Fashion MNIST Image Classification using Deep Learning with Pytorch

By

Kachinga Silwimba (kachingasilwimba@u.boisestate.edu)

Boise State University (BSU), Idaho, USA

December 2022

Deep Learning Class-Project ECE-662

1. Pattern Recognition

In this project, we explore classifying the Fashion MNIST images by implementing various convolutional neural network (CNN) architectures using PyTorch. In PyTorch, a CNN is composed of several layers, including convolutional layers, pooling layers, and fully-connected layers.

1.1 Problem Statement

Image classification is one of the most foundational problems in computer vision, which has a variety of practical applications, such as image and video indexing. Moreover, the capacity of machine learning (ML) models to learn from datasets offers various advantages. However, there are limitations to training a model's behavior on a particular training set. For instance, models could incorrectly categorize real-world data not included in the training dataset. Several deep learning models have been developed to handle such situations and increase the classification accuracy of the models. This project study aims to design and train a PyTorch Convolutional Neural Network (CNN) architecture to classify fashion image classes (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot) of the machine learning benchmarking dataset called Fashion MNIST using the PyTorch framework.

1.2 Background

Convolutional Neural Networks (CNN) have recently been applied in a variety of fields, including the classification of fashion. Criminal law, social media, and e-commerce are all widely applicable in this area. Moreover, deep neural networks have achieved very good performance in many problems (Krizhevsky et al., 2017). The Fashion-MNIST dataset has been used to develop the task of classifying clothing, and the accuracy achieved in some instances exceeds 90% (Nocentini et al., 2022). We will go through some of the work done with the Fashion MNIST dataset using the convolutional neural network and other models. According to Vijayaraj et al. (2022), to improve the classification of the Fashion MNIST dataset, three methods developed batch normalization, residual skip connections, and CNN architectures were developed. A two-layer CNN with batch normalization and skip connections was used to get an accuracy of 92.54%The dataset is accessed from torchvision. The CNN-SVM and CNN Softmax architectures on the Fashion-MNIST dataset achieved roughly 90.72% and 91.86%, respectively. Seo and Shin (2019) designed a hierarchical CNN using VGGNet and the Fashion MNIST dataset and acquired training and testing accuracy of 100% and 93.52%, respectively. Additionally, CNN with the relu activation function, on the other hand, had an accuracy of 91.74%. Meshkini et al. (2019) proposed a simple modification to the already known network architecture (AlexNet, GoogleNet, VGG, ResNet, DenseNet, and SqueezeNet) to improve their performance and speed up the learning process. They reported an accuracy of 93.43% and a loss function of 0.19.This model showed an improved accuracy of 2% compared to other existing systems in the literature on the benchmark Fashion MNIST dataset. Nocentini et al. (2022) designed the Multiple CNN with 15 convolutional layers (MCNN15) and obtained the accuracy of 94.04% on the Fashion MNIST dataset.

1.3 Approaches and Ideas for the Network Architecture

This section lists some techniques we shall use to design the CNN network.

(i) Design and implement a PyTorch CNN architecture on the Fashion MNIST dataset and apply different hyperparameters optimization methods (learning rate, kernel size, different number of fully connected layers, batch size, stride, with or without dropout, etc.) and different feature maps in the convolutional layers to improve the model performance.

(ii) Apply the dropout rate in the CNN layers and try different activation functions, optimizers, hyperparameters, and the preprocessing of the Fashion MNIST dataset (e.g., batch normalization and augmentation) to improve the generalization of the CNN models.

(iv) Compare the performance of the CNN with other image classification algorithms on the Fashion MNIST dataset in the literature and also create CNN models with varying depth (number of convolutional layers) to observe performance.

(v) Keep the designed network architecture unchanged and only vary the kernel sizes to better understand the kernel sizes and hyperparameters effect during the training process.

