Comparative Study of Deep Learning Approaches on Fashion MNIST Dataset

Yameen Ajani[1], Mohammed Farhan Baluch[1] and Jasmin Patel[1]

[1]*University of Windsor, Windsor, Ontario, Canada*

Abstract

This paper addresses the classification problem and emphasizes the importance of datasets in deep learning. The performance of a model is heavily reliant on the quality of the data, and therefore, it is crucial to use a high-quality dataset. The Fashion MNIST dataset is used in this study, which contains images of different clothing items. The objective of this research is to compare the performance of different models for image classification. Various models are developed and tested, and their results are compared to determine which model performs the best. The accuracy, precision, recall, and F1-score are used as evaluation metrics to measure the performance of the models. Additionally, the paper discusses the importance of hyperparameter tuning and how it can affect the performance of the models. The results of the models are compared and analyzed, and it is observed that some models outperform others while some show overfitting. The research demonstrates the significance of selecting appropriate architectures and techniques in building effective classification models.

Keywords

Image Classification, Deep Learning, Convolutional Neural Networks (CNNs), Dropout, Pooling

1. Introduction

With the increasing popularity of deep learning techniques in computer vision, researchers are interested in exploring the effectiveness of different deep learning architectures on the Fashion MNIST dataset. This research aims to investigate the following questions: Which deep learning architectures are most effective for fashion image classification on the Fashion MNIST dataset?

Convolutional Neural Networks (CNNs) have been the most widely used deep learning architecture for image classification tasks, including on the Fashion MNIST dataset. However, there is a growing interest in exploring alternative deep learning architectures. Answering this research question will provide insights into the strengths and weaknesses of different deep learning architectures for fashion image classification on the Fashion MNIST dataset and will inform the development of more accurate and efficient deep learning models for this task.

Final Project (COMP 8610 - Neural Network and Deep Learning)

ajaniy@uwindsor.ca (Y. Ajani); baluchm@uwindsor.ca (M. F. Baluch); patel8m6@uwindsor.ca (J. Patel)

110096721 (Y. Ajani); 110093799 (M. F. Baluch); 110095248 (J. Patel)

1.1. Dataset Description

The Fashion MNIST dataset is a popular benchmark dataset for testing computer vision algorithms, particularly for image classification tasks. The dataset consists of a collection of grayscale images representing 10 different types of clothing items. The dataset is widely used for research and development of machine learning algorithms, particularly in the field of computer vision.

The Fashion MNIST dataset is composed of 70,000 grayscale images of 28x28 pixels each. These images are categorized into 10 different classes of clothing items such as T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot. The dataset is divided into two main parts: a training set of 60,000 images and a test set of 10,000 images. The images in the dataset have been collected from a variety of sources and preprocessed to ensure they are of consistent size and shape. This makes it easier for researchers and developers to use the dataset for benchmarking and testing different machine learning algorithms.

The Fashion MNIST dataset is a challenging dataset that is frequently used to benchmark machine learning algorithms in the field of computer vision. Compared to the original MNIST dataset, which consists of images of handwritten digits, the Fashion MNIST dataset is more complex and more challenging. The clothing items represented in the dataset are often similar to each other, which makes it more difficult for algorithms to differentiate between them. This makes it an ideal dataset for testing and improving the accuracy of machine learning algorithms, particularly for image classification tasks.

The Fashion MNIST dataset has been widely used for research and development in the field of computer vision. It is often used as a drop-in replacement for the original MNIST dataset, which has become too easy for modern machine learning algorithms. The Fashion MNIST dataset has been used for a variety of applications such as image classification, object detection, and image segmentation. It is an important resource for researchers and developers who are working on improving the accuracy of machine learning algorithms for image analysis.

1.2. Dataset Exploration

Figure 1 shows that the Fashion MNIST dataset has a balanced class distribution, with each class containing the same number of images. Specifically, there are 6,000 images in each class in the training set, and 1,000 images in each class in the test set. This means that the dataset is evenly distributed across all 10 classes, and there is no class imbalance to worry about.

Figure 2 is displaying a sample of images from each class to get an idea of the complexity and variability of the dataset.

In figure 3, we can see that the pixel intensity distribution is centered around a value of 125, with a range of values from 0 to 255. This suggests that the images in the dataset are

