Predict Real Images Classes by Using Fashion MINIST

Keval Patel
Kp14@myscc.ca

Introduction

With vast amounts of multimedia and internet data growth, we are facing the huge quantities of clothing image data, if we manually annotate the semantic attributes of clothing images to classify them. Then we need to spend a lot of effort and time, and the semantic attributes cannot fully express rich information of the clothing image, so the classification effect may not reach what we expected. To meet our expectation, we need a deep machine learning model which can help us highly specific features that can be detected anywhere on input images [1], this is what exactly a convolutional neural network good at.

The purpose of this project is to first explore how to construct a convolutional neural network to classify images with Fashion-MNIST dataset and get a test accuracy at least 90%, then try how to apply improvement to the CNN model so that we can get a 90%- 95% test accuracy and make sure there is no overfitting and underfitting problem in our model, and next explore how to save the model and use it to make predictions on new unseen data which we download from internet and import and resize them for making model predictions.

Fashion MNIST dataset was published by Zalando research [2]. And it contains 60,000 training and 10,000 testing images with each 28*28 in gray scale. The dataset contains images with 10 different fashion products categories which represent as ten labels (0-9): T-shirt/top (0), Trouser (1), Pullover (2), Dress (3), Coat (4), Sandal (5), Shirt (6), Sneaker (7), Bag (8), Ankle boot (9).

From our understanding, the reason why CNN can work efficiently in image classification is that it first scans an image chunk by chunk from left to right and top to bottom to extract and learn features by using Convolution layer with filter (Kernel) to produce a feature map, and a CNN model can have several convolutional layers. Then it uses Pooling layer to reduce the dimensionality size of each feature map to keep the important information. [3] Next, it flattens the output with Flatten layer and feed them into Dense layer. Last, it can calculate the output based on the probabilities by using SoftMax activation and classify the images into their corresponding classes according to maximum probabilities.

Related Work

Prior discoveries have demonstrated the inadequacy of point estimate Convolutional Neural network (CNN) to provide predictions which exceed our expectation. In ECE Department of REVA University, they made tremendous progress in area of apparel classification. They took 2D structure of Convolutional Neural network for detecting fashion cloths or items that person wears or hold in image [4]. Zalando Research, where they present Fashion-MINIST dataset with 70,000 fashion items with 10 different classes and compared it with its original version of MINIST dataset by using different machine learning algorithms [5]. In Research school of Computer Science of Australian National University, for batch normalization and skip connections they use Long short-term memory networks (LSTMs) because they think there were too many related works available on CNNs so, by using LSTMs they improve their accuracy up to 92.54% and minimize the training time [6]. eBay research, where they use Fashion MINIST dataset to detect fashion objects since the position of fashion item is closely connected to human and they use Region-based Convolutional Neural Network(R-CNN) to select most informative parts of each fashion items and learn automatically from data [7].

When we compare these related works with ours, we can say that [4,7] are highly related to our work. In [4,7] they are detecting fashion items which could be next thing to do after predicting correct image classes.

Methods

In order to achieve our desired goal, I tried many methods, which can be mainly summarized as the following steps:

Data preprocessing [8]**:** Firstly, I load the Fashion-MNIST dataset and import necessary packages which include NumPy, pandas, matplotlib.pyplot, and Keras, and load both training set and testing set. Visualize the data. View the training image data by using imshow( ) function under the matplotlib library. Secondly, I select features (X_train) and target (y_train) for training and testing data, normalize the data, and change the data type. As the grayscale values for the images lie between 0-255 which is a large range, so I need to normalize data from both training and testing set to a range from 0 and 1. To do this, I divide each pixel value with 255, and we transform them into a float 32 array of shapes. There is another reason for doing this because of each neuron store value between 0 and 1. Thirdly, I reshape training and testing image data, so that I can get a proper input shape (height*width*channels) for training the CNN model, and evaluating it. Proper training data shape is 60000*28*28*1 and proper testing data shape is 10000*28*28*1, the number 1 here means channels=1 because we are using grayscale images and 60000 and 10000 are train set and test set images accordingly.

The first simple neural network model building [8]**:** I create a sequential model which is a linear stack of layers. Add in a Flatten layer to flatten the 3 dimensions input (28*28*1) into 1 dimension. Then I add in a fully connected layer (Dense layer) to connect the neurons to all activations in the previous layer and a final Dense layer with 10 outputs and a softmax activation which can help us to calculate the output based on the probabilities and make a classification.

CNN model building [9]**:** On the first CNN model building, I add in the first convolutional layer with number of filters= 32, filter (kernel) size = 3*3, activation function=Relu, input shape(height*width*channels) =28*28*1. Besides, I add in the first Max pooling layer with the pooling window with size 2*2 to reduces the dimensionality of each feature and number of parameters and shorten the training time. Then, I compile the model by specifying which loss and optimization functions I want to use in the training of our model. And, I choose sparse_categorical_ceossentropy as our loss function because our problem is a classification problem, and our targets can only belong to a single class. The reason why I choose Adam Optimiser is that its memory requirements are very low. After that, I fit the model with training data and set epochs=10 which means 10 forward pass and backward pass of all the training examples.

