be taken into account when working with the library. The values of the three colors range from 0 to 255 again with 0 being black and 255 being the respective color. The code in listing II.2 of the Capstone Code Addendum divides the image into three sections and three channels. The resulting image is red, green, and blue as shown in Figure II.4.

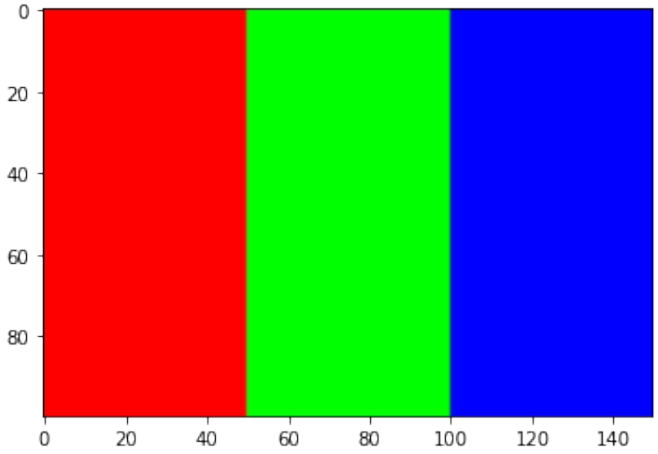


Figure II.4: RGB Example

2 Image Transformations

Table Of Contents

Image transformations are calculated with matrices. Learning the basics of transformations leads us to a more comprehensive understanding of what a convolution or filter is in a deep learning algorithm. One of the simplest transformations is rotation. This transformation is learned by undergraduate students when they first learn the difference between a scalar and a vector quantity. Starting with the following grayscale image of Eau-Claire students walking to class, Figure II.5,w we will rotate the image by forty-five degrees.

Figure II.5: Initial Image

The matrix used for rotation is defined as R and is typical two dimensional rotation matrix with angle θ

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{II.1}$$

For this example $\theta = 45°$ and this leaves the matrix

$$R = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}. \end{pmatrix} \tag{II.2}$$

To apply this transformation to the image the following algorithm must be applied

1. The image I and rotation matrix R are given.

2. Transform image extents to get the resultant image size.

3. Initialize the resultant image with all zeros.

4. Compute the inverse transformation matrix: $T_{Inverse}$.

5. For each pixel location (a, b) in the resultant image:

   (a) On a point (a, b) the inverse matrix $T_{Inverse}$ can be applied to get (c, d).

   (b) Apply interpolation on (c, d) and get intensity g.

   (c) Copy intensity g on location (a, b) in the resultant image.

This list comes from AI Sciences [9]. The image is interpolated after the rotation transformation because of four reasons:

1. Image resolution: When an image is transformed it can change the resolution. Interpolation changes the resolution of an image by adding or removing pixels. This can be used in zooming, resizing, and rotations.

2. Image quality: Interpolation can improve the quality of an image by smoothing out edges and reducing pixelation. Thus this is useful when working with low-resolution or compressed images.

3. Coordinate mapping: When an image is transformed, the coordinates of the pixels may no longer align with the original grid. To map the pixel values to the correct new values interpolation is used. This allows the image that is transformed to be correctly displayed.

4. Consistency: Interpolation helps to keep the consistency of the image. It helps to keep the image in the same size and resolution for the following processing steps.

For rotation, the third reason for interpolation is important because after the transformation most of the image is not on the coordinate grid. Thus when the final image is produced most of the original image will not be correctly transformed into the visible part of the reference frame. The calculation of interpolation uses the pixel value from adjacent pixels is used in a weighted average. There are many other ways to interpolate the pixel values: to calculate this transformation bilinear interpolation will be used. The function to calculate the interpolation is

in the Capstone Coding Addendum [8]. Bilinear Interpolation uses each axis in a two-dimensional plane and weights the values in each dimension using an average. To calculate the weights the distance to the interpolated point from the adjacent points is averaged. The distances are switched because the closer pixel is weighted more than the other that is farther away. Using Interpolation and transforming with the inverse transformation matrix yields Figure 11.6.

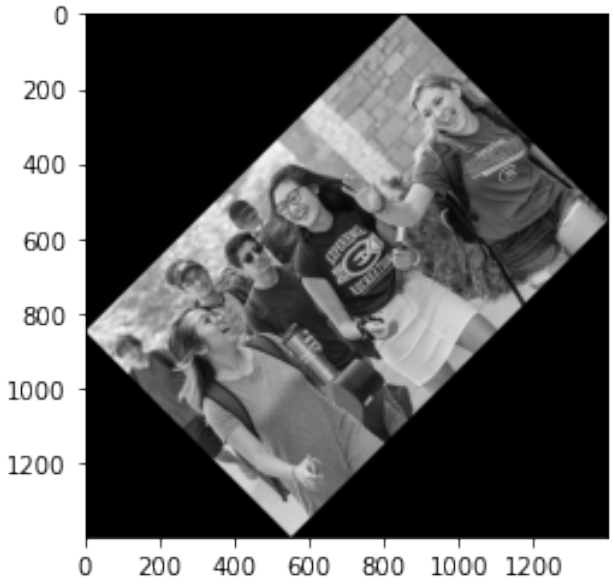


Figure II.6: Transformation Computer Vision Example

The resulting image is rotated in a counter-clockwise direction. Moreover, this is just an example of one type of transformation on images. There are many other transformations that make up the fundamental basis of computer vision. A brief summary of image transformations is

1. Rotation: This transformation rotates the image by an angle θ and rotation matrix R.
2. Scaling: This transformation changes the size of the image by using a transformation matrix.
3. Translation: This transformation shifts the image in the two-dimensional plane.
4. Reflection: This transformation flips the image horizontally or vertically.
5. Cropping: This transformation removes a sub-region of the image.
6. Shearing: This transformation skews the image by a specified angle along the axes.
7. Brightness/Contrast adjustment: This transformation changes the brightness and contrast of the image.
8. Blurring: This transformation applies a blur filter to the image to reduce noise or to create a smoothing effect.
9. Sharpening: This transformation reinforces the details of the image and makes the edges more defined.
10. Color map: The final transformation applies a color map to the image, which can be used to change the colors of the image.

Many other topics could be explored but for brevity, this simple example is presented. The purpose of this section is to present basic transformations on images that will introduce the concept of using matrices with images. This concept is pertinent to Deep Learning since the images will use matrices to make it possible to train neural networks on images.

3 Convolution Filters

Convolutional neural networks (CNN) are used for image and video deep learning algorithm training. The filter that is applied is called a convolution filter and this is how deep learning algorithms are able to train on images. The network learns a set of filters that can detect certain features in an image instead of each pixel in the image. These filters are then applied to the entire image, allowing the network to learn patterns. Additionally, CNNs reduce the number of dimensions of the feature maps using pooling layers. Applying pooling layers makes the computations simpler and allows more complicated images to be trained with deep learning. This makes CNNs much better than traditional neural networks for images.

As an example of a convolutional filter, the following matrices X and C will be used as an example. This example is a basic image of the letter X. The filter used is just a simple diagonal convolutional filter C. This example is described in AI Sciences [10]. The black pixels are

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{II.3}$$

From this image, there are three convolutional filters. The first convolutional filter C_1 matchs the features that are the portion of the "X" that is the diagonal of the X matrix.

$$C_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{II.4}$$

The next convolutional filter C_2 matches the feature that is the other diagonal of the "X" in the X matrix.

$$C_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{II.5}$$

Finally, the convolutional filter C_3 is the feature that is the middle part of "X".

$$C_3 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{II.6}$$

These matrices are not multiplied in the typical manner where rows and columns are multiplied. The multiplication is one-to-one each matrix position multiplies the equivalent position in the other matrix. The convolutional matrices C_1, C_2, C_3 are moved over the image and match different features in the image. In this case these three features could be used to classify the image on an X.

4 Neural Network Basics

The last section described the basics of convolutional filters and this section will describe the basics of neural networks. In general, neural networks mimic features of brains found in many different creatures. The main analogy is the connected nodes or neurons that calculate the model by finding the optimized minimum of error. A single node or neuron is a perceptron. This is the simplest of all neural networks and is a great place to start calculating neural networks.

Neural networks typically take real numbers as input and then output some sort of model. The neural network trains on the input data to find the optimized model. There are different weights associated with each node and these weights are optimized during the training. The first step of training a neural network is to give it a general set of labeled example data. Then the weights are optimized such that the final model has some amount of generalization to other data.

There are many different types of neural networks some notable ones include: recurrent neural networks, convolutional neural networks, and feedforward neural networks. Each type of neural network is suited for different types of tasks and data. Neural networks are used for many different tasks in many different areas of study, such as image classification, object detection, natural language processing, video games, and many different tasks in the physical sciences.

There are a plethora of neural networks. Figure II.7 from The Asimov Institute [11], illustrates different the neural networks' taxonomy. For this Capstone the only ones that will be used are the convolutional neural networks Figure II.8 from The Asimov Institute.

Overall, there are many different neural networks. With such a rich diversity there is so much possibility in the future for technological development. The rest of this chapter will focus on Convelutional Neural Networks to train on an image set. This is just one out of the many in Figure II.7 and therefore each algorithm has such a deep knowledge base to learn.

(a) Tensor Fundamentals

A tensor is a multi-dimensional array that is used in different mathematical abstractions. Tensors are used in many different fields, including physics, engineering, and machine learning. Tensors are generalizations of scalars, vectors, and matrices. Tensors make image classification and object detection possible with deep learning.

A scalar is invariant to rotations and is usually a real number: this is a tensor of order zero. The next-order tensor does transform under rotations and is called a vector or usually a tuple of real numbers: this is a tensor of order one. A matrix is a two-dimensional array of real

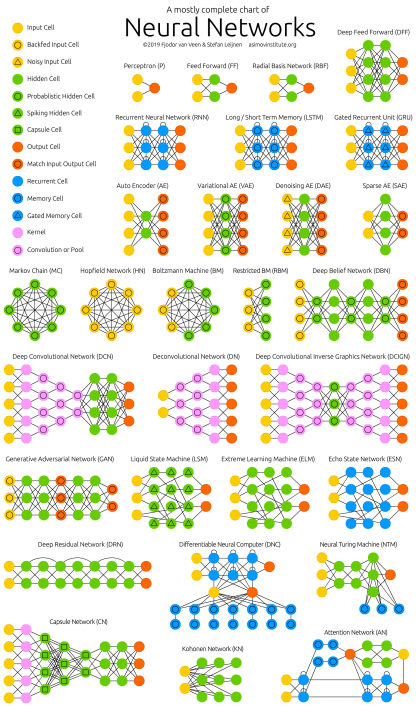


Figure II.7: Neural Network Zoo from The Asimov Institute [11]

numbers and is a 2nd-order tensor. Anything greater than a 2nd-order tensor has no traditional mathematics analogy and just has more indices at each order.

Tensors can be used in data science and other fields, including image and video processing, natural language processing, and deep learning. TensorFlow and PyTorch are popular libraries for working with tensors and building machine-learning models. Tensor notation is also used in physics to describe various physical phenomena.

In the article "Tensor Methods in Computer Vision and Deep Learning" the authors Panagakis et al [12] summarize many of the modern uses of tensors used in CNN's. This article describes many different advanced techniques. One interesting technique is fibers and unfolding which is a way to think about tensors with rows and columns. The advanced techniques in [12]

are too complicated to thoroughly review in this capstone but this article shows how tensors are used in advanced CNN's. A great software described in the article is Tensorly which is used for tenser learning.

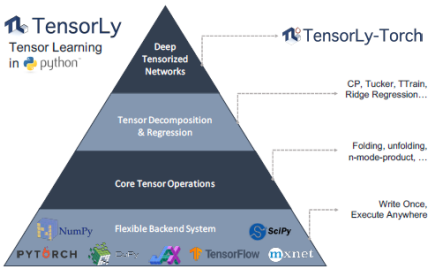


Figure II.8: Tensorly: A Python Library used for Tensor Learning [12].

(b) Deep Learning Example

This section will give an example that will aid the reader in comprehending the basics of deep learning. The data that is used will be structured to present the basic features of the calculations that will be later done on unstructured data. The calculation follows the methodology in [10]. Moreover, the ability of deep learning algorithms to handle non-linearities in the data is the main conclusion of the calculation. This first step is to generate some random data using PyTorch in Python that has a non-linear binary classification. The steps are summarized below

- Import necessary libraries (torch, numpy, matplotlib, torch.nn, and sklearn).
- Generate a binary classification dataset using the "make_circles" function from scikit-learn with 500 samples.
- This function has the following variables: random_state set to 123, noise equals 0.1, and factor is 0.2.
- Convert the generated dataset into PyTorch tensors defined as x_data and y_data.
- Plot the data points with matplotlib's scatter function, where the x and y axis represent the two features of the data and the color of the points represents the class label 0 or 1.

The code generates two concentric circles randomly and is shown in Figure II.9. Next a deep learning algorithm in PyTorch will be used to classify the points.

Next, we define the neural network. The activation function used is the sigmoid function. This neural network is a basic one. The neural networks used in the CNN's have many more layers that have different purposes. The

- Defined a class "Model" that extends the functionality of PyTorch's nn.Module class.
- In the class constructor (__init__), the input size, first hidden layer size, and output size are defined and defined as variables.

