Executive Report: Neural Systems Coursework

Student ID: P2963116

1 Background

This report outlines my implementation of artificial neural networks to build a classifier for use with the Fashion-MNIST [7] image dataset using PyTorch [2]. The dataset contains 70,000 greyscale images, 28x28 pixels in size. The images depict 10 categories of fashion items [Table **??**] and is split into two subsets, 60,000 for training and 10,000 for testing. A random sample of 10,000 is taken from the training set for use in validation, tests performed during training to monitor how the model performs. In training, I recorded the loss; an error metric used to adjust model parameters and while in testing a series of performance metrics such as accuracy, precision, recall and F1 Score were measured for each category.
I implemented two types of neural network when building the classifiers. The first is a Multi-Layer Perceptron or ANN [Table 2a], a collection of 'neurons' each of which calculate a weighted linear sum based on the training parameters and input. Before passing the result to the neurons in the next layer, we pass the value through an activation function, helping the model learn complex patterns in the training data.The original structure was later modified to reflect innovations as outlined in Section 2.
I also created a Convolutional Neural Network (CNN) [Table 3, Fig 1], a network type particularly effective at dealing with tabular data such as images. We perform convolution operations on the input with smaller matrices to generate feature maps and downsize the input matrix. The image is then flattened and passed through dense layers of similar structure to the ANN.

2 Findings

Initial training was completed over 30 epochs to emphasise the impact of potential issues. One of which is high validation loss compared to training loss, a phenomenon is known as overfitting.
Mitigating overfitting was achieved through modifying the ANN (due to faster training time) in a number of ways. Dropout [4] randomly disables connections between layers during training, which helps to stabilise the loss over each epoch. Similarly, batch normalisation [1] attempts to stabilise training and improve convergence speed. Both significantly mitigated the impact of overfitting in early epochs [Fig 3b].
I also experiemented with optimiser choice. Testing [Fig **??**] showed that Adam **??** was the best performing algorithm, with RMSProp **??**, SGD **??** and Adagrad **??** performing worse or having convergence speed issues.
Data augmentation was also performed, applying random transforms [Fig **??**] to training data to increase the number of samples and prevent overfitting. Use of augmented data led to worse test performance, but led to no overfitting even when training over 60 epochs. As such it is worth considering higher epoch counts when training the CNN or future models.
For the CNN we took all the above findings and trained on an augmented dataset for 60 epochs.

2.1 Key Metrics

Model	Training Time	Accuracy	Train Loss	Validation Loss	Epochs
ANN	4.5 minutes	88.43%	0.093	0.473	30
ANN (modified)	5 minutes	89.94%	0.153	0.296	30
ANN (modified+data aug)	14 minutes	87.05%	0.422	0.375	60
CNN	9 minutes	91.79%	0.122	0.225	30
CNN (data aug)	36 minutes	90.14%	0.308	0.265	60

Table 1: Key metrics after training

Data was visualised using Seaborn [5], Pandas [6] and SciKit Learn [3] through a series of graphs (Figure 3 and 4) and by plotting the Confusion Matrix (Figure 2) associated with each model.

3 Evaluation

An alternative fix to overfitting not implemented is through early stopping to prevent divergence of loss values occurring using a patience system. It would also be reasonable to adjust the architecture of my networks to reflect existing high performance classifier architectures such as ResNet, GoogLeNet and VisionTransformer, all of which would likely get higher accuracy scores when compared to my models due to their increased complexity.
Overall, while more complex, a CNN is far superior when it comes to computer vision classification task. This is likely due to the architecture facilitating the storage and understanding of spatial information by virtue of the matrix structure.

