Training Fashion - MNIST dataset with ResNET - 50

Jainy Patel

Computer Science
University of Maryland
College Park, MD 20740
jpatel98@terpmail.umd.edu

Shiv Patel

Computer Science
University of Maryland
College Park, MD 20740
sp99@terpmail.umd.edu

Abstract

In this project, we trained a model on the pre-standardized Fashion-MNIST dataset. We aimed to create a classification model that could categorize the outfits into different pieces of clothing. We tested different model parameters in hopes to create the most efficient but still accurate model, to ensure we didn't reach diminishing returns, in reference to time and resources spent in training.

1. Introduction

Fashion-MNIST is a dataset with about 60,000 training images as well as 10,000 testing images. For this project, we chose the Fashion-MNIST dataset over the MNIST dataset due to increased complexity of each dataset. The purpose of replacing the original MNIST dataset is because it's commonly used to train models detecting handwritten digits. In our case, we are using images of different parts of clothing and classifying them as 10 different labels.

We sought to create a lightweight, but still accurate model for a classification problem. We used a fashion related dataset, because our final product was related to fashion. We wanted our model to be able to successfully classify each piece of clothing into its respective clothing "type" category. We also needed to find the point of diminishing returns, to ensure we were able to reach the highest point of accuracy, while maintaining proper training times, and resource usage.

2. Data

The data we use is imported from the TensorFlow machine learning library. It contains four parts of the data we have used for the project including training images and labels as well as testing images and labels, 60,000 and 10,000 images respectively. The images are 28x28 with pixel values from 0 to 255. The labels range from “shirts” to “shoes” to “accessories” and contain a total of 10 different classes. By default, all training and testing images are assigned to one of the classifying labels.

We wanted to spend as much time as possible on creating a reliable model that worked efficiently, but also didn’t have gaps in accuracy. To ensure we still had a properly sized training set, we used one that was already public. We settled on the MNIST-Fashion library because it had similar images to what our final use case would be, but also because it provided sixty-thousand labeled images, all standardized to a twenty-eight by twenty-eight pixel grayscale image. We saved lots of time on this front by using the MNIST dataset, and aimed to use this time to fledge out our training model, and test it regarding training, and testing runtimes, and accuracy.

3. Related Work

Initial research work has been done by Kashif Rasul and Han Xiao from Zalando Research group. Their project was to create a new dataset that would be a drop-in replacement for the original MNIST dataset. MNIST dataset is generally used to train models to identify handwritten letters from an image. Their purpose of creating a new separate dataset just for clothing is to be able to make it easier for developers to create their models and be able to use this dataset on a wide range of machine learning libraries.

A year after Fashion-MNIST was created, many publications were issued from top AI different institutes including nine unique publications from google followed by seven from University of Cambridge. As stated by one of the initial dataset providers Han Xiao,

> “Fashion-MNIST has collected 4000 stars; it is referred in more than 400 repositories, 1000 commits and 7000 code snippets. On Google Scholar, more than 250 academic research papers conduct their experiments on Fashion-MNIST. It is even mentioned in the Science Magazine from AAAS” (Xiao 2018).

A list of all research done after the publication up to September 2018 can be found on a Google Sheets made by Xiao, compiling everyone’s research, models, and accuracy results. This research has not only covered American institutes but it has been used globally as well. Top number of publications were made by The United States of America with ninety-four unique publications, followed by China with forty-four unique publications. Outside of politics, these two countries have “co-authored 10 publications with Fashion-MNIST”.

Since Fashion-MNIST has been released, many different research groups have used it for their purposes. As per one article published in the magazine “Science” by AAAS, a machine learning research group from UCLA mentioned that they “created 3D-printed D^2NNs that implement classification of images of handwritten digits and fashion product”. The approach they had followed to create this product is highly optimal and they say it “works at the speed of light”.

We used Fashion-MNIST in a ResNet-50 model we created while others used it to test their own models before testing it for their own respective use case. While most of the submitted models found here break the 90% accuracy mark, not many of them had used a model with a fifty layer neural network. The most number of layers found for a neural network submitted using a ResNet model is eighteen.

4. Our Approach

For our model, we used an implementation of ResNet-50. We started off the project in hopes of being able to implement both ResNet-18 and ResNet-50 models, to be able to compare them. The problem with implementing ResNet-18 was a lack of resources and time. ResNet-18 required much more GPU resources that we couldn't provide with Google's Resource Allocation, this hindered our testing and training process, but more importantly our debugging process. We decided to fully implement the ResNet-50 architecture first, alongside the theory that increasing the number of layers in our model would also increase accuracy. We also applied different activation and gradient optimizers, comparing between the multitudes provided by the tensorflow library.