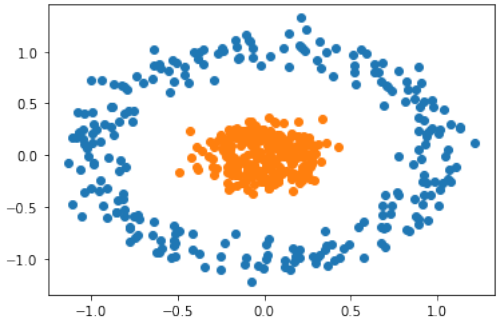


Figure II.9: Classification Data Example

- Two fully connected linear layers, self.linear and self.linear2, are defined using PyTorch's nn.Linear class and initialized with the input size and first hidden layer size, and the first hidden layer size and output size, respectively.
- A forward pass method forward is defined which takes an input x passes it through both linear layers and returns the result after passing it through a sigmoid activation function.
- A predict method is defined which takes an input x, calls the forward method on it to get the predicted value, and returns 1 if the prediction is greater than or equal to 0.5, otherwise returns 0.

Set the loss, optimizer, and train the model with one thousand epochs. The steps to implement are summarized below.

- A seed is set for PyTorch's random number generator to ensure the reproducibility of results.
- An instance of the Model class is created with input size 2, first hidden layer size 4, and output size 1.
- The parameters of the model are printed.
- A binary cross-entropy loss function criterion is defined using PyTorch's nn.BCELoss class.

- An optimizer is defined using PyTorch's torch.optim.Adam with the model parameters and a learning rate of 0.1.
- A loop is executed for 1000 epochs to train the model.
- In each iteration of the loop, the model's forward pass is executed on the input data x_data and the output is stored in y_pred.
- The binary cross-entropy loss between the predicted and target outputs is calculated using the criterion and stored in loss.
- The loss value is printed and appended to the losses list.
- The gradients are zeroed out, the backward pass is executed to calculate the gradients, and the optimizer take a step to update the model parameters.
- Finally, plot the loss and the decision boundary.

Next the loss is plotted using the following steps. The graph of the output is shown in Figure

- Uses the plt.plot function from the matplotlib library to create a plot.
- The x-axis of the plot is defined by the range function with a range of values equal to the number of epochs.
- The y-axis of the plot is defined by the losses list.
- The y-axis is labeled as 'Loss' using the plt.ylabel function.
- The x-axis is labeled as 'epoch' using the plt.xlabel function.

- A function plot_decision_boundary is defined that takes in two arguments, X and y.
- The x and y ranges of the grid are defined as x_span and y_span, respectively.
- np.meshgrid is used to create a mesh from the x and y ranges.

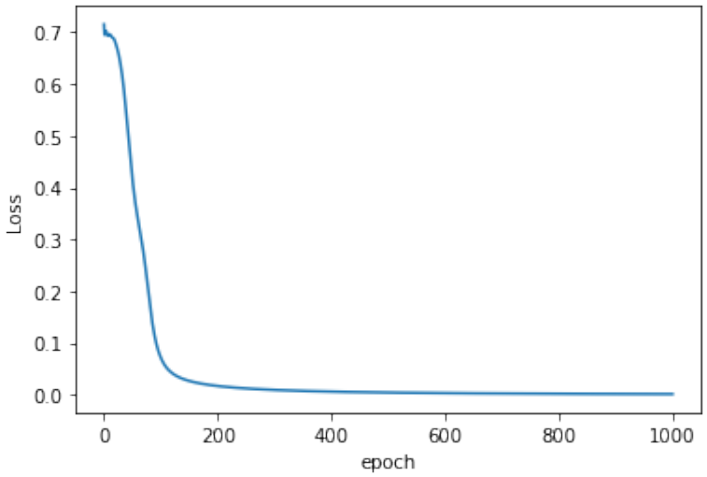


Figure II.10: Deep Learning Classification Loss

- A tensor grid is created from the mesh.
- The forward pass of the model is applied to the grid tensor to get the prediction pred_func.
- The prediction is converted to a numpy array z and reshaped to match the shape of the mesh.
- The plt.contourf function is used to create a filled contour plot with xx, yy, and z as inputs.
- The plot_decision_boundary function is called with X and y as arguments.
- A separate function scatter_plot is called, though its definition and details are not provided in this code.

This example displays the methodology of Deep Learning models. The non-linear nature of data can be modeled and this classification is shown in Figure II.11. This model is calculated using the PyTorch library. In the proceeding chapters, TensorFlow and PyTorch will be used to calculate the CNN's on the stroke image data. Although, before calculating those let's review CNN's in the next chapter.

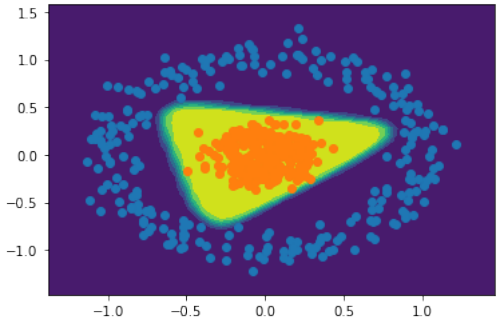


Figure II.11: Deep Learning Classification Example

5 Convolution Neural Networks

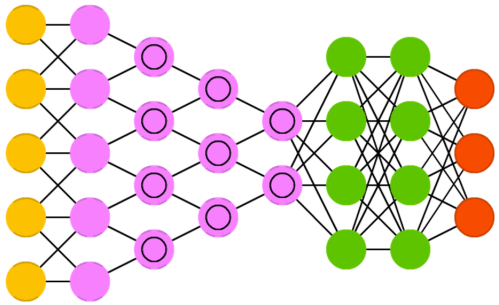


Figure II.12: CNN or deep convolutional neural network from The Asimov Institute [12]

Convolutional neural networks (CNNs) are specifically for calculating algorithms on images. In Figure II.12 the a general CNN is shown. Other neural networks do not take into account the patterns in the spatial structure of images. Moreover, trying to train other algorithms is computationally impossible because there would have to be a hidden layer for each pixel. Convolutional neural networks have much fewer parameters than other neural networks. Another benefit of using CNNs is the pooling layer helps prevent overfitting the data when compared with other neural networks.

Convolution basically is just applying filters to the image in terms of matrix multiplication. The first part of a CNN is the filter. Filters are a matrix that is much smaller that the image parameters, that contain weights. The smaller filter matrix slides over the image and the weights multiply the values of the image. A given filter can have multiple purposes. A filter can be used to yield high activation values when the part of the image that is pertinent is being filtered.

The final parts of a CNN are stride, padding, and pooling. The stride is how much the kernel is translated across the image at each iteration. Although sometimes the image is not able to fit the strides and filter into the image size. Thus, padding is used to fit the filter and stride over the image. To pad the image zeros are placed on the outside such that the stride and filter can always fit on the resulting image. The next part of a convolutional neural network is pooling. Pooling aggregates a subsection of the image. Pooling reduces the number of pixels needed to be stored. The final part of the convolutional part of the neural network is the flatten layer or the fully connected layer. This layer takes the matrix values and makes it a linear set of numbers or "flattens". Then the final flat data is sent to the hidden layers and finally the output is obtained. A great introduction to Convolutional Neural Networks can be found in O'Shea et al [13].

6 Object Detection Basics

Object detection is a computer technology related to computer vision and image processing that deals with detecting objects belonging to different classes, such as humans, buildings, or cars. Using deep learning, object detection is used in many applications, such as self-driving cars, surveillance systems, and object tracking. The basic pipeline of object detection includes two stages: (1) object proposal generation, which aims to generate a set of candidate regions that may contain objects, and (2) object classification, which classifies the objects in the proposed regions. The most popular deep learning frameworks for object detection are RCNN, Fast RCNN, Faster RCNN, YOLO, and SSD.

In this section, the OpenCV library will be used to show how object detection works. The face detection method from OpenCV will be used on the previous image that was used for the

transformation section. First, start off with the full-color image of the students at the Eau Claire campus below in Figure II.13.

Figure II.13: Object Detection Sample Image

The steps to do this calculation in Python are summarized below

- Load the cv2 library into python.
- From the cvlib.object_detection library import draw_bbox for drawing the boundary boxes.
- Use the cv.detect_face function on the image.
- Create a for loop that goes over the image and places the label of person.
- Use the draw_bbox function to draw the bounding boxs on the image and save it in a new image.
- Use plt.imshow function to display the final image with object detection.

The final image from this calculation is in Figure II.14.

Figure II.14: Object Detection Example

The example presented in this section presents the basics of how Object detection works in python. Moreover, object detection in Python can be performed using various libraries such as OpenCV, TensorFlow, and YOLO. The process typically involves feeding an image or video into the model, which then outputs the bounding boxes and labels of the detected objects. The model is trained on a large dataset of annotated images to learn the features that distinguish the objects of interest. Once trained, the model can be used for a variety of tasks such as object recognition, tracking, and counting. In the next section a basic example of YOLO will be presented by using the training weights and is the final example presented in chapter 2.

7 YOLO

YOLO (You Only Look Once) is a popular real-time object detection system. It is a single convolutional neural network that can detect objects in an image and predict their bounding boxes. YOLO divides an image into a grid of cells, and for each cell, it predicts the probability of the presence of an object and the coordinates of the bounding box for that object. This allows YOLO to be faster than other object detection methods, such as R-CNN and its variants. Other object detection systems need to process each region separately. YOLO also uses anchor boxes,

a set of predefined bounding boxes, to handle objects of different aspect ratios. YOLO has been updated in its versions v2, v3, and v4 to improve its accuracy and speed, and it's used in many real-world applications, such as self-driving cars, surveillance, and facial detection.

The final part of this section will present the basics of YOLO and Darknet by using it to detect objects from an image of Eau-Claire. These libraries come with pre-trained weights and can be used to detect objects, place a bounding box, and print a name string by the bounding box. The following calculation will use YOLO object detection on an image of Eau Claire students walking on campus in Figure II.15.

Figure II.15: Object Detection of University of Wisconsin Eau Claire

The code that is described in this section can be found in the Capstone Code addendum [8]. Furthermore, the methodology and code will be used to detect the different types of strokes after the custom weights are calculated. This along with many other different types of algorithms will be calculated in this Capstone. To use YOLO to detect an object, you need to follow these steps:

1. Install the YOLO library, which includes the pre-trained weights for the CNN.
2. Load the image or video stream that you want to detect objects in.
3. Run the YOLO algorithm on the image or video stream, using the pre-trained weights.
4. The algorithm will output bounding boxes around the detected objects, along with the class labels and confidence scores.
5. You can then display the image or video stream with the bounding boxes and labels overlaid on top.

6. You can also fine-tune the pre-trained YOLO model on a new dataset to improve its accuracy for specific objects.

The steps in the previous enumeration for YOLO are an overview of what it takes to run the pre-trained algorithms. The code that was used in this Capstone can be found at the end of the Coding Addendum [8]. The final output of the algorithm is the image in Figure II.16.

Figure II.16: Object Detection of University of Wisconsin Eau Claire

---

The object detection in Figure IV.2 picked up many people, backpacks, a clock, and a cell phone. This is just an introductory calculation. There is much more work to be done and this is just a sample.

8 Literature Review Summary

Object detection research started in the seventies. The research originally used template-matching techniques, Fichler and Elschlager 1973 [14]. Neural networks were not introduced until the late nineties in publications similar to Rowley, H. A. et al. (1998) [15]. From this initial research, many different applications have come about since the nineties. Most notable is facial recognition and this technology was shown in the Basic Object Detection section with the OpenCV library. Most object detection systems use a schema known as a sliding window. This schema uses a classifier to search over and over again for different objects. There are other schemas like bag-of-words (Weinland et al., 2011; Tsai, 2011) [16] and sampling patches and finding key points. Each schema has its own pros and cons and is used in different approaches to object detection.

The first approach that is most pertinent to this Capstone is Deep Learning. The first successful application of convolutional neural networks (CNN) was (Delakis and Garcia, 2004) [17]. In [17] the authors used CNNs on images of faces and found a high detection rate with a low amount of false positives. In this Capstone this is the main method that will be used to calculate the algorithms with TensorFlow, PyTorch, and then finally with the YOLO object detection system. The benefits of using Deep Learning are that the feature engineering is learned but the drawback is that there needs to be a large number of images for it to work correctly. There are other notable approaches to object detection. The Dictionary approach uses the bag of words schema and is used to detect a single object per image (Lampert et al. 2009) [18]. Another notable approach is Boosted Classifiers. This approach uses boosted cascade classifiers (Viola and Jones 2002) [19]. Boosting allows the complexity to be controlled at each stage making it useful for efficiency.

There are many different current research problems going on with object detection. The most pertinent research is to increase efficiency and reduce computational power. Again using Boosted Classifiers is the best-known way to achieve efficiency. Although, it should be noted

that efficiency does not mean that the object detection system will have real-time performance (Felzenszwalb et al. 2010b) [20]. For real-time performance, Deep Learning has the best choice. Another problem with object detection is when objects are combined with background pixels. One example of this is pedestrian detection Dollar et al. (2012) [21].

Computer Vision and Object Detection have greatly impacted the medical field. There have been generally four areas of publication. The first is using multiple imaging techniques to analyze a medical image. One example of this is in Monti et al. (2017) [22] where positron emission tomography (PET) imaging is combined with magnetic resonance imaging to achieve better results in complex regions like the head and neck. Furthermore, feature extraction is a difficult task for medical images. In Esener et al. (2017) [23] they developed a multistage classification for computer-aided diagnosis (CAD) system for breast cancer diagnosis. Both of these papers give great examples of how computer vision is used in the medical field for the first category of analyzing medical images.

