

Assignment 4

**CNN Analysis on Fashion MNIST Dataset:
Image Identification of Clothing Items**

Theodore Fitch

Department of Data Analytics, University of Maryland Global Campus

DATA 640: Predictive Modeling

Dr. Steven Knode

March 5th, Spring 2024

Introduction:

The purpose of this report was to use a Convolution Neural Network (CNN) to accurately identify images from the Fashion MNIST dataset and demonstrate how changing parameters affects accuracy results. CNNs are a powerful class of deep learning models designed for problems like image classification, object detection, or image segmentation. There are several key components to understanding CNNs. They fundamentally require convolutional layers (which convert the input data using kernels/filters in order to capture patterns or features of the images), a ReLU layer (a.k.a. a rectified linear unit layer, which introduces non-linearity by replacing the negative values with zero), a pooling layer (a.k.a. subsampling, which decreases computational load by reducing the spatial dimension of the input), and a fully connected layer (after several ReLUs and convolutional layers, this layer flattens the information one-dimensionally, and then uses the information to classify the image via the Softmax activation function)(Kayed et al., 2020). By increasing the complexity of the model by increasing number of layers and other numbers (like filters, neuron count, etc.), the model should be able to gain greater accuracy. Image classification is an incredibly useful tool that has countless applications (of which include: disease detection, robotics, brake light detection, self-driving cars, and more)(Garg, 2022). The Fashion MNIST dataset has already been studied by many using a variety of mechanisms, and a previous study was able to attain a 99.37% accuracy (Kadam et al., 2020). This study aimed to change parameters of a basic CNN model, and add tools from Keras (n.d.) in order to show how the parameters affect accuracy, and increase accuracy of the model. Thus, several, new, unique CNN models will be generated with hopes to equal or surpass current accuracy rates on the Fashion MNIST dataset.

Exploratory Data Analysis and Preprocessing:

The Fashion MNIST is an upgrade from the traditionally used MNIST dataset featuring handwritten numbers. While that dataset was incredibly helpful and ubiquitous, it also had several critical drawbacks: It became too easy to model (traditional ML models could easily achieve 97% accuracy and CNNs could achieve 99.7%). It also doesn't represent modern computer vision tasks well; recognizing handwritten numbers is a good baseline, however modern CNNs should be able to accomplish much more than just this (Zalando research, n.d.). Thus, this dataset was generated by Zalando in order to make a more complex dataset that could replace MNIST but still contains the same dimensionality. It contains 10 classes of clothing articles, and each example is a 28 x 28 greyscale image labelled 0-9 (Table 2). The training set contains 60,000 entries with a test set of 10,000 images. A database of classifier models was previously generated for comparison showing that traditional ML models (gradient boosting, SVC, decision trees, etc.) maintain around 80% accuracy. The first 25 images and their associated label were pulled from the training dataset as an example (Figure 1). Furthermore, the first image was processed such that the 28 x 28-pixel grid was labelled and each pixel was labelled with its associated greyscale value (Figure 2). This image makes it easy to understand how the CNN can map out the information in the image. No preprocessing was needed on this dataset (any changes to the images themselves are performed during modeling, and it was already partitioned for training and testing).

Models and Methods:

A total of 8 models were generated and explored in this report coded using Python in Google Colab (Table 3). First, a baseline CNN model was given by the classroom (Knode, n.d.). This was run, adjusted, and tweaked; but ultimately, a different structure of CNN was found and used as the baseline CNN simply to generate differentiation between this report and others made

by the classroom (West, 2024). This **Model 1** started off with 3 convolutional layers, 2 pooling layers, 1 dense layer, 10 training epochs and no dropout. It used a convolutional layer (Conv2D) with 32 filters and a kernel size of 3,3. There are a couple of general pooling types: MaxPooling (which takes the maximum value of each region of the input feature map) and AveragePooling (which takes the average value of each region of the input feature map)(Keras, n.d.). It also had a training batch size of 64, and a training validation split of 80:20. The next 6 models were variations of this model with different parameters adjusted in order to determine their effect on the resultant accuracy. **Model 2** increased the filters on the third convolutional layer, increased the number of dense layers, added a dropout function, and increased training epochs to 15. The dropout function is a regularization technique which helps to prevent overfitting. It functions by randomly “dropping out” some amount of input units during training. This forces the model to become more robust during training (Keras, n.d.). Next, **Model 3** kept the changes made in Model 2, and stretched the epochs to 50 to see if increased training would result directly in increased accuracy.

