Student Number: 234558

1. Abstract

An investigation into 6 key hyperparameters for the task of classifying FashionMNIST images. This included an exploration of the hyperparameters, and a Random Search of some ranges found to be optimal for maximising accuracy and minimising loss.

2. Methodology

2.1. Approach

Initially I approached this project with intent to make it as performant as possible, so started with a (somewhat modular) Convolutional Neural Network, and it performed well, but there was little difference in the first hyperparameter tests I performed (Filter Size and Batch Size, and to a lesser extent, Batch Norm, and Max Pool), which would make investigating differences in hyperparameters difficult. Based on this, I switched tactics to instead construct a modular fully-connected Neural Network, with the added benefit (beyond clearer performance differences) being that some of the hyperparameters I found interesting to investigate from a theoretical standpoint are FCNN-exclusive (Dropout, Width [Nodes per Layer] and Depth [Number of Layers]).

To start with my FCNN, I created a modular baseline class, *BaseMLP*, to work from and edit hyperparameters one at a time from using my function, *run_series*, to run series of experiments in averaged repeats of 3, with 10 epochs each to give each model and hyperparameter value a fighting chance. This baseline also automatically decreased layer size from end to end of the NN to avoid mismatching widths.

From here I explored several hyperparameters of interest in isolation with several different values for each, then performed a random search of 50 trials, based on the acceptable ranges I found in my exploration.

In experiments, a Training/Validation split was used to train and validate models ability to generalise before validation on testing data. One optimiser, Adam, was used for all networks, and was provided with all parameters, and one constant step-based learning rate scheduler was used.

2.2. Hyperparameter Selection

Analysing once at a time (so judged based on deviation from my baseline model), I explored Learning Rate, Non-linearity (of the Activation Function), Hidden Layer Widths, Number of Hidden Layers (or 'Depth'), Dropout, and Batch Size. Table 1 shows the hyperparameters explored and their values, along with the optimal ranges/values I identified shown in bold. Performance in this section refers to the ability of a model to both increase its accuracy to a higher value than its peers, and decrease its loss lower than its peers.

Based on this, I defined the following ranges to perform a random search with:

- Learning Rate: 0.0005 to 0.005, half of and 5x the Optimum 0.001 in order to investigate its order of magnitude.
- Activation Function: ReLU, as it was found to be best and though LeakyReLU was a close second, this was likely due to similarities to ReLU, so was excluded to simplify the selection process.
- Hidden Width: 500 to 1000, as this is the neighbourhood of the input size = 784, and performed best in exploration.
- Hidden Depth: 4 to 8, as networks on both sides of the range performed best in exploration.
- Dropout: 0.1 to 0.4 as the best performance was observed between these two.
- Batch Size: 64 to 128, as both performed similarly and better than the other values explored.

Please see individual graphs in my python notebook for more detail on each tested model's performance.

2.2.1 Random Search

Once I had my ranges (see above list), I performed a Random Search of 50 trials, randomly selecting values in the above ranges, training and validating the produced models and adding the next best performers' hyperparameters to a list based on validation accuracy. The best of the top four recorded was from the 32nd trial with the following values:

- Learning Rate = 0.002
- Hidden Width = 569
- Hidden Depth = 8
- Dropout = 0.333
- Batch Size = 122

I then created my final model using these (in theory) near optimal values (at least for my architecture).

3. Results

Note: Full Training Loss and Training Accuracy graphs can be found in the code (Python Notebook), what follows is averaged accuracy curves to roughly illustrate the change in performance from each hyperparameter value.

Also of note: accuracy isn't the full story, a combination of accuracy and how minimal models made the loss is what ultimately determined the success of a model in my exploration, so if my ranges look slightly misplaced, its because they had better loss minimisation than their slightly more accurate neighbours. E.g. ReLU minimised loss to around 75 whereas LeakyReLU only minimised to 100, but LeakyReLU performed slightly better in terms of pure accuracy.

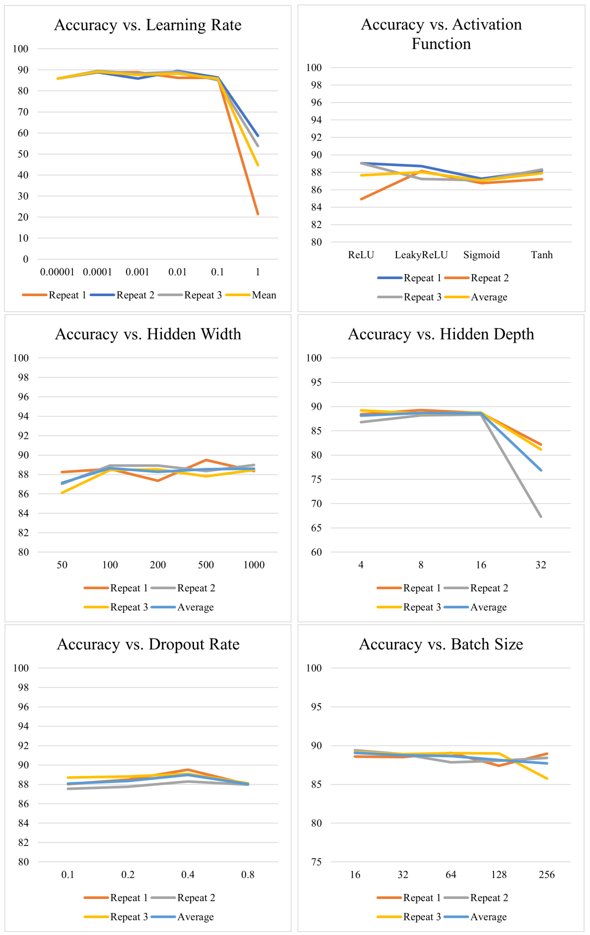

Figure 1. Accuracy Curves for each investigated Hyperparameter.

