Classifying Fashion Images Using a Convolutional Neural Network

Austin Kugler
Spring 2022
CS477: Python for Machine Learning
Department of Computer Science
University of Idaho
Moscow, Idaho
kugl5443@vandals.uidaho.edu
0000-0002-6680-4461

I. Introduction

A. Motivation

Correctly classifying "fashion" images (clothing, shoes, etc.) is useful in inventory management, image search algorithms, and online stores, to name a few. In computer vision, classification of such images also serves as a benchmark for a given machine learning model.

Fashion images provide a better benchmark than traditional computer vision datasets, like the Modified National Institute of Standards and Technology (MNIST) handwritten digit database. Fashion images are better for benchmarking because their classification is much harder than classifying digits. In the original MNIST dataset, two models may both reach near-perfect classification accuracy. This can make them seem equally accurate when in fact their true difference in accuracy does not appear on the simpler digit-classification problem. Using fashion images increases the classification difficulty and provides a better idea of the differences in accuracy between models.

B. Task Description

This paper presents a solution to the problem of fashion image classification. In this problem, we receive many pictures of clothing, footwear, over-wear, and accessories. The goal is to correctly label each of the pictures with the correct item type. Each picture should have a single label. Therefore, we should only use the most likely label as the classification for each image. Fashion-MNIST is a public dataset containing fashion images and their labels.

Effectively solving this classification problem requires a machine learning approach. We use the Keras Python library to implement such an approach. There are three steps common to most machine learning models:

1) Data preparation: load the dataset into the program and ensure its division into a distinct dataset for training the model and another for testing its accuracy. For the Fashion-MNIST dataset, we must also reshape the images and convert the categorical labels to numeric values.
2) Training: fit the model to the training dataset without exposing it to the test dataset. This step requires the use of Keras to create a model for image classification.
3) Prediction: provide the trained model with the test dataset and record the predicted label for each element.
4) Evaluation: compare the predicted and actual labels for the test dataset to calculate the model's test accuracy. We can improve this step using cross-validation, a technique where evaluation occurs on many subsets of the data. This method prevents overfitting the model to a single training and test dataset.

C. The Dataset

The Fashion-MINST dataset is available on Kaggle[1], a popular repository for machine learning focused datasets. Here, it is in comma-separated value format.

However, the dataset is also accessible through a programmatic import from the Keras library. This makes it easier to access for development purposes as developers do not need to download it independent of their code.

Fashion-MINST includes a collection of pictures from the online European shoe and fashion store called Zalando[2]. Zalando SE is also the provider of these images. The fashion images are licensed under the open-source MIT License.

clothing, footwear, over-wear, and accessories.

The dataset includes 60,000 grayscale images for training and 10,000 for testing. Each image is 28 by 28 pixels and belongs to one of ten classes. The classes cover various fashion products including four classes for clothing, three classes of footwear, two over-wear classes, and one accessory class. The ten class labels are as follows:

- 0 T-shirt
- 1 Trousers
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal

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

- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

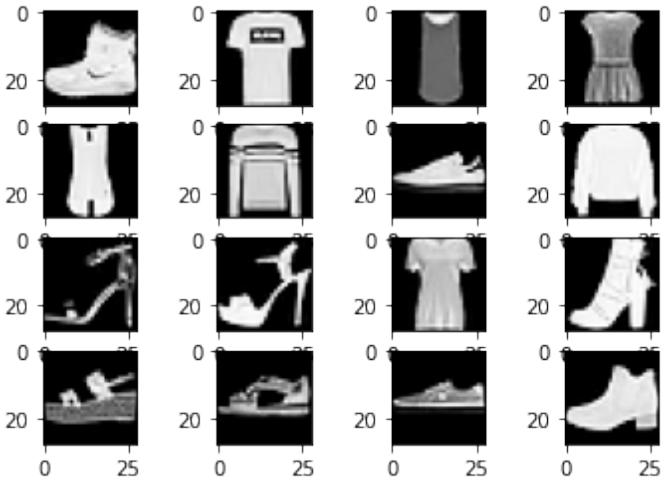

Fig. 1. Selected images from the Fashion-MINST dataset.

Each row of the dataset contains a single image. The first feature is the image's class label and the next 784 contain the darkness of each pixel. Pixel darkness exists in the range 1 (dark-colored) to 255 (light-colored).

II. Method: Convoluted Neural Network

Among deep learning methods, convoluted neural network (CNNs) have grown in popularity for applications in computer vision [4]. One advantage of CNNs is their ability to perform pattern recognition on pixels directly without significant preprocessing. This decreases development time and widens the range of problems the model applies to. For these reasons, we use a CNN for the classification of fashion images.