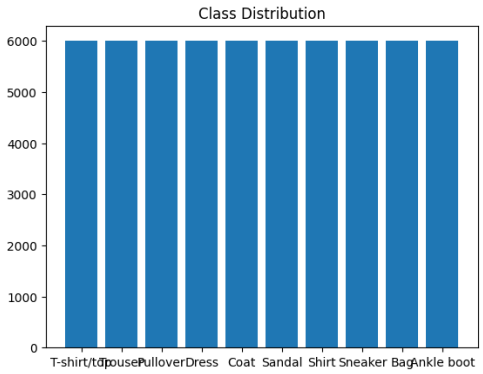

Figure 1: Class distribution bar chart

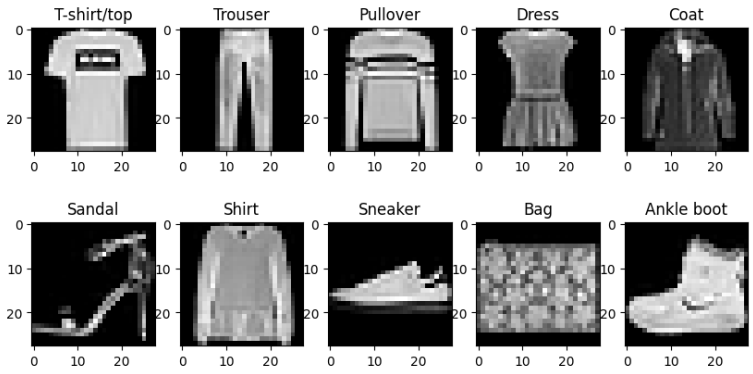

Figure 2: Sample Images

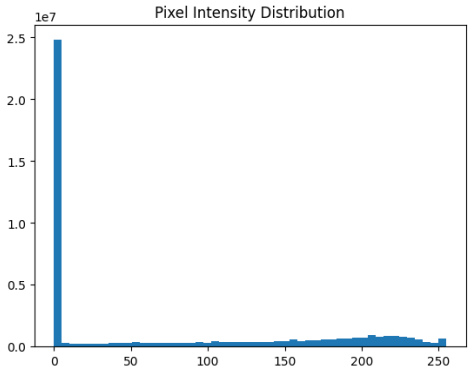

Figure 3: Pixel Intensity Distribution

well-exposed and have consistent lighting conditions.

2. Literature Review

Convolutional Neural Networks (CNNs) have been the most commonly used deep learning architecture for image classification tasks on the MNIST dataset [1]. LeNet-5, a shallow CNN with only 2 convolutional layers, achieved state-of-the-art performance on the MNIST dataset in 1998 [1]. Since then, deeper and more complex CNN architectures have been proposed, such as AlexNet [2], VGG [3], Inception [4], and ResNet [5]. These architectures achieved significant improvements in accuracy on the MNIST dataset by utilizing deeper and more complex networks.

Transfer learning, which involves using a pre-trained model on a large dataset for a different but related task, has also been applied to the MNIST dataset [6]. By transferring knowledge learned from a large dataset to the smaller MNIST dataset, transfer learning can improve the performance of models on the MNIST dataset. For example, models pre-trained on the ImageNet dataset have been fine-tuned on the MNIST dataset and achieved state-of-the-art performance [7].

One recent study explored the use of Capsule Networks for digit recognition on the MNIST dataset [8]. Capsule Networks use a novel architecture that allows the model to learn features and relationships between features in a hierarchical manner. The study showed that Capsule Networks outperformed traditional CNNs on the MNIST dataset, achieving a classification accuracy of 93.6%.

Another recent study investigated the use of Generative Adversarial Networks (GANs) for fashion image generation on the Fashion MNIST dataset [9]. GANs are a type of neural network that can generate realistic images by learning to mimic a given dataset. The study showed that GANs can be used to generate high-quality fashion images that are difficult to distinguish from real images.

3. Code Explanation

The intial code snippet loads and visualizes the Fashion MNIST dataset using the Matplotlib library in Python. The dataset is loaded using the fashion-mnist.load-data() function from the Keras library, with the training and test sets stored in (trainX, trainy) and (testX, testy) respectively.

Then we import the necessary libraries and modules to build and train a baseline CNN model for the Fashion MNIST dataset in Keras. It includes modules for data processing, visualization, cross-validation, model building and training, data augmentation, and optimization. These modules are essential for creating and training a CNN model that can accurately classify images from the Fashion MNIST dataset.

3.1. Base Model (Model 1)

This is a basic implementation of a Multilayer Perceptron (MLP) model for the Fashion MNIST dataset. First, the dataset is loaded using the Keras library. Then, the pixel values of the images in the dataset are normalized to be between 0 and 1. This is done to facilitate better training of the model as large values can slow down the optimization process.Next, the images are flattened into one-dimensional arrays to be fed into the MLP model.

The MLP model is then defined using the Sequential model in Keras. The model contains three layers, the input layer, one hidden layer with 128 units, and one hidden layer with 64 units. The activation function used for the hidden layers is the Rectified Linear Unit (ReLU) and the output layer uses the Softmax activation function as it is a multi-class classification problem.

Overall, the code implements a basic MLP model for the Fashion MNIST dataset, with a single input layer, two hidden layers, and a softmax output layer.

3.2. Base with Dropout Model (Model 2)

We define a Multi-Layer Perceptron (MLP) model with dropout regularization, compiles it, fits it on the training data with 10 epochs and batch size of 32, and evaluates it on the test set. The model architecture consists of three fully connected layers, with the first two layers having 128 and 64 units respectively and using the rectified linear unit (ReLU) activation function, while the output layer has 10 units with a softmax activation function. Dropout regularization is applied to the first two layers with a rate of 0.2 to prevent overfitting.