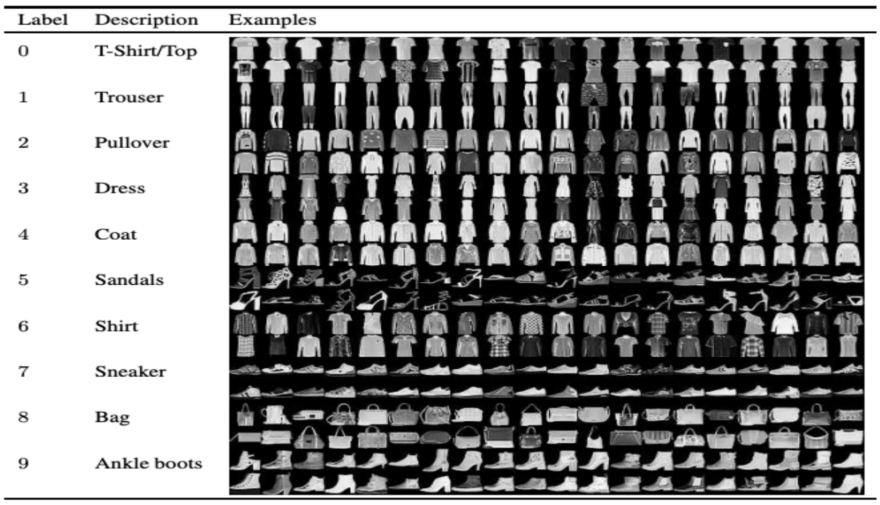
(vi) Use a different loss function, such as the cross-entropy loss or the mean squared error loss, better suited to the Fashion MNIST dataset and evaluation metric. This can help the model optimize for the right objective and improve its performance.

(vii) Use a larger input size: Using larger images as input to the network can improve performance, allowing the network to learn more detailed features.

(viii) Experiment with different optimizers: Different optimizers can have different properties affecting the network's performance.

1.4 Fashion MNIST Dataset

Fashion-MNIST was introduced in 2017 by Xiao et al. (2017) to provide a similarly sized alternative to MNIST, posing a more challenging classification challenge. The Fashion MNIST dataset for training the models and testing the network architecture using the test dataset. The Fashion-MNIST dataset is a collection of Zalando fashion items, with a training set of 70,000 and 10,000 test examples to test the model. Every sample is a 28×28 grayscale image. Each image in the dataset has a label from one of 10 classes, and four files in the dataset contain both the labels and the images shown in the Table below. The 60,000 training set was subdivided into the training and validation sets 50,000 and 10,000, respectively. The examples in Fashion-MNIST are more challenging to classify than those in standard MNIST. This is because the 28×28 images were downscaled from the color clothes images acquired from the Zalando shopping catalog (Xiao et al., 2017). The dataset is downloaded from the torchvision in PyTorch in Tensor format.



Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 1.1: Class names and example images in Fashion MNIST dataset (Xiao et al., 2017).

1.5 Training and Validation of the CNN

In the experimental setup of our network architecture, we trained, validated, and tested our model on the Fashion MNIST dataset described in Section 1.4. The code was written in Python, and for the network model training, we used the PyTorch library with the Adam and Stochastic Gradient (SGD) optimizer and tuned the hyperparameters. Furthermore, various network architectures were designed with different convolutional, fully connected, and pooling layers. During the architecture design, we tried kernels of small dimensions 2×2, 3×3, 4×4, and 5×5. Moreover, we trained the PyTorch CNN using ten epochs, and each complete run of 10 epochs was called a Trial. Each Trial was compared to the preceding Trial by adjusting the hyperparameters. The hyperparameters were hand pick based on the performance compared to the initial model run. The best hyperparameters that performed better on the validation dataset were recorded in Table 1.1 for each experiment (Trial) conducted on the validation and training set, respectively.

1.5.1 Network Architecture 1. Our first PyTorch CNN architecture is composed of two convolutional layers, four fully connected layers, a dropout layer between two fully connected layers, and a pooling layer between the convolutional layer. We tried two different optimizers (Adam and Stochastic Gradient). Table 1.1 shows the Hyperparameters and seven best Trials recorded during the fitting process of our first network architecture design shown below.

```
Fashion_MNIST_Network1(
(conv1): Conv2d(1, 8, kernel_size=(5, 5), stride=(1, 1))
(relu): ReLU()
(maxpoll1): MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(conv2): Conv2d(8, 12, kernel_size=(5, 5), stride=(1, 1))
(maxpoll2): MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(fc1): Linear(in_features=192, out_features=120, bias=True)
(dropout): Dropout(p=0.25, inplace=False)
(fc2): Linear(in_features=120, out_features=60, bias=True)
(fc3): Linear(in_features=60, out_features=40, bias=True)
(fc4): Linear(in_features=40, out_features=10, bias=True))
```

