Neuron Counting Neural Network

Jamie Bergen · Follow

9 min read · Dec 10, 2022





By Kaiya, Austin, and Jamie

Abstract

This project focuses on creating a neural network for counting stained neurons in images taken on a microscope. The process for staining,imaging, and counting cells is incredibly common in neuroscience laboratories and incredibly time consuming. Therefore, a project using transfer learning to create a network to count cells could save many scientists significant lab time to be spent on more experiments and research. The outcome of the network was limited by the small sample size of images due to limitations in their collection. However, due to time, sample size, and computing constraints, the network had limited success.

Introduction to the Project:

Cell staining and counting is one of the most common tasks done in biology labs around the world. While there are ways to automate parts of the staining process (though it is still more common to immunohistochemistry by hand due to the costs of automation) cell counting is more exclusively done by hand. Each stain done on any given cell type may look quite different, and so hardcoded algorithms are typically not very successful for most kinds of cell quantification. Therefore the task of cell counting (with the exception of a few cell types such as blood cells) is generally viewed as too specialized to be done by computer. Enter neural networks.

We have noticed with recent developments in neural networks, the ability to automate more specific tasks such as cell counting has newfound potential. This is because the cost of training a network can be heavily reduced by using transfer learning from a pre-trained network. The goal of this project is to develop a neural network to count neurons stained with the ANNA1 stain and then imaged on a confocal microscope, a very commonly used stain and imaging technique in neuroscience/biology labs. This may function as a proof of concept that training a network for simple yet specialized tasks may be a great way to free up time for scientists spent counting cells, and give them more time to focus on learning and experimenting.

