Classifying Items in the Fashion-MNIST Dataset

Austin Kugler
CS474: Deep Learning
Department of Computer Science
University of Idaho
kugl5443@vandals.uidaho.edu

I. Introduction

The Fashion-MNIST dataset contains pictures of fashion items which include articles of clothing and personal accessories. It provides a training set of 60,000 images and test set of 10,000 images. Each image is 28 by 28 pixels and depicts a single fashion item on a solid color background. The features for a given image include its classification and darkness of each pixel from 1 to 255. The ten classes of fashion items are:

1) T-shirt
2) Trousers
3) Pullover
4) Dress
5) Coat
6) Sandal
7) Shirt
8) Sneaker
9) Bag
10) Ankle boot

We can get the dataset by importing it from the Python Keras library or by downloading it directly from Kaggle[1] under the MIT open-source license.

The Fashion-MNIST dataset is useful for benchmarking machine learning models that deal with images as it is a well-understood problem of reasonable difficulty. It allows us to compare model performance and gives an indication of how a model will perform on other image datasets.

In this paper, we aim to reach the highest possible accuracy in fashion image classification while avoiding overfitting.

A. Preparation

We must first load and prepare the dataset before building a model to interpret it. We can do so by importing and using the Python Keras library through the `keras.datasets.fashion_mnist.load_data()` function. This loads the dataset directly from Keras and makes our project more portable as anyone can run the code without having to first find and download the dataset. Figure 1 displays some example fashion images from the dataset. Once loaded, we must perform a few preprocessing steps before using the data in our model.

We ensure the images are split into separate sets for training and testing. We reshape the data to reflect the number and size of images. We also convert the images to grayscale. Finally, we convert the output datasets from categorical descriptions of the class to one-hot vectors with integer classes. After loading the dataset, we can build and train a convoluted neural network (CNN) that classifies fashion images.

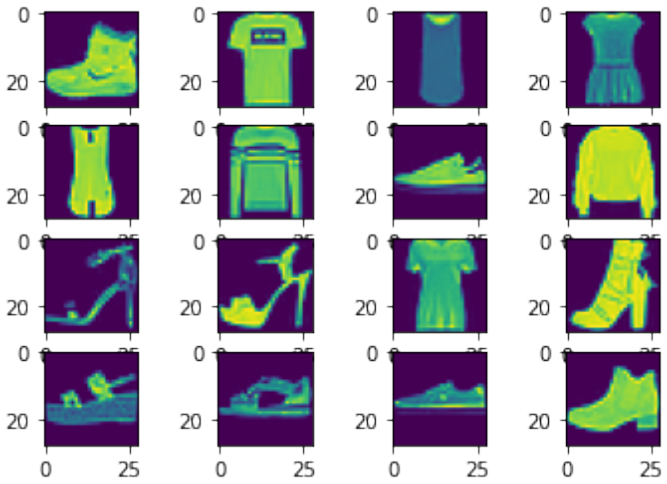

Fig. 1. Some images in the Fashion-MNIST dataset.

II. Proposed Method

Convoluted neural network (CNN) machine learning models have several advantages when solving image classification problems. For one, we can directly feed pixels from fashion images into the model with little preprocessing. Additionally, they reduce inputs with high dimensionality (images in our case) to much simpler feature maps that contain the most relevant patterns of the image. We chose to use a CNN because of these facts and the knowledge that CNNs are a known effective method for image classification.

Our CNN classifier includes the following components:

1) Convolution layer: uses the training dataset to determine which image features or patterns most contribute to its classification. This layer generates feature maps that represent these features or patterns.
2) Pooling layer: summarizes the feature maps by applying a function to parts of the input image. The result is a smaller and less detailed image that decreases the affect of irrelevant inconsistencies in the image.

[1] https://www.kaggle.com/datasets/zalando-research/fashionmnist

3) Activation function: determines the output of a node given many inputs and ensures the model is nonlinear. We use the rectified linear activation function (ReLU).
4) Optimizer: We use the stochastic gradient descent (SGD) optimizer to increase model accuracy by starting at some point in the set of loss function outputs and moving towards the point with the lowest slope.

We begin by initializing our model to use the Keras sequential model builder `keras.models.Sequential()` to create the model and add the convolution layers, pooling layers, activation function, and optimizer.

The first layer added to the sequential model is a `keras.layers.Conv2D()` layer. The purpose of this layer is to perform spatial convolution over images and produce filters that represent the features of our image most relevant to its fashion class. In this layer, we specify the shape of our input images, use the ReLU activation function, and Glorot normal initialization to set the starting weights. ReLU activation is a logical choice due to its standard use in CNN models. We use a Glorot initializer to set the beginning weights of the layer to a normal distribution (with zero as the center). This ensures the starting weights do not start with any bias.

We also specify parameters for the filters that are applied to the input image. These filters translate the input into a feature map that represents a pattern in the image. The feature map helps decrease noise as it puts more focus on the major features of an image. We set the number of filters to 64 in this first Conv2D layer. This number was chosen as during testing it appears to reach a good balance between training time and model accuracy. We also specify the filter size as 3, meaning the 28 by 28 pixel images will result in a filter that is 3 by 3 pixels. We chose 3 as it reduces the images by around 90% and allows faster training while retaining the important features of the image.