First model optimization by increasing filters: I increase the number of filters from 32 to 64 in the first convolutional layer, because increasing filters can give more opportunity for extracting simple features from the input images, especially when I use a very small feature map (3*3).

Second model optimization by adding more layers: I add one more convolutional layer with number of filters= 128, filter (kernel) size = 3*3, activation function=Relu. Then I add dropout layers into our model to avoid overfitting, as it will forces the model to learn multiple representations of the same data by randomly disabling neurons in the learning phase. Furthermore, I set the dropout rate as 0.25, 0.40, 0.30 (Normal dropout rate range: 0.2-0.8), so the layers will respectively and randomly disable 25%, 40%, 30% of its nodes at a given time. Generally, 0.2 dropout is for a smaller dropout rate and 0.8 dropouts are for a larger dropout rate. Moreover, I add more epochs when fitting the training data, and set batch size=100, this means the algorithm takes the first 100 samples (from 1st to 100th) from the training dataset first and trains the network. Next, it takes the second 100 samples (from 101st to 200th) and trains the network again, this process continues until all the samples have been taken for training [10]. And I add more epochs when fitting the model. As the number of epochs increases, a greater number of times the weight is changed in the model.

Third model optimization by tuning parameters: After a lot of trial and error, I change the numbers of filters in the first convolutional layer filters=32 and second convolutional layer filter=64. Besides I add padding = "same" into both convolutional layers so that the zero values are added around the input, and "same" padding can help to make the output has the same size as the input. And then I add a MaxPooling layer with a pooling window with size 2*2 after the second convolutional layer as convolution Layers are often followed by a MaxPooling Layer. Also, I remove the dropout layer after the flatten layer. Also, I change the filter size of the first Dense layer to 128 which means mapping input nodes to 128 and I increase epochs to 100 epochs as well. I tried an abundance amount of combination where I change the number of neurons to the number of layers.

Final model saved for future prediction [18]**:** I define functions for the whole process of the final model in proper order because this is a way to keep things simple and to easily make a prediction on unseen new images. And functions include the following: Load dataset, Preprocessing data, Define model, Fit model with 100 epochs, and 100 batch size, and finally I save the model using save ( ) function. Then I run the function and get the 'final_model.h5' file in our current working directory. And the model can be run anytime by using load_model() function.

Visualization [11]**:** To start with, I import the necessary library and packages which are os, NumPy, pandas, matplotlib.pyplot, TensorFlow, projector from tensor board plugins, and input_data from TensorFlow examples. Then I import our test data which I downloaded from Kaggle and select features as X_test and target as y_test besides I set our log directory where I can store our log data. After that, I use Filewriter from TensorFlow.Summary and give the path of log directory and then I use our test features to save as TensorFlow variable and give a name. Furthermore, I config our projector and embedding which I am using to visualize our high-dimensional data. After that I give our embedding name which is the same as our TensorFlow variable name, then I set the path for our metadata and sprite image path which I going to use in visualization and set the dimension of a single image. Besides, I use Filewriter and projector to visualize embeddings. In the next step, I use the TensorFlow session where I set global_variables_initializer and create saver which I use to save our model with global_variable_initializer in the log directory. Then I create our sprite image and metadata and save in the same log directory so, now I have all the data and I use tensorboard to visualize our data. For the visualization of data, I have to open Anaconda prompt and select a particular instance where I have TensorFlow it has tensorboard in it. I have to activate instance by 'activate TensorFlow' where TensorFlow is instance and then write "tensorboard --logdir='path' "which activate tensorboard after that by using a browser, I can see tensorboard on localhost:6006.

Unseen image import and resize: First of all, I Import glob which is a general term that can be used to define techniques to match specified patterns according to rules related to Unix shell.[12] Then, I use a glob.glob() function to returns a list that contains matches image and labels image names list. After that, I create four for loops, one of them with cv2.imread function to read unseen images that I have downloaded from the internet and convert them into grayscale, and the second one with cv2.resize function [13] to resize unseen images into 28*28, the third one with cv2.reshape function to reshape unseen images into 28*28*1 (grayscale image has only 1 channel), this is a proper input shape of our model. The last for loop is used to transform those unseen images into a float 32 array of shape and divide each pixel value with 255.

Prediction: For Predicting unseen images I downloaded 30 fashion items photos that are randomly downloaded. I use our saved model fashion_model.h5 to predict new image classes. Each image has a different size so, I use the method which I describe before this part and import, resize, and reshape multiple images. Besides I use predict_claases( ) to predict the class of each image.

Results

Model accuracy results [14]: At the very beginning, I got a training accuracy of 91% and 88% test accuracy for the first basic neural network. After building the first CNN model by adding 1 convolutional layer and 1 max-pooling layer on the basic neural network, I got 91% test accuracy. Then I increased our test accuracy from 91% to 92% by doing the first model optimization. As the training accuracy was 98% for the first CNN model, and our model seems like a little bit overfitting. So, for second model optimization, I add dropout layers into our model to avoid overfitting and add one more convolutional layer with proper parameters for each layer on our model, I got 95% training accuracy and approximately 93% test accuracy. Furthermore, I set the value of all the dropout layers to be 0.3 and increase the epochs to be 100 in third model optimization, and then I got 93.40% test accuracy for our final model.