After training, the code plots the accuracy and loss values of the training and validation sets using matplotlib. Finally, the code evaluates the model's performance on the test set by printing the test loss and accuracy. The fashion-mnist dataset is used for training and testing the model, where the images are flattened and pixel values normalized to be between 0 and 1.

3.3. Base with Dropout and Data Augmentation Model (Model 3)

In this implementation, we show an example of training a Multilayer Perceptron (MLP) model with dropout regularization using Keras on the Fashion-MNIST dataset, with data augmentation applied during training. The pixel values of the images are normalized to be between 0 and 1 by dividing by 255.0. The images are then reshaped to be 4D arrays (adding a dimension for the grayscale channel) so that they can be used with Keras data augmentation.

Data augmentation is applied during training using the ImageDataGenerator() function from Keras. This function generates new images by randomly rotating, shifting, and zooming the original images, which helps to reduce overfitting and improve the model's generalization ability.

3.4. CNN with Pooling Model (Model 4-A)

The CNN model architecture is defined using the Sequential model from the Keras library. The model consists of three convolutional layers followed by max pooling layers to reduce the spatial dimensions of the output. The flattened output is then passed through two fully connected layers, which are used to classify the input image. The model is compiled using the compile() function with categorical cross-entropy loss and the Adam optimizer.

The model is trained on the training set using the fit() function from Keras. The training data is reshaped to match the expected input shape of the model. The validation data is also provided to the model during training to monitor its performance. The training is performed for 10 epochs with a batch size of 128.

3.5. CNN with Pooling and Batch Normalization (Model 4-B)

The CNN architecture consists of three convolutional layers with 32, 64, and 128 filters respectively, followed by batch normalization and max pooling layers. The convolutional layers are responsible for extracting relevant features from the input images, while the batch normalization layers help to stabilize the training process by normalizing the activations of the previous layer. The max pooling layers reduce the spatial dimensions of the feature maps, which can help to improve the model's ability to generalize to new data.

After the convolutional layers, the output is flattened to a 1D vector and passed through two dense layers with 128 and 10 units respectively. The ReLU activation function is used for all layers except the last one, which uses softmax to produce a probability distribution over the 10 possible classes.

3.6. CNN with Pooling and Dropout Model (Model 5)

It consists of three convolutional layers with increasing number of filters, followed by max pooling layers, batch normalization and dropout layers to prevent overfitting. After flattening the output of the last convolutional layer, there are two fully connected layers with relu activation and another dropout layer. The output layer is a dense layer with softmax activation to produce the predicted probabilities for each of the 10 classes.

The model is then compiled with the categorical cross-entropy loss function, Adam optimizer, and accuracy metric. It is trained for 10 epochs with a batch size of 128 on the training set and validated on the test set. The training and validation accuracy and loss are plotted using matplotlib.

3.7. Hybrid Model (Model 6)

The model architecture consists of several convolutional layers with ReLU activation, batch normalization, and max pooling, followed by dropout regularization. The final layers include a dense layer with ReLU activation and another dropout layer, and the output layer with softmax

activation.

The model is compiled using categorical cross-entropy as the loss function, Adam as the optimizer, and accuracy as the metric. The model is trained on the training data for 20 epochs with a batch size of 128. The training and validation accuracy and loss values are plotted to visualize the model's performance during training.

3.8. Testing different optimizers

The code trains the hybrid model defined earlier on different optimization algorithms (SGD, Adam, RMSprop, Adagrad, Adadelta, Adamax, Nadam) and records the loss and accuracy of the trained models on a test set. The code imports necessary libraries, including the optimization algorithms to be tested. Then, an empty dictionary called results is defined to store the loss and accuracy of the trained models on the test set. Then, The code loops through each optimization algorithm.

4. Results & Discussion

4.1. Basic Models

We now discuss the results obtained with each of the models. Table 1 shows a summary of the test results obtained. Moreover, we also have a brief discussion about the specifics of the results of each of the eight models.

Models	Epoches	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Model 1	10	0.2386	0.9104	0.3499	0.8789
Model 2	10	0.3114	0.8839	0.3472	0.8775
Model 3	10	0.6256	0.7628	0.5158	0.7984
Model 4-A	10	0.1494	0.9435	0.2763	0.9027
Model 4-B	10	0.0789	0.9702	0.3554	0.9104
Model 5	10	0.2398	0.9119	0.3406	0.8856
Model 6	20	0.1253	0.9534	0.2043	0.9329

Table 1
Table summarizing the performance of all the models.

Model 1 - Baseline: This model is the starting point for the experiment and serves as a benchmark for the other models. It is a basic CNN with only three convolutional layers and a single dense layer. The model uses the categorical cross-entropy loss function and Adam optimizer, and it has ten epochs. The model achieved a high training accuracy of 0.9104 but suffered from overfitting, as the testing accuracy was only 0.8789. Figure 4 shows a plot of all performance metrics for this model.