The second category of computer vision in the medical field is using predictive analytics. An interesting example of this is CT and MRI. P. Gargiulo et al. (2017) [24] where the authors develop methods of using image segmentation to model human skulls in three dimensions and then 3D print this model to assist in neurosurgery. This is an amazing example of computer vision in the medical field and uses many diverse technologies. Furthermore, there is also human activity recognition (HAR) is a permanent computer vision study. HAR is a basis of video surveillance, health care, and human-computer interaction (HCI). A great review of HAR can be found in S. Zhang et al. (2017) [25].

There are many different algorithms that are used for medical images. A couple of notable algorithms are segmentation algorithms used for organ segmentation and random walkers (RW). For organ segmentation C. Pan et al. (2018) [26] use segmentation algorithms to enhance disease prediction and therapy. Moreover, in C. Dong et al. (2017) [27] use a random walkers algorithm and Bayes model for the segmentation of medical images. Although, for the purposes of this Capstone the most interesting algorithms are machine learning algorithms. A great article that

uses CNNs and deep neural networks (DNN) is by Q. Song et al. (2017) [28]. In [28] the authors use deep learning algorithms to classify lung nodules for early diagnosis of lung cancer. Although, there are other great examples of machine learning algorithms used in medical images. In N. D. Kamarudin et al. (2017) [29] the authors use a support vector machine (SVM) learning algorithm and k-means clustering to detect the color of people's tongues which is used in the early detection of imbalances in the body. So, overall there is a plethora of works using computer vision in medical images and within that this field, there is a large subfield of object detection. This is the field most pertinent to this Capstone.

The most commonly known use of Deep Learning algorithms, CNNs, in medical images is in terms of classification. Although, there is a rich amount of academic work on using object detection and image segmentation in medical images. The popular algorithms that are used in object detection are R-CNN, Fast and Faster R-CNN, PFN, PSPNet, SSD, YOLO, CenterNet, and EfficientNet: Liu W et al. (2016) [30]. Object detection has two types of frameworks: one and two-state detection. One-stage frameworks integrate the process carried out in two-step in one. The two-step framework starts with a CNN to extract the features of regions and then uses a category-specific classifier to determine the category labels. The framework used in this Capstone will be You-Only-Look-Once or the YOLO framework which is a one-stage framework.

The R-CNN or region of CNN was a two-stage framework and was developed based on AlexNet. This framework extracts the regions and computes the CNN features, Grishick R et al. (2014) [31]. The Fast R-CNN improved the speed of detection and quality. Then in the Faster R-CNN framework, a feature pyramid model was introduced by Lin et al. (2016) [32]. Although, even with all of the improvements of Faster R-CNN the two-stage frameworks still required too much computational power. The YOLO framework was developed to overcome the problems with the R-CNN models. This framework has 24 convolution layers and two fully connected layers and originates from GoogleLeNet. For the YOLO framework, the inception structure was replaced with a 1 x 1 and 3 x 3 convolution layer Zhuang Z et al.(2020) [33], Redmon J et al. (2016) [34].

Finally, there are many different uses for object detection in the medical field. One of these uses is in Endoscopy where physicians may miss some lesions that could be detected by real-time object detection Hirasawa T et al. (2018) [35]. Another application of object detection in the medical field is detecting exudates in the retina of diabetic patients Cao W. et al. (2018) [36], Li X. et al. (2021) [37]. So, there are many interesting cases where object detection is used for medical images. In this Capstone an object detection system is developed as an example of how to use the technology. Although, maybe there is some sort of real-time analysis that could be applied to the resulting algorithm.

In conclusion computer vision, Deep Learning, and object detection have many interesting applications in the medical field. This Capstone is meant to be a basic application of this technology. Finally, a great review of the different metrics and formulas used in this research is contained in the paper by Shou Y et al. (2022) [38] where the authors introduce a new algorithmic model MS Transformer. That is the only place where the mAP value is described in any of the literature well.

Chapter III

Methodology

Table Of Contents

Chapter 3 will present the methodology of how to calculate classification algorithms with Tensorflow and PyTorch. Moreover, this chapter will describe how the object detection system will be calculated. These calculations will give technologies that are useful to to stakeholders mentioned in the introduction. Furthermore, the classification algorithm methodology gives a great introduction to how the object detection system uses Deep Learning to classify different objects.

1 Stroke Image Pre-Processing Methodology

Table Of Contents

The data is high-resolution stroke images from the Mayo-Clinic. These images range from half a gigabyte to five gigabytes in memory. Since these are to large to process with any normal computer: so, computer vision will be used to reduce the memory size of the images such that the

aspect ratio is preserved along with the dominant features of the stroke. There are many different methods on Kaggle under the Mayo-Clinic competition [3]. But I will use the most popular one which initially gave the best results with the deep learning algorithms. The following Figure is the result after the memory is reduced. The high-resolution images could not be put into a Latex document. A memory reduced image of the stroke is shown below in III.1.

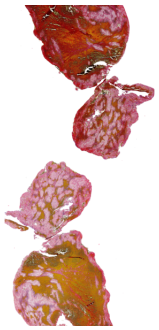


Figure III.1: Memory Reduced Stroke Image

The methodology previously mentioned is of the calculation done by Mulla (2022) [39]. This code will be reviewed and used to produce the final images that will be used in the deep learning algorithms and custom YOLO training. The result of this calculation will be used in the rest of this Capstone as the primary data. I attempted a similar calculation based on different techniques posted on Kaggle to reduce the size of the high-resolution stroke images. Although, those calculations yielded results that were not as good as Mulla's.

First, the author [39] loads the libraries that are required. The following libraries were used to reduce the image memory.

- PIL is the Python Imaging Library.
- Skimage is an image processing package that has algorithms that are used to process images.

- Glob, OS, JSON, cv2, gc, shutil: These libraries work with directories and files, read and write JSON files, images processing, garbarge collection, and deleting files and directories.
- TensorFlow is an open-source library that is used to train deep learning algorithms.
- Keras is a high-level API for neural networks that runs on top of Tensorflow.
- Tqdm is a library that provides a progress bar for iterators to show the progress of a task in Jupyter notebooks.

The next steps are to create different folders to put the final images into and create a dataframe with the paths to the images. The next part of the script in [33] defines six functions that use a mask to reduce the total memory of the images. The first function opens the tiff file.

The next function reads the mask in an image and returns it. The following list enumerates the pertinent parts of the script.

- Read image from image_path using io.imread()
- Remove unnecessary axes with shape 1 using np.squeeze()
- If the first axis of the image has shape 3 (indicating a color image), swap the first and second axes using image.swapaxes(0,1) and the second and third axes using image.swapaxes(1,2)
- Return the processed image.

Finally, the function that produces the mask is summarized by the list below.

- The function slice_images takes four arguments: image_id, image, mask and folder.
- The function prints a message indicating that it is slicing the image with the given image_id.
- The number of possible slices in the x-direction is calculated by dividing the width of the image by IMG_SIZE.

- The number of possible slices in the y-direction is calculated by dividing the height of the image by IMG_SIZE.

- A nested loop is performed over the possible slices in x and y directions.

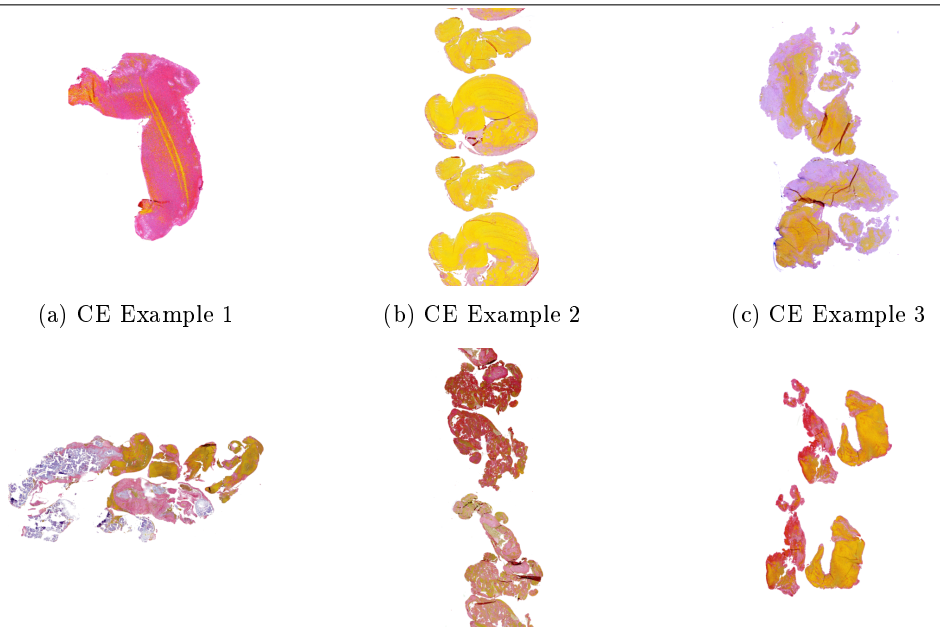
- In each iteration of the loop, an image slice is created by slicing the original image along the x and y-axis. The slice size is defined by IMG_SIZE.

The overall effect of this script described in this section is to reduce the memory of the images to a reasonable amount. These images are the primary data for this Capstone. This is the best way that I could find to reduce the memory of high-resolution images although there are many other methods posted on Kaggle. The images produced from this script produced the best metrics when the deep learning algorithms used them so I will use this data for the rest of the Capstone. Six images run through this script are displayed in Figure III.2. Comparing them to the original high-resolution images there is less detail although the aspect ratio of the images is preseved and the final set of images can be used in deep learning algorithms.

2 Classification with Tensorflow

Table Of Contents

This section will present the methodology of three different calculations using neural networks. The first neural network is a baseline model that will be used for the second calculation which is optimizing that neural network. The final neural network will use transfer learning with the VGG16 network. The calculations follow [40] Nelson, A. and can be found in the Capstone Coding Addendum [8].



(a) CE Example 1 (b) CE Example 2 (c) CE Example 3

(d) LAA Example 1 (e) LAA Example 2 (f) LAA Example 3

Figure III.2: Reduced Memory Stroke Image Data Examples

(a) Stroke Image Classification Baseline Model

TensorFlow is an open-source software library for machine learning that was developed by Google [41]. It provides a platform for creating and training machine learning models. TensorFlow can be used on a wide range of devices, for this Capstone it will be used on a laptop in three different calculations.

The calculation TensorFlow does is a computational graph. This graph is a series of operations on tensors that train algorithms. TensorFlow is a high-level API for building machine learning models and is also has a customizable low-level API. TensorFlow also includes a number of pre-built models for tasks such as image classification, natural language processing, and reinforcement learning, which can be used as a starting point for building custom models. Additionally, TensorFlow has a large and active community of users and developers who contribute to the library and provide support and resources for users.

This section aims to present the methodology for calculating the baseline classification algorithm. The final result will be an algorithm that could be used by the Mayo Clinic and other medical technology companies to classify strokes. This section also serves as an introduction to CNN's that will be used to calculate the YOLO object detection algorithm.

The first task to train a CNN algorithm is to load the images and Keras/Tensorflow uses the class ImageDataGenerator to load the data set. First the different types of strokes need to be put into a folder the script that was used can be found in the coding addendum [8]. It is summarized by the following list

- Open the 'classes_types.csv' file in read mode.
- Read the file line by line using the csv reader.
- For each line, retrieve the value of the first and fifth columns, which are stored in the variables 'name' and 'type' respectively.
- If the 'type' value is equal to 'CE', attempt to rename the file 'train1/'+ name +'.png' to 'train1/CE/' + name+'.png'.
- If the rename operation is successful, print a message indicating that the file has been moved to the CE folder.
- If the file is not found, do nothing.
- If the 'type' value is not equal to 'CE', attempt to rename the file 'train1/'+ name +'.png' to 'train1/LAA/' + name+'.png'.
- If the rename operation is successful, print a message indicating that the file has been moved to the LAA folder.
- If the file is not found, do nothing.

The deep learning algorithm that will be used will have three different convolutional parts and use the Relu and Softmax activation function

$$f_{Relu}(x) = max(0, x),$$

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\Sigma_{j=1}^{K} e^{z_j}}.$$

Furthermore, the loss function for this neural network is given by the binary cross-entropy function

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i)),$$

where y is the true value, $p(y_i)$ is the predicted value, and N is the number of samples. The binary cross-entropy is a loss function is for binary classification problems. It averages the loss over all N samples and measures the dissimilarity between the predicted probabilities and the true labels in a binary classification problem. Dividing by N normalizes the function and makes the function a scalar that is independent of the size of the dataset. It also makes the optimization process stable and allows the loss function to be able to be compared between different datasets and models.

Adam (Adaptive Moment Estimation) is a popular optimization algorithm used in deep learning for training neural networks. It is an extension of the stochastic gradient descent (SGD) algorithm, which computes gradients for each training sample and updates the model parameters after each sample.

Adam tries to address some of the limitations of SGD by using moving averages of the gradient and the squared gradient to provide a running estimate of the first and second moments of the gradients. These moments are used to adapt the learning rate on a per-parameter basis,

allowing Adam to dynamically balance the trade-off between exploration and exploitation during training.