Table 1.1: Training Hyperparameters for the CNN architecture Above

Parameters	**Trial-1**	**Trial-2**	**Trial-3**	**Trial-4**	**Trial-5**	**Trial-6**	**Trial-7**
Learning rate	0.0001	0.001	0.001	0.01	0.0001	0.0001	0.1
Batch size	30	100	120	40	200	256	64
Epochs	15	15	15	15	15	15	15
conv1 (In_channels=)	1	1	1	1	1	1	1
conv1 (kernel_size=)	5	5	5	5	5	5	5
conv1 (Out_channels=)	32	40	48	16	64	72	80
conv1 (stride=)	1	1	1	1	1	1	1
conv2 (In_channels=)	32	40	48	56	64	72	80
conv2 (kernel_size=)	5	5	5	5	5	5	5
conv2 (Out_channels=)	64	80	88	96	104	112	120
conv2 (stride=)	2	2	2	2	2	2	2
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
fc1(in_features=)	1024	1280	1408	1536	1664	1792	1920
Dropout	0.3	0.5	0.4	0.2	0.25	0.5	0.5
fc1(out_features=)	512	640	704	768	832	896	960
fc2(in_features=)	512	640	704	768	832	896	960
fc2(out_features=)	256	320	352	384	416	448	480
fc3(in_features=)	256	320	352	384	416	448	480
fc3(out_features=)	128	160	176	192	208	224	240
fc4(in_features=)	128	160	176	192	208	224	240
fc4(out_features=)	10	10	10	10	10	10	10
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Shuffle	True	True	True	True	True	True	True
Optimizer	Adam	Adam	Adam	SGD	Adam	SGD	Adam
Cost function	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy
Validation accuracy	88.4%	83.7%	87.2%	87.6%	88.1%	86.8%	87.9%

We kept tracking the validation and train loss with respect to their accuracy for each epoch. We observed over-fitting on the network architecture above during the training process on the Fashion MNIST dataset. Therefore, we decided to design another network to improve network performance.

1.5.2 Network Architecture 2. Our second network architecture comprised two convolutional layers and two fully connected layers printed from python. Moreover, we added the pooling layer, batch normalization, and the ReLU activation function between the convolutional layer and the dropout layer between the two fully connected layers and tried different hyperparameters. The network was trained using the Fashion MNIST dataset. We recorded in Table 1.2 the hyperparameter values and the validation accuracy for the seven experiments (Trials) that displayed a better performance on the validation data during the training process. Moreover, we added batch normalization after the convolutional layer to speed up the learning process. Furthermore, batch normalization lessened the network internal covariate shift, which impacts the learning. A change in data distribution is the covariate shift (Ioffe and Szegedy, 2015). Additionally, each convolution layer is followed by a max-pool layer and the ReLU activation function for this framework. Non-linearity is introduced through ReLU because images are naturally non-linear, and noise is reduced with max-pooling. Moreover, the validation data plays a significant role in knowing when to stop training the model to avoid overfitting.

```
Fashion_MNIST_Network2(
(conv1): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(batchnorm1):BatchNorm2d(4,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
(relu): ReLU()
(maxpool1): MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(conv2): Conv2d(4, 16, kernel_size=(5, 5),stride=(1, 1),padding=(2, 2))
(batchnorm2):BatchNorm2d(16,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
(maxpool2):MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(fc1): Linear(in_features=784, out_features=392, bias=True)
(dropout): Dropout(p=0.25, inplace=False)
(fc2): Linear(in_features=392, out_features=10, bias=True))
```

Table 1.2 shows the number of Trials or experiments done on the second-designed CNN architecture by varying hyperparameters used during the training process. The validation accuracy was recorded for each Trial on the printed neural network architecture shown above. The Table is arranged following the flow of the network architecture designed.

