EE 519: Deep Learning Theory & Fundamentals **Spring 2023**

Mini Project Andy Franck

Due: April 28, 2023, 11:59PM PT

Student Name: Andy Franck *Instructor Name: John Lipor*

Problem Description

The objective of this project is to train a multi-output convolutional neural network capable of accurate calculating both the horizontal and vertical diameter of an eyeball in an ultrasound image. After predicting, the model will be studied using guided backpropogation to determine what the trained model is used to make its predictions. The model will be trained on 100 ultrasound images of eyeballs, each with a manually labeled horizontal and vertical diameter.

Because of the nature of the image, the model may be able to make fairly accurate predictions even with the limited data size. This report aims to maximise the performance of the model by utilizing shared layers that branch off later into the model. This will allow the model to learn more general features of the image, and then use those features to make more accurate predictions.

Although basic in scope, the goal of this project is to make steps towards a more advanced method of pathology detection in ultrasound images. Ideally, this project would be the start of a larger project that would be able to determine more complex pathologies from ultrasound images, such as other tissues that are not visible through other methods of analysis.

Exploratory Data Analysis

Because of the nature of the data, it was not necessary to perform much data analysis. It was easy to load directly from the files, and straightforward to work with through both the PyTorch and Numpy libraries. To begin, the ultrasound images were noted to have considerable area that was not useful for training. To fix this, the ultrasound images were cropped by 20% to reduce unnecessary data. Because all of the eyeballs were taken in the center of the screen, and was considerable black space on the surrounding area, it was simple to crop the images without losing any useful data:

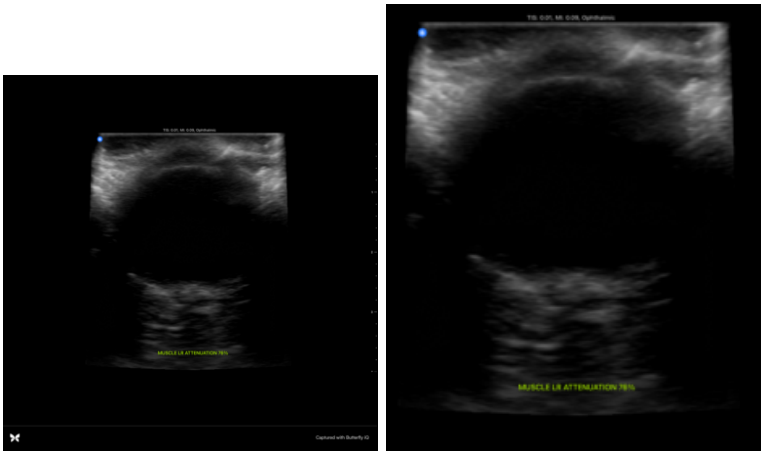

Figure 1: Uncropped (left) and cropped (right) ultrasound images.

Besides basic cropping, the data was also normalized to be between 0 and 1. This was done by dividing each pixel by 255, the maximum value of a pixel. This was done to make the data easier to work with, and to make the model more robust to changes in the data.

Finally, the labels were inspected to check data ranges, values, and distributions. The following are box and whisker plots of both the horizontal and vertical labels. It is clear from the picture that there are not many outliers, so no modifications were made to the labels.