Subsequently, **Model 4** made several changes to Model 2: it increased the first and second dense layer neuron number, it increased dropout layers to 2 and adjusted the numbers to 0.4 and 0.3 respectively, and lastly it also added a batch normalization layer after each convolutional layer (to increase training stability). All these changes were added with the idea that a more complex CNN with more layers would help to increase accuracy. **Model 5** then made two additions: it increased training epochs to 30 and it augmented the data using "ImageDataGenerator" by adjusting images (using zoom, shifts, shear, or rotation) so that the model trains more robustly (Keras, n.d.). The idea behind this being that if the training dataset contains randomly inserted small to moderate distortions in the images, then the model is forced

to become more robust by identifying better patterns on the training dataset. **Model 6** kept the data augmentation added in Model 5, but also increased the filter numbers and the dense layer neurons, while decreasing epoch number. Model 6 also had a learning rate scheduler added to it; this tool helps to dynamically control the learning rate of a model during training so that the model learns quickly but doesn't overtrain (Keras, n.d.).

Model 7 then explored changing the rate of the training batch size for the first time, and it also increased the dense layer neurons; it used Model 2 as a baseline and then made the aforementioned adjustments. Several attempts were made to adjust optimizers on Model 7 but each attempt either showed significant drops in accuracy (SGD) or remained unchanged (RMSprop). Lastly, while exploring several other techniques, **Model 8** was generated using the base structure of a TensorFlow (n.d.) CNN with the dense layer neurons increased to 256 and the epochs increased to 15. This model used a slightly different structure but still optimized using the same methods (MaxPooling, optimized for accuracy). This was added in order to demonstrate a different type of model that could be generated. Example code of Models 7 and 8 show that Model 7 has a generally more complicated structure than Model 8 (Figure 3, Figure 4). Thus, these differences would help explore the accuracy outcomes of multiple different core functions and parameters of CNNs, as well as additional tools. A comprehensive table was made for direct head-to-head comparison of each model, and highlights (in underline) the key features changed in each model (Table 3).

Results and Model Evaluation:

Index	Model Name	Highest Train Accuracy	Highest Validation Accuracy	Test Accuracy	Misclassification Rate	Train-Test Accuracy (Delta)
1	Model 1	93.68%	91.20%	90.49%	9.51%	3.19%
2	Comments: <u>Baseline model</u> given as example (West, 2024)					
3	Model 2	94.67%	91.39%	90.90%	9.10%	3.77%
4	Comments: First iterated model changed from baseline					
5	Model 3	98.62%	90.94%	90.70%	9.30%	7.92%
6	Comments: <u>increasing epochs</u> decreased accuracy, due to overfitting					
7	Model 4	95.04%	90.21%	89.64%	10.36%	5.40%
8	Comments: Added <u>batch normalization</u> layer after each convolutional layer increased overfitting					
9	Model 5	88.01%	89.30%	89.30%	10.70%	-1.29%
10	Comments: Data augmented using " <u>ImageDataGenerator</u> " by adjusting images (using zoom, shifts, shear, or rotation) so that the model trains more robustly. Performed better on test data than train data.					
11	Model 6	86.71%	87.74%	87.74%	12.26%	-1.03%
12	Comments: Kept <u>data augmentation</u> ; added a <u>learning rate scheduler</u> . Performed better on test data than train data.					
13	Model 7	95.39%	91.68%	91.73%	8.27%	3.66%
14	Comments: Model2 adjusted to increase the <u>dense layer neurons</u> and <u>increase batch size</u> increased overall accuracy. Champion model.					
15	Model 8	92.90%	88.54%	87.22%	12.78%	5.68%
16	Comments: Example CNN from TensorFlow (n.d.) with <u>increased neurons</u> and <u>epochs</u> increased <u>overfitting</u> .					

Table 1. Accuracy measures of the 8 models generated evaluated by Accuracy and Overfitting show that Model 9 is the champions with a test accuracy of 91.73% (highlighted red). Test-Train Accuracy represents the subtraction between the two to demonstrate overfitting of models (green fonts show negative numbers showing models that performed better on the test data; red fonts show models where overfitting was observed).

The models primarily needed to be interpreted through accuracy since it primarily mattered if the models correctly identified the image correctly. The sensitivity (or TPR or Recall), precision, and F1 score in this case matters more for the individual categories than the overall models (and so they will be discussed more later). Of the 8 models generated, 3 models had an accuracy greater than the baseline model (2, 3, and 7). Model 7 performed best of all at an accuracy of 91.73%. This is only 1% above the baseline Model 1's performance of 90.49%. The test accuracy did not deviate far from baseline (range of 87.2-91.7%). The first edits made on the

baseline model to generate Model 2 appeared to have negligible results as accuracy barely changed. Adding extra dense layers, increasing the neuron count, increasing epoch time, and adding dropout did not seem to have an effect. Some of these parameters were played with individually (in order to isolate if any had a positive or negative effect) but none was observed as accuracy remained near 90%. Next, drastically increasing epochs of training had an interesting effect: Model 3 clearly overtrained as it achieved a 98.6% train accuracy that dropped to 90.7% on test data. This clearly shows that the Model became too attuned to the training dataset as it displayed the highest sign of overfitting (Delta: 7.92%). Model 4 also showed a high metric for overfitting (Delta: >5%) and accuracy decreased by 1%. This model increased the dense layer neurons, added a second dropout layer, and adjusted the dropout rate. As previously adjusting the neuron layers in Model 2 did not appear to be effective, it was suspected that the dropout rates/layers were causing the decrease in accuracy (and this was tested by individually removing them and re-running the model). The batch normalization layers appeared to make no difference (and was tested by removing/keeping them).