The optimizer used is the Adam algorithm which updates the weights with the following formula

$$w_t = w_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon},$$

where t is the iteration step, η is a hyperparameter that controls the step size of the update, m_t and v_t are the estimated first and second moments of the gradients, epsilon is a small constant added to the denominator to prevent division by zero.

Adam requires little tuning of the learning rate hyperparameter, and it works well for many different problems. However, it is important to note that like all optimization algorithms, the performance of Adam can be sensitive to the choice of hyperparameters, such as β_1, β_2, ϵ.

The neural network that will be used as a baseline has the following architecture:

- It starts with an input layer of shape (100, 100, 3) which corresponds to a 100x100 image with 3 color channels (RGB).
- A convolutional layer with 32 filters of size (3,3), using the ReLU activation function.
- A pooling layer with a pool size of (2,2) which performs down-sampling on the feature maps generated by the Conv2D layer.
- Then there are three more sets of two layers, each having 64, 128, and 256 filters in the Conv2D layers respectively, with the same structure as the previous set.
- The output of the last MaxPooling2D layer is flattened and passed to two fully connected Dense layers with 1024 and 2 neurons, respectively, using the ReLU and softmax activation functions.
- The final layer with 2 neurons will output the class probabilities.

The results of this model are enumerated in Results chapter 5. This is the baseline model and it there are many different techniques that can be used to increase the differtent metrics. The baseline model can be optimized and many different parameters can be changed or different sturcture of the neural network can be used to make the model better. Finnaly, advanced pre-trained neural networks can be used in transfer learning to find different models that might have more appealing characteristics.

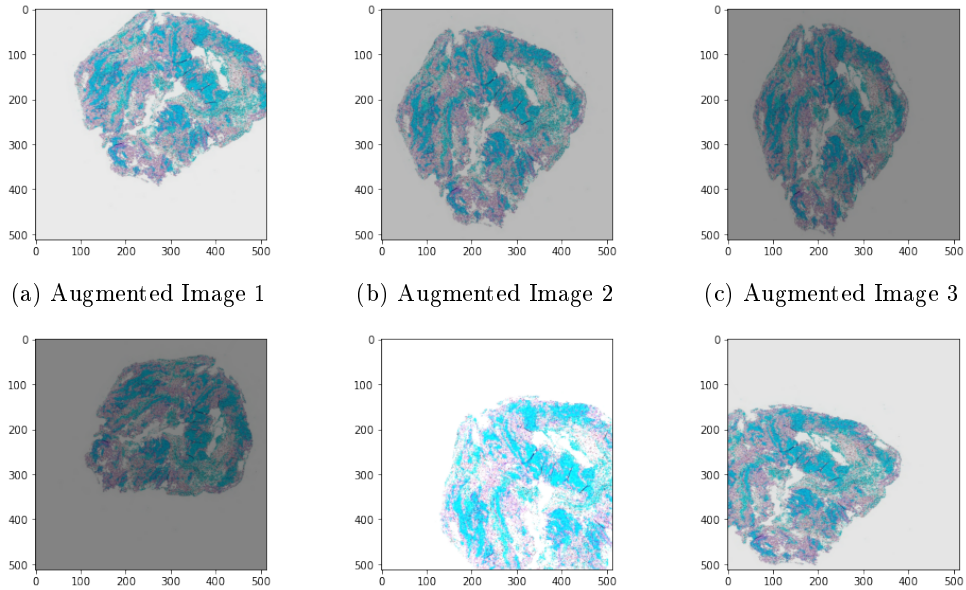
(b) Stroke Image Classification Optimization

For the first try at optimization, I tried to use augmentation. The reasoning behind this is that the data set is small and maybe making more images with augmentation will help the accuracy and overfitting.

The code defines two instances of the TensorFlow ImageDataGenerator class, named "train_datagen" and "test_datagen". This class is used for data augmentation and preprocessing of image data. The different ways the image is augmented is listed below.

- Both instances use the following augmentation techniques:
- Rescale: The pixel values of the images are divided by 255, which normalizes the values between 0 and 1.
- Horizontal Flip: The images are randomly flipped horizontally with a 50
- Rotation Range: The images are randomly rotated by an angle within the range of -20 to 20 degrees.
- Width Shift Range and Height Shift Range: The images are randomly shifted horizontally and vertically by a fraction of their width and height, respectively.

  The use of image augmentation helps to increase the diversity of the training data and reduce overfitting. The image augmentation can be seen in the following Figure III.3.



(a) Augmented Image 1 (b) Augmented Image 2 (c) Augmented Image 3

(d) Augmented Image 4 (e) Augmented Image 5 (f) Augmented Image 6

Figure III.3: Sample of Augmented Images

Next the augmented images will be used to train the baseline algorithm that was described in the previous section. There are many other optimization techniques. A concise summary of these are given below.

- Image Augmentation: This technique involves transforming the original images in the training dataset by applying operations such as flipping, rotating, cropping, scaling, and adding noise. This increases the size of the training dataset and helps the model to generalize better to new images.
- Early Stopping: This technique involves monitoring the validation loss during training and stopping the training process if the validation loss has not improved for a certain number of epochs. This helps to prevent overfitting, where the model starts to memorize the training data instead of generalizing to new data.

- Regularization: This involves adding a penalty term to the loss function to discourage the model from learning overly complex representations. Common regularization techniques include L1 and L2 regularization, Dropout, and Batch Normalization.

- Transfer Learning: This involves using a pre-trained neural network as a starting point for a new task, instead of training a new model from scratch. This can be useful when there is limited data available for the new task, as the pre-trained network has already learned useful features from a similar task.

- Hyperparameter tuning: This involves adjusting the hyperparameters of the model, such as the learning rate, number of hidden units, batch size, etc., to find the best combination of hyperparameters that lead to good performance on the validation set.

The model will use the same neural network methodology as the baseline model presented in the previous section. A great review of image augmentation in deep learning can be found in Yang, S. [42]. In this article the author sytematically goes over many different techniques than the one used in this Capstone.

(c) Stroke Image Classification Transfer Learning

This section will present an initial calculation using Transfer Learning. The neural network that will be used is the VGG16 model. This model was originally presented in the article Y. Lecun et al [43]. Although, there is a rich amount of different algorithms that could be used to do transfer learning. A consise list of a couple is given below.

- ConvNet: The ConvNet, or convolutional neural network, was first introduced in the 1980s by Kunihiko Fukushima, but it was not widely used until the 2010s. The breakthrough paper that popularized the use of ConvNets for image recognition was "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012.

- LeNet: LeNet, a deep learning architecture for handwritten character recognition, was developed in the early 1990s by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. It was one of the first successful applications of ConvNets and was widely used in the banking industry for reading checks.

- VGG: The VGG (Visual Geometry Group) network was developed in 2014 by the Visual Geometry Group at the University of Oxford. The VGG architecture is characterized by a very deep network of small 3x3 convolutional filters, which is thought to enable better feature extraction from images. The VGG network achieved top performance on the ImageNet dataset in 2014 and has been widely used as a starting point for transfer learning in computer vision tasks.

- EfficientNet: EfficientNet is a relatively new deep learning architecture that was developed in 2019 by a team of researchers from Google. The architecture is characterized by a novel compound scaling method that optimizes the network for both accuracy and efficiency. EfficientNet has achieved state-of-the-art performance on a variety of image recognition benchmarks, while using fewer parameters and less computation than many other architectures.

A graph of the evolution of transfer learning algorithms from [43] Jha, A. R. et al is shown in figure III.4

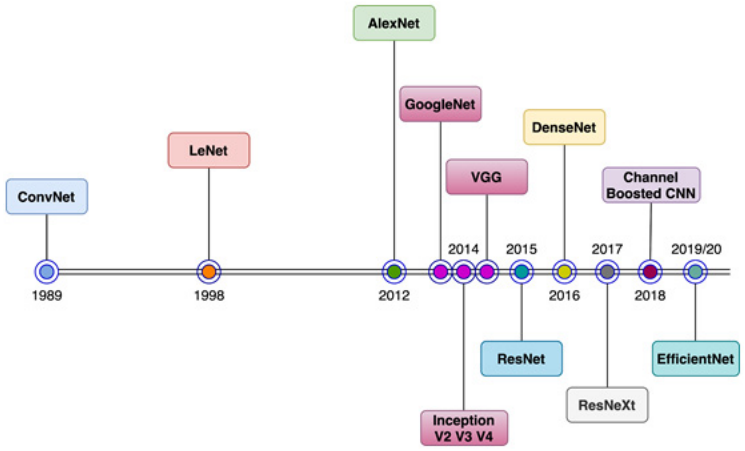


Figure III.4: Tensorflow Optimization Accuracy

VGG16 is a CNN that is used for Deep Learning in images. This is the most complicated CNN used in this capstone and it can be found in [44] Simonyan, K.. The hope is that this complicated neural network can give the best model out of all the different ones attempted in this Capstone.

The architecture of VGG16 consists of 13 convolutional layers and 3 fully connected layers. The convolutional layers are organized in groups of 2-3 layers, each of which is followed by a max-pooling layer. The convolutional layers use small filters (3x3) with a stride of 1. The pooling layers reduce the spatial dimensions of the feature maps, while retaining the important features. VGG16 is shown in Figure III.5 and this visualization comes from HarisIqbal88 [45]

The first five convolutional blocks of VGG16 are designed to learn low-level features like edges and textures. The last eight blocks are designed to learn high-level features like object classes. The last three layers are fully connected and have 4096, 4096 and 1000 neurons respectively. The last fully connected layer produces the final output of the network with 1000 neurons that correspond

to the number of classes, the CNN is trained to recognize.

VGG16 is well known for have great accuracy but it requires difficult computation. It is

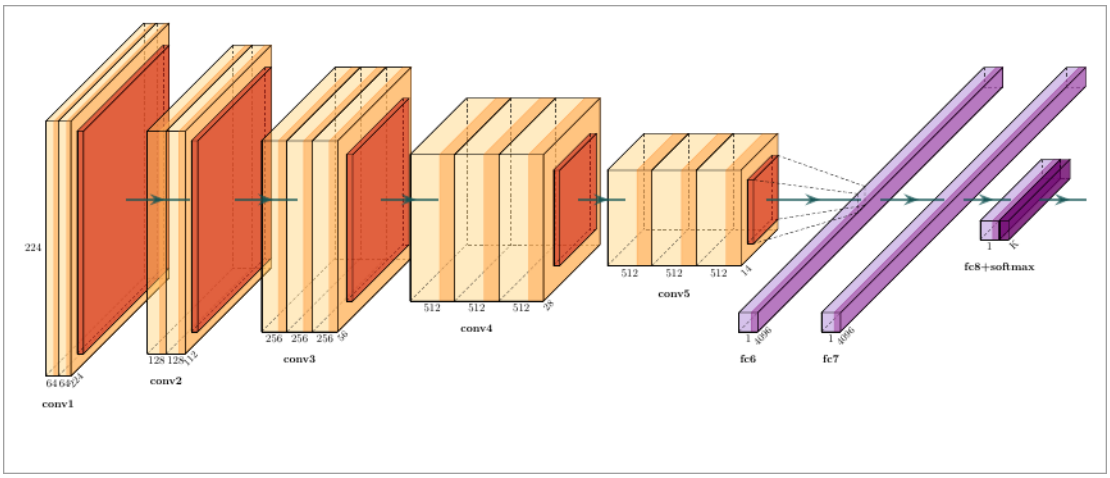


Figure III.5: VGG16 Neural Network

often used as a feature extractor in many other computer vision tasks like object detection. The implementation of VGG16 is summarized below.

- Imports necessary libraries, including TensorFlow, Keras, and VGG16.
- Sets up an image data generator to preprocess and augment training and testing images.
- Loads the pre-trained VGG16 model, sets its layers to be untrainable, adds a new fully connected layer with 512 hidden units and a sigmoid activation function, and combines these layers to create a new model.
- Summarizes the new model and prepares training and testing data iterators.
- Compiles the model using binary crossentropy loss, accuracy metrics, and the Adam optimizer.
- Fits the compiled model to the training data, using the validation data for testing, over 2 epochs, and saves the training history.

3 Classification with PyTorch

Table Of Contents

PyTorch is an open-source software based on the Torch library [46] PyTorch. This library is used for deep learning, computer vision, and natural language processing. It was developed by Facebook's AI research group and has become one of the most popular deep learning softwares. PyTorch provides a high-level interface for building and training machine learning models. It also can use GPUs, making it suitable for large data sets and computationally-intensive calculations. This section follows the calculations in [9] A.I. Sciences.

In this section, PyTorch will be used to calculate the classification models that are similar to the ones calculated in the previous TensorFlow section. The purpose of this section is to present the methodology used to calculate classification algorithms in PyTorch and compare this to TensorFlow. The first neural network that will be calculated will use transfer learning to implement LeNet: a CNN used for image classification. Then this neural network will be optimized using image augmentation. Finally, VGG16 will be used to calculate a model

(a) Stroke Image Classification LeNet

LeNet is a Convolutional Neural Network that was introduced in the late 1990s by LeCun et al [47]. It was one of the first deep learning models to be successful in recognizing handwritten digits from the MNIST dataset, a benchmark for image classification tasks.

