Fashion MNIST: The Classification of Clothing Articles

Zachary Higa
CS 577: Python in Machine Learning
Spring 2023
Department of Statistics
University of Idaho
Moscow, ID USA
higa2426@vandals.uidaho.edu

I. Introduction

The Fashion MNIST dataset consists of images that correspond to 10 different classes of clothing articles. Within the dataset, there are 60,000 training images and 10,000 testing images with each image of size 28 pixels by 28 pixels. This dataset was introduced as a replacement for the original MNIST dataset that is typically used as a benchmark to see if the algorithms will run correctly as "if it doesn't work on MNIST, it won't work at all" [8].

However, some data scientists and researchers have started to stray away from using the original MNIST as a benchmark dataset because classifying using the original MNIST is "too easy" with many classic machine learning algorithms being able to achieve a 97% accuracy on this dataset, and CNN being able to achieve a 99.7% accuracy on the dataset [5]. Therefore, this Fashion MNIST dataset was developed, and as it seeks to take the place of the original MNIST dataset as a benchmark dataset, I was interested in developing my own programs to work with this dataset.

A. *Task Description*

I downloaded the Fashion MNIST dataset and learned about the description and characteristics of the dataset through Kaggle [8], though I ultimately loaded the data as a built-in keras dataset [1]. What I hope to achieve in my program is to be able to correctly classify the images of clothing articles with an accuracy of at least 90% on the test set; though it goes without saying that if I do meet this threshold, then I will continue to optimize my program to achieve the best performance possible.

II. Methods

A. *Selection of Method – Convolutional Neural Network*

In my program, I train a model using a convolutional neural network (CNN). CNN is widely used to train models for the processing of images because within this specific type of artificial neural network, CNN simplifies images by using convolution layers to detect small patterns in an image as well as max pooling to subsample the pixels from the convolution layers to make the images smaller. Through these convolution layers and max pooling layers, we extract a feature map from the original image that we feed into a dense layer where the network is now fully connected, and we then feed the output of this layer to our output layer where we classify the image to one of the ten class labels [3].

In the convolutional layers of a CNN, we define a number of filters of a specified kernel size that will detect small patterns in each of the filters. These convolutional layers are useful for image processing because we find that some patterns in an image are smaller than the entire image and sometimes these patterns appear in different places in different images. Thus, having the convolutional layers to use filters to detect these patterns in smaller sections of an original image is useful to have in image processing [3].

In the maxed pooling layers of a CNN, we define a pooling size that will take a subsample of the original image. This is useful in reducing image sizes, and reducing the image sizes means that there are less parameters needed for the network to process the image. Therefore, from the first convolutional layer and the maxed pooling layer, we output an image that is smaller than the original image, and the number of filters that we specify is the number of channels that will be fed forward into the network [3].

With these things in mind, this is why I chose to use the CNN model to classify the images of the clothing articles of the Fashion MNIST dataset. I will further go in-depth about the specific CNN with the parameter settings and model architecture that I used in my program in a later section.

B. *Preparing the Fashion MNIST dataset*

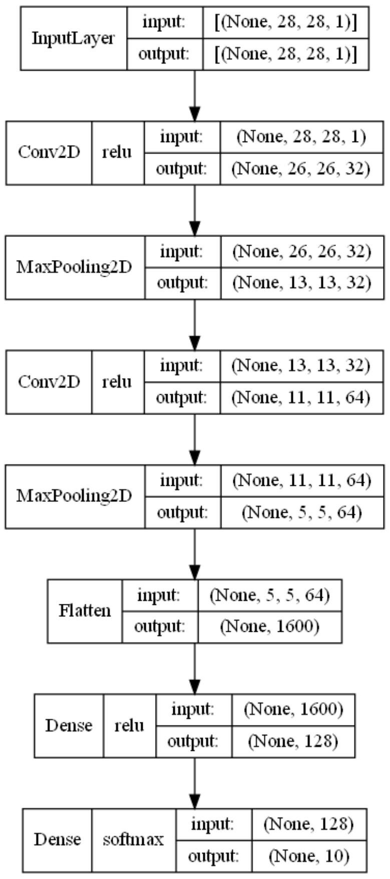
To prepare the Fashion MNIST dataset to be fed into the CNN, I reshaped the training set and test set. After loading the dataset into my program, I saw that the shape of the training dataset is (60000, 28, 28), and the shape of the test dataset is (10000, 28, 28). For this to be fed into the convolutional neural network, I reshape the datasets to (60000, 28, 28, 1) and (10000, 28, 28, 1) as the CNN takes tensors of shape (image_height, image_width, color_channel), where image_height corresponds to the height of the inputted image (28 pixels), image_width corresponds to the width of the inputted image (28 pixels), and color_channel refers to (R, G, B) (set color_channel equal to the value 1 as original image has only one channel) [4].