Next, Model 5 was the first to introduce the data augmentation step. It is critical to note that the data augmentation added to Models 5 and 6 only caused them to be the only models that were not overfit to the train dataset. Each of them had a negative train accuracy minus test accuracy showing that they performed better on the test data than on data they were modeled to. This feature is thus key to keep for further modeling to make them more robust. The second adjustment to this model was increasing epoch number – while epoch number previously was shown to decrease accuracy in Model 3, keeping a lower epoch number (10) actually decreased accuracy to 86%. This makes sense as the data augmentation helps balance models from being overfit, and increasing training epoch number can cause models to be more accurate but also can

be overfit. Thus, the epoch training time simply needs to be fine-tuned, and should be counterbalanced by a method to prevent overfitting (like data augmentation). As previously mentioned, Model 6 had a learning rate scheduler added to it, and also adjusted the filter numbers, and dense layer neurons. No major differences were observed between the Accuracy of Model 5 and 6 except that the latter was slightly lower. They had similar Delta meaning both were not overfit; but, that also means the adjustments made to improve Model 6 did not help.

Due to this, Model 7 took a step back to iterating Model 2 (since adjusting the previous items did not increase accuracy by much). So, Model 2 was adjusted to firstly increase training batch size (for the first time). Then, each parameter was adjusted in order to see what items would increase accuracy: increasing the filter number and dense layer neurons both helped to increase it. The resulting accuracy was 91.73% which was the best observed from this dataset. Many previous analyses of this dataset have been able to achieve >75% and some of the best from Kadam et al. show accuracy of ~93% (2020)(the example CNN provided by the classroom yielded a test accuracy of 70% [Knode, n.d.]). Thus, Model 7 is considered the champion model generated. Two figures were made in order to further explore Models 7 and 8: a confusion matrix and the other model accuracy measures (Figure 5, Figure 6, Figure 7, Figure 8). The confusion matrix shows a few key takeaways: it shows that items that could be commonly confused were confused by the model (e.g. shoes, sandals, and ankle boots all had strong crossover). Shirts were by far the most confused category (for Model 7 only 667 correct predictions, where the second worst was 758 for pullovers). Model 7 appeared to have the most difficulty predicting shirts, t-shirts, pullovers, and coats as these were all of the worst performing categories. This is also reflected in the accuracy measures as shirts had by far the worst F1-score (69%) followed by: pullovers (78%), coats (80%), and t-shirts (84%). The Model 8 results reflect the same trend

(with slightly different F1-score and a different order of worst predictions, however shirts is still the worst category). F1-score is the critical measure to use from this table since it is the harmonic mean of precision (accuracy of positive predictions made) and recall (TPR). In many business cases, it would be critical to identify all positive values and missing any would be costly, and thus TPR is the critical accuracy measure. However, in image classification problems, F1-score is more important since it balances both TPR and precision, as image classification usually needs to identify all categories. As a caveat, there are some image classification problems in which TPR is critical (like disease diagnosis using radiological or microscope biopsy images).

To visualize how Model 8 was working, a plot was made to show the first 25 images of the test dataset and how the model predicted them (Figure 9). This showed how the model correctly/incorrectly predicts the images and shows how it gets them confused with other categories. A subsequent plot was made showing the first 25 incorrectly predicted images in the dataset. This helps us visualize how the model is incorrectly predicting items (Figure 10). We as observers can now see how the Model incorrectly predicts items and how certain images do have similarities to others. For example, the third incorrect prediction is a boot with a white insignia that makes it appear more like a sandal, which it was incorrectly predicted to be. This gives the idea that these models may perform better if contrast is adjusted using the ImageDataGenerator to help the models differentiate the classes more precisely. As mentioned before, Model 8 was generated using a generally different and simpler CNN architecture. It also showed signs of overfitting (5.7%) as the training accuracy (92.9%) was higher than the test accuracy (87.2%). This shows that it is possible to have a simpler CNN structure but this consequently leads to less accurate outcomes.