Counting Neurons

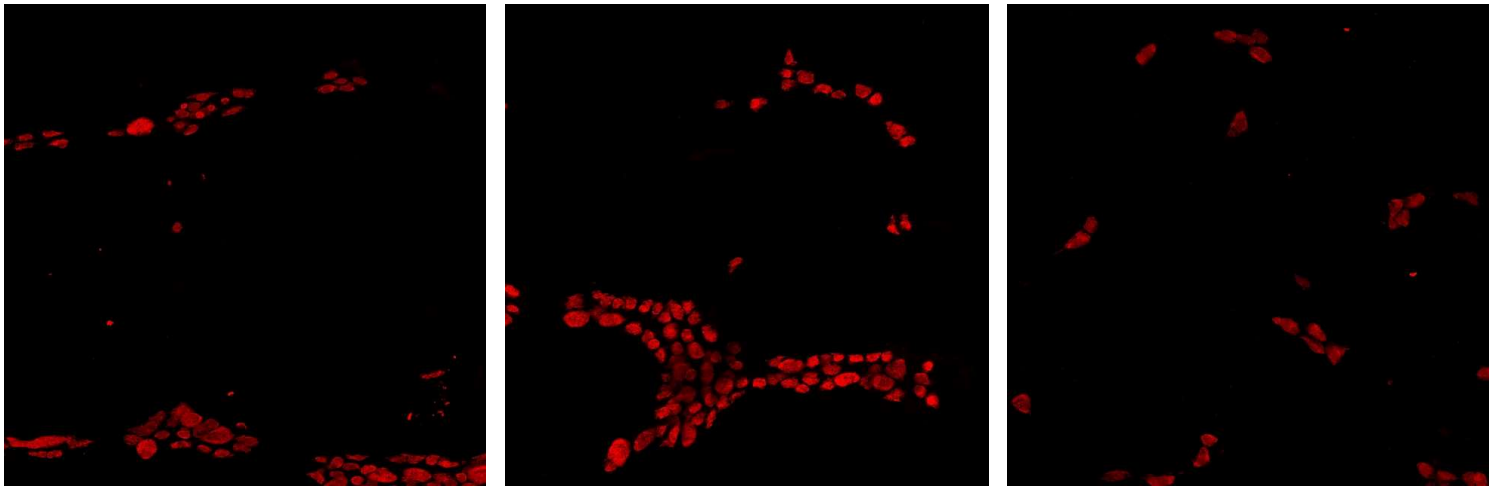

Fig. 1.0 ANNA1 Neurons

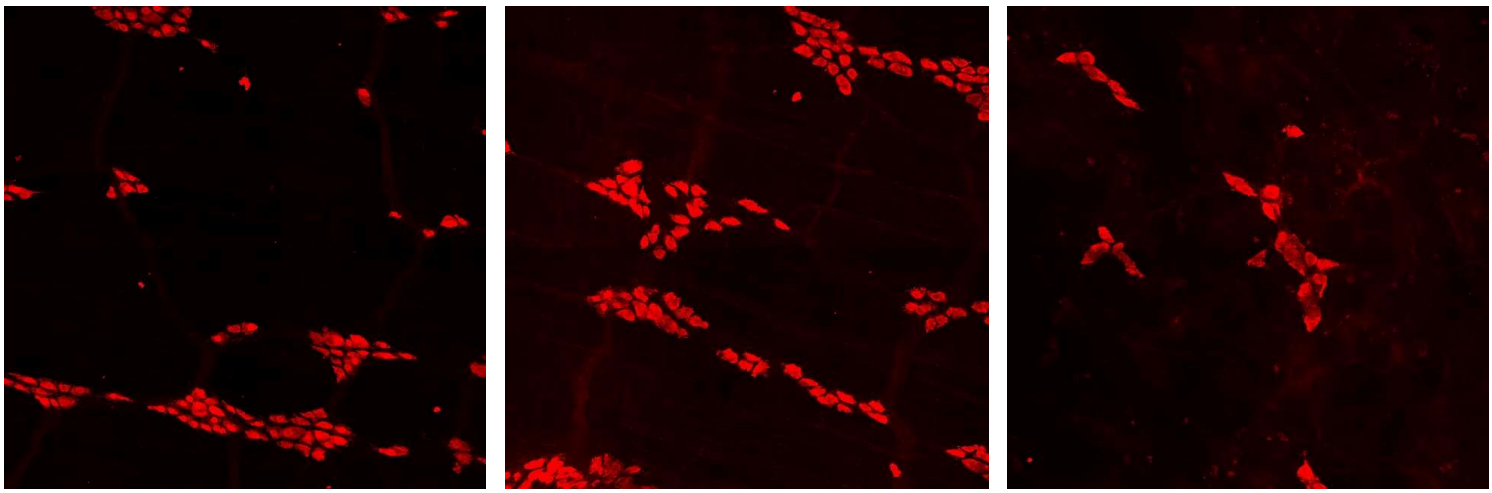

Fig. 1.1 Additional ANNA1 Neurons

Images (examples shown above, Fig 1.0, 1.1) were sourced from the Agalliu Lab the at Columbia University Irving Medical center, which were generated as follows:

Two layers of the gut, submucosal plexus (SP) and myenteric plexus (MP), of mice were dissected out and stained for ANNA1 (1:32,000) which is a protein to stain for total neuron populations. The slides were then coverslipped with a mounting medium to preserve them (Vector Laboratories) and stored at -20 degrees before imaging. The SP and MP were imaged on a confocal

microscope taking three 4x4 tiled images per plexus at 20x magnification. The treatment groups of the mice were not visible to the researcher performing the quantification, to avoid bias.

The images were then normalized and converted to JPG format on ImageJ using a macro we wrote, and neurons were hand counted to be used to train the network. Due to limitations in working with real-life data from a lab, a very small sample size of images was used to train the network (88 images).

Link to the dataset here: Link

Network Design:

In trying to decide on how to structure the network, we were torn between building a CNN from the ground up, or using transfer learning to assist in some of the earlier. The primary advantage of a CNN is that you are able to have more control over all the steps of the network you are creating, but it also requires a larger dataset and it is more computationally expensive.

Because of these limitations, particularly our small dataset we ended up choosing a transfer learning based model with layers of dense nodes. These layers of dense nodes were determined through trial and error which will be discussed further in the Playing with Training: Fighting overfitting section of this paper. The code and a diagram showing the structure of the network are shown below.