A Tables and Figures

Layer	Type	Size	Activation
Input	Image	28x28	-
Flatten	Flatten	784	-
1	Linear	392	ReLU
2	Linear	196	ReLU
3	Linear	98	ReLU
Output	Linear	10	Log Softmax

(a) Initial ANN

Layer	Type	Size	Activation
Input	Image	28x28	
Flatten	Flatten	784	
1	Linear	392	ReLU
	Batch Norm		
	Dropout	p=0.2	
2	Linear	196	ReLU
	Batch Norm		
	Dropout	p=0.2	
3	Linear	98	ReLU
	Batch Norm		
	Dropout	p=0.2	
Output	Linear	10	Log Softmax

(b) Modified ANN

Table 2: Network Structures

Layer	Type	Feature Maps	Output Size	Kernel	Stride	Activation
Input	Image		28x28x1			
1	Convolution	128	28x28x128	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.25			
2	Convolution	128	28x28x128	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.25			
	Max Pooling		14x14x128	2x2	(2,2)	
3	Convolution	256	14x14x256	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.25			
4	Convolution	256	14x14x256	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.25			
	Max Pooling		7x7x256	2x2	(2,2)	
	Global Avg. Pool		1x1x256	7x7		
	Flatten		256			
5	Linear		128			ReLU
	Batch Norm					
	Dropout		p=0.25			
6	Linear		64			ReLU
	Batch Norm					
	Dropout		p=0.25			
Output	Linear		10			Log Softmax

Table 3: CNN Structure

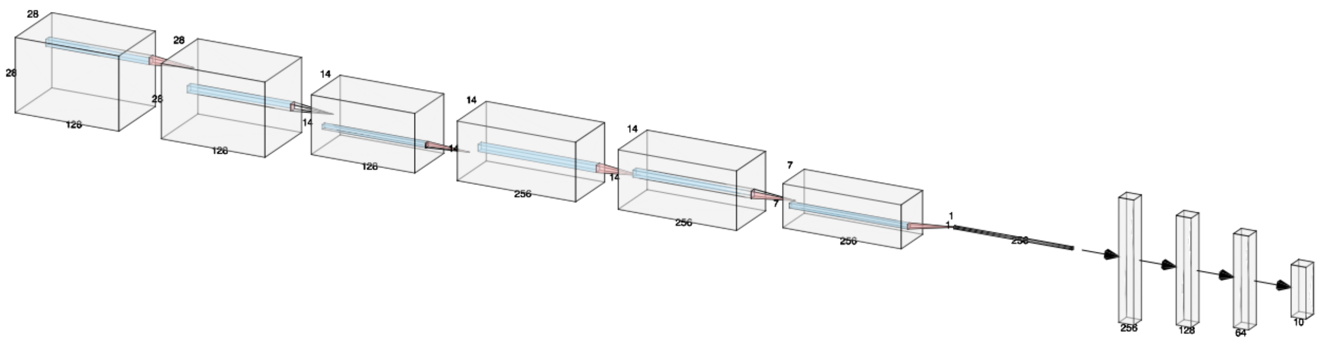
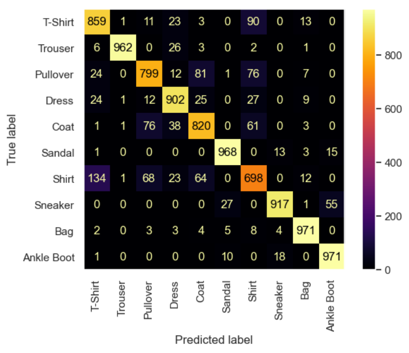
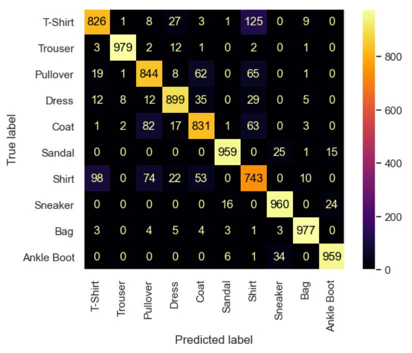


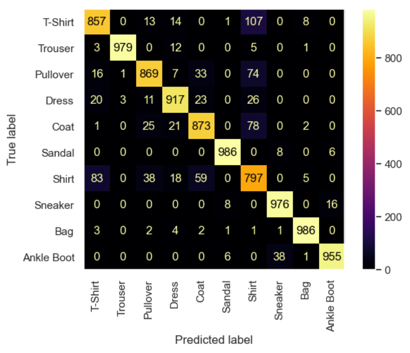
Figure 1: AlexNet-like Visualisation of my CNN Structure
Source: NN SVG [?]