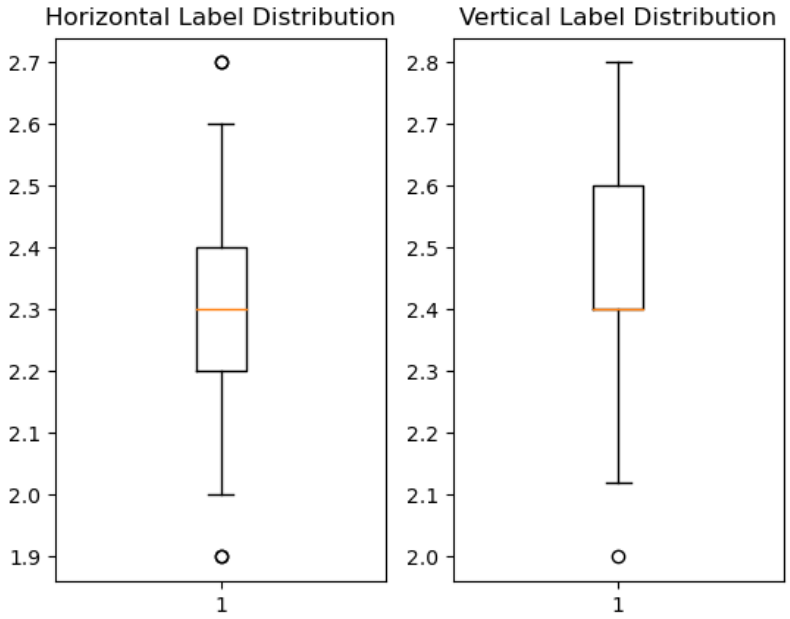

Figure 2: Box and whisker plots of the horizontal and vertical labels.

Challenges

The largest challenge for this project was determining how many shared and seperate layers to use in the neural network model. Optuna was applied to the model however it was not able to find any significant improvements to the model, so it was discarded. Instead, the model was manually tuned to find the best combination of shared and seperate layers.

A dataset class was also made to load data properly into the dataloader with both y labels. This was done by creating a custom dataset class that would load the data and labels, and then return both the image and the labels. This was then passed into the dataloader, which was able to load the data into the model.

Additionally, it was difficult to determine exactly how accurate the model was predicting, since it is a standard regression problem. It was determined that if the prediction was truncated to the nearest hundreth, then it was a correct prediction.

Finally, the guided bkacpropogation analysis proved very difficult to implement. After adding hooks for the ReLU layers in the model, the analysis was subject to both exploding and vanishing gradients. Fortunately, this was solved by utilizing a different method to calculate the saliency maps without directly hooking the ReLU functions. [1]

Approach

The intial step in developmen involved creating a baseline Network in Network model with two outputs and all shared layers to get a baseline accuracy. The model implemented both batch normalization and dropout layers into the model based on the discoveries from the previous geothermal energy project. This model was used to make sure all data was being loaded properly, training was working as intended, and to establish a learning baseline to be build upon.

```
self.net = nn.Sequential(
    nin_block(96, kernel_size=5, strides=3, padding=0),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(256, kernel_size=3, strides=1, padding=2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(384, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(num_classes, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten())
```

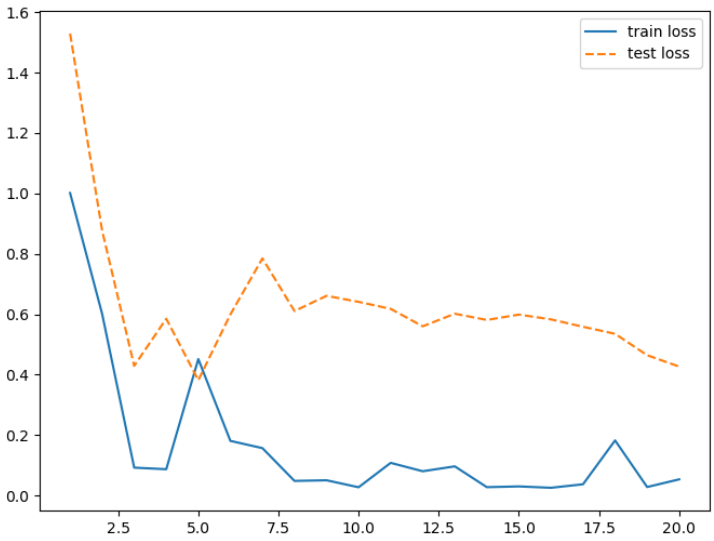

Figure 3: Baseline model architecture and loss graph.