CNNs have three major components: the convolution layer, pooling layer, and activation function [4]. The convolution layer attempts to discover which feature representations are necessary for classification. The convolution layer differs from a fully connected layer as it has significantly fewer connections for the same number of inputs and outputs [5]. This allows it to better scale for images which generally have many times more inputs and outputs than other data.

The pooling layer reduces the feature representation and separates its environment. Essentially, this layer summarizes the feature maps. This is done by applying a function to portions of an input image to produce a smaller output [5]. In max pooling, the function chooses the maximum value of a larger area as the output. Pooling decreases the impact of unwanted inconsistencies in pictures such as orientation or lighting.

The activation function determines what the output of a node is, given an input or many inputs. They add a nonlinear aspect to the model which is vital for complex classification problems. The rectified linear activation function (ReLU) is a popular option for convoluted neural networks.

In addition to these components, many other machine learning techniques apply to CNNs. It is still valuable for CNNs to include loss functions, regularization, and optimization. One of the more popular optimizers for image classification is stochastic gradient descent (SGD). The SGD optimizer aims to minimize error by starting at a random point in the model's loss function and iteratively moving towards the point of lowest slope [7].

A. Model Architecture

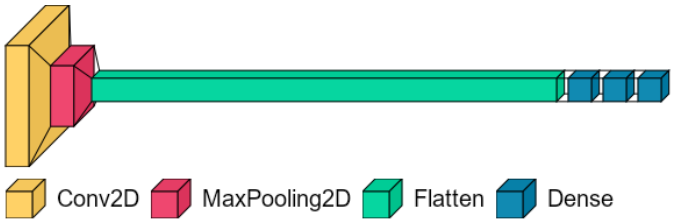

Fig. 2. Layers of the classifier.

We use several Keras components to define an architecture for our classification model. We begin with a base sequential model. We then add a 2D convolution layer, max pooling layer, flatten layer, dense layers, and output layer (see Fig. 2). Together, these components predict the class label for a given fashion image. The addition of a SGD optimizer further improves the model.

1) Sequential Model: Kera's provides an application programming interface (API) for building machine learning models sequentially, layer-by-layer [6]. Creating a sequential model is an important part of the classifier's architecture. In Python, we access this part of the Keras library using the `keras.models.Sequential()` syntax. We initialize the `model` variable as a sequential model, allowing the addition of layers later.

2) Conv2D Layer: Another key part of the classification model is the `keras.layers.Conv2D()` layer. This layer represents two-dimensional inputs—images in the case of the Fashion-MNIST dataset. The 2D convolution layer allows specification of several key model parameters.

One parameter is the number of filters that the convolution layer should learn. Each filter translates the entire image input into an output for a specific pattern in the image, called a feature map. Together, the feature maps influence the final predicted label. Filtering also decreases noise by creating a feature map smaller than the original image. For the fashion classifier, we specify the number of filters as 32. This value reflects the relative simple nature of the images and makes model training less resource intensive.

The second parameter in the `Conv2D` layer determines the pixel size of each filter. In the fashion classifier, we set this size to 3 by 3 pixels (9 total). This is a reasonable filter size compared to the input size of 28 by 28 pixels (784 total).

In this layer, we also specify the activation function applied to the layer's output. We choose the ReLU activation function due to its outperformance for CNN models [1].

Finally, we specify a kernel initializer to determine the beginning weights of the layer. This initializer sets initial weights to a normal distribution centered on 0. We use this initializer due to its ability to outperform logistic sigmoid activation in deep learning [8].

3) MaxPooling2D Layer: We include a max pooling layer in the architecture. This layer selects the maximum element from feature map subsets. Therefore, it acquires the most notable patterns in the inputted feature map. We use the default 2 by 2 pooling window, reducing the feature map by half. In Keras, the max pooling layer is accessible with the syntax `keras.layers.MaxPooling2D()`.

4) Flatten Layer: A flatten layer exists immediately following the max pooling layer. The flatten layer converts the multi-dimensional vectors from the pooling feature maps into one single-dimensional feature vector. This layer ensures the final dense and output layers can accept the output of the convolutional part of the model. Keras provides access to the flatten layer at `keras.layers.Flatten()`.

5) Dense Layers: Two dense layers occur directly after the flatten layer. Dense layers are a core part of the architecture for a deep learning model or CNN. Every neuron in a dense layer connects to the every neuron in the previous layer, making them deeply connected.

The unit count of a dense layer specifies the dimension of its output vector. We specify a unit count of 128 for the first dense layer and 64 for the second. These values were arbitrarily chosen initially and then adjusted based on the resulting accuracy. We also took into account the required training time when experimenting with unit counts. The result of this process was the final values of 128 and 64.

6) Output Layer: The final layer in the classification model is the output layer. This dense layer outputs the final predicted label of the inputted fashion image. It has a unit count of 10, matching the number of possible class labels. This layer uses the softmax activation function to determine the probability distribution for the ten labels [2]. Many models use softmax in the output layer because of its ability to provide probabilities for many class labels. We use it for this same reason.