(a) ANN

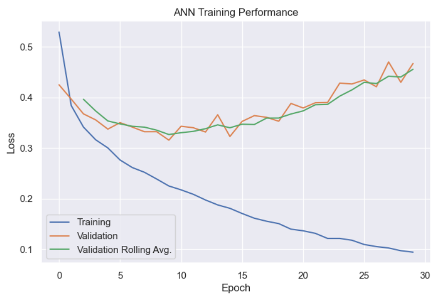


(b) ANN (modified)

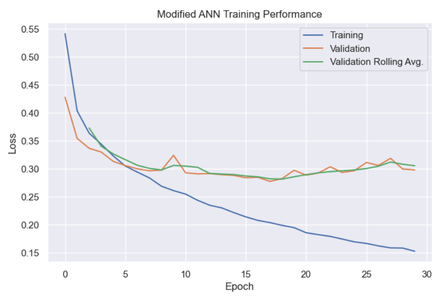


(c) CNN

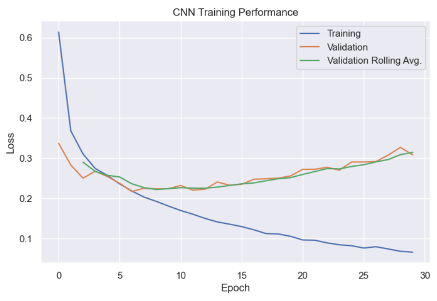
Figure 2: Confusion Matrices



(a) ANN

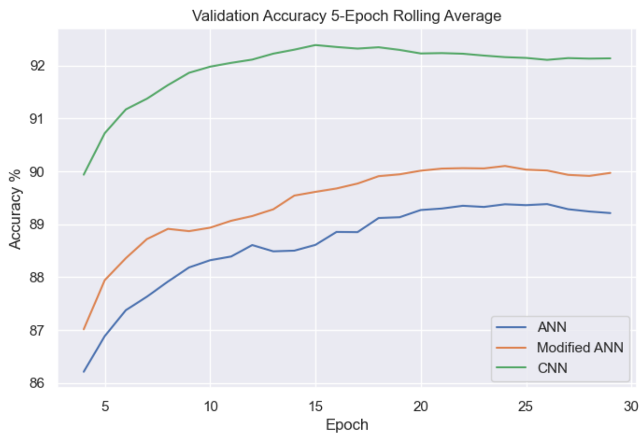


(b) ANN (modified)

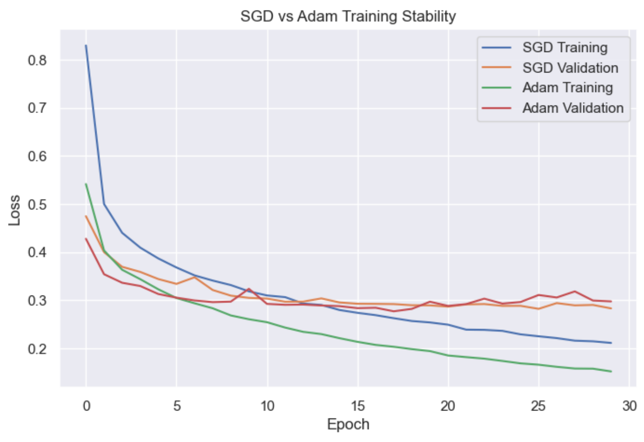


(c) CNN

Figure 3: Loss History Plot During Training



(a) Rolling Avg. Validation Set Accuracy



(b) Optimiser Comparison

Figure 4: Comparison Graphs

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[5] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[6] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[7] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.