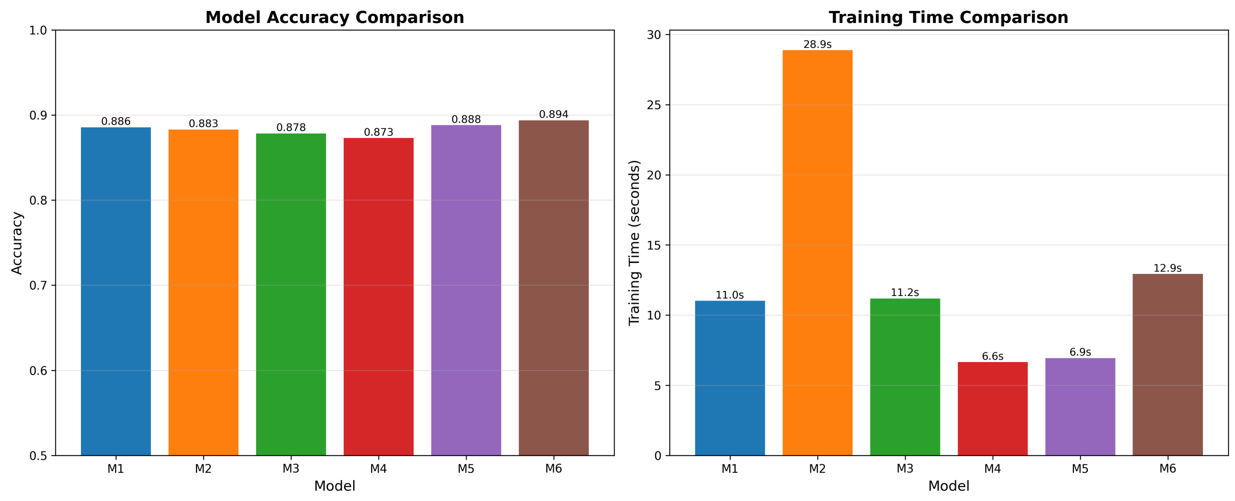
Bode Packer
Math 5750/6880: Mathematics of Data Science
Project #3 Final Report
December 3, 2025

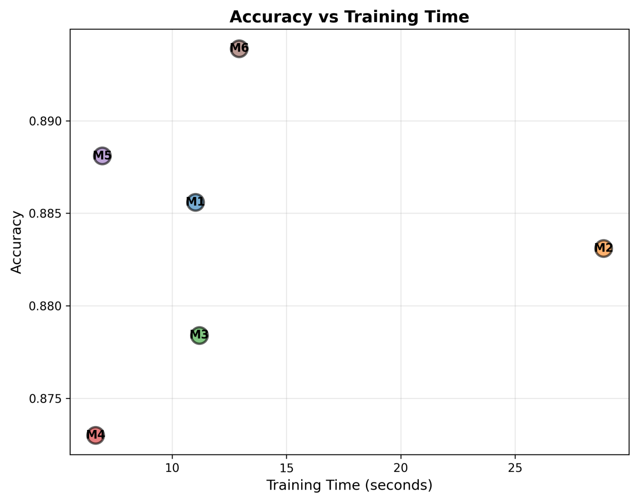
My GitHub Project3 repository is located here:

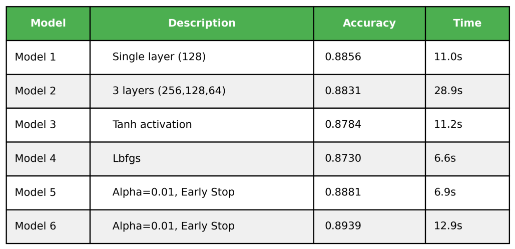
https://github.com/bodePacker/Math-for-DS-A3