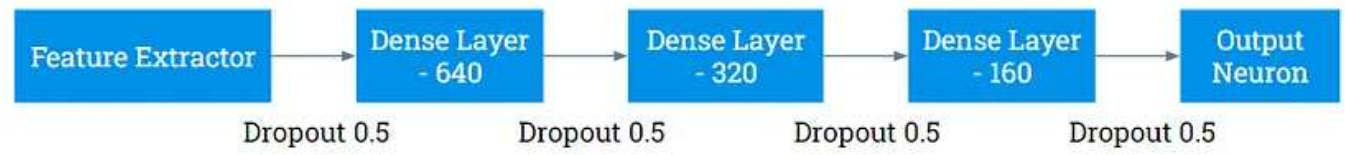

Fig. 1.3 Structure of the Neural Network

```
ropout(rate=0.5),
ense(640, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
ropout(rate=0.5),
ense(320, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
ropout(rate=0.5),
ense(160, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
ropout(rate=0.5),
ense(1,
     kernel_regularizer=tf.keras.regularizers.l2(0.01))

MAGE_SIZE+(3,))
```

Choosing a Feature Extractor Model:

Our choice of pre-trained feature extractor has the most influence on the accuracy of our model. The more complex the feature extractor is, the more detail the pre-trained model will be able to extract from the neuron images. However, the achievable feature extractor size is limited by the amount of computing time and memory that Kaggle can allocate to the model.

We primarily experimented with four sizes of the EfficientNet v2 model. EfficientNet is trained on the 2011 ImageNet database of photos of one thousand different categories of objects. When used as a feature extractor

(not a classifier), EfficientNet outputs a vector with an abstract representation of an image's features (ImageNet Dataset | Papers With Code).

We started with the XL version of the EfficientNet model, which had too many parameters to retrain on Kaggle. Therefore, the complete model was not specialized to the neuron dataset. The B0 model had less than 5% as many neurons as the XL version, so it was able to train quickly. However, it made no significant improvement in testing accuracy when it was trained for more epochs (Figure 1.4), which is a clear sign of overfitting. We got promising results from the M version since both testing and training loss decreased throughout training, as seen in Figure 1.5. Unfortunately, we were unable to load this model again for further training due to Kaggle's memory limitations.

We ultimately selected EfficientNet v2 S, because it strikes a balance between model complexity and ability to run on Kaggle. When testing with one hundred epochs, this model had promising testing loss after about 40 to 50 epochs (Figure 1.6).

Model Summaries:

- EfficientNet v2 XL 21k– 207M parameters, 580x580 input resolution
- EfficientNet v2 B0– 7M parameters, 224x224 input resolution

