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 [1] 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 2a] 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 3a], 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 4, 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 [Fig 2a], 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 [3] randomly disables connections between layers during training, which helps to stabilise the loss over each epoch. Similarly, batch normalisation [4] attempts to stabilise training and improve convergence speed. Both significantly mitigated the impact of overfitting in early epochs [Fig 2b].
I also experiemented with optimiser choice. Testing [Fig 3] 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 4a] 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 [Fig 4b]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	Precision (mean)	Recall (mean)	F1 Score (mean)	Epochs
ANN	4.5 minutes	88.43%	0.891	0.890	0.888	30
ANN (modified)	5 minutes	89.94%	0.901	0.901	0.901	30
ANN (modified+data aug)	14 minutes	87.05%	0.872	0.871	0.870	60
CNN	9 minutes	92.39%	0.926	0.926	0.926	30
CNN (data aug)	20 minutes	91.42%	0.917	0.916	0.916	60

Table 1: Key metrics after training

Data was visualised using Seaborn [5], Pandas [6] and SciKit Learn [7] through a series of graphs (See Appendix A) and by plotting the Confusion Matrix (Figure 6) associated with each finalised model. This lets us consider all the key metrics for our models to determine which performs best. In all cases higher is better, so we clearly see the CNN outperform here. We must note that loss with data augmentation is higher so given more time training for longer would be ideal. In addition, I have visualised examples of the output prediction of the model [Fig 7] along with some of the filters and feature maps from my CNN [Fig 9].

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. In a similar vein applying pre-trained models to the problem would also alleviate time constraints on training larger models (however it did not feel fitting to do so due to the nature of the project brief).
Overall, while more complex, a CNN is far superior when it comes to computer vision classification task. All my CNNs produced more precise models with better accuracy, recall and F1-score across all prediction categories when compared to an ANN with equivalent configuration.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[2] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[3] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[4] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[5] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[6] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[9] Alex Lenail. *NN SVG*. Oct. 25, 2025. URL: https://alexlenail.me/NN-SVG/AlexNet.html (visited on 10/25/2025).

[10] Jeremy Fix. *First steps in PyTorch. Classifying fashion objects (FashionMNIST)*. CentraleSupelec. URL: https://teaching.pages.centralesupelec.fr/deeplearning-lectures-build/00-pytorch-fashionMnist.html.

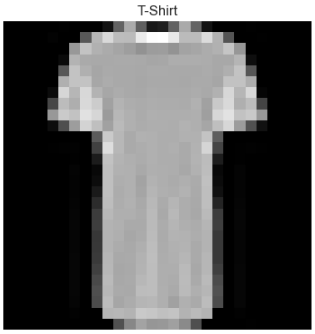
[11] Subhajeet Das. *PyTorch Feature Map and Filters Visualization*. 2024. URL: https://www.kaggle.com/code/subhajeetdas/pytorch-feature-map-filters-visualization.

[12] Overleaf. *Learn LaTeX*. Nov. 1, 2025. URL: https://www.overleaf.com/learn.

A Tables and Figures

Index	Label
0	T-Shirt
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

(a) Clothing Categories



(b) Example Image

Table 2: Fashion MNIST

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 3: 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=0.2			
2	Convolution	128	28x28x128	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.2			
	Max Pooling		14x14x128	2x2	(2,2)	
3	Convolution	256	14x14x256	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.2			
4	Convolution	256	14x14x256	3x3	(1,1)	ReLU
	Batch Norm					
	Dropout		p=0.2			
	Max Pooling		7x7x256	2x2	(2,2)	
	Global Avg. Pool		1x1x256	7x7		
	Flatten		256			
5	Linear		128			ReLU
	Batch Norm					
	Dropout		p=0.2			
6	Linear		64			ReLU
	Batch Norm					
	Dropout		p=0.2			
Output	Linear		10			Log Softmax