4. Discussion

4.1. Conclusions

My hyperparameter search was partially successful as I successfully isolated the best values from my explored ranges, but these were not very useful in my random search as I found another ok model rather than a good combination of my findings in the hyperparameter search.

I was able to find a generally good model in my final model2, but this was true for a lot of the models I investigated before performing a random search. This leads me to believe my random search was flawed in terms of being either too restrictive in value ranges, or not extensive enough to find a good combination of my ranges 1.

In reference to the correlation matrices in Figure 3...

In my final model, the first strongest non-self correlation was between Sneakers (Class 8, shown here as 7 due to 0-indexing) and Ankle Boots (Class 10, shown as 9). So it seems that my model learnt a strong correlation between sneakers and ankle boots which makes sense as they're the only two types of closed-toed footwear in the data-set. Strangely it also found correlation between sneakers and trousers (Class 2, shown as 1) which I can only assume is down to the examples I've seen taking up similar amounts of their respective images.

My model has many more weaker correlations whereas ResNet18 shows just a few stronger ones, namely between pullovers (Class 3, shown as 2) and shirts (Class 7, shown as 6), which makes sense as an association since they are often very similar pieces of clothing, and the strongest non-self correlation being between sandals (Class 6, shown as 5) and sneakers. A similar association to the sneakers/ankle boot combination learnt by my model.

So my model learnt an association between sneakers

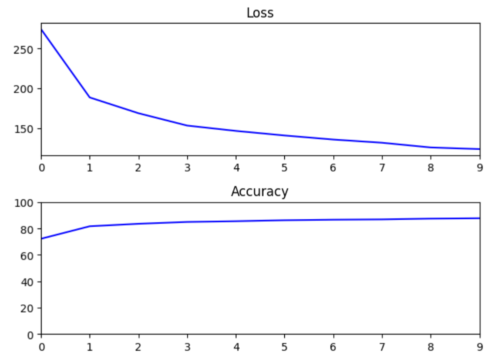

Figure 2. Training loss and accuracy of my final model.

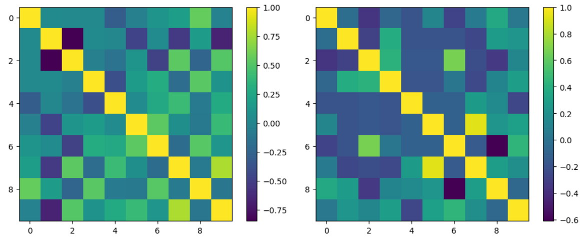

Figure 3. Correlation Matrices for my final model (Left), and ResNet18 (Right).

and ankle boots, whereas ResNet18 demonstrated an association between sneakers and sandals. Possibly due to its training data-set being higher resolution, it was able to generalise the smaller footwear together but not the larger footwear. This could have been since larger footwear less detail to sort since they're often single-colour whereas sneakers and sandals are more distinctive was harder for it to discern.

4.2. Retrospectives

- My chosen architecture in making a single modular network to avoid creating many similar but different ones may have restricted experimentation such that I did not achieve results above 90% accuracy. A more custom set of layer widths and utilising different optimisers fed different subsets of the hyperparameters would likely have improved performance.
- I didn't clean up my input data, nor did I augment it to expand it and possibly allow for more ability generalisation in my models. In terms of augmenting, were I to do this project again, I would do horizontal and vertical flips, as well as some small crops and if I was really confident in my models, some warping. I would avoid doing large crops as they could make the task extremely difficult since we only have 28*28 pixels to play with.
- Weight initialisation and different regularisation techniques should have been attempted especially as the former could have drastically sped up training.
- If I were to do this project again I would make another effort to use a Convolutional Neural Network again since I can't deny the speed benefit of CNNs over FC-NNs.

5. Appendix

Hyperparam	Values					
LR	0.00001	0.0001	**0.001**	0.01	0.1	1
Activation	**ReLU**	LeakyReLU	Sigmoid	Tanh		
Width	50	100	200	**500**	**1000**	
Depth	**4**	**8**	16	32		
Batch Size	16	32	**64**	**128**	256	
Dropout	**0.1**	**0.2**	**0.4**	0.8		

Table 1. Hyperparameter Values investigated, and my found optimal ranges in **bold**

6. Code

Can be found in the attached Jupyter/Colab Notebook *234558.ipynb*, as well as on GitHub: https://github.com/danbates1452/neural-networks.

Libraries:

- PyTorch + torchvision [4]
- TQDM [1]
- Matplotlib [2]
- Sci-Kit Learn (for Confusion Matrices and Metrics that went unused)[5]
- NumPy [3]

References

[1] tqdm documentation. https://tqdm.github.io/. 3
[2] Matplotlib — Visualization with Python. https://matplotlib.org/, 2023. 3
[3] NumPy. https://numpy.org/, 2023. 3
[4] PyTorch. https://www.pytorch.org, 2023. 3
[5] scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/, 2023. 3