



Deep Learning Classifying Images

Roy Phelps
Royphelps1@gmail.com




Table of Contents

Introduction.....	2
Dataset.....	2
Deep Learning Models Developed	3
Results.....	5
Conclusions and Takeaways	8
References.....	9
Appendix.....	10

Introduction

The purpose of the deep learning neural network being developed is to classify images into one of ten predefined categories based on the input pixel values. Utilizing a Convolutional Neural Network (CNN) with the libraries TensorFlow and Karas in deep learning for the precise classification of clothing and fashion items, can automate object identification effectively when successful.

Why use a CNN? The CNN's have several different trainable filters that can convolve on a given image. They can detect features like edges and shapes spatially. The high filters can learn to capture these features based on the learned weights through back propagation and stacked layers (Saha, 2018).

This classification task has broad applications, from image tagging to autonomous driving and content recommendation systems. Developing an accurate model for this problem type holds significant value, as it can enhance the efficiency of various real-world applications by automating the process of image categorization and recognition (Mishra, 2020). The model's ability to accurately classify images can save time and resources, improve user experiences, and facilitate the development of innovative technologies.

Dataset

The dataset consists of 60,000 examples for a training set and 10,000 examples for a test set. In each example, there is a 28x28 1D grayscale image associated with a label from 10 classes (Han Xiao, 2017). Each pixel in a grayscale image is represented by a single intensity value ranging from 0 (black) and 255 (white) in an 8-bit image. If, however, the image was color, there would be a depth dimension because the color images have color channels, typically represented by RGB, (Red, Green, Blue) color space (Pramoditha, 2021). An example would be 28x28x3, meaning there are three 28x28 arrays, one with each color channel.

Each training and test example is assigned to one label, or class, ranging from 0 to 9, constituting 10 total examples, 0 is counted. They are the following, 0 for **T-shirt/top**, 1 for **Trouser**, 2 for **Pullover**, 3 for **Dress**, 4 is for **Coat**, 5 equals **Sandal**, 6 is **Shirt**, 7 for **Sneaker**, 8 is for **Bag**, and 9 for **Ankle Boot**. The classes can be seen here, [Figure 1](#).

To see some of the grayscale images in the data, some coding is conducted in Python, [Figure 2](#). The first twenty images in the dataset can be seen in [Figure 3](#).

Deep Learning Models Developed

There were eight models developed for this analysis with varying parameters on each in hopes of gaining a higher accuracy and lower training time. Below in the **Models Created** table, the Deep Learning models are structured with convolutional layers (Conv1D) and various settings for training. Each model has a specific configuration that defines its architecture and hyperparameters. Let's break down these models down to explain their settings.

Models Created						
Model Number	Layer	Conv1D Settings	Maxpooling1D Settings	Droupout Settings	epochs	Batch Size
1	1	Filters = 256. Kernal Size = 2	Pool Size = 2	0.2 or 20%	10	128
	2	Filters = 128. Kernal Size = 2	Pool Size = 2	0.2 or 20%	10	128
2	1	Filters = 256. Kernal Size = 3	Pool Size = 3	0.2 or 20%	10	20
	2	Filters = 128. Kernal Size = 3	Pool Size = 2	0.2 or 20%	10	20
	3	Filters = 64. Kernal Size = 2	Pool Size = 2	0.2 or 20%	10	20
3	1	Filters = 256. Kernal Size = 3	Pool Size = 3	0.1 or 10%	10	20
	2	Filters = 164. Kernal Size = 3	Pool Size = 2	0.1 or 10%	10	20
	3	Filters = 64. Kernal Size = 2	Pool Size = 2	0.1 or 10%	10	20
4	1	Filters = 256. Kernal Size = 7	Pool Size = 3	0.2 or 20%	20	70
	2	Filters = 128. Kernal Size = 5	Pool Size = 3	0.2 or 20%	20	70
5	1	Filters = 256. Kernal Size = 7	Pool Size = 3	0.1 or 10%	20	70
	2	Filters = 128. Kernal Size = 5	Pool Size = 3	0.1 or 10%	20	70
6	1	Filters = 256. Kernal Size = 7	Pool Size = 3	0.3 or 30%	20	70
	2	Filters = 164. Kernal Size = 5	Pool Size = 2	0.3 or 30%	20	70
	3	Filters = 64. Kernal Size = 1	Pool Size = 1	0.3 or 30%	20	70
7	1	Filters = 256. Kernal Size = 7	Pool Size = 3	0.1 or 10%	20	70
	2	Filters = 164. Kernal Size = 5	Pool Size = 2	0.1 or 10%	20	70
	3	Filters = 64. Kernal Size = 1	Pool Size = 1	0.1 or 10%	20	70
8	1	Filters = 256. Kernal Size = 10	Pool Size = 10	0.2 or 20%	20	70