LeNet is a feedforward network, which means that the data flows through the network in one direction, from input to output. It consists of several layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers in LeNet are designed to extract features from the input image. In this CNN filters slide over the input image and compute the dot product between their weights

and the overlapping pixels. This makes feature maps similar to what was discussed in Chapter 2. These maps capture various aspects of the input image such as edges, corners, and textures.

The pooling layers in LeNet are used to reduce the spatial dimensions of the feature maps while retaining the most important information. This process is called subsampling, and it helps to reduce computational cost and mitigate overfitting.

The fully connected layers in LeNet perform the final classification task. They receive the output from the final pooling layer and use it to compute the probability of the input image belonging to each of the possible classes. The class with the highest probability is chosen as the final prediction.

Figure III.6 show a visualization of LeNet from [45] HarisIqbal88.

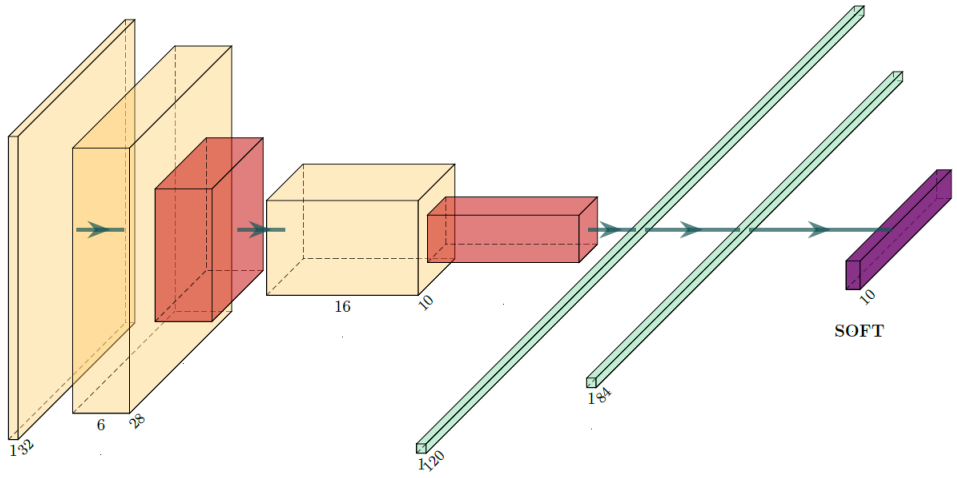


Figure III.6: LAA Labeled Stroke Image

The libraries used in this calculation are listed below

- torch: PyTorch library for deep learning, providing functionality for creating and training neural networks.
- matplotlib.pyplot: A plotting library in Python used for creating visualizations, like line charts, bar charts, and scatter plots.
- numpy: A library for numerical computations, providing support for arrays, linear algebra, and random number generation.
- torch.nn.functional: A module in PyTorch that contains a large number of neural network functions, such as activation functions and loss functions.

- torch.nn: A module in PyTorch that contains the basic neural network functions and modules, such as linear layers and convolutional layers.
- torchvision.datasets: A module in PyTorchVision that contains popular datasets for computer vision tasks, such as MNIST and CIFAR.
- torchvision.transforms: A module in PyTorchVision that contains image and video processing functions, such as cropping, resizing, and normalizing.
- torchvision.models: A module in PyTorchVision that contains pre-trained models for computer vision tasks, such as image classification and object detection

The next itemization walks thru step by step implementation of PyTorch LeNet. This network will be our baseline model.

Initialize the number of epochs, training and validation loss and accuracy histories.

- Loop through the number of epochs:
  - Initialize the running loss and running accuracy for the training set.
  - Loop through the training data using the training_loader:
    * Transfer the inputs and labels to the device.
    * Pass the inputs through the model to get the outputs.
    * Calculate the loss using the criterion function.
    * Backpropagate the gradients and update the model parameters using the optimizer.
    * Update the running loss and running accuracy by summing the current loss and accuracy.
  - Loop through the validation data using the validation_loader:
    * Transfer the validation inputs and labels to the device.
    * Pass the validation inputs through the model to get the validation outputs.

    * Calculate the validation loss using the criterion function.
    * Update the validation running loss and validation running accuracy by summing the current validation loss and accuracy.
- Calculate the epoch loss and accuracy by dividing the running loss and running accuracy by the number of training samples.
- Append the epoch loss and accuracy to the training loss and accuracy histories.
- Calculate the validation epoch loss and accuracy by dividing the validation running loss and validation running accuracy by the number of validation samples.
- Append the validation epoch loss and accuracy to the validation loss and accuracy histories.
- Print the epoch number, training loss and accuracy, and validation loss and accuracy.

(b) Stroke Image Classification Optimization

In this section PyTorch's image augmentation methodology will be presented. Image augmentation is a technique used to increase the size of a training dataset by creating modified versions of existing images. PyTorch provides several functions to perform image augmentation, including random cropping, horizontal flipping, rotation, and color jittering. These functions are used during training to improve the generalization of a deep learning model. PyTorch also supports the creation of custom image augmentation functions, allowing for greater flexibility in the types of modifications that can be made to images.

To optimize the LeNet transfer learning algorithm in the last section the following transformations are performed on the training set.

- Resize((32,32)): Resizes the image to a 32x32 size.
- RandomHorizontalFlip(): Flips the image horizontally with a 50% probability.
- RandomRotation(10): Rotates the image by a random angle between -10 and 10 degrees.
- RandomAffine(0, shear=10, scale=(0.8,1.2)): Applies a random affine transformation with a maximum shear of 10 degrees and a random scaling factor between 0.8 and 1.2.
- ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2): Adjusts the brightness, contrast, and saturation of the image randomly.
- .ToTensor(): Converts the image to a PyTorch tensor.
- Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)): Normalizes the tensor by subtracting the mean of 0.5 and dividing by the standard deviation of 0.5 for each color channel.

(c) Stroke Image Classification Transfer Learning

This section will run the previous section's code with VGG16. The model that is produced will be compared to the one calculated by Tensorflow. In PyTorch, we can load VGG16 with the nn.Module class. The layers of the network are defined in the __init__ method. The features layer is a sequential block of convolutional layers, followed by ReLU activations and max pooling layers. The classifier layer consists of fully connected layers, with dropout regularization applied between the two hidden layers. The last layer in the classifier block is a linear layer with the number of output classes as the output size.

The input to the VGG16 network is an image with 3 color channels. The first layer of the features block is a convolutional layer with 64 filters, each of size 3x3, and a padding of 1. This layer applies 64 filters to the input image, which can be thought of as 64 "feature maps" or 2D arrays of numbers that represent different aspects of the image. The output of this layer is then passed through a ReLU activation function, which applies an element-wise non-linearity to the output.

This is followed by another convolutional layer with 64 filters and a ReLU activation function, and then a max pooling layer with a kernel size of 2x2 and a stride of 2. This reduces the size of the feature maps by a factor of 2. The network then applies a similar pattern of convolutional and max pooling layers to further downsample the feature maps and extract more abstract features.

Once the features have been extracted using the features block, they are passed through an AdaptiveAvgPool2d layer to get a fixed size output. This layer computes the average of the input feature maps, which results in a fixed-size representation of the input image regardless of its original size.

Finally, the fixed-size feature map is flattened into a 1D vector using the flatten function, and passed through the classifier block. This consists of three fully connected layers, with ReLU activations and dropout regularization applied to the hidden layers. The last layer is a linear layer with the number of output classes as the output size. The output of this layer represents

the class probabilities of the input image

The results of this calculation are going to be discussed in the next chapter: Results. Moreover, the code that was described in this section can be found in the Capstone Code Addendum [8].

4 Custom YOLO Object Detection Calculation

Table Of Contents

In this section, a custom YOLO model will be calculated and tested. The final code will be similar to the pre-trained script in chapter IV. Where the two classes will be the two types of strokes: CE (Cardioembolic) and LAA (Large Artery Atherosclerosis).

(a) Custom YOLO Object Detection Introduction

Table Of Contents

Before compiling the object detection system Darknet has to be set up and tested. To compile Darknet the make file needs to be executed. Then Darknet can be tested on a sample image that comes with it to assess whether it is working or not.

The next step to building an object detection system is to label the images. This can be done by hand for all the images with the heartexlabs GitHub [48]. For example, each image can be classified into each stroke class by drawing a box around the pertinent parts of the image. First, all of the CE stokes are labeled in Figure III.7.

Figure III.7: CE Labeled Stroke Image

Next, the LAA strokes are labeled in Figure III.7. The image below shows an example of an LAA stroke labeled by labeling.

After labeling all of the images there is a text document that is used by the YOLO algorithm to create the system. Next, the YOLO configuration file in Darknet needs to be changed such that it will compile for our specific case. There are two classes and this needs to be changed in the configuration file. Moreover, the number of filters needs to be changed based on the number of cases. There are $3(N_{classes} + 5)$ filters, where $N_{classes}$ is the number of classes. For our particular problem there are two classes, so there are twenty-one filters. This is changed in each section of the CNN before the YOLO layer's definition. Moreover, the maximum batches are set to four thousand. This will take ten hours or so to train using Google Colab. This section follows the calculations in [49] Nelson A.

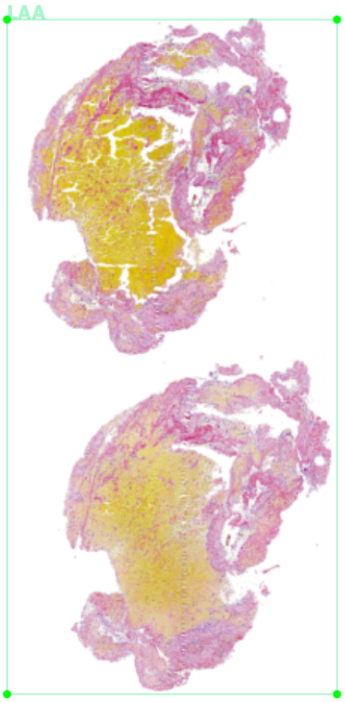


Figure III.8: LAA Labeled Stroke Image

(b) Training the Weights

Table Of Contents

To train the weights a powerful graphics card is required. So, the model will be trained on Google Colab because they offer expensive GPUs for free. Training the model takes about half a day. It will be interesting to compare the YOLO's accuracy and loss to the loss calculated in the previous sections. The following methodolgy will be followed to train the custom weights.

- Data preparation: You need to split the collected data into training, validation, and testing sets. You may also need to perform data augmentation techniques like flipping, rotation,

and scaling to increase the diversity of the training data.

- Configuration file: You need to create a configuration file in Darknet format that specifies the model architecture, the training parameters, and the paths to the training data and the trained weights.
- Model training: You can start the training process by running the Darknet command line with the configuration file. Darknet will automatically update the model weights using the backpropagation algorithm based on the annotated data.
- Model evaluation: During the training process, you can use Darknet to evaluate the performance of the model on the validation set and monitor the training progress.
- Model fine-tuning: Based on the performance on the validation set, you may need to fine-tune the model by adjusting the hyperparameters or changing the architecture in the configuration file.
- Model testing: After training the model, you need to evaluate the performance on the test set and save the weights of the final model.

Chapter IV

Results

Table Of Contents

In this section the results of the neural networks trained using Tensorflow, Pytorch, and YOLO will be presented. The code for chapters three and four is combined in the Capstone Code Addendum [8].

1 Overview

The final section will present the results of the calculation whose methodology in chapter three.

2 Image Classification Results

The results of the algorithms that were calculated with the methodology in chapter three section b yielded the baseline algorithm. There are generally a couple of different parameters that can be analyzed when trying to assess the validity of a deep learning algorithm. The results of the deep

learning algorithm in TensorFlow could be affected by the quality of the data. Our images were generated using the techniques presented in chapter 3 section a, so there could be much better ways to produce the images with less memory. Other techniques could yield different features that would have higher accuracy. Also, the number of training samples could be changed along with the choice of optimization algorithm, and the values of the hyperparameters could also be altered. The list below summarizes all of the parameters that could be analyzed from the resulting algorithm.

- The training loss measures the difference between the predicted and actual values in the traning data. The training loss should get smaller over time because the model is learning the data.

- The validation loss measures the difference between the predicted and actual values for the validation data. Furthermore, the validation loss is used as a measure of the generalization performance of the model. If the validation loss is larger than the training loss: the model might have overfitting.

- The accuracy applys the model to the test set to measure if they are classified correctly. The accuracy is a measure of the model's generalization.

- A confusion matrix is a table that has the number of true positive and negative predictions compared to false positive and negative predictions. The confusion matrix is used to calculate precision, recall, and other performance metrics.

- There is also the precision and recall metrics. The precision uses the confusion matrix to compute the proportion of true positive predictions relative to all the positive predictions $Precision = \frac{tp}{tp+fp}$. Recall measures the proportion of true positive predictions relative to all true predictions $Recall = \frac{tp}{tp+fn}$.

- The ROC (Receiver Operating Characteristic) curve is a plot of the true positive rate against the false positive rate for different thresholds.