Model 2 - Baseline + Dropout: This model is an extension of the baseline model, where dropouts are added to prevent overfitting. Dropouts randomly drop out some of the neurons

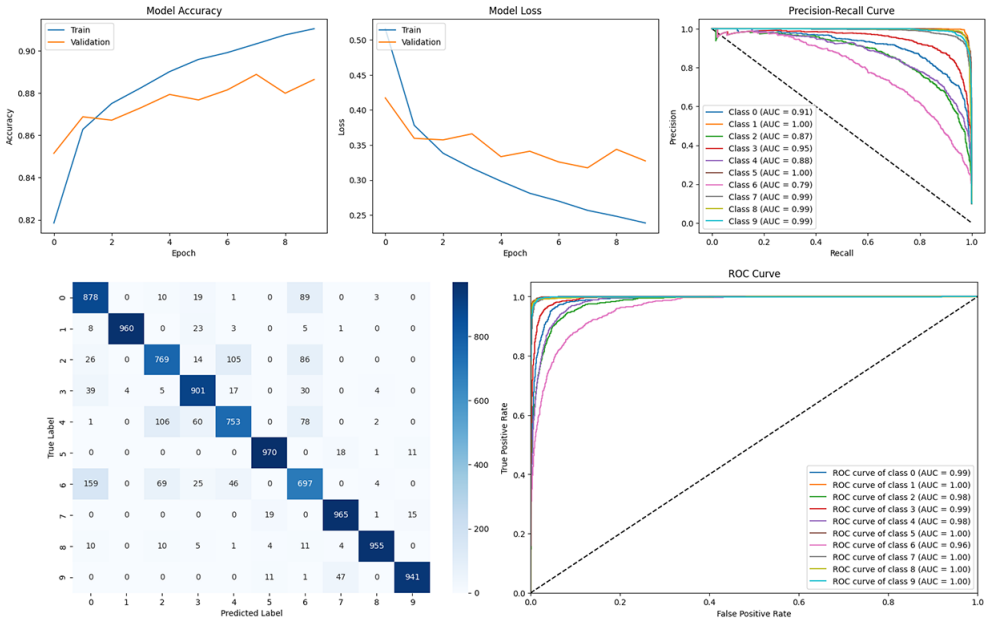

Figure 4: Performance metric of Model 1

during training, which helps the model generalize better. The model achieved a training accuracy of 0.8839 and a testing accuracy of 0.8775, which is slightly better than the baseline model but still suffers from overfitting. Figure 5 shows a plot of all performance metrics for this model.

Model 3 - Baseline + Dropout + Data Augmentation: In this model, data augmentation techniques are added to generate more training data. The model applies various transformations to the existing training data to increase the diversity of the training set. However, this technique did not yield better results and even decreased the testing accuracy to 0.7984. The model also suffered from overfitting. Figure 6 shows a plot of all performance metrics for this model.

Model 4-A - Baseline + Pooling: This model is an extension of the baseline model, where max-pooling layers are added to reduce the spatial dimensions of the feature maps. This technique reduces overfitting by reducing the number of parameters in the model. The model achieved a training accuracy of 0.9435 and a testing accuracy of 0.9027, which is a significant improvement over the baseline model. Figure 7 shows a plot of all performance metrics for this model.

Model 4-B - Baseline + BatchNorm + Pooling: This model includes batch normalization layers along with max-pooling, which further improves the model's accuracy. Batch normalization normalizes the input to each layer, which reduces the internal covariate shift and

