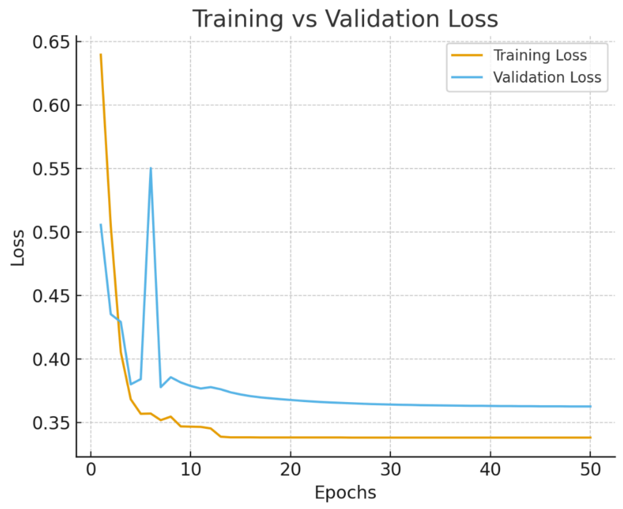
Trevon Woods
Computer Vision - ITAI - 1378
Patricia McManus
09/25/2025

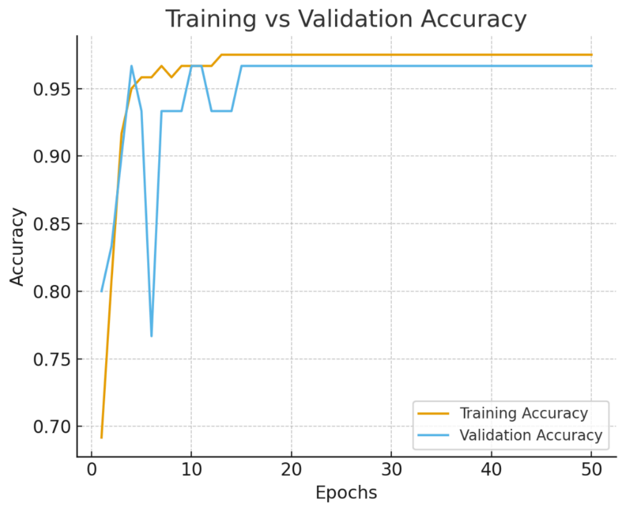
Chihuahua or Muffin – Linear

The main objective of this assignment was to train a neural network classifier that determines whether an image is a Muffin or Chihuahua. We used several techniques for this lab starting off with the creation of a neural network that subclasses the nn.Module base class. In the code cell we defined the height and width of our images that will be fed into the model. I made a couple additions to the internals of the network but I will mention that later. Next we did the loading and preprocessing of the dataset images using torchvision.datasets.ImageFolder which houses the images and torchvision.transforms. We use transforms to resize, normalize the pixel values, and convert the images to tensors. After we created the dataloaders by using the function torch.utils.data.Dataloader to load and batch all of the data into an iterable object. Moving on we defined the loss function we use to determine the model's loss simply by calling the function nn.CrossEntropyLoss. We also define an optimizer called nn.SGD (Stochastic Gradient Descent) to update the weights of the model based on the loss. Last but not least, we train the model for 10 epochs, both forward and backward passes, and track the training and validation losses of each run. After training the model we visualize the performance of the model based on its predictions on the validation data.

I learned a few things during this lab such as Image Classification, and Hyperparameter Tuning methods. Image Classification is when a model analyzes an image and assigns it to one or more classes. It has three main components, input, feature extraction, and inference. It is prevalent in medical imaging, self-driving cars, security (facial recognition, surveillance), and things like eCommerce for product search using the image or image tagging. As for Hyperparameter tuning, it is when we change and train our model using different values for the settings that have an effect on different aspects of the model's learning. For this lab the hyperparameters were learning rate, batch size, epochs, optimizer, number of layers/layer dimensions, image size, and augmentation transforms.

I just want to preface this before moving forward, any mention of accuracy from this point onward is referring to the validation accuracy unless otherwise stated. The challenge I encountered in this lab was my model's accuracy being around 56% after training — using all of the base hyperparameter values and an image size of 28 x 28. To remedy this I dove right into tweaking hyperparameters. I started by trying to increase the number of epochs to around 50 but the model didn't learn anything new the longer the training went. I kept the epochs at 50 and tried upping the image size to 128 x 128 which seemed to improve my models accuracy to around 83%. With this improvement I decided to leave the image width and height at these values and moved onto trying to increase the accuracy by tweaking the learning rate and optimizer. I tried a smaller learning rate of 0.01 and 0.001, these values didn't really help the model achieve a better accuracy instead the accuracy bounced around 60-80%. Since that didn't work I put the learning rate back to 0.1 and tried out another optimizer. The optimizer I used this time was Adam but this made the accuracy fall back into the 50% range so I went back to SGD. Moving on I tried different values for the batch size because depending on the batch size a model could learn more or less features. Initially I had a batch size of 32 and I decreased the batch size to 5 for a training cycle but my accuracy dropped to around 60-78%. Then I increased the size to 15 and obtained around 90-93% accuracy on each epoch. This was good news, but I wanted to obtain 100% accuracy so I kept going. I tried some higher batch sizes like 20 or 60 but they didn't improve the model, they mostly decreased the accuracy. Next I moved onto playing with the internals of the network, the layers, by adding or taking away some layers. One setup of note was a 5 layer setup where the output of layer 0 was 512. This, believe it or not, made my accuracy drop to around 86% which was odd because typically the larger the hidden layers the more complex features the model learns. It was also the same for any other configuration whether I added more or less — the model's accuracy did not improve. So finally I decided to try out larger image sizes as inputs. Nothing improved the accuracy until I reached around 512 x 512. With this I was finally able to achieve a little better accuracy at around 97.5% training 96.6% validation accuracy. The model showed some instability for the validation runs in the beginning but it still ended up converging at around 96.6% early around 15 epochs.





This is not the first time I've done image classification but this lab helped me get some additional hands-on experience particularly in the hyperparameter tuning department. It is surprisingly a lot of work to get a model to stably converge at an optimal training and validation accuracy. I gained insights into how one the wrong hyperparameter value can drop my models accuracy significantly or be the reason it does not learn important features. I can see how this whole process can be overwhelming for some developers because there are so many aspects of training that need to be factored in and considered. In my opinion the most important factor to the success of my model was the batch size, this hyperparameter allowed my model to jump from around 83-86% accuracy all the way up to 93-96% accuracy. The potential real world applications for the techniques learned in this lab could be for surveillance footage or medical imaging. We could use this model to look at x-ray images of a person and — depending on what we train our model to look for — it can tell us if there are any abnormalities in the images.