Conclusion, Limitations, and Improvements:

In total, 8 CNNs were modeled on the Fashion MNIST dataset generating a range of accuracy from 87.22% to 91.73% representing an increase from the baseline model provided by the classroom of 17-21%. Thus, there are a few key takeaways. Adjusting dense layer number or neuron count didn't seem to have an effect on accuracy. Adjusting dropout rates and layers had a negative effect. The data augmentation must be kept as this forces the models to become more robust and increase their accuracy on test datasets versus the training dataset. Any further modeling should include this feature. The learning rate scheduler did not appear to have a positive effect on accuracy but further experimentation with this should be performed.

Depending upon the business application, the accuracy measures can be used to determine how to fine tune the models. If shirts are critical to differentiate, then TPR/Recall/Sensitivity should be maximized. More attention can be given to rewarding the model for correct predictions and penalizing it for incorrect predictions. Possible future model adjustments should include adding the data augmentation step to the champion model, and if it would correct the overfitting observed on Model 8. Specifically, ImageDataGenerator should attempt to adjust contrast on images to see if this increases accuracy to differentiate the problem categories (shirts, pullovers, coats, and t-shirts). Increasing epochs and adding the data augmentation together should increase accuracy while preventing overfitting from occurring. Furthermore, there are countless other options from the Keras documentation that could be adjusted (2024): the optimizer could be adjusted further (SGD, RMSprop, Adadelta, Adamax, etc.); further data augmentation could be performed (like increasing/decreasing contrast in the images to prevent mix-ups between the hard to differentiate categories); adjusting the kernel number of the convolutional layers; adjusting the pooling method (attempting averaging versus maxing); and adjusting each layer number further.

References:

- Garg, A. (2022). How to make an image classification model using Deep Learning?. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2022/11/how-to-make-a-image-classification-model-using-deep-learning/>
- Kadam, S. S., Adamuthe, A. C., & Patil, A. B. (2020). CNN model for image classification on MNIST and fashion-MNIST dataset. Journal of scientific research, 64(2), 374-384.
- Kayed, M., Anter, A., & Mohamed, H. (2020). Classification of garments from fashion MNIST dataset using CNN LeNet-5 architecture. In 2020 international conference on innovative trends in communication and computer engineering (ITCE) (pp. 238-243). IEEE.
- Keras (n.d.). Keras 3 API Documentation. Keras. <https://keras.io/api/>
- Knodel (n.d.). Model Fashion MNIST Database.ipynb. Google Colab. https://colab.research.google.com/drive/1ITdNyCGyTW5V5bK4sfU70y6_LYMZMrMJ
- TensorFlow (n.d.). Basic classification: Classify images of clothing. TensorFlow. <https://www.tensorflow.org/tutorials/keras/classification>
- West, M. (2024). Convolutional Neural Networks : An implementation. Bouvet Norge. <https://www.bouvet.no/bouvet-deler/understanding-convolutional-neural-networks-part-2>
- Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.
- ZalandoResearch. (n.d.). ZalandoResearch/fashion-mnist: A mnist-like fashion product database. benchmark. GitHub. <https://github.com/zalandoResearch/fashion-mnist>

Appendix:

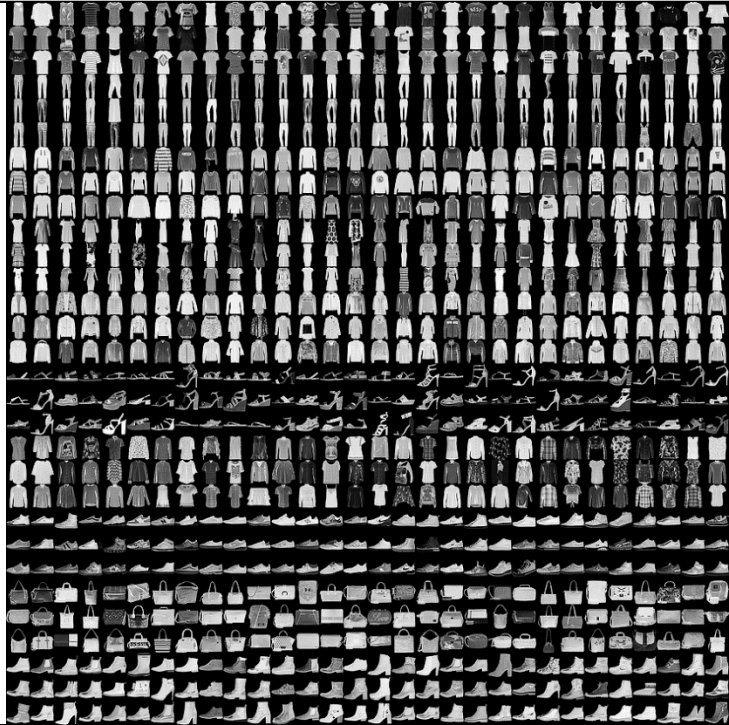
Label	Description	Example
0	T-shirt/top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandal	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boot	

Table 2. The 10 dataset variable labels, descriptions, and image examples (Zalandoresearch, n.d.)