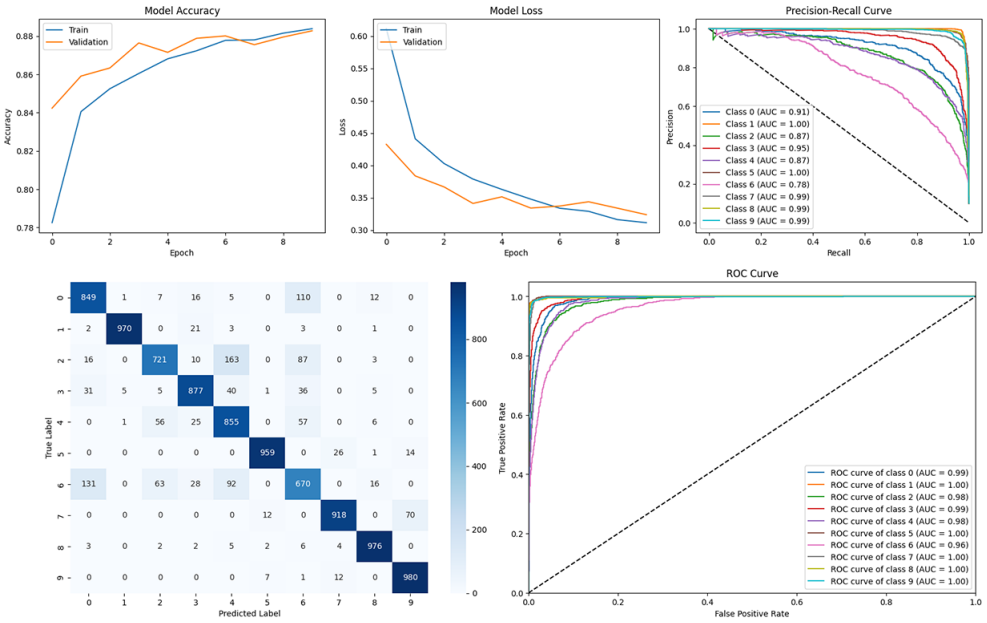

Figure 5: Performance metric of Model 2

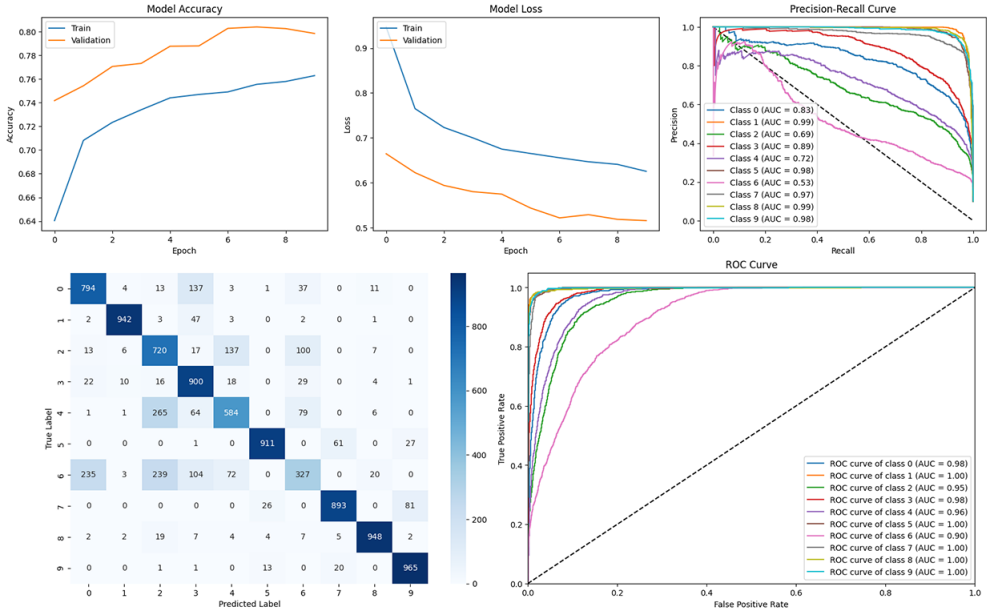

Figure 6: Performance metric of Model 3

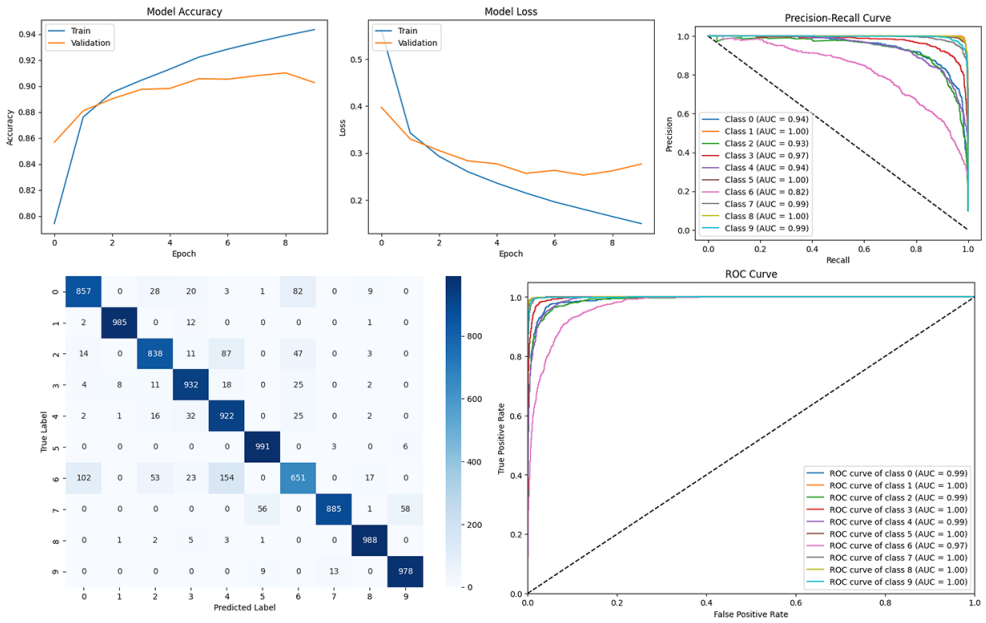

Figure 7: Performance metric of Model 4-A

makes training faster and more stable. The model achieved a high training accuracy of 0.9702 and a testing accuracy of 0.9104. However, the model still suffered from overfitting. Figure 8 shows a plot of all performance metrics for this model.