7) SGD Optimizer: The final component in the model architecture is the stochastic gradient descent (SGD) optimizer. We include the SGD optimizer to help minimize error in the model. It is accessible in Keras through the syntax `keras.optimizers.SGD()`.

In the fashion classifier, we provide the SGD optimizer with two parameters: learning rate and momentum. The former instructs the degree to which the model changes in response to its error. The latter allows the model to change based on previous gradients generated by the optimizer [7]. For these parameters, we use the default learning rate of 0.01 and specify 0.9 for momentum.

III. Experimental Results

We achieve accurate evaluation of the Fashion-MNIST classifier through a few methods. We separate the data into training and test sets. The model is then fit to the training set and predicts classifications for the values in the test set. Comparing the predictions with the actual values allows calculation of an accuracy value—the proportion of correct predictions as a percentage. The resulting value is usable in comparisons with models of other researchers.

We also add the method of k-fold cross validation to our evaluation plan. Cross-validation intends to eliminate bias when determining the accuracy of a model. It does so through resampling, where the training and test datasets are derived from k subsets (folds) of the original dataset [3]. This allows many evaluations of the model, decreasing bias that may occur in a single evaluation. The basic process is:

1) Randomize the entire dataset.
2) Divide the dataset into k unique groups.
3) For each k group:
    a) Split the group into a training and test dataset, respectively.
    b) Fit a model using the training set.
    c) Evaluate the model on the test set, saving the accuracy.
4) Evaluate the overall model using the saved k accuracies.

We set the value of k to 5. We then evaluate the overall model by calculating the mean accuracy and standard deviation of the folds. The accuracy of each fold and the overall evaluation is as follows:

TABLE I
Evaluation of Fashion-MNIST CNN Classifier

K-fold Cross-Validation		
k	***Train Accuracy***	***Test Accuracy***
1	96.46%	91.21%
2	95.25%	90.92%
3	94.95%	90.47%
4	94.85%	90.61%
5	95.67%	90.28%
mean	95.44%	90.70%
std dev	+/- 0.59%	+/- 0.33%

Overall, the CNN classifier reached mean training accuracy of 95.44%. This result indicates the training was successful in capturing the patterns present in the fashion images. A value closer to 100% may indicate overfitting has occurred and a significantly lower value may indicate underfitting. However, it seems neither of these undesired outcomes occurred.

The mean test accuracy was 90.70%, comparable to the results of other researchers. On the Kaggle repository, with the most voted model implementations reaching test accuracy in the 90-92% range[3]. The accuracy is reasonable but may be improvable by the adjustments described in section IV-A.

[3]https://www.kaggle.com/datasets/zalando-research/fashionmnist/code?datasetId=2243&sortBy=voteCount

IV. Conclusion

In this paper, we describe the problem of classifying fashion images from one of ten possible categories. We describe an architecture that uses a convolutional neural network (CNN) to solve this problem. As part of the architecture design process, we examine the deep learning concepts of sequential models, convolution layers, max pooling, dense layers, and optimizers. We implement the proposed architecture using the Keras Python library. We then train it on the Fashion-MNIST dataset and evaluate its performance using k-fold cross-validation. The resulting mean test accuracy was 90.70%, proving the efficacy of the model.

A. Future Work

The model proposed in this paper has room for improvement. The addition of more convolution filters may increase the model's accuracy. Future work may include evaluating the model with an increasing number of filters to determine the most effective. A padding layer may also improve the model by allowing more widespread application of other convolution layers. Currently, no input padding occurs; this reduces the total applicable area of a convolution layer.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, 2012.

[2] J. Brownlee, "Understand your data, create accurate models and work projects end-to-end," *Machine Learning Mastery*, 2018.

[3] J. Brownlee, "Statistical methods for machine learning: discover how to transform data into knowledge with Python," *Machine Learning Mastery*, 2018.

[4] J. Gu, Z. Wang, J. Kuen, et. al., "Recent advances in convolutional neural networks," *Elsevier Pattern Recognition*, vol. 77, pp. 354-377, May 2018.

[5] N. Ketkar, "Convolutional neural networks," In: Deep Learning with Python: A Hands-on Introduction, *Apress*, pp. 61-77, 2017.

[6] N. Ketkar, "Introduction to keras," In: Deep Learning with Python: A Hands-on Introduction, *Apress*, pp. 97-111, 2017.

[7] N. Ketkar, "Stochastic gradient descent," In: Deep Learning with Python: A Hands-on Introduction, *Apress*, pp. 113-131, 2017.

[8] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks," In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, *Proceedings of Machine Learning Research*, vol. 9, pp. 249-256, May 2010.