The parameter *filters* specify the number of filters, or channels, in the convolutional layer. Think of them as small windows that slide over the input data to learn features. The *kernel size* defines the size of the filter window, meaning if the kernel size is ten, then the filter looks at a window of ten sequential elements at a time. *MaxPooling* reduces the spatial dimensions of the data by taking the maximum value within each window of the specified size. If the MaxPooling is ten, then it reduces the sequence length by a factor of ten. The *dropout* layer is a regularization technique that randomly sets a fraction of input units to zero during each training update, which helps prevent overfitting. The *batch normalization* helps to stabilize and speed up the training by normalizing the inputs to each layer in a mini batch. It can also mitigate the vanishing/exploding gradient problems during training (Göllner, 2022).

An *epoch* represents one complete pass through the entire training dataset, where the model processes and updates its weights based on the entire dataset's gradients. *Batch size*, on the other hand, defines the number of data samples processes in each forward and backward pass through the neural network during a single epoch. Smaller batch sizes provide more frequent updates to the model but may result in noisy gradients, while larger batch sizes can speed up training but may require more memory and potentially converge to a suboptimal solution. A visual diagram of what a CNN model does is in [Figure 4](#).

In the first model, **Model 1**, there are two convolutional, hidden, layers. The first layer has 256 filters with a kernel size of 2, followed by max-pooling with a pool size of 2. A dropout of 20% is applied after this layer. The second layer has 128 filters with a kernel size of 2, followed by max-pooling with a pool size of 2 and another 20% dropout. In this model, each layer is trained for 20 epochs with a batch size of 128.

Model 2 comprises of three convolutional layers. The first layer has 256 filters and a kernel size of 3, with a max pooling size of 3 and 20% dropout. The second layer consists of 128 filters, a kernel size of 3, max pooling with a pool size of 2, and a 20% dropout. The third layer has 64 filters with a kernel size of 2, max pooling of 2, and a 20% dropout. This model is run with 10 epochs and a batch size of 20.

The remaining models can be explained similarly, with variation in the number of convolutional layers, filter sizes, kernel sized, pooling sized, dropout rates, and training parameters such at epochs and batch sizes in the **Models Created** table.

The reason for changing these parameters is to produce a more accurate, timely, and reduction in data loss for an optimal model. For example, **Model 2** has three hidden layers instead of two which add additional layers to provide the network with more capacity to learn complex patterns and representations for the data. Changing the pool size to 2 in layers 2 and 3 down samples the spatial dimensions of the feature maps, reducing computational complexity and helping to capture more abstract features.

Results

The results of the eight different models, are best viewed as a table to visually see the performance metrics of each one in hopes to find the best and worst models. Below, is the **Results** table.