Model 5 - Same as Model 4b + Dropout: In this model, dropouts are added to the architecture of model 4b to further reduce overfitting. The model achieved a training accuracy of 0.9119 and a testing accuracy of 0.8856. Although this model reduced overfitting, the testing accuracy decreased compared to model 4b. Figure 9 shows a plot of all performance metrics for this model.

Model 6 - Complex: This model is the most complex architecture, including more convolutional layers, increased epochs, and all the useful techniques from the previous models. The model has 20 epochs, which is double the epochs used in the previous models. The model achieved the best results overall, with a high training accuracy of 0.9534 and a testing accuracy of 0.9329. This model has the lowest overfitting among all the models, making it the best-performing model. Figure 10 shows a plot of all performance metrics for this model.

4.2. Hyperparameter Tuning

Figure 11 shows that the Adam optimizer had the lowest initial and final train loss, with initial and final values of 0.5992 and 0.1199, respectively. It also had the highest initial and final train accuracy, with initial and final values of 0.794 and 0.9552, respectively. Additionally, the Adam optimizer had the highest final validation accuracy of 0.9367. RMSprop also performed well,

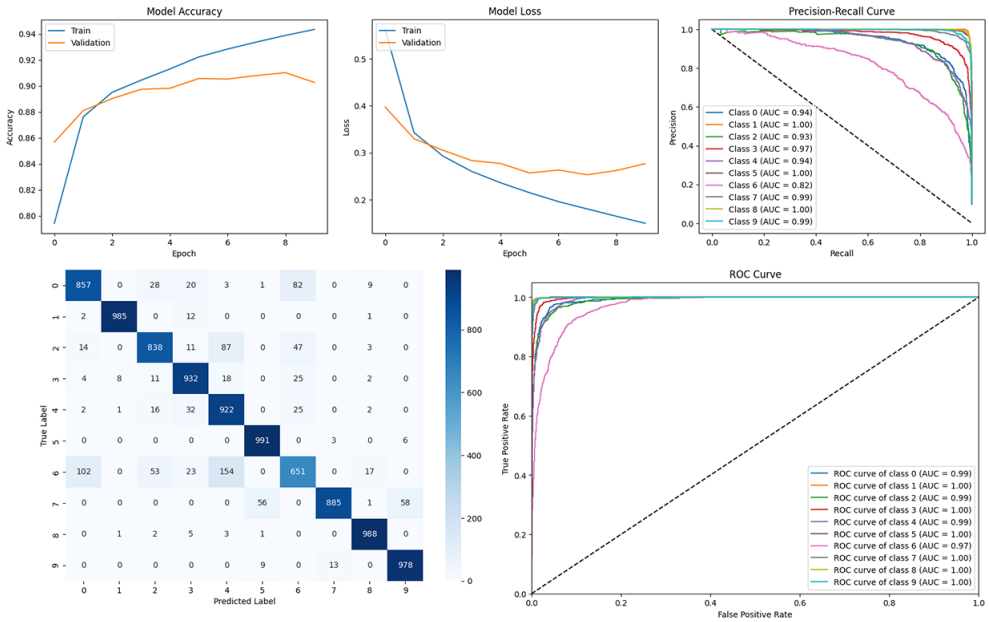

Figure 8: Performance metric of Model 4-B

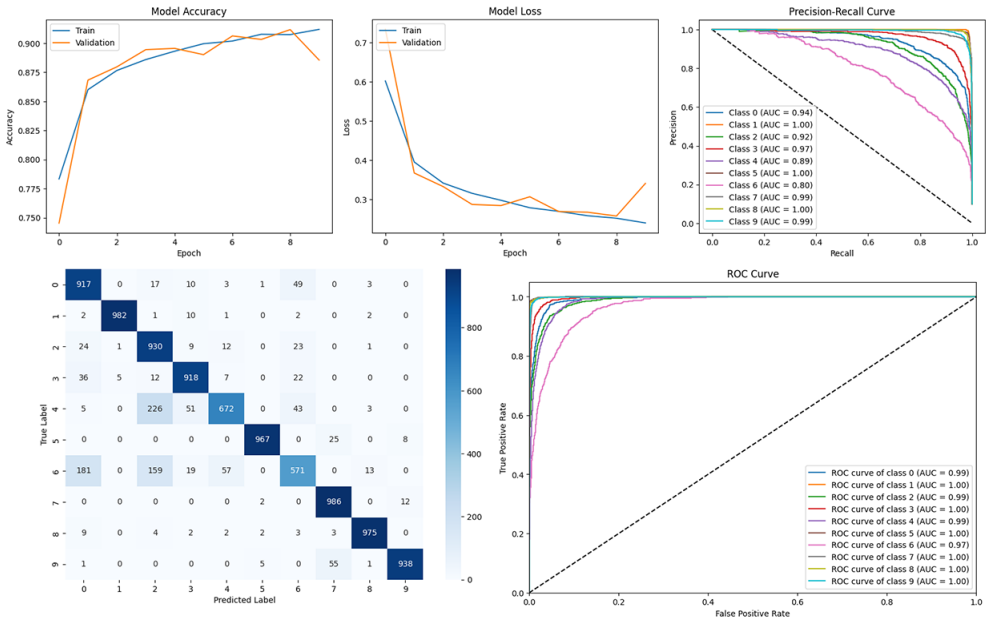

Figure 9: Performance metric of Model 5

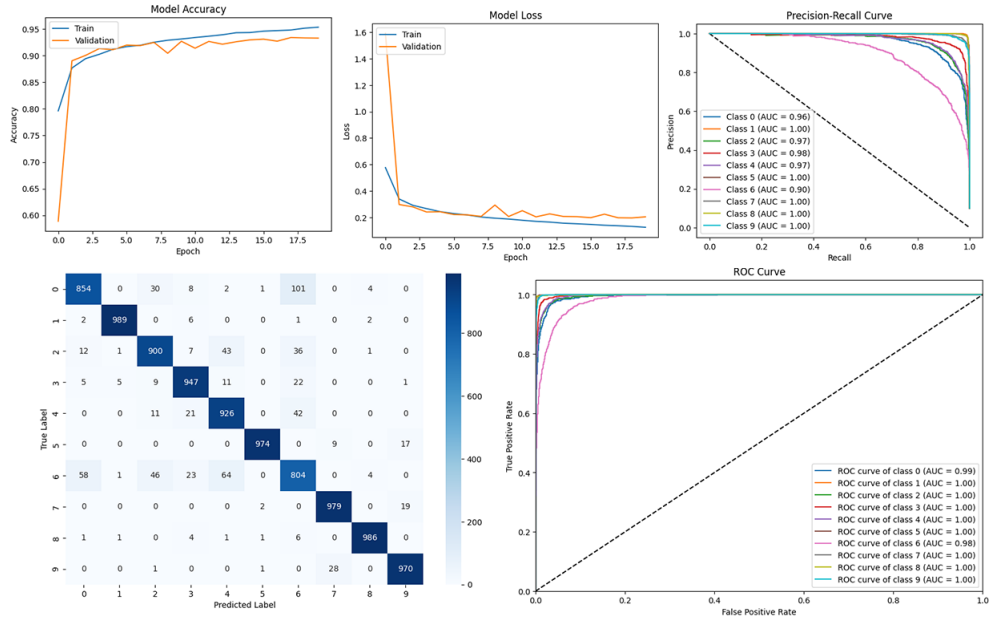

Figure 10: Performance metric of Model 6

Optimizer	Initial Train Loss	Final Train Loss	Initial Val Loss	Final Val Loss	Initial Train Accuracy	Final Train Accuracy	Initial Val Accuracy	Final Val Accuracy