4.1 Imports

Tensorflow is widely used to design our model. To be specific, we used the ResNET model provided by TensorFlow via Keras. Karas is the high-level API used to build and train models in deep learning.

Along with that, many other libraries have also been used for minor purposes such as time, numpy and matplot. Each of these play a special role to make the user understand their model better. Numpy helped us keep track of the number of epochs, loss, accuracy and as well as time spent to run each epoch. Matplot makes it easier to include diagrams in a model. We used it to print data visually in a graph to show changes in model accuracy throughout the total number of epochs.

4.2 Initial Two layers

To simplify the input, we start by max pooling the dataset. However, to make the process of max pooling easier, we start by applying batch normalization to the input first. The purpose of doing this is because it helps with speed, accuracy and efficiency in training the model using the dataset.

After that, we proceed with applying ReLU activation function to the output given from batch normalization as well as applying max pooling operation to the output from the activation function. When added to a model, max pooling reduces the dimensionality of images by reducing the number of pixels in the output of the previous layer. For our purposes, our model downsizes the input to a 3 x 3 and uses a stride of two for max coverage of the image before it proceeds next to the bottleneck architecture.

4.3 Architecture Used

For our architecture, we greatly took influence from the 2016 research paper written by He et al., titled "Deep Residual Learning for Image Recognition." With a fifty-layer structure, a model can be very inefficient to train and test large amounts of input images. For an optimal solution, we decided to structure our model with the bottleneck architecture. This is used in deep learning to simplify computational considerations. In simple terms it is just a layer with less neurons than the layer above or below it. The purpose of having such layers encourages the network to compress representations to best fit the available space to get the accurate loss during training.

A basic block is what is used for ResNet 18 and ResNet 34 because it is affordable using GPU ram however when it comes to having as many as 50 neural network layers in between, using a basic block wastes much of the GPU ram to run expensive 3 x 3 convolutions. A simple bottleneck uses a 1 x 1 convolution to reduce the channels of the input before performing the expensive 3 x 3 convolutions (Sachan, 2019). Once this has been done, it uses another 1 x 1 convolution to restore the dimensions back to the original shape. For our project, we have used sixteen residual blocks with different input dimensions taken from the output of the previous layer. This makes it another forty-eight layers in between and adding this along with the initial two layers make this a fifty layer model.

To simplify the number of residual blocks used in one run, we divided all sixteen blocks into four stages with different numbers of blocks in every stage. Stage 1 contains three blocks, stage 2 contains four blocks, stage 3 contains six blocks and stage 4 contains three blocks.

Throughout the residual blocks, we also used ReLU activation at every point after the dimensions of the data were resized. Activation functions help determine the output of the model itself by defining its accuracy and efficiency of the training model which is why it is very crucial to use them in networks. For our purposes, we used ReLU because it easily overcomes gradient problems which helps models to learn faster and perform efficiently.

4.4 Closing Structure

Towards the end of the structure, we need something that can simplify the outputs returned from the layers. For this, we proceeded by using Average Pooling. Average pooling is used to compute the average for each patch of the feature map. Since at this point we have completed computing all residual blocks therefore to get accurate results we choose to use average pooling over max pooling. In simple terms, the difference between max pooling and average pooling is mainly the fact that max pooling will extract the most important features of an image like edges which is why we chose to apply max pooling at the beginning of the model whereas average pooling takes everything into account and returns an average value which is beneficial for us since now we read though all of our input images.

This would then create a dense layer that would need to be reduced to simplify the output. A simple, softmax activation is used to compute losses from the previous layers. By the end of this, we conclude with one complete epoch.

5. Experiments & Results

In our experimentation, we realized the optimal number of epochs with batch size of 128 was 100. Post 100 epochs, the model was making negligible accuracy improvements, while still taking linear time to train and consume GPU resources. As can be seen in Figure 1 and Figure 2. Our results indicated that after epoch 80, the accuracy improvements greatly decreased per epoch, decreasing from an improvement of .00083 to .00002, a factor of 40 times, which is seen in Table 1. Regarding batch size, 128 was found to be a midpoint between accuracy and time to train, ensuring that while training time wasn't unbearably long, we were still able to maintain a high amount of accuracy. Epoch tuning made more of a difference on this, so we focused more efforts on epoch training and tuning.