Figure 1. Example data from the Fashion MNIST dataset showing the first 25 images in the set along with their labels (West, 2024).

Index	Model Name	Convolutional Layers	Filter Number	Kernel Number	Pooling Layers	Dense Layer Neurons	Training Epochs	Training Batch Size	Dropout Rate
1	Model 1	3	32, 64, 64	3,3	2	64	10	64	N/A
2	Comments: Baseline model given as example (West, 2024)								
3	Model 2	3	32, 64, 128	3,3	2	128, 64, 10	15	64	0.5
4	Comments: First iterated model changed from baseline; added <u>layers</u> , increased <u>filters</u> , <u>neurons</u> , and <u>epochs</u> . Added <u>dropout</u> .								
5	Model 3	3	32, 64, 128	3,3	2	128, 64, 10	50	64	0.5
6	Comments: <u>increased epochs</u> to see if this increases accuracy or overfits the model								
7	Model 4	3	32, 64, 128	3,3	2	256, 128, 10	15	64	0.4, 0.3
8	Comments: Added <u>batch normalization layer</u> after each convolutional layer. Added <u>second dropout layer</u> .								
9	Model 5	3	32, 64, 128	3,3	2	256, 128, 10	30	64	0.4, 0.3
10	Comments: Data augmented using " <u>ImageDataGenerator</u> " by adjusting images (using zoom, shifts, shear, or rotation) so that the model trains more robustly.								
11	Model 6	3	64, 128, 256	3,3	2	512, 256, 10	10	64	0.4, 0.3
12	Comments: Kept <u>data augmentation</u> ; added a <u>learning rate scheduler</u>								
13	Model 7	3	64, 128, 256	3,3	2	256, 128, 10	10	128	0.5
14	Comments: Model2 adjusted to increase the <u>dense layer neurons</u> , and <u>increase batch size</u>								
15	Model 8	N/A	N/A	N/A	N/A	256, 10	15	64	N/A
16	Comments: Example CNN from TensorFlow (n.d.) adjusted <u>neuron count</u> and <u>epoch number</u>								

Table 3. Overview of the 8 CNN models generated to predict Fashion MNIST dataset. Red writing indicates any changes from previous model (most models iterated from the previous model unless comment denotes otherwise).