SGD	0.8394	0.2555	0.9935	0.3219	0.7094	0.9065	0.6806	0.8785
Adam	0.5992	0.1199	1.108	0.1917	0.794	0.9552	0.6548	0.9367
RMSprop	0.6362	0.1142	0.589	0.2313	0.7928	0.9585	0.785	0.9352
Adagrad	1.3362	0.4286	1.1674	0.3594	0.5843	0.8435	0.583	0.8662
Adadelta	4.2911	1.0688	2.6078	0.7215	0.1263	0.6369	0.1005	0.729
Adamax	0.7092	0.1414	1.0205	0.2126	0.7541	0.9462	0.6046	0.9278
Nadam	0.5913	0.1214	1.2907	0.1955	0.7925	0.9542	0.632	0.9371

Figure 11: Table showing results of different optimizers.

with the lowest final validation loss of 0.2313 and the second-highest final validation accuracy of 0.9352. Its initial train loss was 0.6362, which was higher than the Adam optimizer but lower than some of the other optimizers. Adadelta had the highest initial train loss of 4.2911 and the lowest initial and final train and validation accuracies, with initial and final train accuracies of 0.1263 and 0.6369, respectively, and initial and final validation accuracies of 0.1005 and 0.729, respectively. Adagrad had the highest initial train loss of 1.3362 and the lowest final validation accuracy of 0.8662. Its final validation loss was 0.3594. Overall, the table provides a comprehensive summary of the performance of different optimizers on the dataset and allows for easy comparison between their respective initial and final values of train and validation metrics. Also looking at graph it is cleared that Adadelta is not converging to result fast, while Adagrad and SGD are also below average and Adam in once again proved one of best optimizer for neural networks. Figure 12 shows a plot of all performance metrics for different optimizers.