For the loss graphs there are four different things to look for in the graphs

- The training loss and validation loss should decrease over time. If the training loss decreases but the validation loss starts to increase, it may be a sign of overfitting. On the other hand, if both the training and validation loss fail to decrease over time, the model may be underfitting.
- The loss should plateau after some time. If the loss continues to decrease over time, it may be a sign of overfitting, as the model is becoming too specialized to the training set.
- The magnitude of the loss can be difficult to interpret, as it will depend on the specific problem and the scale of the data. It is more important to focus on the trend of the loss over time.
- The shape of the loss graph may be different depending on the specific problem and the model architecture. Some problems may require longer training times to achieve good performance, while others may see rapid improvement at the beginning of training.

| The Baseline models described in chapter 3 from Tensorflow and Pytorch have their loss displayed in Figure IV.1 and Figure IV.2. The Tensorflow model has an ideal loss graph. Both the training and test set losses are matched so there isn't any underfitting or overfitting. It plateaus after one epoch and it's trend is one that is ideal for loss. This graph of the loss achieves good performance at the beggining of training. The Baseline model used in the first model of Pytorch does not have the greatest loss graph. The validation loss starts to increase at two epochs while the training loss stays low which shows that this model is overtrained. Niether of the losses have a plataeu and the trend of the loss is increasing. So, this model needs its hyper-parameters to be adjusted.

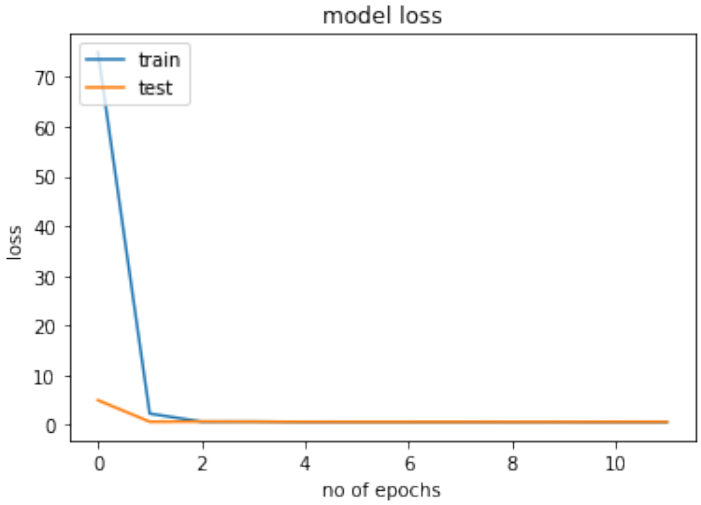


Figure IV.1: Baseline Model Loss

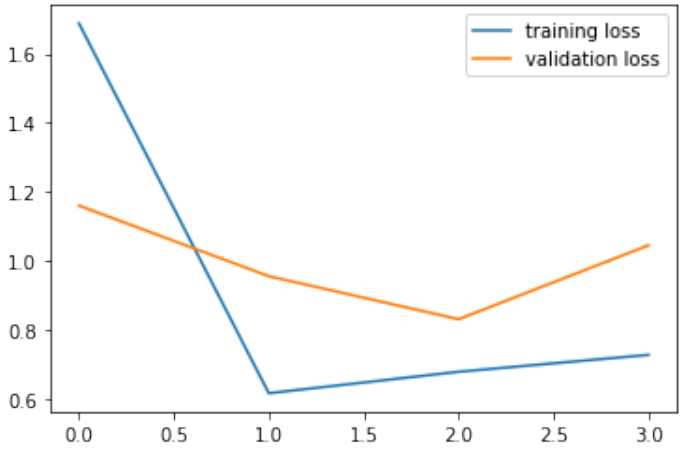


Figure IV.2: PyTorch Baseline Loss

The optimized models from chapter 3 loss graphs in Figures IV.3 and IV.4 had less features that would indicatate a model with good performance. The graph of the optimized model from Tensorflow has a good trend but no plateau. At six epochs the training and validation accuracy

starts to increase so this could be a sign of undertraining and possiblie the model could use more epochs. Overall, this algorithm could be experimented with until is creates a better plateau.

The optimized LeNet from chapter three shows overfitting. This model needs to be adjusted. Possible different types of image transformations or using other optimization techniques. With the validation loss being so much greater than the training loss this model is overtrained.

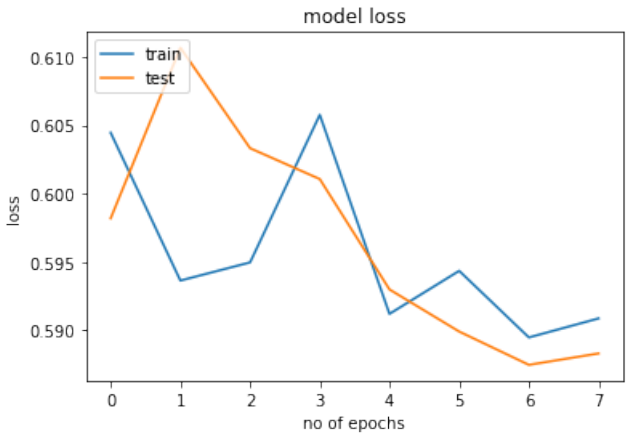


Figure IV.3: Tensorflow Optimization Loss

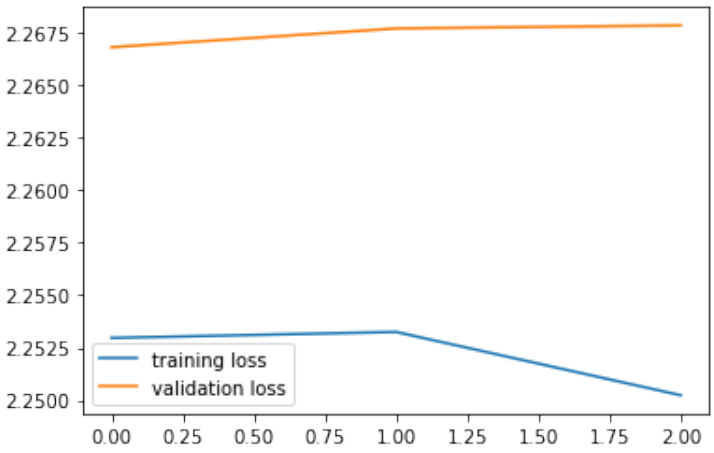


Figure IV.4: PyTorch Optimized Loss

The VGG16 transfer learning loss graphs are below in Figures IV.5 and IV.6. Boths graphs different problems with them. For the Tensorflow model the overall trend of the loss graph is decreasing. Although, the training loss is slightly greater than the validation loss or the algorithm is undertrained. More epoch could be used to further train the model. The Pytroch graph is clearly overtrained. The overall trend is decreasing and when more epochs where included it started increase. Both the VGG16 models give a high accuracy but their graphs show that these models need more training or are not generalizable.

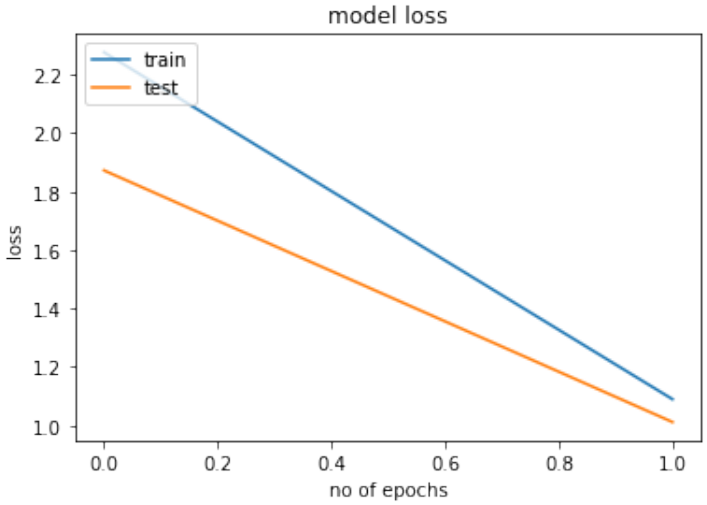


Figure IV.5: Tensorflow Transfer Loss

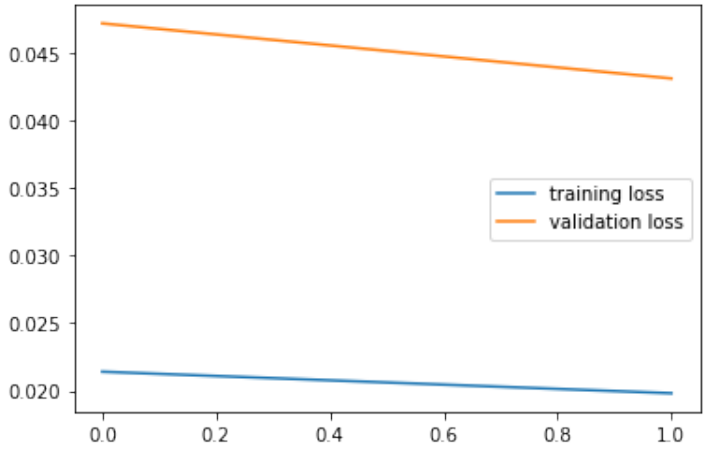


Figure IV.6: PyTorch Transfer Loss

Next the accuracy graphs in Figures IV.7 and IV.8 from the image classification section will be analyzed. There are a similar set of criteria to loss used to analyze accuracy graphs but subtly different. The following criteria will be used to analyze the acceleration graphs:

- The general trend of the accuracy graph gives the most pertinent information. If the accuracy of the model is increasing then the model is becoming more trained. Plateuing in accuracy can mean that the model become as trained as possible or could be overfitting.
- The accuracy can also depend on the learning rate. Capitulations in accuracy can mean that the learning rate needs to be changed. If the accuracy increases rapidly then the learning rate is too high and if it drops then the learning rate is too low.
- Comparing the validation and training accuracy is the main goal of analyzing the graphs. If the training accuracy is much higher than the validation accuracy, it may indicate that the model is overfitting.
- If the accuracy has plateaued, then the model might not be able to gain anything from further training. Plateaus can also be an indication that the model is overfitting.

The graph of the Tensorflow model shows signs of a plateue. There is also capitulations in it that could be attributed to the learning rate being too low do the the sudden drops after two epochs. The overall trend of the two lines is similar and this model has the least overfitting. The value of the accuracy for the this model is around 0.725 and is the best model for classification developed in this capstone.

The PyTorch accuracy graph shows overfitting. The validation capitulates and again could benifit from changing the learning rate. The accuracy graph for Pytorch doesn't really have a trend. Compared to the Tensorflow graph this one is a much better model.

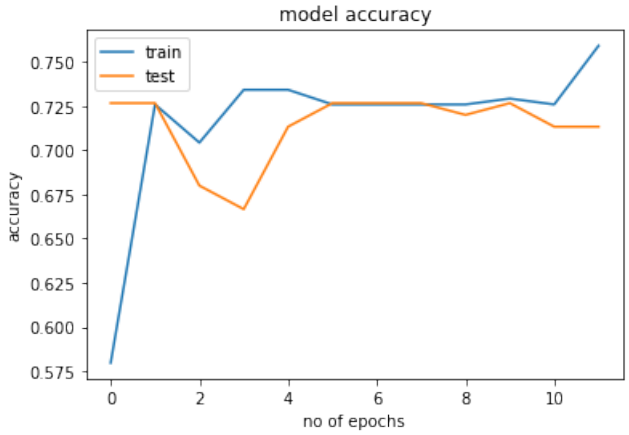


Figure IV.7: Baseline Model TensorFlow Accuracy

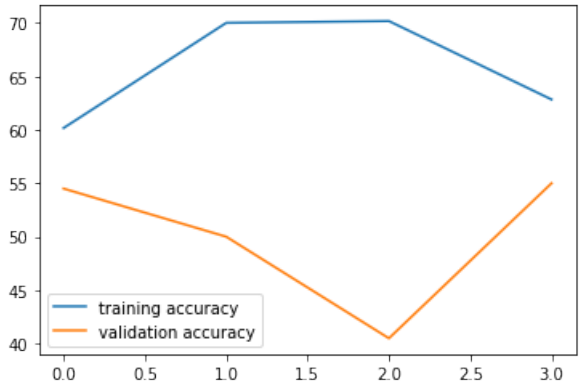


Figure IV.8: PyTorch Baseline Accuracy

The next set of graphs, in Figures IV.9 and IV.10 are from the optimization of the baseline models. The Tesorflow accuracy shows overtraining and it is just constant. There is something wrong with the data or another architecture could be tried. For the Pytorch graph the validation capitulates opposite to the training curve. The overtraining just gets worse as the model trains. Both of the models for this section did not work out well. There must be something in the optimization that is not working.