1. Fashion-MNIST image classification using sklearn

After training the models and creating the corresponding charts (Figures 1 and 2), we can see several interesting things. Firstly, the default parameters used in the MLPClassifiers from SKLearn are already quite well optimized. This makes sense since SKLearn is a public library designed for accessible data analysis tools, but I felt was a needed aside as all accuracies are quiet good. Analyzing the six MLPClassifier variants tested, each one has a single parameter change to observe its impact on training speed and accuracy. Model 1, which is our control, used a hidden layer size of 128 (an increase from the default 100) and reduced the maximum iterations to 50 (down from 200) to speed up training. This resulted in a reasonable training time of 11.0 seconds and a strong accuracy of 0.886. Model 2 increased the depth by using larger hidden layers (256, 128, 64), theoretically boosting the model's capacity due to a much higher parameter count. However, this led to a substantial increase in training time to 28.9 seconds and actually decreased accuracy compared to the base model. This drop in accuracy is likely due to overfitting in which there are too many parameters on a simple dataset which often can hinder generalization. Model 3 experimented with a tanh activation function instead of ReLU. This resulted in a slightly faster convergence compared to Model 1, but with lower accuracy. Model 4 switched the solver from ADAM to L-BFGS, in which we use the Hessian instead of momentum plus RMSProp. While this drastically sped up training, it also significantly reduced accuracy, making it a trade off to consider. Model 5 applied higher regularization along with early stopping, which greatly decreased training time and improved accuracy. Finally, Model 6 combined information from all the previous tests and uses more hidden layers (128, 128), higher regularization, early stopping, an adaptive learning rate, and a higher iteration limit. This model achieved training times similar to the base while improving accuracy by about one percent, which is more than any other method accomplished by itself. These findings align with content from class as well with my own MLP experience. In general more parameters and increased regularization can improve performance, provided training times remain under control. Also looking at Figure 2, which analyzes the base model, the best single optimization model, and our hand tuned model, we see that all of the models struggled with shirts vs t-shirts, but overall did quite well on almost every other category. Each of the clothing articles have some recognizable feature that even a basic model is able to capture, and then from there we make minor optimizations to the way we handle said information inside the model to produce a more accurate output.

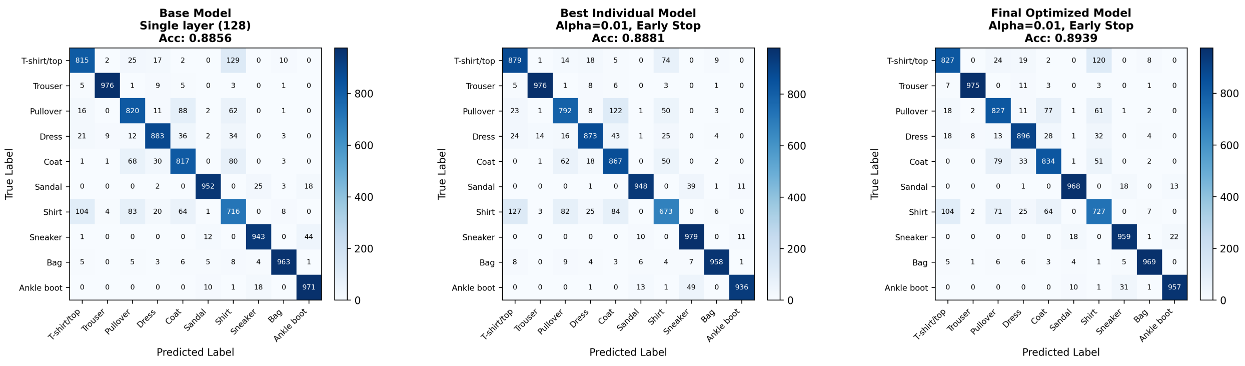






Model	Description	Accuracy	Time
Model 1	Single layer (128)	0.8856	11.0s
Model 2	3 layers (256,128,64)	0.8831	28.9s
Model 3	Tanh activation	0.8784	11.2s
Model 4	Lbfgs	0.8730	6.6s
Model 5	Alpha=0.01, Early Stop	0.8881	6.9s
Model 6	Alpha=0.01, Early Stop	0.8939	12.9s

FIGURE 1. This chart shows the 5 different model accuracy, training time, the relationship, and what the unique characteristics of each model are.



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	815	2	25	17	2	0	129	0	10	0
Trouser	5	976	1	9	5	0	3	0	1	0
Pullover	16	0	820	11	88	2	62	0	1	0
Dress	21	9	12	883	36	2	34	0	3	0
Coat	1	1	68	30	817	0	80	0	3	0
Sandal	0	0	0	2	0	952	0	25	3	18
Shirt	104	4	83	20	64	1	716	0	8	0
Sneaker	1	0	0	0	0	12	0	943	0	44
Bag	5	0	5	3	6	5	8	4	963	1
Ankle boot	0	0	0	0	0	10	1	18	0	971