Table 4: CNN Structure

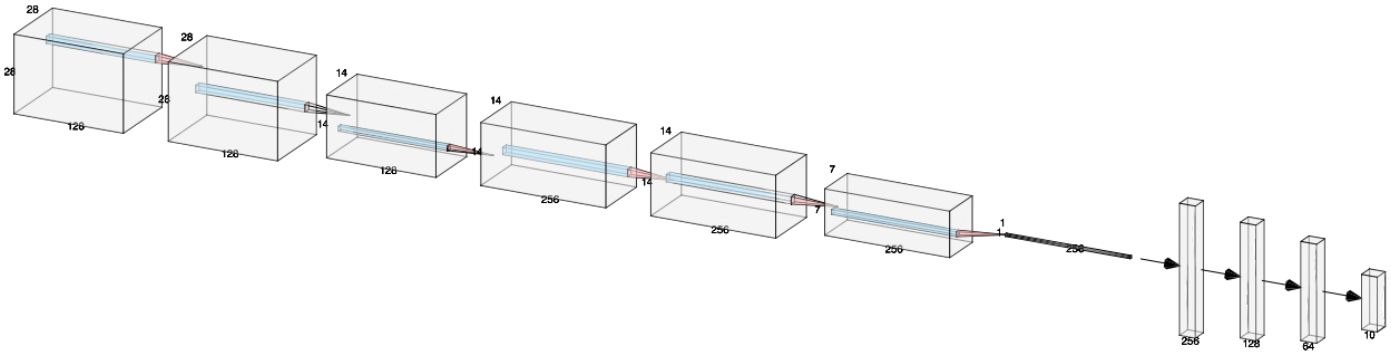
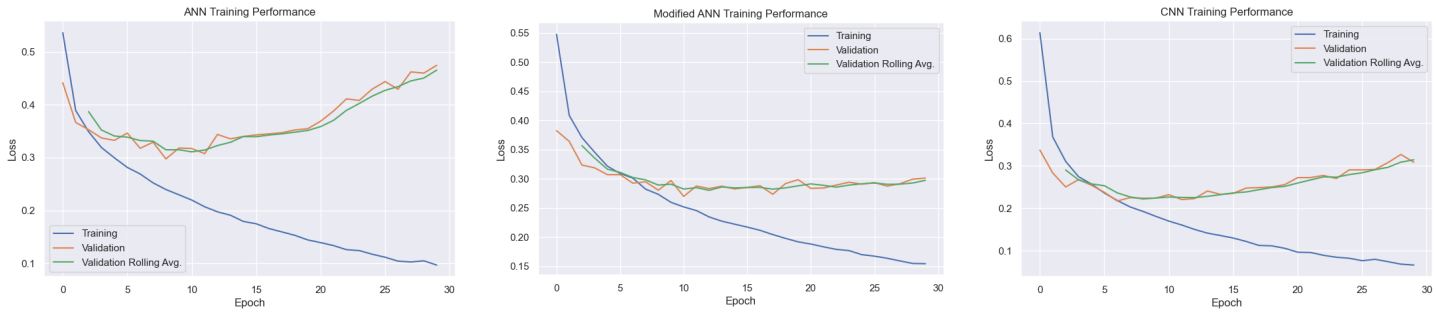
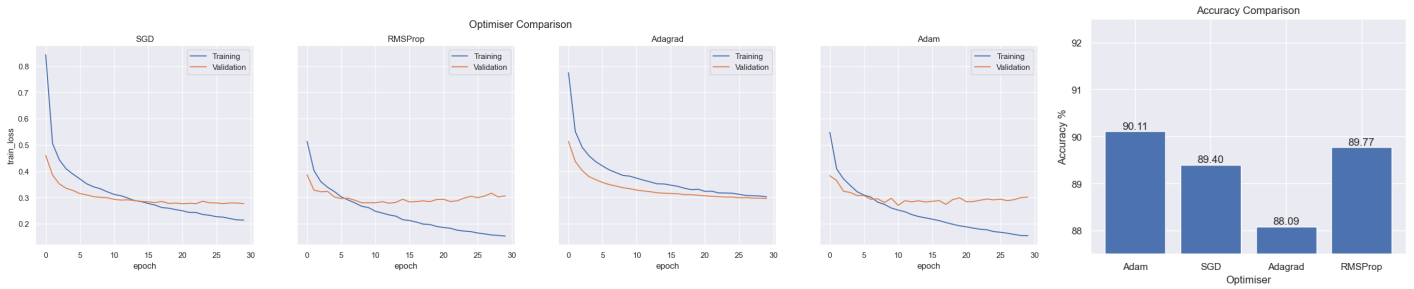