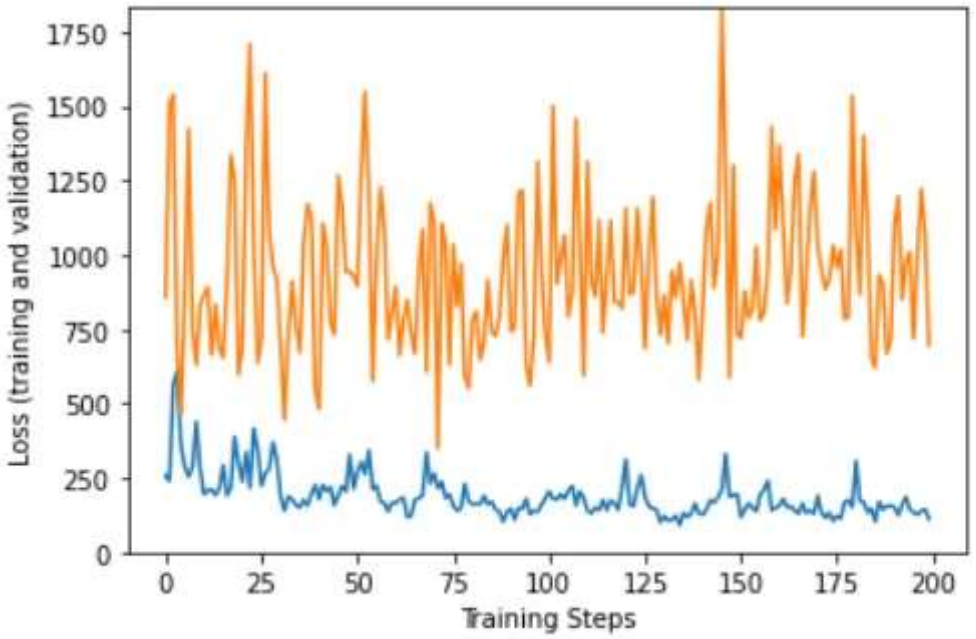

Figure 1.4 EfficientNet v2 B0 Training

- EfficientNet v2 M 21k– 54M parameters, 480x480 input resolution

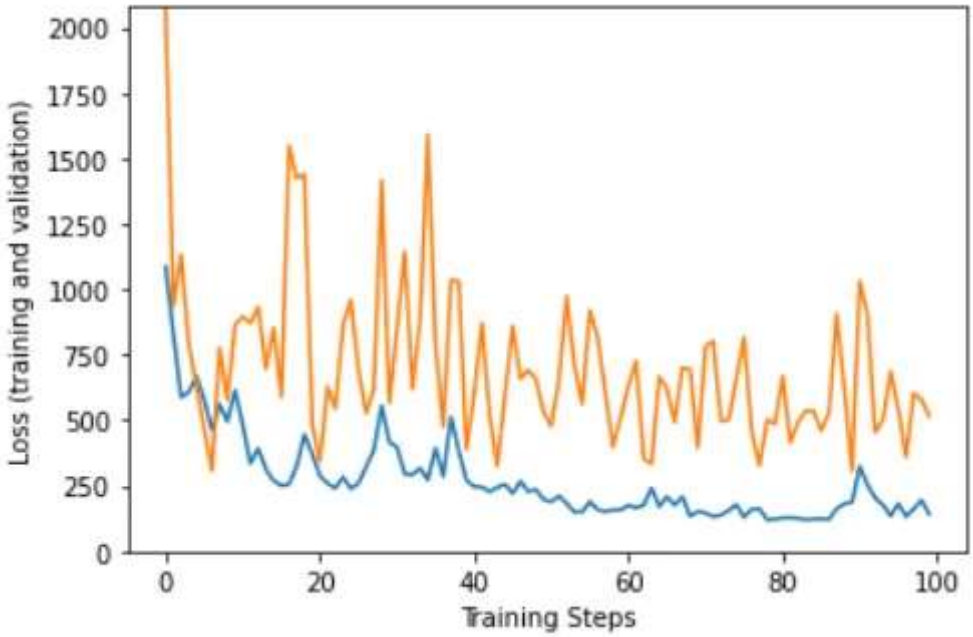

Figure 1.5 EfficientNet v2 M Training

- EfficientNet v2 S 21k– 21M parameters, 384x384 input resolution

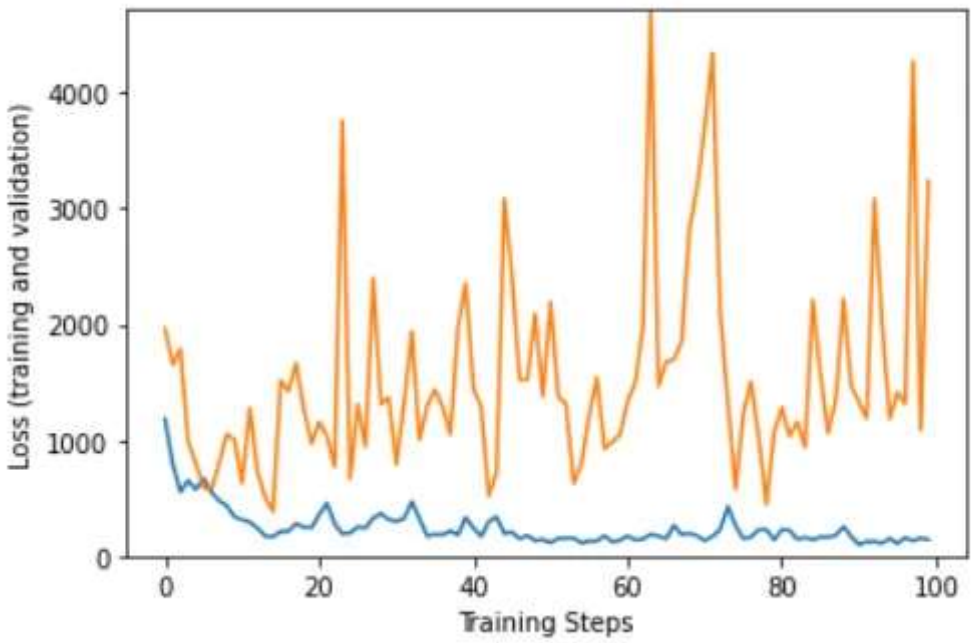

Figure 1.6 EfficientNet v2 S Training

Our choice to use a classifier as a feature extractor for a regression model raises some limitations. EfficientNet was originally trained to identify distinct features of everyday objects, however we are feeding it extremely visually similar images. Therefore, retraining the feature extractor with our neuron data is essential. In an ideal world, we would freeze the first layers of the feature extractor, which might identify basic features like lines and shapes, and retrain the later layers of the model to identify properties of cell images. Unfortunately, our current integration of EfficientNet with our model requires us to retrain the entire model.