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	879	1	14	18	5	0	74	0	9	0
Trouser	5	976	1	8	6	0	3	0	1	0
Pullover	23	1	792	8	122	1	50	0	3	0
Dress	24	14	16	873	43	1	25	0	4	0
Coat	0	1	62	18	867	0	50	0	2	0
Sandal	0	0	0	1	0	948	0	39	1	11
Shirt	127	3	82	25	84	0	673	0	6	0
Sneaker	0	0	0	0	0	10	0	979	0	11
Bag	8	0	9	4	3	6	4	7	958	1
Ankle boot	0	0	0	1	0	13	1	49	0	936

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	827	0	24	19	2	0	120	0	8	0
Trouser	7	975	0	11	3	0	3	0	1	0
Pullover	18	2	827	11	77	1	61	1	2	0
Dress	18	8	13	896	28	1	32	0	4	0
Coat	0	0	79	33	834	1	51	0	2	0
Sandal	0	0	0	1	0	968	0	18	0	13
Shirt	104	2	71	25	64	0	727	0	7	0
Sneaker	0	0	0	0	0	18	0	959	1	22
Bag	5	1	6	6	3	4	1	5	969	0
Ankle boot	0	0	0	0	0	10	1	31	1	957

FIGURE 2. This chart shows the Confusion Matrices for the base model from SKLearn and then the best configured model that I was able to make.

2. Pytorch

Having worked with Pytorch previously in my Machine Learning and Deep Learning classes, I am pretty familiar with how it works and its benefits. Pytorch is a super useful library and now that I am taking Deep Learning in which we have to write the underlying functions that Pytorch supplies by hand, I am even more grateful that it exists! I did learn going over the docs that there is an option to use MPS or "Metal Performance Shaders" which is a CUDA replacement on Mac. This is a great discovery as the code I wrote for Part 3 would run about 5 times slower without it on my Mac. This also means that I can work on these ML assignments on the go instead of using my desktop or CoLab. I also learned that PyTorch comes with a built in dataset loader and converter. Having manually downloaded and converted some datasets into tensors in the past, this is a nice time saver and makes sense to be integrated. Also as a note in future years, pytorch appears to have FASHION-MNIST as a default dataset, which would allow us to not use the tensorflow library at all, saving lots of space in my python environment, and simplifying the preprocessing code!

3. Fashion-MNIST image classification using PyTorch

When classifying Fashion-MNIST with PyTorch versus SKLearn, we have found a couple of interesting observations. As an overview, I tested four models using PyTorch: a basic Neural Net, similar to the MLPClassifier in Scikit-Learn, a Convolutional Neural Net with two layers, a basic Vision Transformer, and ResNet-18 based model with transfer learning to fit it to our dataset. Doing an analysis of the different architectures, starting with the basic Neural Net, it was interesting to see how much the validation accuracy fluctuated over the first 15 epochs. This is likely due to the Adam optimizer overshooting optimal weights or just a sample variance from the validation due to it being relatively small. Looking at the architecture I implemented for this model, it was super simple, with just two blocks. Each of those blocks has a linear layer, a batch norm to normalize the activations, RELU to propagate the useful values, and then a dropout of 0.3 to prevent overfitting. Two of these blocks and then an output layer from 128 to 10 gives us a simple but effective NN. I was impressed with the final **test accuracy** of 89.29%, which narrowly beat the MLPClassifier from Scikit-Learn. I believe this demonstrates the optimization and utility of Scikit-Learn, as it required significantly less configuration compared to the PyTorch implementation. This also showcases, that with some basic ML knowledge, a simple NN in PyTorch can be effective for these types of tasks.
Moving on to the actual model we want to examine, the CNN. This model was the best (other than RESNET-18) with a **test accuracy** of 92.4%. This makes sense as CNNs are some of the best models for image classification. My model consisted of two convolutional blocks, each comprising a convolutional layer and a MaxPooling layer, followed by two fully connected layers that performed the actual prediction. This takes the image data and extracts the relevant information, allowing the fully connected layers to make more accurate predictions. This feature extraction enables it to perform better than a basic fully connected network, without significantly increasing training time.

Next up is the Vison Transformer, which I chose to attempt to train as a personal challenge and to see how it would stack up on a small dataset against a CNN. PyTorch did most of the heavy lifting, but it was significantly more complicated, requiring the splitting of the image into patches, keeping track of positional embeddings, and utilizing the CLS token for actual classification. In the end, it was less accurate than all of the other models with a **test accuracy** of 87.35%. This was likely due to having a small set of training data for the transformer to generalize across, or just to a minor mistake in the implementation on my part.