Figure 1: AlexNet-like Visualisation of my CNN Structure
Source: NN SVG [9]



(a) ANN (b) ANN (modified) (c) CNN

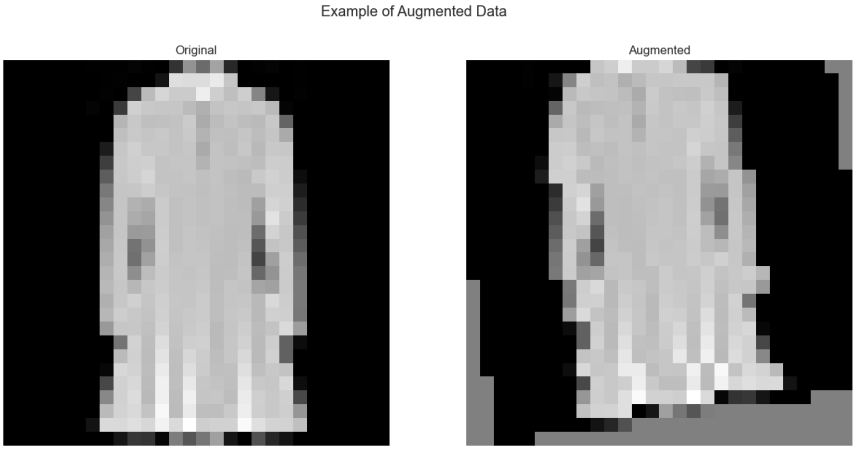
Figure 2: Loss History Plot During Training