Playing with Training– Fighting Overfitting:

Neural networks involve many hyperparameters, with many possible values for each. Usually, programs are written to find optimal values for these

parameters, but that added a level of complexity outside the scope of this project. Instead, we tested some major hyperparameters by hand.

We started by varying the epoch number (5–20) and drop rate (0.2–0.8), before realizing that we needed more images and introducing additional data augmentation. Since zooming and translating the images might have invalidated the associated neuron counts, we created three 90 degree rotations of each image, effectively quadrupling the size of our data set.

With the data augmentation implemented, the next thing we tackled was network size. We added several Dense layers, of half the size of each previous layer (starting from 640 for half of the feature extractor's output) until the model stopped improving at 160 neurons.

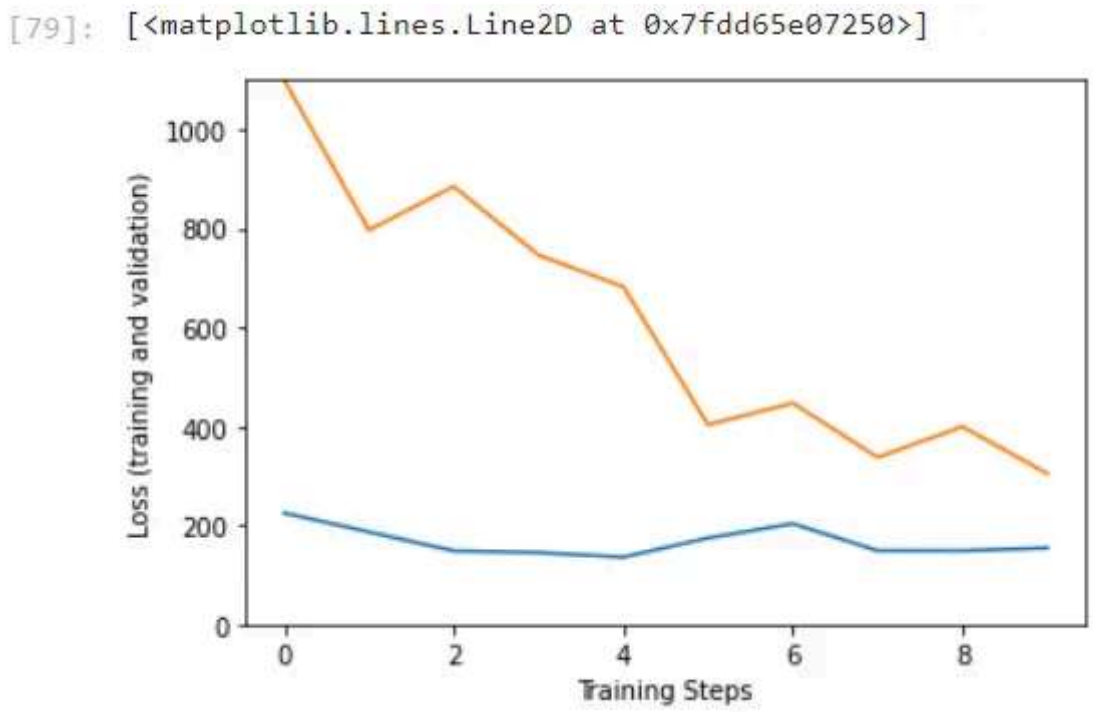

Figure 1.7 160 Neurons

Feature extractors proved to be a fight between having many neurons and being able to adjust those neurons. We had the option to freeze or unfreeze

all neurons in the feature extractor, which led to memory issues when unfreezing all but the smallest feature extractors. Unfreezing the smaller networks showed definite improvement over large frozen networks, but still displayed clear signs of overfitting.