Finally, the most challenging and time consuming implementation, the RESNET-18 based model that was minorly altered to work with Fashion-MNIST. In this case, we took a pretrained RESNET-18 model from the torchvison library and replaced the final layer with our own FC layer to get the 10 classification outputs that Fashion-MNIST has. To get this model to work with said dataset, it needed to be upscaled from 28x28 to 226x226, as well as converted from gray scale to RGB. I used the included transform class in PyTorch to do this while loading the data to simplify it. I attempted to do this manually, by converting from a PIL image to a transformed tensor, but it was overly complicated and slow. This RESNET-18 based model was the best model overall, while also having vastly longer training times, with a **test accuracy** of 94.37%. As this is a supremely larger and more complex model, this score is lower than I initially expected, but anything over 90% is excellent.
In conclusion, there are many different architectures and parameters that can be used to perform classification, each with its pros and cons. I would say for smaller datasets like Fashion-MNIST at around 70,000 images, a CNN is by far the most effective and quickest model to use. With my basic version reaching 92% accuracy and a more complex one likely approaching the 95% boundary, it feels like the clear choice. For larger and more complex datasets (100k+ images or 15+ classes), vision transformers or ResNet-based models will pull ahead as the sample space becomes increasingly complex. Below are the training graphs for each of the models talked about in this section.

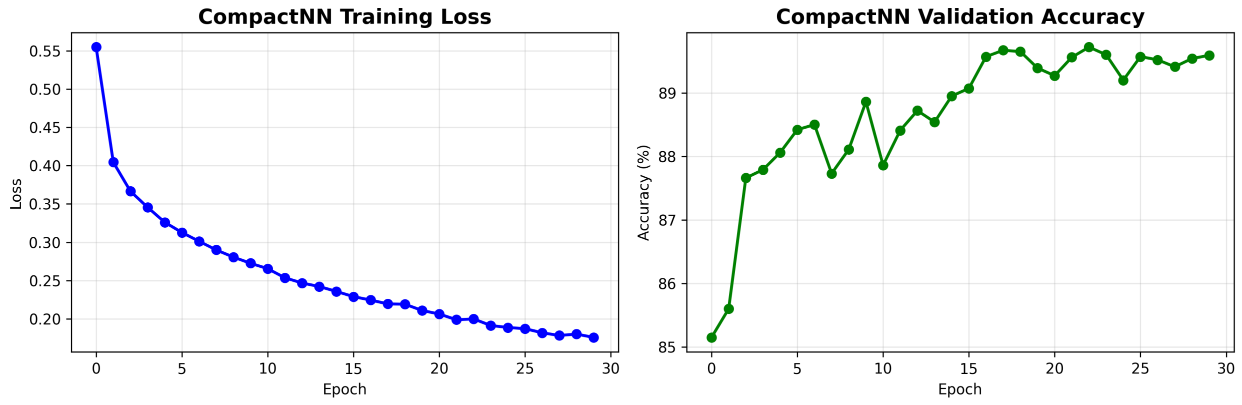


FIGURE 3. This chart shows the training loss and validation accuracy over the epochs for the basic compact NN.