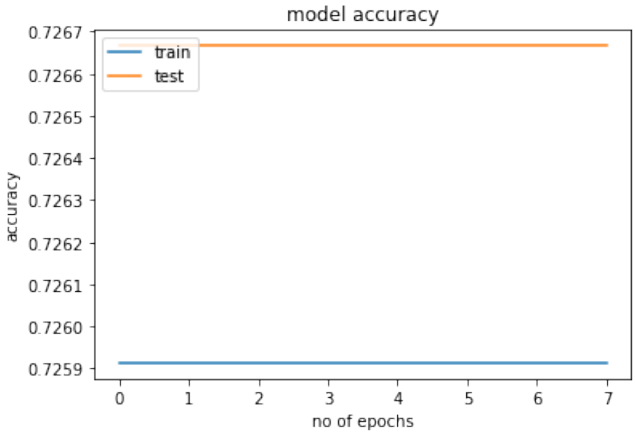


Figure IV.9: Tensorflow Optimization Accuracy

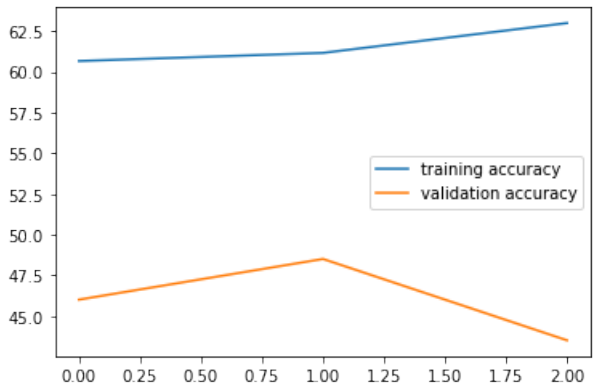


Figure IV.10: PyTorch Optimized Accuracy

The final accuracy graphs, in Figures IV.11 and IV.12, come from training the VGG16 models. The overall trend of the graphs is decent and look like it is going to plateau. Although, the PyTorch graph shows overtraining. Furthermore, the Tensorflow graph also has overtraining after one epoch. So, both of these models need to be worked on. There is no capitulation in the graphs so the learning rate is most likely fine. This neural network does the deepest training but the final metrics need some work.

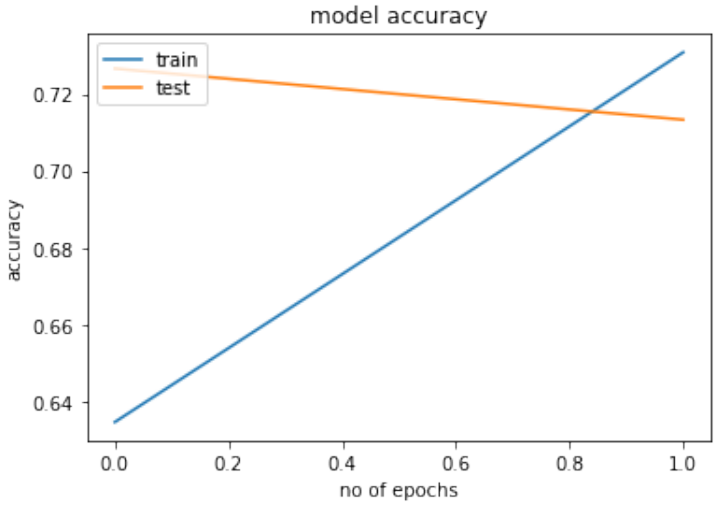


Figure IV.11: Tensorflow Transfer Accuracy

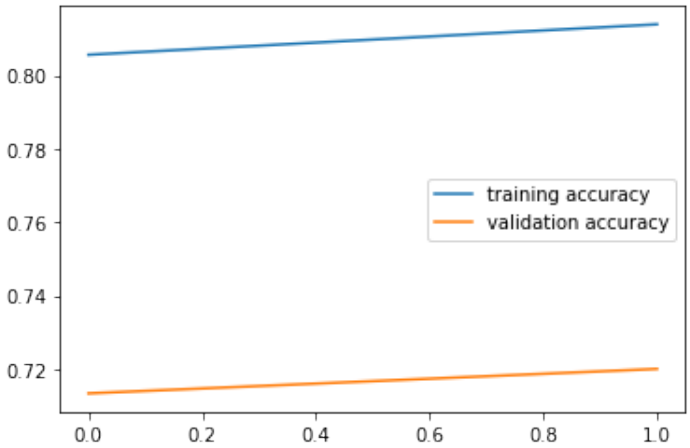


Figure IV.12: PyTorch Transfer Accuracy

(a) PyTorch and Tensorflow Comparison

Table Of Contents

TensorFlow and PyTorch are both open-source deep learning libraries. The following list enumerates some of the pertinent differences.

1. Google developed TensorFlow and released it in 2015, while PyTorch was developed by Facebook and released in 2016.

2. PyTorch is considered to be more user-friendly, while TensorFlow is more difficult to use.

3. The graph system each one uses is different: PyTorch uses a dynamic computational graph, and TensorFlow uses a static computational graph. Dynamic graphs are able to update during training while statics ones do not change.

4. Both TensorFlow and PyTorch can handle Big-Data tasks, but TensorFlow is generally considered to be more scalable. PyTorch is considered to be better for research and debuging.

For the algorithms trained for this Capstone the two libraries gave similar results. The best accuracy came from the optimized model with TensorFlow although this model and many others showed overfitting. Overall the best model came from the VGG16 which was compiled with TensorFlow. This model had a training accuracy of 0.7133 and a validation accuracy of 0.7133. So, this model had the least overfitting and should be generalizable. Overall, many of the models showed overfitting. Augmentation was attempted to increase the size of the data set in the optimization section. Although, that model with Tensorflow did not train well. I tried many different parameters but it didn't help. The PyTorch model that used similar optimization had a low accuracy but the model came did train but had overfitting at higher epochs.

There is still much work that could be done on this chapter. Many other types of transfer learning algorithms could be compiled and their metrics could be analyzed. A great introduc-

tion to transfer learning can be found by Hosna et al [45]. It would be interesting to compile GoogLeNet (InceptionV3) or ResNet50. These could possibly yield better results than the ones presented in this Capstone. In the next section, YOLO object detection will be compiled and presented. A great presentation of the YOLO neural network can be found in Redmon [46].

3 Custom YOLO Object Detection Results

(a) Training the Weights

In this section the final weights will be presented and analyzed. The final loss graph is shown in figure IV.13. The overall trend of the loss is good and it plateau's around 2000 epochs out of 4000. This plateau means that there isn't much is being gained by training the model past 2000 epochs. Although, the loss does slightly decrease as it is trained up to 4000 epochs but the mAP for these epochs is problematic.

The red curve is the mAP which stands for mean average precision and is a common metric in object detection. The trend of the mAP is overall not the best. It does reach fifty five percent around 1300 epochs. Then it trends lower to about thirty seven percent. The final mAP could be improved by training the algorithm again to 1600 epochs.

Overall, the final result does work but it needs some additional work to achieve at minimum a consistent mAP score of fifty percent. This could be due to the size of the data set being small and also the way that the images where reduced. So, this object detection system could be made better but for this Capstone this is the final result.

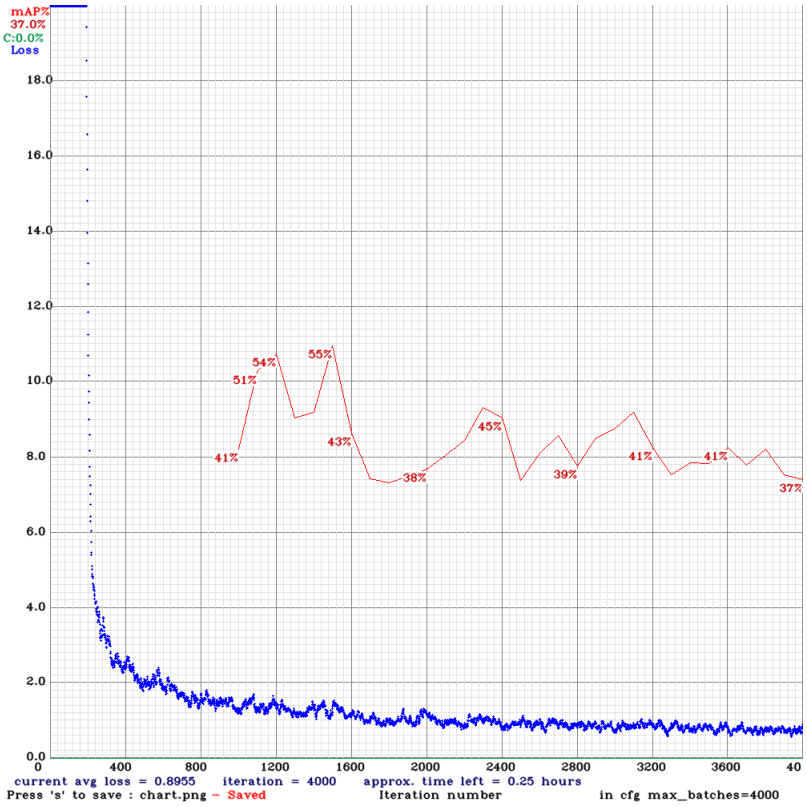


Figure IV.13: YOLOv4 Loss

(b) Testing the Model

Table Of Contents

To test the model the code from chapter 4 section 2 must be altered to use the newly trained weights and the classes for the stroke images. This code can be found in the Capstone adendum [8]. In figure IV.14 the CE stroke is labeled correctly with 96% accuracy. It would be interesting to have some new images to run the object detection model on. But this is a good result for this Capstone.

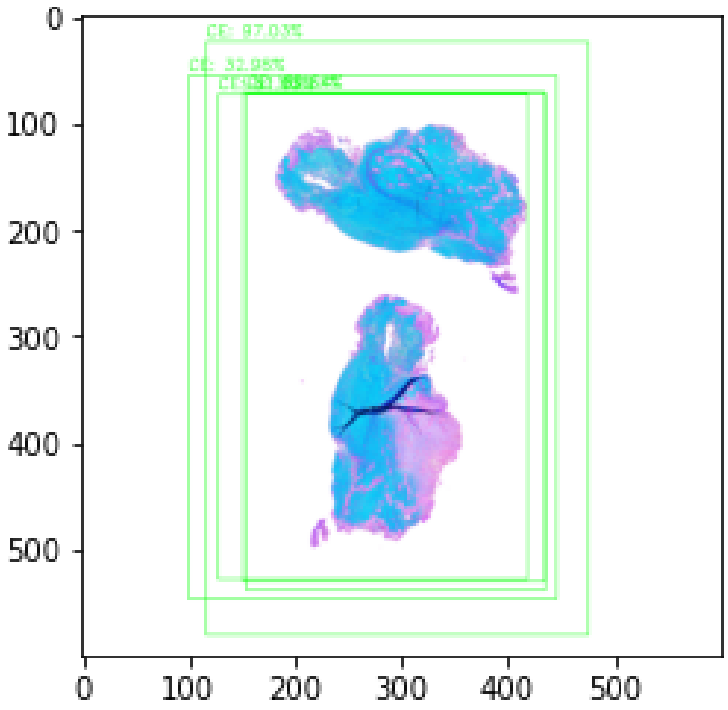


Figure IV.14: YOLOv4 Loss

In figure IV.15 there is a LAA stroke that is classified with 90.66% accuracy. Overall, the object detection system is working and this is the final result for this Capstone. A working object detection system was trained and tested. Although, there could be changes made such that the system has a better mAP.

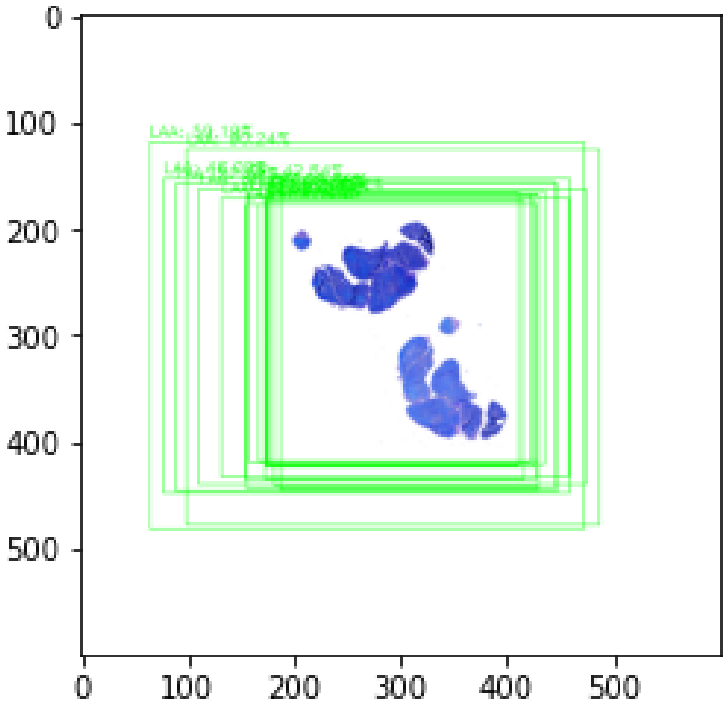


Figure IV.15: YOLOv4 Loss

Chapter V

Summary, Recommendations, Conclusion

Table Of Contents

In conclusion, many different skills were used for this Capstone. The algorithms that were built could be further developed and become useful for institutions like the Mayo Clinic and others. The overall goal of this Capstone is to explore how computer vision technologies along with Deep Learning can help medical professionals make better diagnoses and for people suffering from strokes to have better prognoses. The technologies used and presented in this Capstone offer a great future for many different applications and for the medical profession. A great revolution in medicine is occuring now and these new technologies show that the future of medicine is an amazing endever.

The methodolgy of this Capstone started off dealing with high-resolution images and managing their memory. Using the OpenCV library the images had their memory reduced such that they can be used in a Convolutional Neural Network. Using the code from Kaggle found in Mulla [39], the high-resolution images are reduced using a function in OpenCV found in chapter III section 2. This is an interesting calculation and has many other solutions to it. The one used in

the capstone was one that gave the most useful images within the time limits. Although, there are many other approuches to reducing the memory of the images and produce better results when the deep learning algorithms are applied.