In addition to the reshaping of the dataset, I converted the training and test sets to take the data type of a 32-bit floating point number. We do this conversion because each of the pixels in the original image has a pixel-value to it to indicate the lightness or darkness of that pixel, and the pixel-value ranges from an integer value of 0 to 255 [8]. However, for our CNN, we want each of the pixel-values to be rescaled to the range of 0 to 1, so we divide each of the original pixel-values by 255 which is why the conversion to a 32-bit floating point number is necessary.

The final step in preparing the dataset to be fed into my CNN is to convert my class labels to one-hot encoding vectors. We need to convert the class labels numbered 0 to 9 to those one-hot encoding vectors because our network will not be able to work with categorical data values directly. Therefore, we convert the class label numbers to a row vector that consists of all zeros and a value of 1 for the class that the label corresponds to [2]. Now the preparations for the dataset are complete, and we can now feed the dataset into my CNN.

C. *Convolutional Neural Network Model Architecture*

In my CNN, I utilize two convolutional layers with two max pooling layers that feed into one fully-connected layer before the classification of the images in the output layer. Using the plot_model() function in my code, I was able to produce the figure below a outlining the model architecture of my CNN such that [6]:



For the first convolutional layer that takes in the inputted data of shape (28, 28, 1), and we feed the data into 32 filters with a kernel size of (3, 3). Then from this first convolution, we use a max pooling layer that uses a pool size of (2, 2), so the size of each feature map will decrease by a factor of 2 for both dimensions of the data. From the first convolutional and max pooling layer, we output data of shape (13, 13, 32).

For the second convolutional layer that takes in the data of shape (13, 13, 32), and we now feed the data into 64 filters with a kernel size of (3, 3). Then from this second convolution, we use another max pooling layer that uses a pool size of (2, 2), and once again, the size of each feature map will decrease by a factor of 2 for both dimensions of the data. From the second convolutional and max pooling layer, we output data of the shape (5, 5, 64).

Since the output of the second convolutional layer and max pooling layer is a three-dimensional tensor, we then want to flatten the output to a one-dimensional vector to feed into a dense layer now that the network is fully-connected. The dense layer consists of 128 nodes, and we then feed this into our output layer with 10 nodes for the classification of the images to the ten class labels.

Both convolutional layers and the dense layer when the network is fully-connected network uses the rectified linear units (Relu) activation function. Relu is given by

$$g(x) = \max\{0, x\}$$

$$= \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}, \quad (1)$$

where the output of the activation function will be the input x itself if the input is positive, and it will output 0 otherwise. Relu was proposed for deep neural networks, and it is an activation function used for hidden layers [2].

Then, the activation function used for the output layer is softmax. The softmax activation function is given by

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \in (0,1), \forall i, \quad (2)$$

where we have that

$$softmax(x) = \begin{pmatrix} S(x_1) \\ S(x_2) \\ \vdots \\ S(x_n) \end{pmatrix},$$

and $S(x_1) + S(x_2) + \cdots + S(x_n) = 1$. What this softmax activation function does is that the outputs of the neural network will become a vector of probabilities that corresponds to the probabilities of the inputted data belonging to each of the ten class labels [2].

D. Compilation and Training of the Model

Now that we have constructed our CNN model, we can now compile the model to prepare for training. The loss function that we use to compile the model is the categorical cross-entropy function. This categorical cross-entropy function takes the vector of probabilities from the output layer with the true class labels to calculate the cross-entropy loss. The loss function is defined by

$$L = \sum_{i=1}^{10} y_i \log(S_i), \quad (3)$$

where we have that y_i is the true class label for the ith class, and S_i is the probability of the ith class from the output of the softmax activation function. Thus, we want to minimize the loss function [2].