Results									
Model #	Total Train Time in Seconds	Train Loss	Train Accuracy	Test Loss	Test Accuracy	Delta in Train loss and test loss	Delta in Train Accuracy an Test Accuracy	Input Shape	Output Shape
1	62	1.071	0.622	0.569	0.797	50.20%	17.50%	28, 28	27, 256
2	168	0.966	0.664	0.746	0.746	22.00%	8.20%	28, 28	26, 256
3	169	0.823	0.713	0.691	0.758	13.20%	4.50%	28, 28	26, 256
4	86	0.700	0.749	0.614	0.778	8.60%	2.90%	28, 28	22, 256
5	96	0.621	0.779	0.572	0.797	4.90%	1.80%	28, 28	22, 256
6	122	0.836	0.716	0.696	0.757	14.00%	4.10%	28, 28	22, 256
7	123	0.663	0.772	0.617	0.787	4.60%	1.50%	28, 28	22, 256
8	46	0.776	0.729	0.697	0.756	7.90%	2.70%	28, 28	19, 256

Model 1 trained for 62 seconds and achieved a relatively low train loss of 1.071 and a train accuracy of 0.622. However, its test loss of 0.569 and test accuracy of 0.797 suggest a significant overfitting problem, with a large delta between train and test loss of 50.20% and accuracy 17.50%. It is not the best choice due to overfitting.

Model 2, trained for a longer period, 168 seconds, and achieved a train loss of 0.966 and a train accuracy of 0.664. its test loss of 0.746 and test accuracy of 0.746 show a relatively balanced performance. However, it still suffers from overfitting, with a significant delta in loss of 22% and accuracy at 8.2%. it is better than **model 1** but could be improved.

Model 3 trained for 169 seconds, achieving a train loss of 0.823 and a train accuracy of 0.713. its test loss of 0.691 and test accuracy of 0.758 demonstrate better generalization compared to **model 1** and **model 2**. The overfitting is less pronounced with a delta in loss of 13.2% and a delta in accuracy of 4.50%. **model 3** seems to be a better choice compared to the first two.

Model 4 has a training time of 86 seconds and achieving a train loss of 0.700 with a train accuracy of 0.749. Its test loss of 0.614 and test accuracy of 0.778 indicate good generalization

and a smaller delta in loss at 8.6% and accuracy 2.9% compared to the previous models. **Model 4** is a strong contender for the best model.

Model 5 was trained for 96 seconds and achieved a train loss of 0.621 and a train accuracy of 0.779. it performed even better on the test data with a test loss of 0.572 and a test accuracy of 0.797. the delta in loss at 4.9% and accuracy of 1.8% is small, indicating minimal overfitting. This suggests that **model 5** not only learned the training data effectively, but also generalized its knowledge to unseen examples, making it a strong candidate for deployment in real-world application. The balance between training and test metrics in **model 5** is a positive indicator of its performance.

Model 6 trained for 122 seconds, achieving a train loss of 0.836 and a train accuracy of 0.716. Its test loss of 0.696 and test accuracy of 0.757 show good generalization. The delta in loss 14% and accuracy 4.1% suggests some overfitting, making it a slightly less favorable choice.

Model 7 trained for 123 seconds and achieved a train loss of 0.663 with a train accuracy at 0.772. its test loss of 0.617 and test accuracy of 0.787 demonstrate a good balance between training and testing performances. The delta in loss 4.6% and accuracy 1.5% is relatively small, making **model 7** a strong candidate for the best choice.

Model 8 stands out as the fastest to train, taking only 45 seconds, and it maintained a good tradeoff between training and testing performance. It achieved a train loss of 0.776 and a train accuracy of 0.729. its test loss of 0.697 and test accuracy of 0.756 indicate a balanced performance. The delta in loss 7.9% and accuracy 2.7% points to some overfitting. This model is a strong candidate for tasks where computational resources and training time are limited while still obtaining competitive results.

Conclusions and Takeaways

The model that stands out is **model 5**. Its performance metrics clearly demonstrate its superiority in multiple aspects. First, when examining the training loss, **model 5** achieves an impressive value of 0.621, the lowest among all models. This signifies that it learns to fit the training data very well, indicating a high level of model convergence. Furthermore, its training accuracy of 0.779 is the highest of all the models. It demonstrates its ability to correctly classify data points in the training set. A graph of training accuracy and validation accuracy can be viewed here in [Figure 5](#).

What truly sets **model 5** apart is its ability to generalize its learning to unseen data. It has a test loss of 0.572 which is the second lowest, but the real highlight is its test accuracy of 0.797. This indicates not only learning the training data well but also managed to generalize its knowledge effectively to new examples. The delta in train loss and test loss is a mere 4.9%, showcasing robustness.