Next, we add the `keras.layers.MaxPooling2D()` layer to the sequential model. The purpose of a max pooling layer is to select the most significant patterns from an input feature map. It does this by calculating the max value in each part of a feature map and uses those values to create a new smaller feature map.

After this layer, we add a `keras.layers.Dropout()` layer. This layer aims to prevent overfitting by randomly setting input units to 0 at a specified rate. We chose 0.25 for our rate as it finds a good balance between too much dropout leading to underfitting and too little leading to overfitting.

We add another instance `keras.layers.Conv2D()` as our third layer. Images in the Fashion-MNIST dataset have fairly obvious patterns in their appearance. Adding another Conv2D layer further emphasises these patterns and improves our model accuracy. We use the same parameters as in our previous Conv2D layer.

Our fourth addition to the model is the `keras.layers.Flatten()` layer. Flatten layers translate multi-dimensional feature map vectors into one-dimensional feature vectors. This is a necessary step as the dense and output layers require flattened input.

After flattening, we add two dense layers using instances of `keras.layers.Dense()`. Dense layers are an integral part of any deep learning model. Each neuron in these dense layers has a connection with every neuron in the previous layer making them deeply connected. We set a unit count for each dense layer that determines the dimensions of the output space. We chose arbitrary values for the unit parameters and adjusted them based on test performance. The resulting dense layers we chose have output dimensionality of 64 and 128 units, respectively. We again use the ReLU activation function.

The final layer we add is the output layer. This is a dense layer with the output dimensionality equal to the number of fashion item class labels. We use the softmax activation function to get probabilities for the class labels.

During model compilation, we specify our optimizer as stochastic gradient descent (SGD). First, we specify the learning rate parameter as the default value of 0.01. This value controls how much the model adjusts based on its error. The momentum is set to 0.9 and determines the significance of model changes in response to changes in previously generated gradients. We access this optimizer from Keras using `keras.optimizers.SGD()`.

The overall makeup of the fashion item classifier is shown in the Figure 2 visualization.

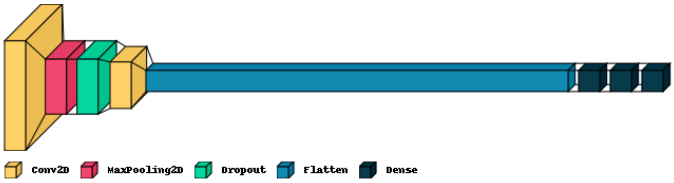

Fig. 2. Sequential layers of the fashion image classifier.

III. Experimental Results & Discussion

We take several steps to validate the model's experimental results. First, we prevent overfitting by separating the dataset into training and test sets to ensure we do not test on the same data the model using for training. We use k-fold cross-validation to ensure the measured accuracy has little variation across multiple trainings for different subsets of training and test sets. This process appears as follows.

1) Randomize the entire dataset.
2) Divide the dataset into K unique groups.
3) For each k group in 1 to K:
   a) Split the group into two subsets: the training set and test set.
   b) Fit the model to the k training set.
   c) Evaluate the model on the k test set and save the accuracy.
4) Calculate the mean train and test accuracy using each of the k recorded accuracies.

During testing we use a K value of 7. The train and test accuracy for each k-fold and overall mean is shown in table I. The mean training accuracy is 99.08% with standard deviation

TABLE I
MODEL EVALUATION USING K-FOLD CROSS-VALIDATION

k	*Train Accuracy*	*Test Accuracy*
1	97.00 %	91.51%
2	98.72 %	94.32%
3	98.57 %	96.05%
4	99.71 %	98.53%
5	99.80 %	99.07%
6	99.91 %	99.66%
7	99.83 %	99.65%
mean	99.08% +/- 0.99%	96.97% +/- 2.9%

0.99% across all *k*-folds. Mean test accuracy is 96.97% with standard deviation 2.9%. The high test accuracy of the model, validated by k-fold cross-validation, indicates it successfully found which features of a fashion image determine its class label. The high test accuracy proves underfitting is not present. Overfitting may be a concern if the test accuracy was in the 98% to 100% range. We prevented overfitting in this model by splitting the dataset and using a dropout layer. Furthermore, the accuracy is reasonably similar to what other models have reached. Most models made available on Kaggle[2] were in the 92% to 97% range for test accuracy.

IV. CONCLUSION

In conclusion, we aimed to build a model that classifies fashion images from the Fashion-MNIST dataset. We built our CNN model using the Keras sequential model and added Conv2D, MaxPooling2D, Dropout, and Dense layers. We then validated the model using k-fold cross-validation and measuring the train and test accuracy. Our overall training accuracy was 99.08% +/- 0.99% and test accuracy was 96.97% +/- 2.9%. We discuss why this accuracy is reasonable and likely does not reflect over or underfitting.

Future work may improve on the model by adding further dropout layers to further decrease potential overfitting. Test accuracy could benefit from additional max pooling layers to increase the impact of obvious patterns (such as the fashion item's outline shape). Finally, experimenting with optimizers other than SGD could lead to better accuracy as only SGD was tested during this project.

[2]https://www.kaggle.com/datasets/zalando-research/fashionmnist/code