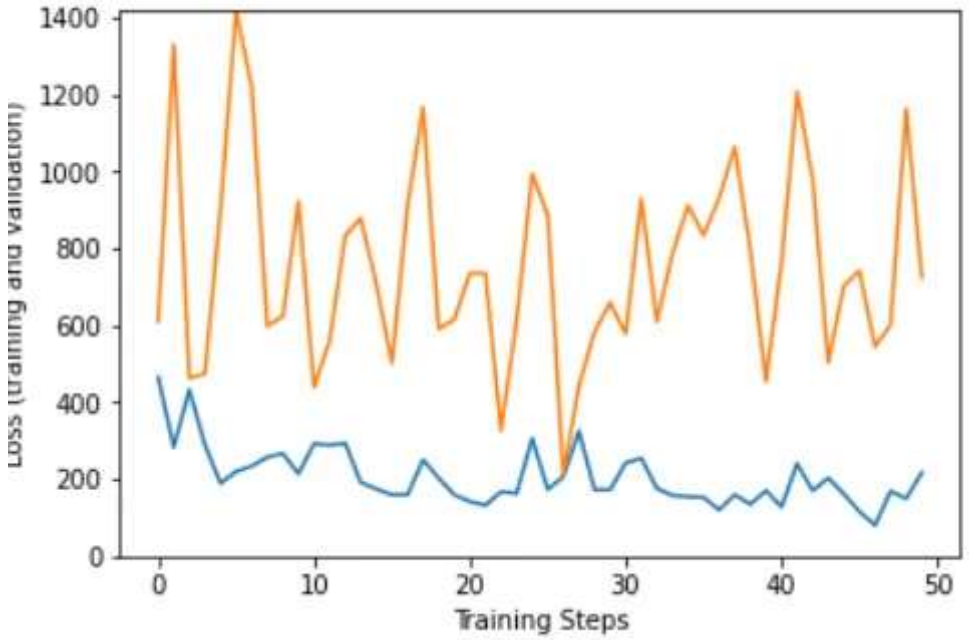

Figure 1.8 efficientnetv2-s-21k with fine tuning

The final hyperparameter we adjusted was batch size. We originally had a batch size of 16, and found it to be the best for our dataset's size. Smaller batch sizes allowed for even more overfitting, and larger ran into both memory issues, as well as overfitting (due to having too few batches, where the smaller batches led to too much weight given to any individual image)

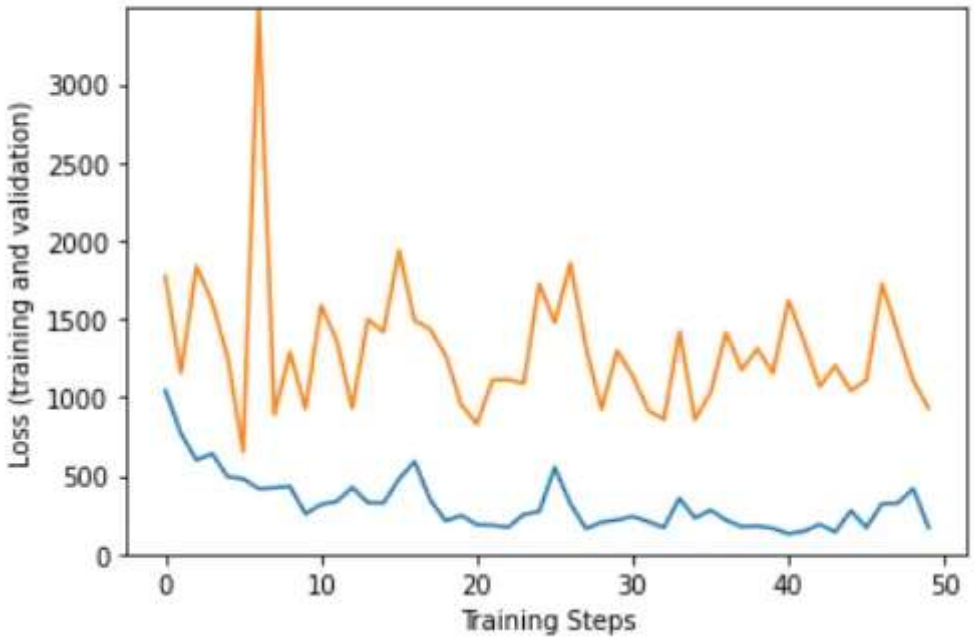

Figure 1.9 batch size of 20

The final model configuration is outlined below:

- Feature Extractor: EfficientNet v2 S 21k
- Dense Layers: 640 nodes, 320 nodes, 160 nodes, 1 output node
- Dropout Rate: 0.5
- L2-Regularization Rate: 0.01
- Batch Size: 16
- Number of Training Epochs: 50

Results:

Our model learned to make predictions within the range of 5 to 50, however, the predicted count is almost always off by several neurons. The model appears to have learned to make predictions that are close to the average number of neurons across all photos, in the same way, that a Recurrent Neural Network might learn to output mostly whitespace because that is the most common character in English text samples. This behavior caused the model to occasionally make very accurate predictions, as shown in Figure 2.0, however, the majority of predictions looked like those in Figure 2.1.

Good Prediction:

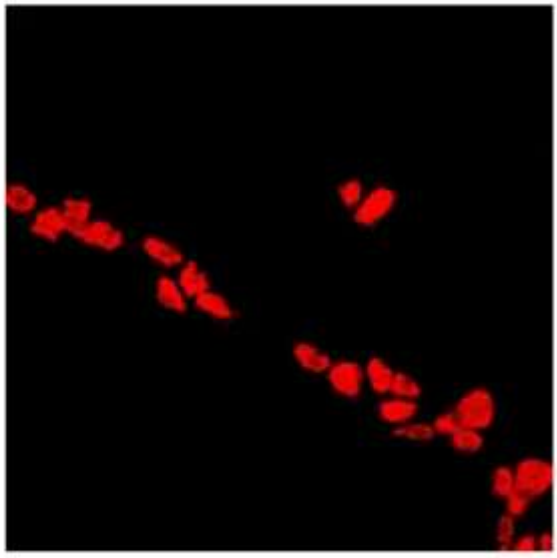

Figure 2.0Good Prediction

Poor Prediction:

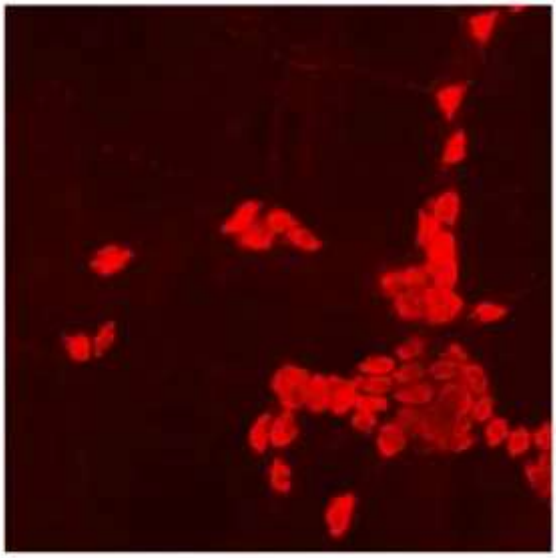

Figure 2.1 Poor Prediction

Unfortunately, our model is unable to consistently produce predictions that are accurate enough to use in a neuroscience research lab. An ideal model should give neuron count predictions that are just as accurate as humans labeling the data. Such a model could be used to drastically reduce the time needed to process neuron images, with human oversight and cross-checking.

Practical Limitations:

Even though we were unable to achieve our goal of creating a neuron-counting model that is accurate enough to replace manual counting, the process of training our model was a valuable learning experience. It was more difficult than we anticipated to coordinate with a neuroscience lab to get real-world data, which limited the amount of time and data we had for training. Furthermore, it required lots of experimentation to simply create a

model that could accept our custom-formatted dataset, as opposed to the simplicity of using pre-prepared datasets in previous projects. It was also frustrating that the size of our feature extractor was limited by Kaggle's memory restrictions. In future iterations of this project, this issue could likely be solved by doing more research into Kaggle's capabilities.

Finally, we were overwhelmed by the number of model permutations due to hyperparameters such as feature extractor type, dropout and L2-regularization rate, number of dense layers, batch size, and number of training epochs. Larger neural network research projects write programs to test different model configurations, which would be a more efficient strategy than manual testing if this project is continued.

This project has given us practical experience in data collection and neural network model design. We believe that developing a model that can be quickly trained to count neurons in different types of cell images would be extremely helpful to neuroscience labs. We hope that the designs and experiences shared in this post will inspire others to improve on our model!

Neural Networks Neuroscience