Table 1.2: Training Hyper-parameters for the CNN Architecture above

Parameters	**Trial-1**	**Trial-2**	**Trial-3**	**Trial-4**	**Trial-5**	**Trial-6**	**Trial-7**
Learning rate	0.001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
Batch size	90	90	100	256	190	100	230
Epochs	30	30	30	30	30	30	30
conv1 (In_channels=)	1	1	1	1	1	1	1
conv1 (kernel_size=)	3	3	3	3	3	3	3
conv1 (Out_channels=)	4	4	8	16	24	48	56
conv1 (stride=)	1	1	1	1	1	1	1
conv1 (padding=)	1	1	1	1	1	1	1
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
max_pool2d (stride=)	2	2	2	2	2	2	2
conv2 (In_channels=)	4	4	8	16	24	48	56
conv2 (kernel_size=)	5	5	5	5	5	5	5
conv2 (Out_channels=)	16	16	24	32	40	72	80
conv2 (stride=)	1	1	1	1	1	1	1
conv2 (padding=)	2	2	2	2	2	2	2
max_pool2d (kernel_size=)	2	2	2	2	2	2	2
max_pool2d (stride=)	2	2	2	2	2	2	2
fc1(in_features=)	784	784	1176	1568	1960	3528	3920
fc1(out_features=)	392	392	588	784	980	448	512
Dropout	0.5	0.5	0.4	0.2	0.1	0.5	0.5
fc2(in_features=)	392	392	588	784	980	448	512
fc2(out_features=)	10	10	10	10	10	10	10
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Shuffle	True	True	True	True	True	True	True
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Cost function	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy	Cross entropy
Validation accuracy	89.3%	91.6%	90.3%	90.4%	89.7%	88.9%	88.7%

Finally, from the training Table 1.2, we selected **Trial-2** hyperparameters highlighted in **yellow** and the printed second-network architecture because it gave better validation accuracy of 91.6% compared to other trials and minimized the loss function by giving a smaller validation and train loss was constantly decreasing shown in the plots below. The second preferred network architecture did not overfit during the training. Hence, the framework will generalize well to the unseen Fashion MNIST test datasets.