The model did not appear to be learning very well. Two methods were utilized to improve the performance of the model.

To artifically increase the data size, image augmentation was utilized via the torchvision.transforms library [2]. This library allows for the creation of a transform object that can be applied to a dataset. The transform object contained both random flips and gaussian noise. The transform was applied after loading each batch at the start of the training loop, so that each batch was augmented differently.

Secondly, the model was modified to utilize both shared and seperate layers. This was done by taking the last two Network in Network layers and splitting them into a different Sequential object. This allowed for the model to learn more general features of the image, and then use those features to make more accurate

predictions for both the horizontal and general diameters.

```
self.shared = nn.Sequential(
    nin_block(96, kernel_size=5, strides=3, padding=0),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(256, kernel_size=3, strides=1, padding=2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2),
    nin_block(384, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.MaxPool2d(3, stride=2)
    )

self.split = nn.Sequential(
    nin_block(num_classes, kernel_size=3, strides=1, padding=1),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten())
```

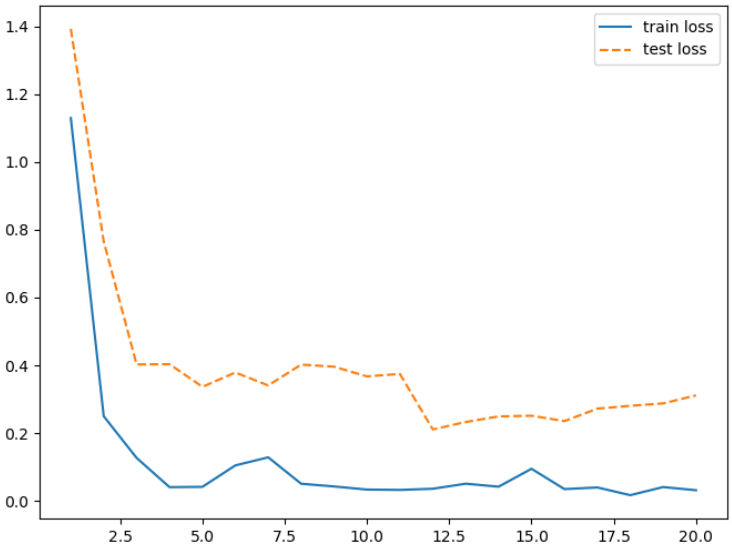

Figure 3: Updated model architecture and loss graph.

Saliency maps of the model were generated using guided backpropogation, following the tutorial [1] and [3]. Originally, a hook was added to every ReLU activation layer in the network, and then the model was run on a single image. The gradients were then calculated at the output layer with respect to the input image. However, this method resulted in many gradient values equal to zero, so a modification was made following [1] to calculate the gradients without directly hooking the ReLU layers.

Evaluation and Summary

What I Learned

This project was excellent practice for experimenting with more in-depth models. Data management and EDA were both very straightforward in this project, which allowed for more time to work on optimizing model parameters and layers in a multi-output format.

Additionally, now having some basic experience with image augmentation methods, it was interesting to discover more advanced methods to inflate the dataset. This was not particularly important for the project due to how quickly the model learned, however it was still interesting to learn about.

Finally, this projects use of saliency maps without classification was an interesting twist. Because it is difficult to determine a "correct" prediction, the nature of the maps and their usefulness was different than in previous projects. It was interesting to see how the maps were able to highlight the features of the image that the model was using to make its predictions, even without a classification task.

References

[1] I. A. Khalid, "Saliency map for visualizing deep learning model using pytorch," 2021, https://towardsdatascience.com/saliency-map-using-pytorch-68270fe45e80.

[2] https://pytorch.org/vision/stable/transforms.html.

[3] K. Chung, "Guided backpropogation with pytorch and tensorflow," 2021, https://www.coderskitchen.com/guided-backpropagation-with-pytorch-and-tensorflow/.