(a) Loss

(b) Accuracy

Figure 3: Optimiser Comparison



(a) Example transform

(b) Training loss with augmented data

Figure 4: Data Augmentation

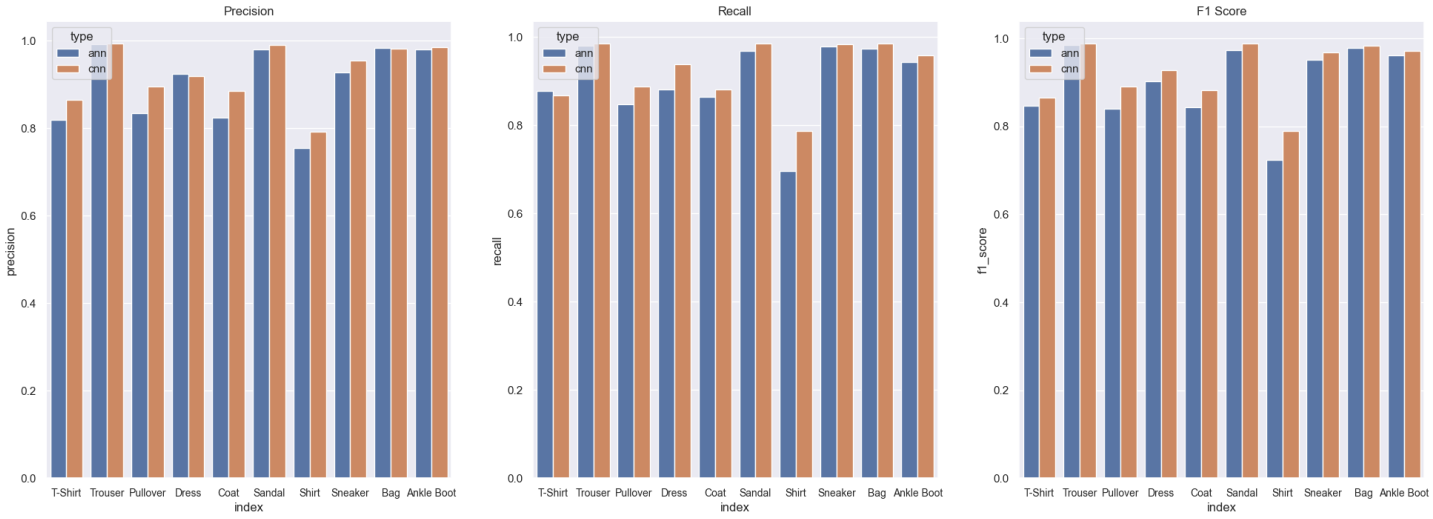
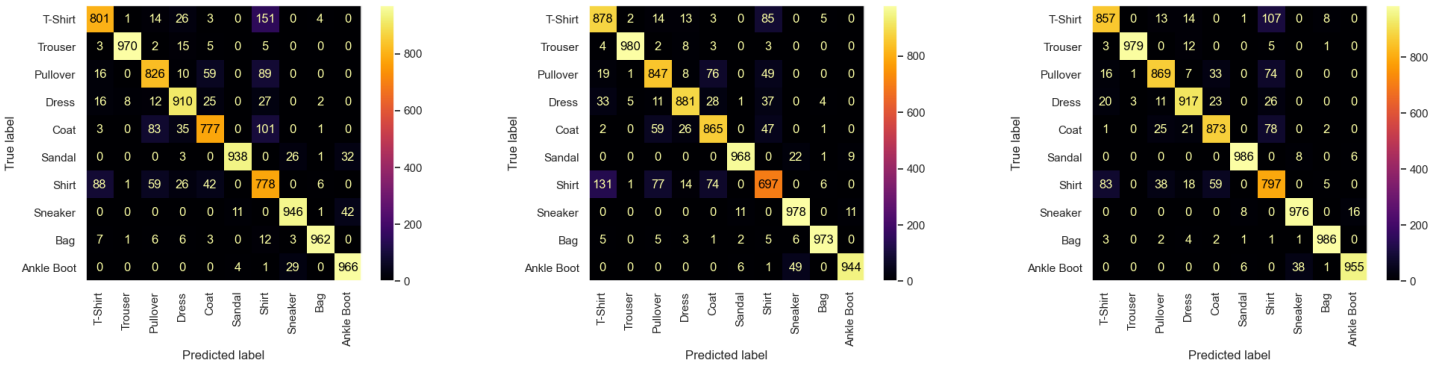


Figure 5: Metric Comparison



(a) ANN

(b) ANN (modified)