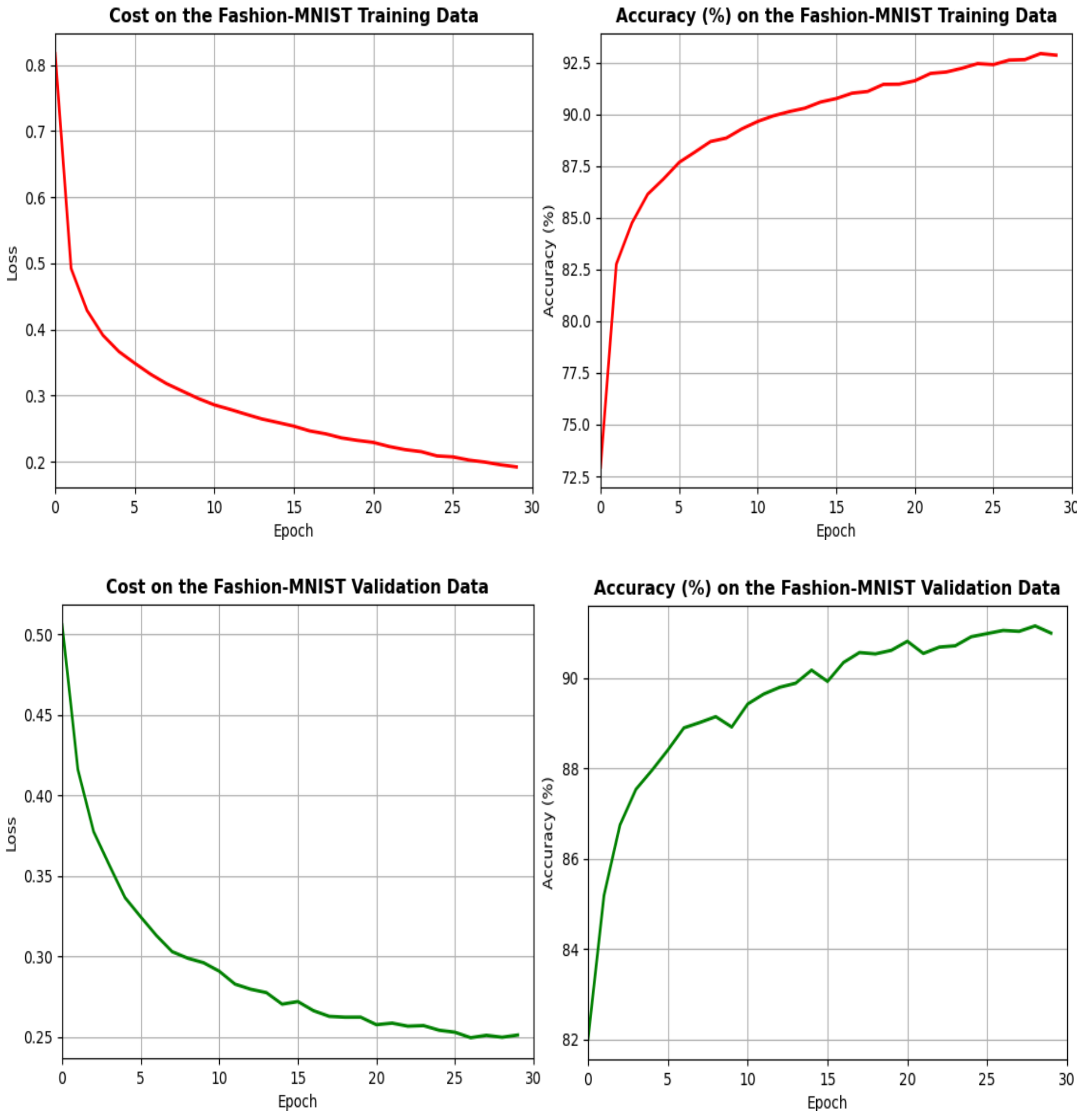


Figure 1.2: Figures show the validation and training loss and also validation and training accuracy

1.6 Final Network Architecture

The final CNN architecture consists of two convolutional layers, and two fully connected layers shown below was selected. Table 1.3 shows the full model summary of our final proposed CNN architecture. The network used Adam (Kingma and Ba, 2014) optimizer with a learning rate of 0.0001 to update the weights and used the ReLU activation function. Table 1.3 shows the hyperparameters used during the training. In addition, a cross-entropy loss function was used to optimize the network by adjusting the model weights to minimize errors while training the network. Finally, a dropout rate of 0.50% between two fully connected layers was used for regularization to prevent the neural network from overfitting.

```
Fashion_MNIST_Network2(
(conv1): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(batchnorm1):BatchNorm2d(4,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
(relu): ReLU()
(maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(conv2): Conv2d(4, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
(batchnorm2):BatchNorm2d(16,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
(maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(fc1): Linear(in_features=784, out_features=392, bias=True)
(dropout): Dropout(p=0.5, inplace=False)
(fc2): Linear(in_features=392, out_features=10, bias=True))
```

1.6.1 Final Network Architecture Description. The first convolutional layer (`Conv1`) takes in a channel of 1 dimension, and the kernel size of 3×3 with a stride of 1 and output of this convolutional layer is set to be 4 channels implying that it will extract **4 feature maps** using **4 kernels**. To ensure that the input and output dimensions are identical, we pad the image using a padding size of 1. The output dimension at this convolutional layer is $\mathbf{4 \times 28 \times 28}$. Then, a max pooling layer with a kernel size of 2 and a stride of 2 is applied to it. The feature translates to dimensions of $\mathbf{4 \times 14 \times 14}$ as a result of the downsampling through max pooling.

The second convolution layer (`Conv2`) has **4 input channels**. It extracts **16 feature maps** because we chose an output channel size of 16 with stride 1. The kernel size for this layer is 5. To keep the input and output dimensions constant, we once more utilize a padding size of 1. This layer's output size is going to be $\mathbf{16} \times \mathbf{14} \times \mathbf{14}$. Then, a max-pooling layer with a kernel size of 2 and a stride of 2 is applied to it. The feature translates to dimensions of $\mathbf{16} \times \mathbf{7} \times \mathbf{7}$ as a result of the downsampling.

Finally, two fully connected layers (`fc1` and `fc2`) are used. The first fully connected layer (`fc1`) receives a flattened copy of the feature maps. Therefore, it must have a dimension of $\mathbf{16} \times \mathbf{7} \times \mathbf{7}$, or 784 nodes. This layer joins a 392 node layer that is fully connected (`fc2`). The output dimension matches the total number of classes, which is 10, as this is the last layer. Thus, we have two fully connected layers that are $\mathbf{784} \times \mathbf{392}$ and $\mathbf{392} \times \mathbf{10}$, respectively. A dropout layer between the two fully connected layers was added to dropout connections to reduce overfitting. Table 1.3 shows a summary of the final network architecture selected with the input and output shape.