In addition to the loss function, we use an optimizer to find the model parameters. The optimizer that we use is the Adam optimizer with the default learning rate of 0.001. The Adam optimization is a stochastic gradient-descent method to obtain the model parameters [7].

Before training the model, I defined an early stopping callback for the model to keep track of the validation accuracy, so we can stop the training before all the iterations that I specified are complete if it appears that the validation accuracy is not improving. Along with this callback, I set the validation split to 0.25, the number of epochs (iterations that the training is performed on) to 100, the batch size equal to 128, and the verbose equal to 1.

III. Experimental Results

After training my model, I looked to evaluate the accuracy of the trained model on both my training set and test set. The classification accuracy of the model on my training set was always around 96% for each time that I ran my program. Though for the more important results, I found that the accuracy of my trained model on the test set was always falling around 90 – 91% each time I ran my program.

IV. Conclusion and Discussion

From looking at the results of my trained model on the training set and testing set, it appears that my trained model works well in classifying the clothing article images. I was able to achieve my task of constructing a program that would have an accuracy that exceeds 90% on the test set. Unfortunately, with the given time that I had in working on this program, I was not able to increase the accuracy of the model to greater than 91%.

Originally, the CNN model architecture for my program included three convolutional layers (with no maxed pooling layer used for the third convolutional layer). The two convolutional layers that are currently in the CNN model architecture remained the same with number of filters and kernel sizes for both. The additional convolutional layer that was included in the model architecture contained 128 filters with a kernel size of (3, 3). From this original architecture, I excluded using a third max pooling layer since the dimensions for the output would be very small, and we would lose a lot of the parameters.

However, I ended up using just the CNN model architecture with two convolutional layers and max pooling layers because the original model architecture would yield around the same 90 – 91% accuracy that the current model achieves, but it would take 10 additional seconds for each epoch than the current model in the training process. This computation time is probably accounted for as the CNN model with three convolutional layers had around 20000 more parameters than the current CNN model.

In addition, note that from the experimental results section, that the performance of the trained model on the training set was around 96% compared to the performance on the testing set being around 90 – 91%. Thus, we see that there is around a 5% better performance on the training set than the test set. It is possible that since there is this 5% difference in performance on the training and test set, then there may be a slight issue with overfitting. Therefore, we could potentially incorporate a dropout layer that was discussed in one of our final class lectures that could handle this type of issue.

Overall, I thought that having to construct and train a model using CNN to classify images of clothing articles was a valuable learning experience. Earlier in the semester, we looked at classifying handwritten digits from the original MNIST dataset, and it was insightful to have the experience of creating an artificial neural network. Then building from that experience was very useful in building a CNN since it was the first time having to use one. Also, with some curiosity, I tested the code that I used for the handwritten digits classification assignment on the Fashion MNIST dataset, and I was able to achieve around an 85% accuracy rate on the test set. Since I was able to achieve a 90 – 91% accuracy rate using CNN, it really emphasized that CNN was more powerful than the ANN model that we used earlier in the semester, and in the future, I hope to gain further experiences in being able to find and develop more powerful models to gain even better performances for given tasks.

V. References

[1] K. Team, “Keras Documentation: Fashion Mnist Dataset, an alternative to mnist,” Keras, https://keras.io/api/datasets/fashion_mnist/ (accessed May 12, 2023).

[2] M. Xian, “CS 477/577 Course Notes,” 2023

[3] H. Lee, “Deep Learning Tutorial,” Apr. 26, 2023

[4] “Convolutional neural network (CNN),” TensorFlow, https://www.tensorflow.org/tutorials/images/cnn (accessed May 12, 2023).

[5] “A mnist-like fashion product database,” GitHub, https://github.com/zalandoresearch/fashion-mnist (accessed May 12, 2023).

[6] K. Team, “Keras Documentation: Model plotting utilities,” Keras, https://keras.io/api/utils/model_plotting_utils/ (accessed May 12, 2023).

[7] K. Team, “Keras Documentation: Adam,” Keras, https://keras.io/api/optimizers/adam/ (accessed May 12, 2023).

[8] “Fashion mnist,” Kaggle, https://www.kaggle.com/datasets/zalando-research/fashionmnist?resource=download (accessed May 12, 2023).