Table 1: Accuracy Over Epoch Iterations

Epochs	Train Accuracy	Test Accuracy	Time Elapsed	Accuracy Improvement
10	0.87858	0.87318	15.98	N/A
20	0.90346	0.88866	33.47	0.01548
30	0.91821	0.89424	47.79	0.00558
40	0.92908	0.89741	63.51	0.00317
50	0.93753	0.89876	79.11	0.00135
60	0.94403	0.90037	94.71	0.00161
70	0.94957	0.90144	110.46	0.00107
80	0.95402	0.90227	126.11	0.00083
90	0.95776	0.90229	141.8	0.00002
100	0.96101	0.90368	157.5	0.00139

110	0.96355	0.90411	172.86	0.00043
120	0.96578	0.90452	188.1	0.00041
130	0.96779	0.90486	203.44	0.00034
140	0.96955	0.90526	219	0.0004
150	0.97105	0.90559	234.56	0.00033
160	0.97236	0.90574	250.06	0.00015
170	0.97364	0.90607	265.57	0.00033
180	0.97439	0.90623	281.25	0.00016
190	0.97541	0.90644	296.9	0.00021
200	0.97636	0.90663	312.61	0.00019

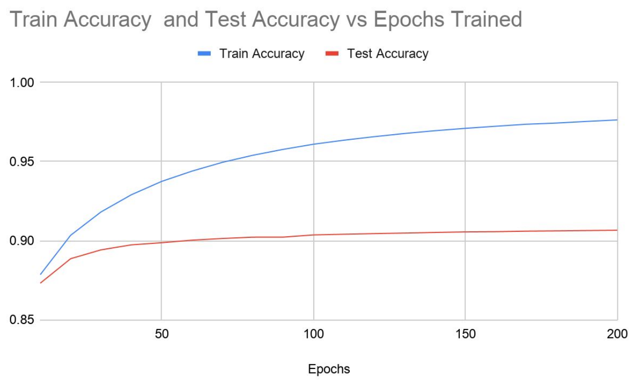

Figure 1: Train Accuracy and Test Accuracy vs Epochs Trained

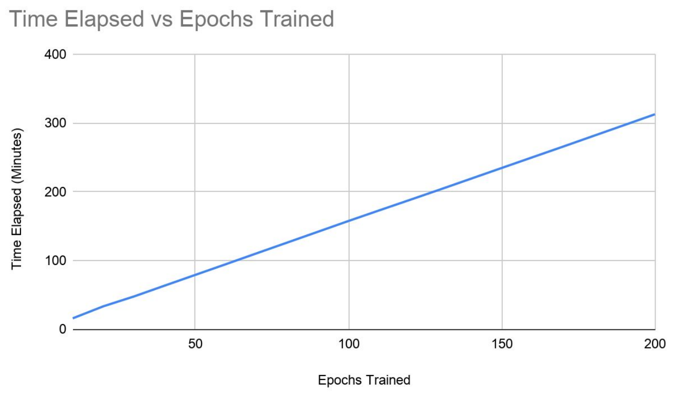

Figure 2: Time Elapsed vs Epochs Trained

6. Conclusion & Discussion

In conclusion, while the network works and runs at a pretty high accuracy, it’s nowhere near as efficient as the group was hoping it would be. In future projects, we’d probably run with a neural network from figure 3 that produced higher accuracy, but with less operations, and one that is less computationally resource demanding. ResNet-50 did provide an accuracy the group was satisfied with, but when it came time to fine tune, it became evident that our network was too heavy to be tuned to be truly efficient. We don’t believe it would be plausible to run this on a mobile platform while still expecting the phone to be considered in a “usable” state.

Some other problems we ran into were computational limitations. Choosing batch sizes was limited by GPU Memory limitations, and Google Colab’s resource allocation. We reached ceilings in terms of both. Also making more intensive computations showed the inefficiency in our network, leading us to study more into applying bottleneck architectures to offload some of the load.

It was also interesting making comparisons between home-brewed implementations, and predefined library implementations. The Keras and TensorFlow library oftentimes provided solutions for problems we faced, however, using library defined functions sometimes created more problems that required intense amounts of macgyvering to even get code to compile, let alone train. Oftentimes, I found it useful to try and spend time and resources in getting library functions to work, or researching what was required to make a library function to work, but sometimes looking back, it would’ve made more sense to just start on my similar but unique (to my situation) implementation.