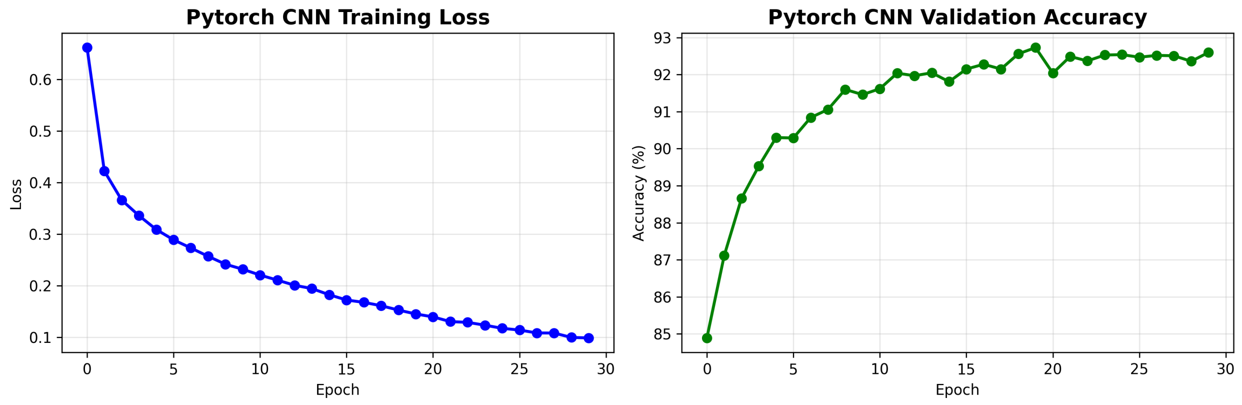


FIGURE 4. This chart shows the training loss and validation accuracy over the epochs for the Convolutional NN.

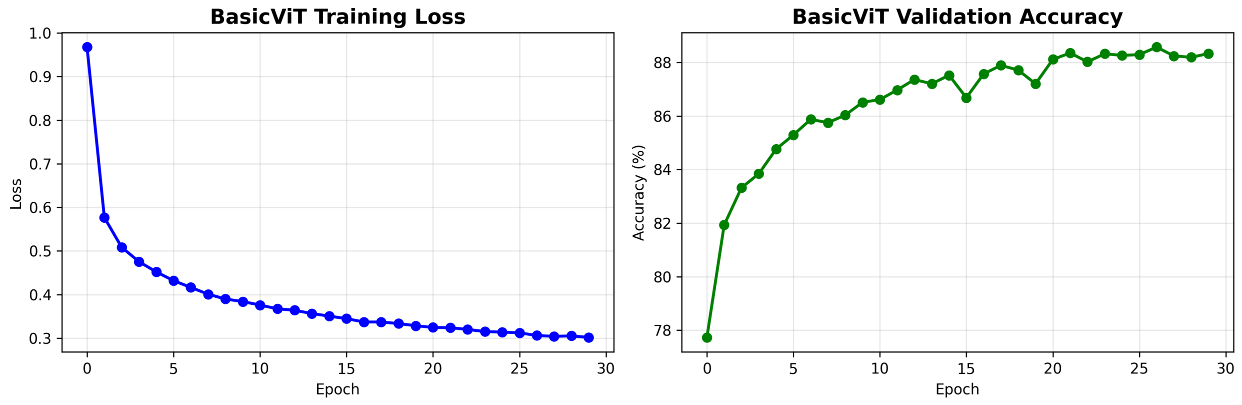


FIGURE 5. This chart shows the training loss and validation accuracy over the epochs for the basic vision transformer.

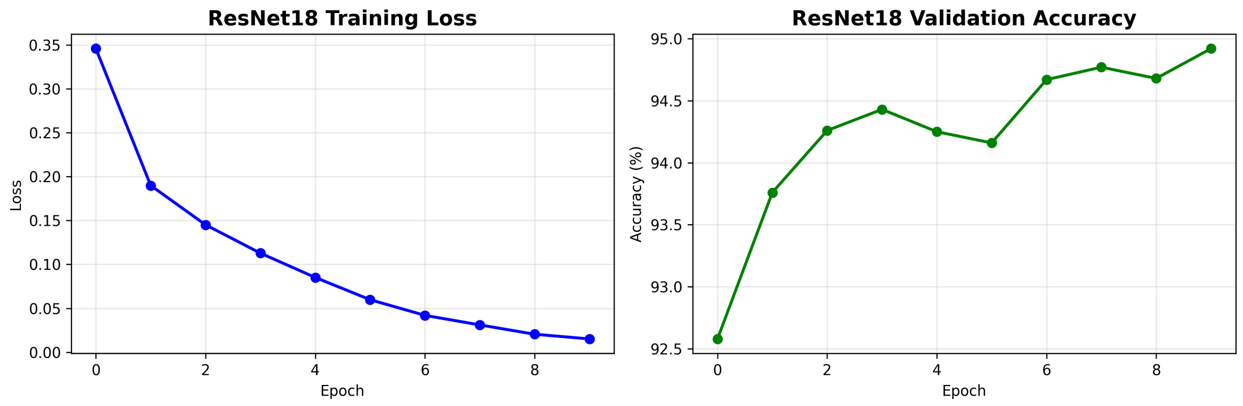


Figure 6. This chart shows the training loss and validation accuracy over the epochs for the RESNET-18 based model.