Model 5 can be improved in the future with a faster training time possibly with reducing the number of hidden layers; however, this could lead to not capturing all the data. Another way could be to increase the number of layers giving it more opportunities to address unseen data. But this also could lead to overfitting. Adding additional epochs to the model, could increase its accuracy. On the other hand, it would increase training time if resources were limited.

In either case, **model 5** is the best choice for deployment in a real-world situation for identifying grayscale images out of the eight models developed.

References

- Göllner, S. (2022, July 14). *How to reduce training parameters in CNNs while keeping accuracy >99%*. Retrieved from Towards Data Science: <https://towardsdatascience.com/how-to-reduce-training-parameters-in-cnns-while-keeping-accuracy-99-a213034a9777>
- Han Xiao, K. R. (2017, August 28). *zalando research / fashion-mnist*. Retrieved from git hub: <https://github.com/zalando research/fashion-mnist>
- Mishra, M. (2020, August 26). *Convolutional Neural Networks, Explained*. Retrieved from Towards Data Science: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>
- Pramoditha, R. (2021, December 21). *How RGB and Grayscale Images Are Represented in NumPy Arrays*. Retrieved from Towards Data Science: <https://towardsdatascience.com/exploring-the-mnist-digits-dataset-7ff62631766a>
- Saha, S. (2018, December 18). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved from Towards Data Science: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

Appendix

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Figure 1: Label Description

```
# Show images in the data
import matplotlib.pyplot as plt
import numpy as np

# train_x is a list or array of images
num_images_to_display = 20 # Display the first 20 images
images_to_display = train_x[:num_images_to_display] # Select the first 20 images

# Calculate the number of rows and columns for the 5x4 grid
num_rows = 5 # Number of rows in the grid
num_columns = 4 # Number of columns in the grid

# Create a subplot grid for displaying images
fig, axes = plt.subplots(num_rows, num_columns, figsize=(12, 4)) # Figure size

# Loop through and display the selected images
for i, ax in enumerate(axes.flat):
    ax.imshow(images_to_display[i], cmap='gray') # Images are grayscale
    ax.axis('off') # Turn off axis labels

plt.show()
```