After reducing the images memory the methodology and results of training and testing a classification model using deep learning algorithms was presented. Two different libraries was used to compile these models on the stroke images. First Keras/Tensorflow will be utilized to try to find the optimal model. This was the most successful of all the different attempts between the two libraries. Keras was easier to test out different neural networks and lead to an algorithm that had the best metrics and least overfitting.

Using Pytorch was not as successful, although I'm fairly new to the software. The models had overfitting and generall poor metrics. Both Pytorch and Tensorflow models results from optimization did not turn out as intended. It would be interesting to try out different experiments using different techniques to reduce the images memory and then test out different optimization parameters.

Finally, the object detection system was compiled and was used to detect the two different types of strokes. The results for the mAP could have been because the number of images that came with the data is small compared to what is optimal. Although, the number of images being small does help with compiling the model in a reasonable amount of time. The final object detection system had a decent loss and possible could have better results if Darknet was trained on less epochs. This object detection system would be the most important for stakeholders and could be used to build great applications to help medical professionals.

Overall, there are many different applications of this capstone to different research and markets. The stakeholders could use the weights calculated in this Capstone to create an application that would detect strokes with a video or image and help physicians diagnose patients and help people who suffer from strokes. Moreover, this technology has an interesting future. I hope to develop more in the field of computer vision and deep learning.

Finally, this capstone was a difficult endeavor. It was brutal to get to the end of the object

detection system and have it not classified correctly and many hard lessons were learned and the final product still needs work. Although, in the process, I learned a lot about deep learning and computer vision. I hope to become an expert in unstructured data in the future. My next project will be in NLP and I'm currently looking for a project in computer vision. Hopefully, this is just a stepping-stone to a long career in data science.

Chapter VI

References

Table Of Contents

[1] Frost & Sullivan (2016) Frost & Sullivan From $600 M to $6 billion, artificial intelligence systems poised for dramatic market expansion in healthcare. https://ww2.frost.com/news/press-releases/600-m-6-billion-artificial-intelligence-systems-poised-dramatic-market-expansion-healthcare/ 2016

[2] Lakhani & Sundaram (2017) Lakhani P, Sundaram B. Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. Radiology 2017

[3] Mayo Clinic - strip AI. Kaggle. (n.d.)., from https://www.kaggle.com/competitions/mayo-clinic-strip-ai

[4] Vishnu VY, Vinny PW. The Neurologist and Artificial Intelligence: Titans at Crossroads. Ann Indian Acad Neurol. 2019 Jul-Sep;22(3):264-266. DOI: 10.4103/aian.AIAN_493_18. PMID: 31359934; PMCID: PMC6613413.

[5] raghavgarg12. (2022, September 27). Data pre-processing. Kaggle., from https://www.kaggle.com/code/raghavgarg12/data-pre-processing

[6] Kim M, Yun J, Cho Y, Shin K, Jang R, Bae HJ, Kim N. Deep Learning in Medical Imaging. Neurospine. 2019 Dec;16(4):657-668. doi: 10.14245/ns.1938396.198. Epub 2019 Dec 31. Erratum in: Neurospine. 2020 Jun;17(2):471-472. PMID: 31905454; PMCID: PMC6945006.

[7] Ahuja AS. The impact of artificial intelligence in medicine on the future role of the physician. PeerJ. 2019 Oct 4;7:e7702. doi: 10.7717/peerj.7702. PMID: 31592346; PMCID: PMC6779111.

[8] Patrickhearin. (n.d.). Patrickhearin/capstone_code_addendum. GitHub., from https://github.com/patrickhea

[9] Sciences, A. I. (n.d.). Computer vision theory and projects in Python for beginners. O'Reilly Online Learning, from https://www.oreilly.com/library/view/computer-vision-theory/9781801815949/

[10] Slim, R., Slim, J., Sharaf, A., &; Tanveer, S. (n.d.). Pytorch for Deep Learning and Computer Vision. O'Reilly Online Learning., from https://learning.oreilly.com/videos/pytorch-for-deep/9781838822804/

[11] Veen, F. van. (2017, April 1). Fjodor van Veen, author at the Asimov Institute. The Asimov Institute., from https://www.asimovinstitute.org/author/fjodorvanveen/

[12] Panagakis, Y., Kossaifi, J., Chrysos, G. G., Oldfield, J., Nicolaou, M. A., Anandkumar, A., & Zafeiriou, S. (2021, July 7). Tensor methods in computer vision and Deep Learning.

arXiv.org. Retrieved January 24, 2023, from https://arxiv.org/abs/2107.03436

[13] O'Shea, K., & Nash, R. (2015, December 2). An introduction to Convolutional Neural Networks. arXiv.org., from https://arxiv.org/abs/1511.08458

[14] Fischler, M. A., and Elschlager, R. (1973). The representation and matching of pictorial structures. IEEE Trans. Comput. C-22, 67–92. doi:10.1109/T-C.1973.223602

[15] Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural network-based detection. IEEE Trans. Pattern Anal. Mach. Intell. 20, 23–28. doi:10.1109/34.655647

[16] Weinland, D., Ronfard, R., and Boyer, E. (2011). A survey of vision-based methods for action representation, segmentation and recognition. Comput. Vis. Image Underst. 115, 224–241. doi:10.1016/j.cviu.2010.10.002.

[17] Delakis, M., and Garcia, C. (2004). Convolutional face finder: a neural architecture for fast and robust face detection. IEEE Trans. Pattern Anal. Mach. Intell. 26, 1408–1423. doi:10.1109/TPAMI.2004.97

[18] Lampert, C. H., Blaschko, M., and Hofmann, T. (2009). Efficient subwindow search: a branch and bound framework for object localization. IEEE Trans. Pattern Anal. Mach. Intell. 31, 2129–2142. doi:10.1109/TPAMI.2009.144

[19] Viola, P., and Jones, M. (2002). "Fast and robust classification using asymmetric adaboost and a detector cascade," in Advances in Neural Information Processing System 14 (Vancouver: MIT Press), 1311–1318.

[20] Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010b). Object detec-

tion with discriminatively trained part-based models. IEEE Trans. Pattern Anal. Mach. Intell. 32, 1627–1645. doi:10.1109/TPAMI.2009.167

[21] Dollar, P., Wojek, C., Schiele, B., and Perona, P. (2012). Pedestrian detection: an evaluation of the state of the art. IEEE Trans. Pattern Anal. Mach. Intell. 34, 743–761. doi:10.1109/TPAMI.2011.155

[22] Monti S, Cavaliere C, Covello M, Nicolai E, Salvatore M, Aiello M. An Evaluation of the Benefits of Simultaneous Acquisition on PET/MR Coregistration in Head/Neck Imaging. J Healthc Eng. 2017;2017. doi: 10.1155/2017/2634389. PMID: 29076331.

[23] Esener, Idil & Ergin, Semih & Yüksel, Tolga. (2017). A New Feature Ensemble with a Multistage Classification Scheme for Breast Cancer Diagnosis. Journal of Healthcare Engineering. 2017. 1-15. 10.1155/2017/3895164.

[24] Gargiulo P, Árnadóttir I, Gíslason M, Edmunds K, Ólafsson I. New Directions in 3D Medical Modeling: 3D-Printing Anatomy and Functions in Neurosurgical Planning. J Healthc Eng. 2017;2017. doi: 10.1155/2017/1439643. PMID: 29068642.

[25] Zhang S, Wei Z, Nie J, Huang L, Wang S, Li Z. A Review on Human Activity Recognition Using Vision-Based Method. J Healthc Eng. 2017;2017:3090343. doi: 10.1155/2017/3090343. Epub 2017 Jul 20. PMID: 29065585; PMCID: PMC5541824.

[26] Pan C, Xu W, Shen D, Yang Y. Leukocyte Image Segmentation Using Novel Saliency Detection Based on Positive Feedback of Visual Perception. J Healthc Eng. 2018 Feb 1;2018:5098973. doi: 10.1155/2018/5098973. PMID: 29599951; PMCID: PMC5823411.

[27] Dong C, Zeng X, Lin L, Hu H, Han X, Naghedolfeizi M, Aberra D, Chen YW. An Improved Random Walker with Bayes Model for Volumetric Medical Image Segmentation. J Healthc Eng. 2017;2017:6506049. doi: 10.1155/2017/6506049. Epub 2017 Oct 23. PMID: 29201332; PMCID: PMC5672701.

[28] Song Q, Zhao L, Luo X, Dou X. Using Deep Learning for Classification of Lung Nodules on Computed Tomography Images. J Healthc Eng. 2017;2017:8314740. doi: 10.1155/2017/8314740. Epub 2017 Aug 9. PMID: 29065651; PMCID: PMC5569872.

[29] Kamarudin ND, Ooi CY, Kawanabe T, Odaguchi H, Kobayashi F. A Fast SVM-Based Tongue's Colour Classification Aided by k-Means Clustering Identifiers and Colour Attributes as Computer-Assisted Tool for Tongue Diagnosis. J Healthc Eng. 2017;2017:7460168. doi: 10.1155/2017/7460168. Epub 2017 Apr 20. PMID: 29065640; PMCID: PMC5416652.

[30] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: single shot Multi-BoxDetector. In European Conference on Computer Vision, Cham: Springer. (2016). p. 21–37 doi: $10.1007/978-3-319-46448-0\_2$

[31] Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition. Columbus, OH. (2014). p. 580–587. doi: 10.1109/CVPR.2014.81

[32] . Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature pyramid networks for object detection. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016). p. 936–944. doi: 10.1109/CVPR.2017.106

[33] Zhuang Z, Liu G, Ding W, Raj ANJ, Qiu S, Guo J, et al. Cardiac VFM visualization and

analysis based on YOLO deep learning model and modified 2D continuity equation. Comput Med Imaging Graph. (2020) 82:101732. doi: 10.1016/j.compmedimag.2020.101732

[34] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016). p. 779-88.

[35] Hirasawa T, Aoyama K, Tanimoto T, Ishihara S, Shichijo S, Ozawa T, et al. Application of artificial intelligence using a convolutional neural network for detecting gastric cancer in endoscopic images. Gastric Cancer. (2018) 21:653–60. doi: 10.1007/s10120-018-0793-2

[36] Cao W., Czarnek N., Shan J., Li L. Microaneurysm detection using principal component analysis and machine learning methods. IEEE Transactions on NanoBioscience . 2018;17(3):191–198. doi: 10.1109/tnb.2018.2840084.

[37] Li X., Xiang J., Wang J., Li J., Wu F.-X., Li M. Funmarker: fusion network-based method to identify prognostic and heterogeneous breast cancer biomarkers. IEEE/ACM Transactions on Computational Biology and Bioinformatics . 2021;18(6):2483–2491. doi: 10.1109/tcbb.2020.2973148.

[38] Shou Y, Meng T, Ai W, Xie C, Liu H, Wang Y. Object Detection in Medical Images Based on Hierarchical Transformer and Mask Mechanism. Comput Intell Neurosci. 2022 Aug 4;2022:5863782. doi: 10.1155/2022/5863782. PMID: 35965770; PMCID: PMC9371842.

[39] Mulla. (2022, July 7). Mayo Clinic Image Dataset 1024 JPG. Kaggle. Retrieved November 28, 2022, from https://www.kaggle.com/code/robikscube/mayo-clinic-image-dataset-1024-jpg

[40] Nelson, A. (n.d.). Deep learning using keras - a complete and Compact Guide for Beginners. O'Reilly Online Learning., from https://learning.oreilly.com/videos/deep-learning-using/9781803242835/

[41] Tensorflow. TensorFlow. (n.d.). Retrieved February 12, 2023, from https://www.tensorflow.org/

[42] Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., & Shen, F. (2022). Image Data Augmentation for Deep Learning: A Survey. ArXiv. https://doi.org/10.48550/arXiv.2204.08610

[43] Jha, A. R., &; Pillai, D. G. (n.d.). Mastering pytorch. O'Reilly Online Learning., https://www.oreilly.com/library/view/mastering-pytorch/9781789614381/

[44] Simonyan, K., &; Zisserman, A. (2015, April 10). Very deep convolutional networks for large-scale image recognition. arXiv.org., 2022, from https://arxiv.org/abs/1409.1556

[45] HarisIqbal88. (n.d.). Harisiqbal88/PlotNeuralNet: Latex Code for making Neural Networks diagrams. GitHub., from https://github.com/HarisIqbal88/PlotNeuralNet

[46] Pytorch. PyTorch. (n.d.)., 2023, from https://pytorch.org/

[47] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[48] Heartexlabs. (n.d.). Heartexlabs/labelimg: LabelImg is now part of the label Studio Community. the popular image annotation tool created by Tzutalin is no longer actively being developed, but you can check out label studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data. GitHub., from https://github.com/heartexlabs/labelImg

[48] Hosna, A., Merry, E., Gyalmo, J. et al. Transfer learning: a friendly introduction. J Big

Data 9, 102 (2022). https://doi.org/10.1186/s40537-022-00652-w

[49] Nelson, A. (n.d.). Computer vision: You only look once (YOLO) custom object detection with colab GPU. O'Reilly Online Learning., from https://learning.oreilly.com/videos/computer-vision-you/9781800563865/

[

HarisIqbal88. (n.d.). Harisiqbal88/PlotNeuralNet: Latex Code for making Neural Networks diagrams. GitHub., from https://github.com/HarisIqbal88/PlotNeuralNet