(c) CNN

Figure 6: Confusion Matrices

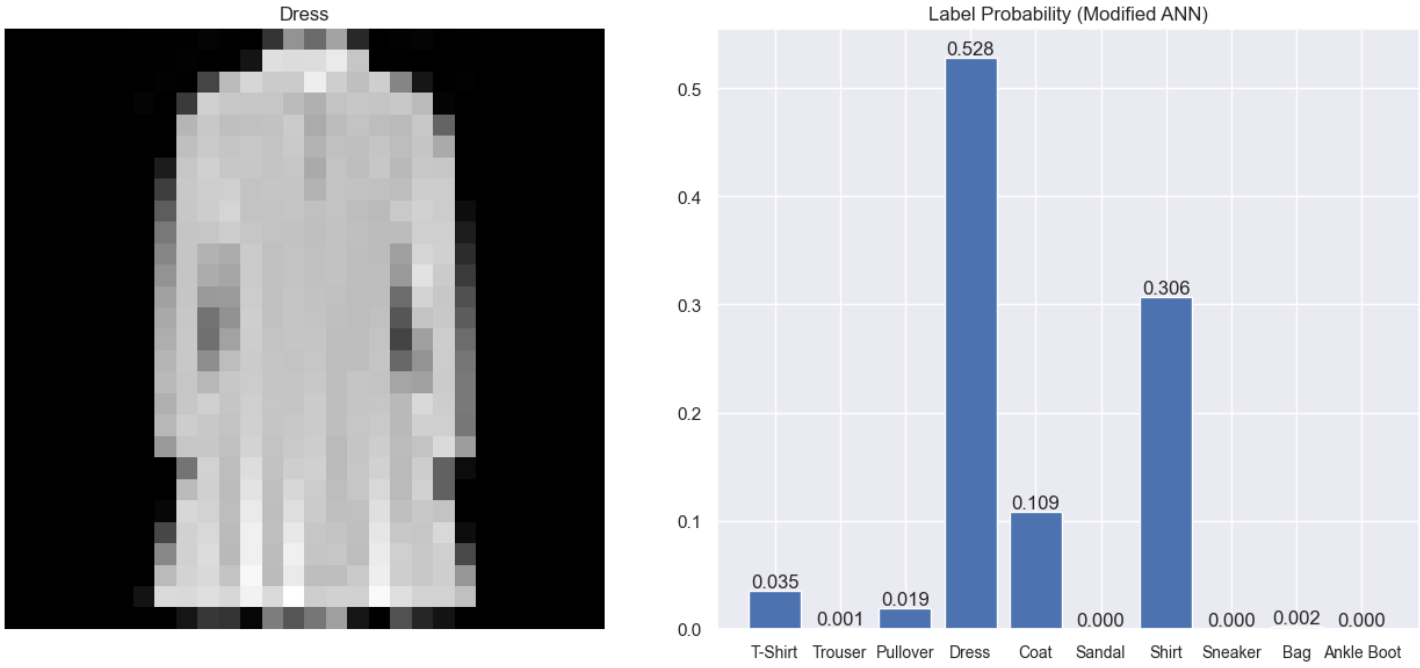


Figure 7: Example Prediction

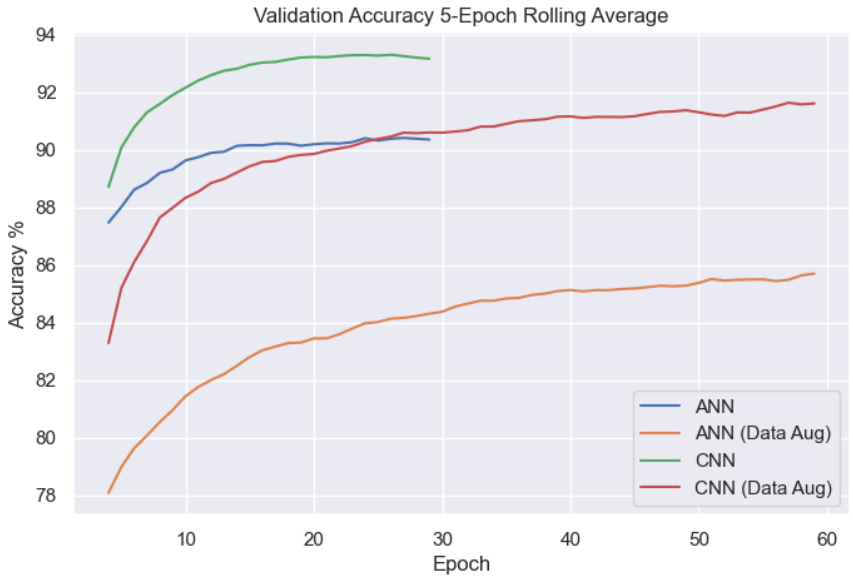