Figure 2: Code Snippet

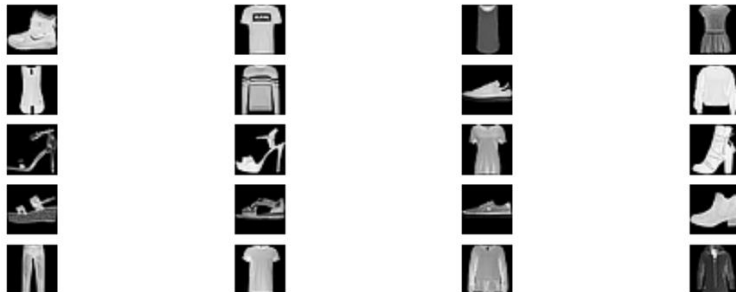


Figure 3: First 20

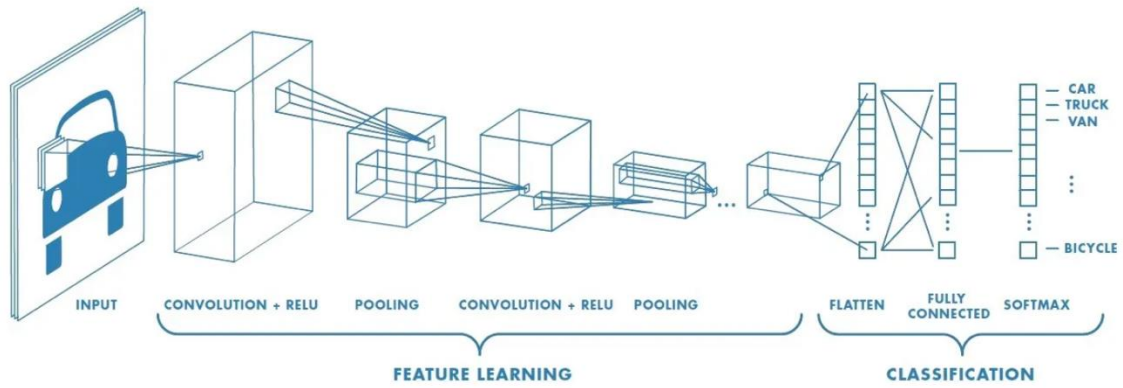


Figure 4: Visualization of CNN

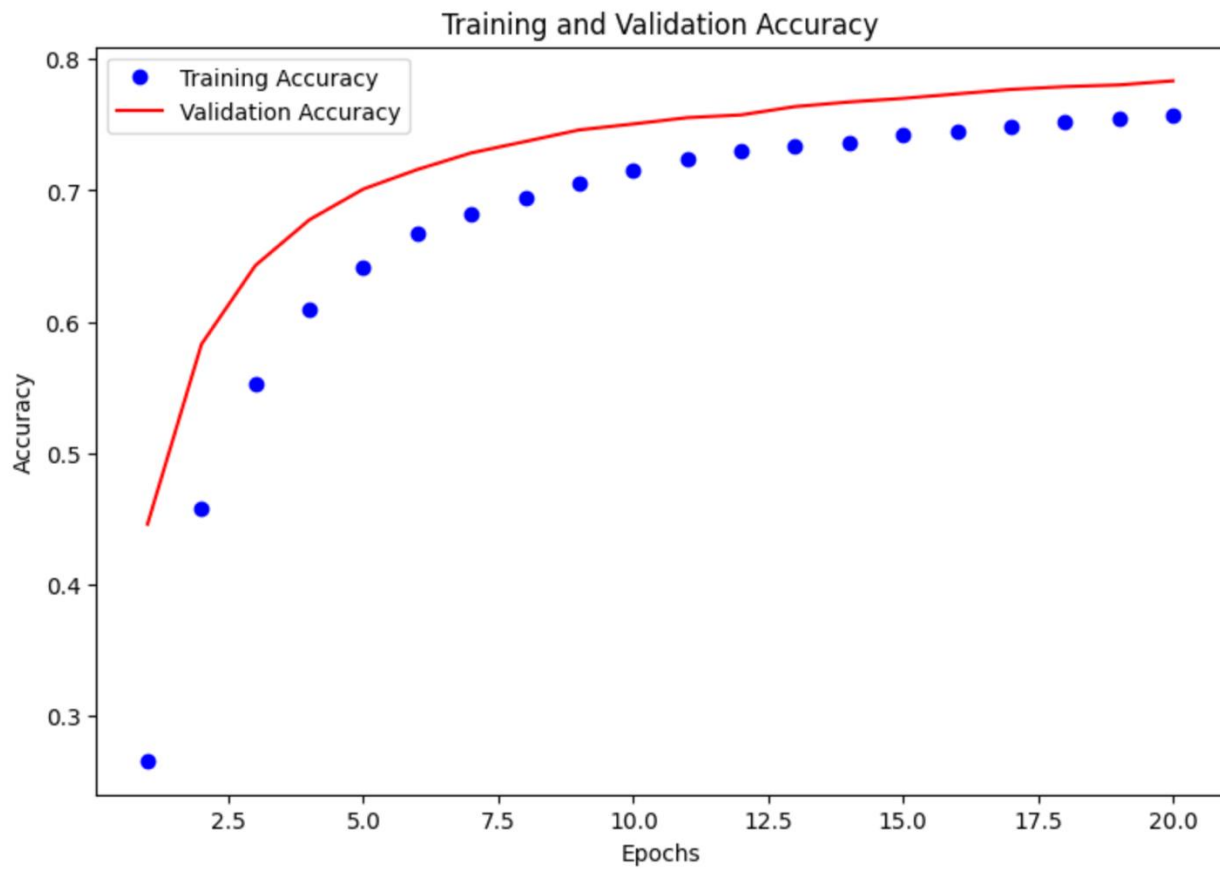


Figure 5: Training and Validation Accuracy