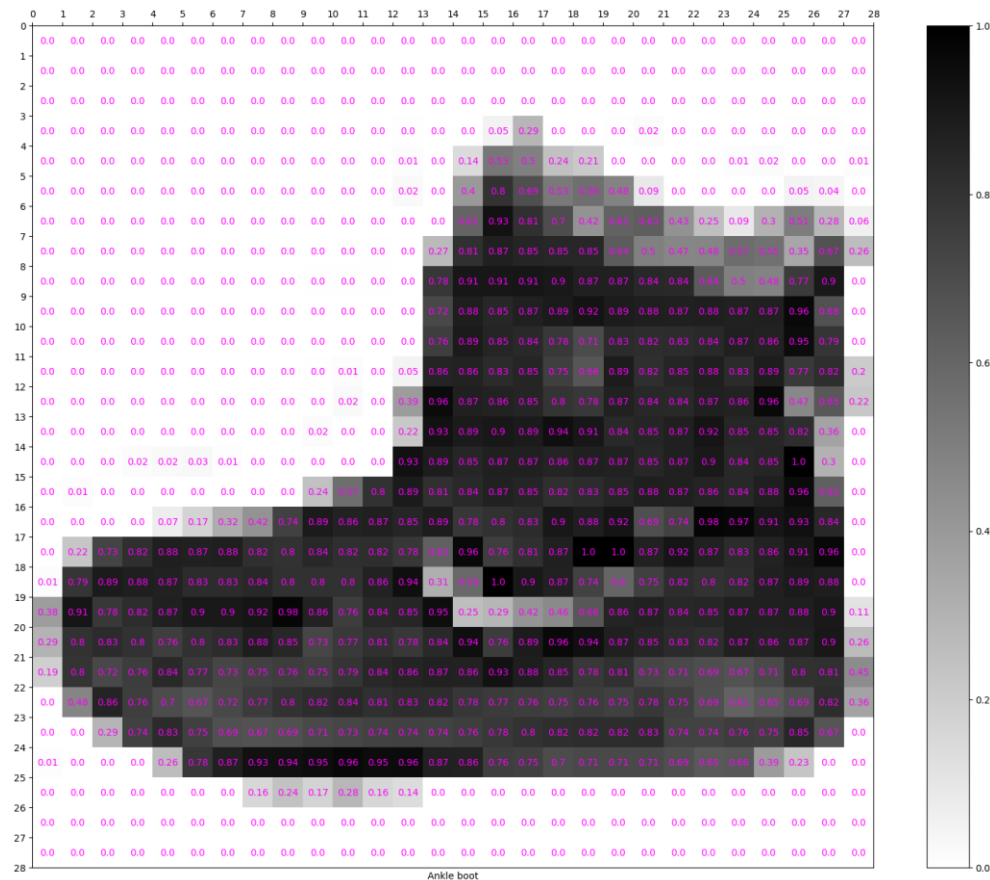


Figure 2. The first image from the Fashion MNIST dataset shows the 28 x 28-pixel grid along with the greyscale value associated inside each pixel (West, 2024).

```

# Model17: Adjusted CNN model from Model 2, increased batch size and dense layer neurons
model17 = models.Sequential()
model17.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model17.add(layers.MaxPooling2D((2, 2)))
model17.add(layers.Conv2D(128, (3, 3), activation='relu'))
model17.add(layers.MaxPooling2D((2, 2)))
model17.add(layers.Conv2D(256, (3, 3), activation='relu')) # Increased filters
model17.add(layers.Flatten())
model17.add(layers.Dense(256, activation='relu')) # Increased neurons
model17.add(layers.Dropout(0.2)) # Added dropout for regularization
model17.add(layers.Dense(128, activation='relu'))
model17.add(layers.Dense(10, activation='softmax'))

# Compile Model17
model17.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Train Model17
model17.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)

# Evaluate Model17
predictions = model17.predict(test_images)
predicted_labels = [np.argmax(prediction) for prediction in predictions]
true_labels = [np.argmax(label) for label in test_labels]

```

Figure 3. Example python code shows the architecture of Model 7 had a more complicated CNN structure than Model 8.

```

# Model18: Modified TensorFlow (2024) model with additional metrics and validation split
model18 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(10)
])

model18.compile(optimizer='adam',
                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

# Train Model18 with validation split
model18.fit(train_images, train_labels, epochs=15, validation_data=(val_images, val_labels))

# Evaluate Model18
predictions = model18.predict(test_images)
predicted_labels = [tf.argmax(prediction).numpy() for prediction in predictions]

```

Figure 4. Example python code shows the architecture of Model 8 has a less complicated CNN structure than Model 7.

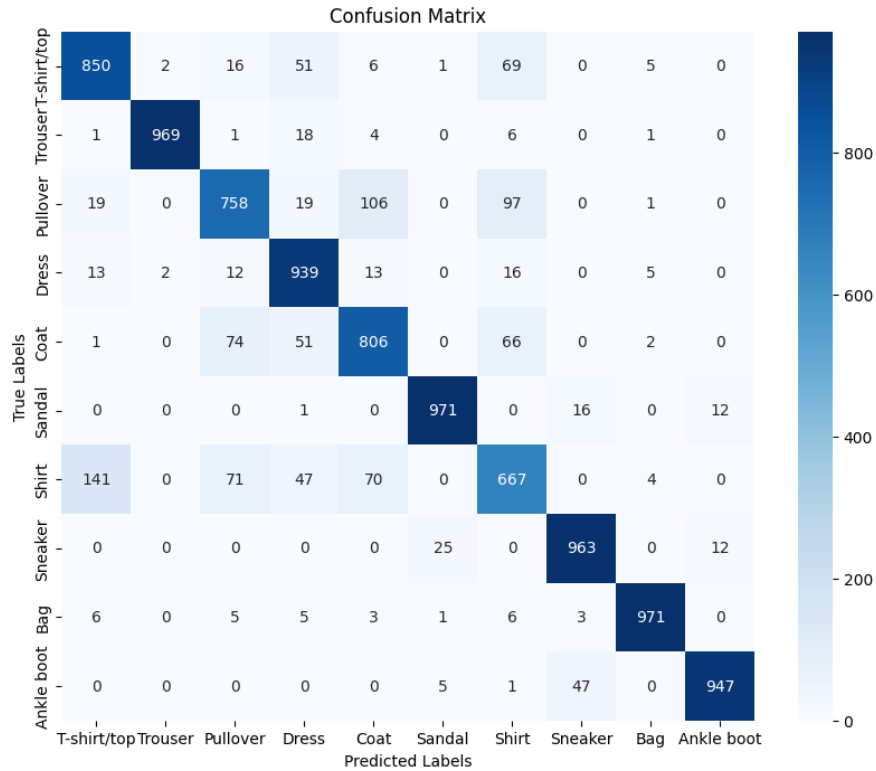


Figure 5. The confusion matrix generated for Model 7 shows the correct/incorrect predictions of each of the 10 categories on the test dataset. The most commonly incorrectly predicted categories (with high crossover) were shirt, t-shirt, pullover, and coat.