Visualization results: By visualizing data with embedding on tensorboard, I can see data in 3D with help of PCA and T-SNE. By PCA (Principal Components Analysis) I can see a group of fashion items and by selecting I can see the nearest neighbor of that particular image. After that, with T-SNE I can see clusters of fashion items with help of perfect perplexity, learning rate, and iterations. Furthermore, by clusters, I can define that which fashion items are nearest to each other. I can also understand how the model can recognize images but sometimes it can misunderstand image class which is nearby that image.

Prediction results: Our model predicts few images correct and others cannot identify correctly. By this, I can say that model is not going to work on every image I use as input. There are some facts about the model such as model can only identify those kinds of images which it learns in training set so, if an image is shifted or flipped then it’s not going to predict the correct class of that image and also when I reduce the size of the particular image it also reduces some features so, the image doesn't look same as it before.

Discussion

When I start my project, I think that predicting real images with computer-generated data is a legendary idea. First, I need to find something by which I can learn these methods and I find one on Coursera "Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning" with Laurence Moroney [15]. I learn Data Processing, how to build a neural network, the building of a convolutional neural network how I can improve the performance and accuracy of the model by adding additional convolutional layers or altering parameters [15]. After First model optimization, I want something healthier which can defeat the last result which is around 92% on the test set and 97% on training set so, for that tried to find something on the internet which can help us and I got one article by Matthew Stewart on towards science "Simple Guide to Hyperparameter Tuning in Neural Networks" and I find out about dropout layers by which I optimize our model and reduce overfitting [10]. After that our instructor suggest we run the model for more epoch and check what happens. When tried to run the model for more than 50 epochs but it took so much time for that for 1 epoch it took at least 10 minutes so, I need to find a different way for that and I heard that convolutional model can run faster on GPU instead of CPU so I tried to find how I can run our TensorFlow on GPU and I find information on TensorFlow official site which describe that I need Cuda software which is well-matched with particular TensorFlow-GPU [16]. I strained a lot but still, I wasn't able to run our model on GPU and then I find one answer on stack overflow which solved our problem [17]. Next thing is to save the model so, I don't need to run whenever I open the file or try to do some experiments with the model. For that, I find a way to save the model [18]. Then I want to visualize our data for that I find one video but still, I need to change some codes to get appropriate results [11].

Conclusions

In conclusion, after training and testing the dataset through different methods, my final optimized model' s test accuracy is 93.40%, which achieved our goal, and there is no overfitting and underfitting in my model. However, my final model is not accurate as I expected in predicting the classes of new images that are out of the test dataset. In the future, we can improve my model by taking some steps Firstly, we can continue to improve my model to get a more accurate model. On the other hand, we can improve the way we change the size and shape of the new images, decrease the bad effect on the predicted results during image resizing and reshaping so that we can get better predictions. We also use transfer learning to train new images on the same model so, the model can recognize more image classes.

References

[1] How to Develop a Deep CNN for Fashion-MNIST Clothing Classification: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/

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

[3] Understanding of Convolutional Neural Network (CNN): https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[4] Sri Devi. J.E, R. Priyanka, Soumya.R.K, Sowmyashree . H.R Assistant Prof. Sarveda. Apparel Classification Using CNN S ECE Department, REVA University

[5] Han Xiao, Kashif Rasul, and Roland Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," 2017

[6] Shuning Shen, Image Classification of Fashion-MNIST Dataset Using Long Short-Term Memory Networks Shuning Shen Research School of Computer Science, Australian National University, Canberra

[7] Hara, Kota, Vignesh Jagadeesh, and Robinson Piramuthu. ‖Fashion Apparel Detection: The Role of DeepConvolutional Neural Network and Pose-dependentPriors.‖ arXiv preprint arXiv:1411.5319 (2014)

[8] introduction to TensorFlow Coursera week 2: https://bit.ly/34QuyKV

[9] Introduction to TensorFlow Coursera week3: https://bit.ly/38gOCZ4

[10] Simple Guide to Hyperparameter Tuning in Neural Networks: https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

[11] Tensor board tutorial (MINIST): https://bit.ly/2rULxNt

[12] Python glob() function: https://www.poftut.com/python-glob-function-to-match-path-directory-file-names-with-examples/

[13] Read multiple images on folder (OpenCV) stack overflow: https://bit.ly/2Ygs2Li

[14] GitHub link

[15] Introduction to TensorFlow Coursera: https://bit.ly/33VtEvo

[16] TensorFlow on GPU: https://bit.ly/35ZXIr8

[17] How to run TensorFlow on GPU – Stack overflow: https://bit.ly/2RmbOPv

[18] How to save and load your keras deep learning model: https://bit.ly/2rZTm4u