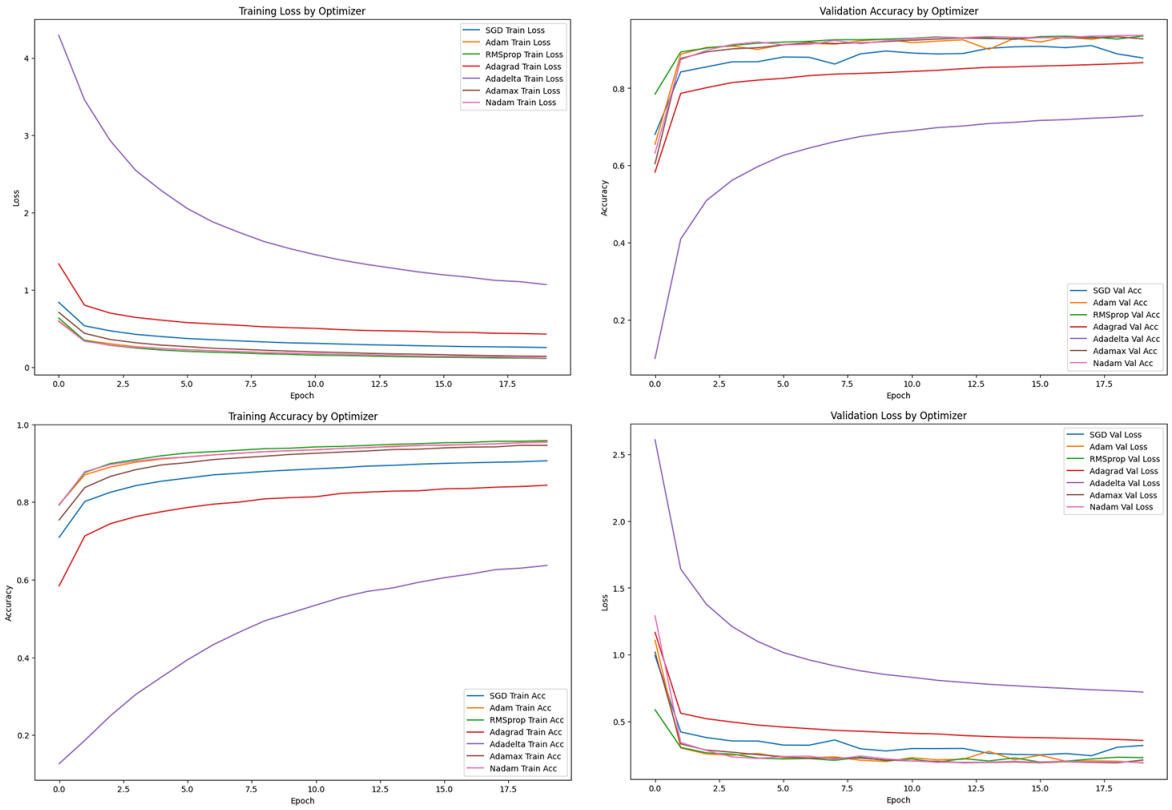

Figure 12: Performance metric of different optimizers

5. Future Works

The results of this study show that deep learning models can achieve high accuracy in image classification tasks. However, overfitting remains a common issue that needs to be addressed. Future work could focus on developing novel techniques to overcome overfitting. Additionally, the potential of transfer learning could be investigated by using pre-trained models on similar datasets to improve the accuracy of the models. The trained models can be fine-tuned on other datasets for different classification tasks.

The Fashion-MNIST dataset can also be used for object detection tasks. The models trained on this dataset can be used to detect clothing items in images and videos, which can be helpful in developing smart shopping assistants. Moreover, the dataset can be used to train generative models like Variational Autoencoders (VAE) and Generative Adversarial Networks (GANs). These models can be used to generate new images of clothing items, which can be used for virtual try-ons, product catalogues, and various other applications.

Overall, this study provides a foundation for future research in improving the accuracy of deep learning models in image classification tasks.

6. Conclusion

In this study, we have presented an analysis of several deep learning models applied to the Fashion MNIST dataset. We have evaluated the performance of different models, including a baseline model, a model with dropouts, a model with data augmentation, and models with various pooling and normalization techniques. We have also proposed a complex model architecture with increased epochs that achieved the best results in terms of accuracy and overfitting.

Our results demonstrate that deep learning models can achieve high accuracy in image classification tasks. However, overfitting remains a common issue, which can be addressed by adding dropouts, pooling, and normalization techniques. Data augmentation, on the other hand, did not improve the performance and even decreased the accuracy of the models.

In conclusion, this study highlights the importance of selecting the appropriate techniques and architecture for deep learning models in image classification tasks. The proposed complex model achieved the best results, but it may not be the most practical solution in real-world applications. Thus, future work should focus on developing more efficient and scalable models that can be applied to larger datasets and real-world scenarios.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324. doi:10.1109/5.726791.

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 25, Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[5] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

[6] R. V. Singh, A. Sethi, Mnistnet: A deep convolutional neural network architecture for image classification on mnist dataset, 2018.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] S. Sabour, N. Frosst, G. E. Hinton, Dynamic routing between capsules, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/2cad8fa47bbef282badbb8de5374b894-Paper.pdf.

[9] H. Zhang, Generative adversarial networks for fashion, 2019.