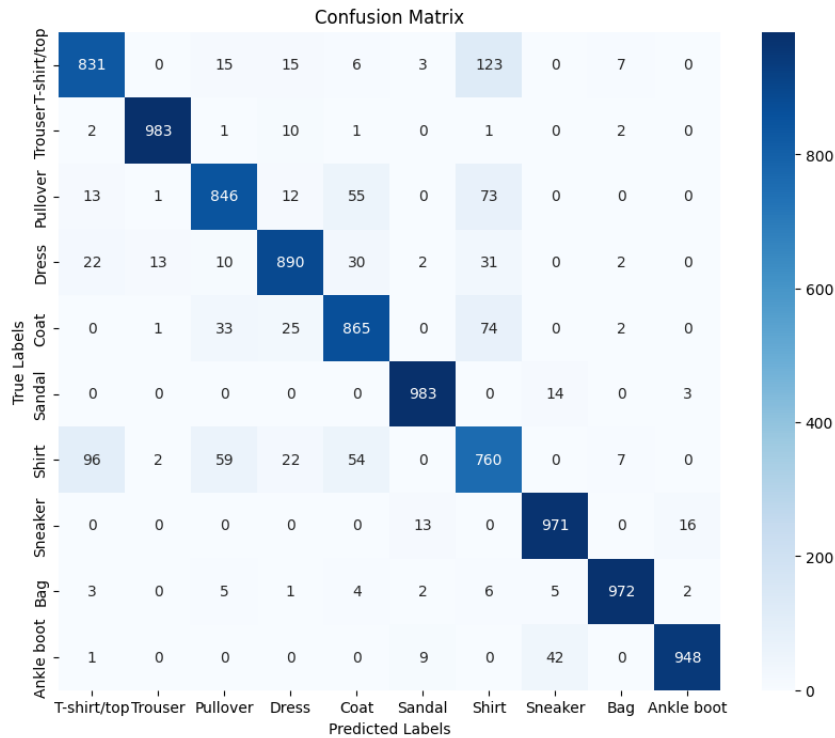


Figure 6. The confusion matrix generated for Model 8 shows the correct/incorrect predictions of each of the 10 categories on the test dataset. The most commonly incorrectly predicted categories (with high crossover) were shirt, t-shirt, pullover, and coat.

Classification Report:

	precision	recall	f1-score
T-shirt/top	0.82	0.85	0.84
Trouser	1.00	0.97	0.98
Pullover	0.81	0.76	0.78
Dress	0.83	0.94	0.88
Coat	0.80	0.81	0.80
Sandal	0.97	0.97	0.97
Shirt	0.72	0.67	0.69
Sneaker	0.94	0.96	0.95
Bag	0.98	0.97	0.98
Ankle boot	0.98	0.95	0.96

Figure 7. The other accuracy measures generated for Model 7 of precision, recall (sensitivity/TPR), and F1-score.

Classification Report:

	precision	recall	f1-score
T-shirt/top	0.86	0.83	0.84
Trouser	0.98	0.98	0.98
Pullover	0.87	0.85	0.86
Dress	0.91	0.89	0.90
Coat	0.85	0.86	0.86
Sandal	0.97	0.98	0.98
Shirt	0.71	0.76	0.74
Sneaker	0.94	0.97	0.96
Bag	0.98	0.97	0.98
Ankle boot	0.98	0.95	0.96

Figure 8. The other accuracy measures generated for Model 8 of precision, recall (sensitivity/TPR), and F1-score.