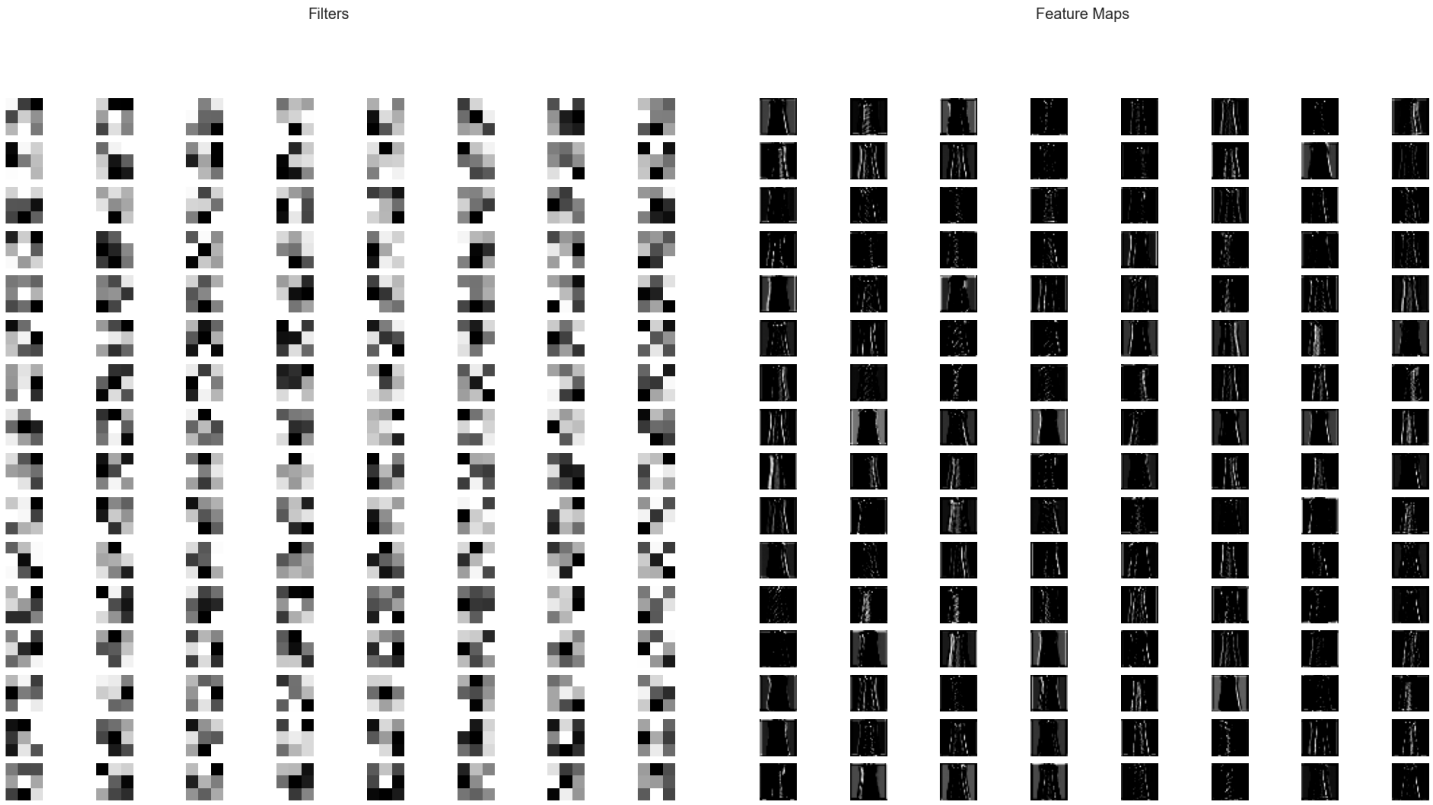


Figure 8: Rolling Avg. Validation Set Accuracy



(a) Filters (b) Feature Maps

Figure 9: Filters and Feature map extraction