Table 1.3: Pytorch model summary of our final CNN architecture

Operational Layer		**Input Shape**	**Kernel Size**	**Stride Size**	**Padding**	**Output Shape**
Convolution Layer (`Conv2D_1`)	Covolutional	(1,28,28)	(3,3)	(1,1)	(1,1)	(4,28,28)
Relu (`ReLU`)		(4,28,28)	-	-	-	(4,28,28)
Pooling Layer (`MaxPool2D_1`)	Max pooling	(4,28,28)	2	2	0	(4,14,14)
Convolution Layer (`Conv2D_2`)	Covolutional	(4,14,14)	(5,5)	(1,1)	(2,2)	(16,14,14)
Relu (`ReLU`)		(16,14,14)	-	-	-	(16,14,14)
Pooling Layer (`MaxPool2D_2`)	Max pooling	(16,14,14)	2	2	0	(16,7,7)
Dense Layer (`fcon_1`)	Fully connected	(784)	-	-	-	(392)
Dropout Layer (`drpout`)	Dropout ($\mathcal{P} = 0.5$)	(392)	-	-	-	(392)
Dense Layer (`fcon_2`)	Fully connected	(392)	-	-	-	(10)

Table 1.4 shows the final hyperparameter extracted from Table 1.2 for the final convolutional neural network architecture considered for the classification of 10 Fashion MNIST classes.

Table 1.4: Hyperparameters of final training CNN Architecture

Parameter Name	**Value**
`Learning rate` (η)	0.0001
`Batch size (batch_size)`	90
`Number of Epochs (num_epochs)`	30
`Dropout` ($\mathcal{P}$)	0.5
`Shuffle`	True

1.7 Network Results On Test Dataset

Finally, we tested our best model on the unseen Fashion MNIST test dataset. We achieved an accuracy of approximately $\mathbf{90.4}\%$ on the test dataset. Throughout the training process, the model was not exposed to this dataset. Our model's performance was now independently assessed, thanks to this. The test accuracy compares well with the results in the literature review from the background of the Fashion MNIST dataset. For example, McKenna (2017) reported the accuracy range from 51% to 86.8% for the non-neural network (NN) models, while 94.9% for the ResNet18 model. Moreover, our CNN model outperforms the non-NN models based on the accuracy reported by Xiao et al. (2017) on the Fashion MNIST dataset. Moreover, Khanday et al. (2021) reported a 92.68% accuracy on the Fashion MNIST dataset using the CNN model with 3×3 pixels kernel size. Additionally, Meshkini et al. (2019) presented the accuracy of approximately 93% for the AlexNet, GoogleNet, VGG, ResNet, and DenseNet. Hence, we can conclude that our network architecture aligns well with the literature reviewed.

1.8 Conclusion

In this project, different simple convolutional neural network architecture was used on the Fashion MNIST dataset for each designed network architecture. The architecture was kept unchanged to track the effect of hyperparameters and kernel sizes until a desirable accuracy was achieved. On the test set, our selected network architecture and hyperparameters achieved an accuracy of 90.8%. This demonstrates how effective convolutional networks are in predicting images. Moreover, from the experiment results in Tables 1.2 and 1.1, kernel size and feature maps significantly affect classification accuracy, training accuracy, and validation. For example, the accuracy for the network architecture with kernel size 2×2 pixels and 3×3 pixels was higher than the one with a kernel size of 5×5 pixels. A larger kernel size was observed to increase the number of parameters in the model, making the model more computationally expensive. In future work, we would like to improve the model's accuracy.

References

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR.

Khanday, O. M., S. Dadvandipour, and M. A. Lone (2021). Effect of filter sizes on image classification in cnn: A case study on cfir10 and fashion-mnist datasets. *IAES International Journal of Artificial Intelligence 10*(4), 872.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*(6), 84–90.

McKenna, M. (2017). A comparison of activation functions for deep learning on fashion-mnist. *arXiv preprint arXiv:1708.07747*.

Meshkini, K., J. Platos, and H. Ghassemain (2019). An analysis of convolutional neural network for fashion images classification (fashion-mnist). In *International Conference on Intelligent Information Technologies for Industry*, pp. 85–95. Springer.

Nocentini, O., J. Kim, M. Z. Bashir, and F. Cavallo (2022). Image classification using multiple convolutional neural networks on the fashion-mnist dataset. *Sensors 22*(23), 9544.

Seo, Y. and K.-s. Shin (2019). Hierarchical convolutional neural networks for fashion image classification. *Expert systems with applications 116*, 328–339.

Vijayaraj, A., V. Raj, R. Jebakumar, P. Gururama Senthilvel, N. Kumar, R. Suresh Kumar, and R. Dhanagopal (2022). Deep learning image classification for fashion design. *Wireless Communications and Mobile Computing 2022*.

Xiao, H., K. Rasul, and R. Vollgraf (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.