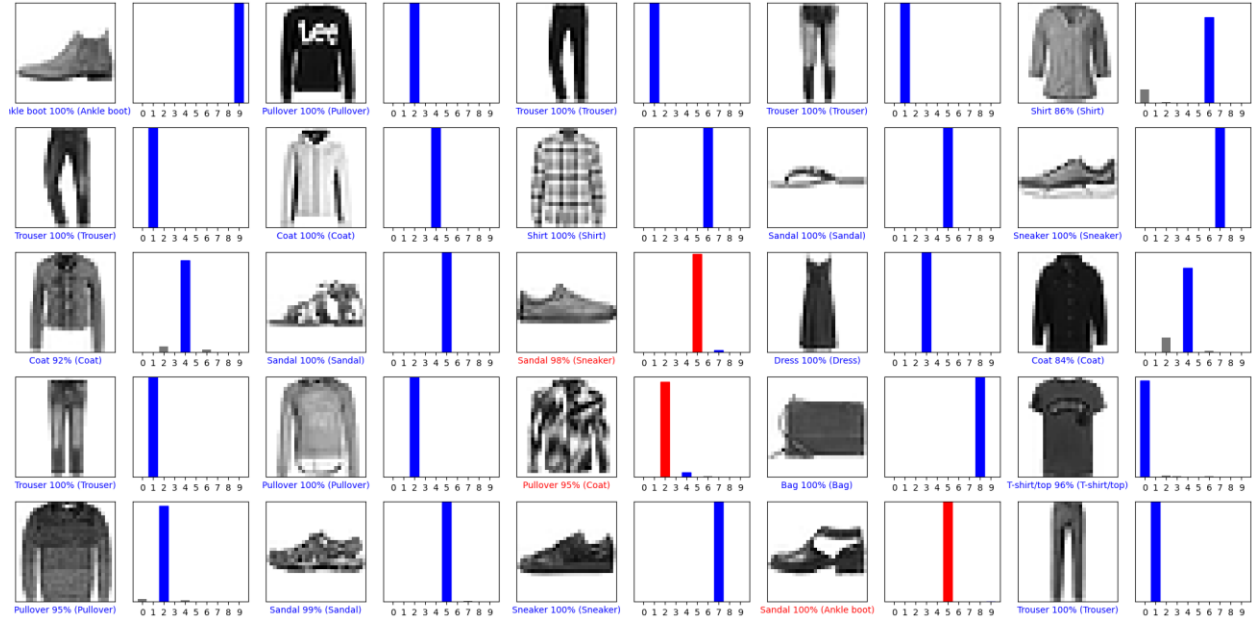


Figure 9. The first 25 images in the test dataset and model 8's prediction for each item. Blue is correctly predicted, red is incorrectly predicted, and grey is potentially chosen but not categorized (for example, entry 8 above was categorized as a shirt correctly with 90% confidence, but there was 10% confidence it should have been a coat). Category index (from Table 2) printed on x-axis. Correct category printed in parentheses and predicted category printed to the left (with confidence percentage).

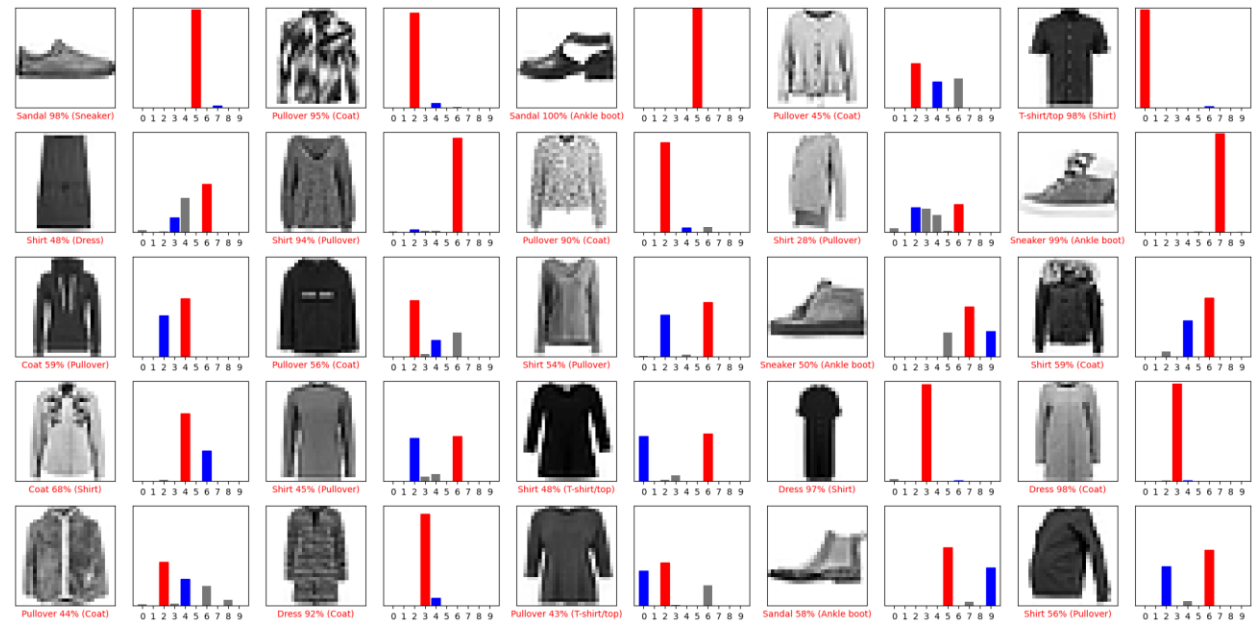


Figure 10. The first 25 incorrectly predicted images in the test dataset and model 8's prediction for each item. Blue is correctly predicted, red is incorrectly predicted, and grey is potentially chosen but not predicted. Category index (from Table 2) printed on x-axis. Correct category printed in parentheses and predicted category printed to the left (with confidence percentage).