Lastly, relating to libraries, and library functions, it was quirky seeing the differences in formatting and syntax between library functions. My group and I had used PyTorch throughout the semester to accomplish homeworks where we were allowed to customize our implementation, as that was what we were familiar and comfortable with (and had an in class lecture with). However, in this implementation, we found more documentation regarding ResNet implementations in TensorFlow and Keras, it was an experience finding the differences and similarities between the libraries, as they’re all known for machine learning and their end goals in my mind were all similar, but their offerings can be very different when looked deeper than face value. It took the group a while to forget the way things were run in PyTorch and to ensure we followed proper TensorFlow documentation to ensure we encountered the least amount of errors as possible.

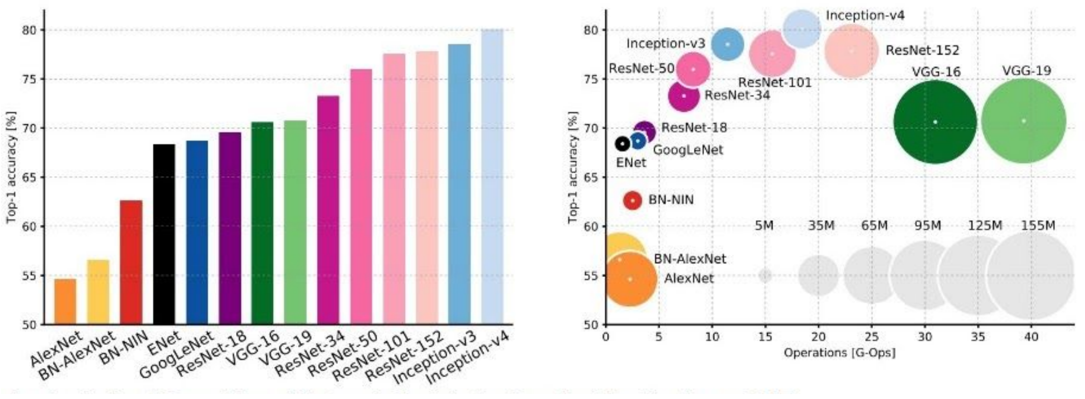

Figure 3: Analysis for Deep Neural Network Models

7. Future work

Our original intention for this project was to set the footing for a long term project that would work on mobile. We wanted to be able to create a neural network that could classify and detect the individual pieces of clothing in a person’s closet, and draft out hypothetical outfit ideas, allowing the user to then “like” or “dislike” outfits, introducing bias into the algorithm to consistently produce “like-able” outfits for each user’s specific tastes.. In the timeframe we were given, we were able to find a proper dataset for our use case, and create, tune, test, and train our model.

With 2 more weeks time, we would implement another, perhaps lighter, model and see how much of an accuracy tradeoff we’d be making in the sense of accuracy vs. portability, since we’d want this network to be light enough to run on mobile platforms. With two months worth of time, the team would introduce more classifications into the network, the two months would give us enough time to introduce more exotic data to the new networks, and create new classifications, like patterns, colors or customizations (ex: Distressed Denim, Striped Sweater, Yellow Jacket).

With a semester’s worth of extra time, the team would work on tackling the larger picture. With the models trained and primed, we would move further to creating the mobile interface, which would allow users to take pictures of everything in their closet and create a virtual inventory. Using said virtual inventory, users would then be able to visualize individual pieces to bring together an entire outfit. The core components of this implementation would be the classification and the learning required to detect the fashion sense of the person choosing and wearing the outfits, which would adapt based on the feedback of the user. The end product’s goal would be to eliminate decision making anxiety, and time wasted in choosing an outfit.

8. Presentation

Presentation Link: here

References

Fung, V. (2017, July 17). *An overview of ResNet and its variants*. Medium. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Hassan, M. A. (2019, January 25). *ResNet (34, 50, 101): Residual CNNs for image classification tasks*. Neurohive - Neural Networks. https://neurohive.io/en/popular-networks/resnet/

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2016.90

Introducing TensorFlow datasets. (n.d.). The TensorFlow Blog. https://blog.tensorflow.org/2019/02/introducing-tensorflow-datasets.html

Jay, P. (2018, April 8). *Understanding and implementing architectures of ResNet and ResNeXt for state-of-the-art Image Classification: From Microsoft to Facebook [Part 1]* Medium. https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

Sachan. (2019, September 17). *Detailed guide to understand and implement ResNets*. CV-Tricks.com. https://cv-tricks.com/keras/understand-implement-resnets/

TensorFlow. (n.d.). *Tf.keras.applications.ResNet50*. https://www.tensorflow.org/api_docs/python/tf/keras/applications/ResNet50

Xiao, H. (2018, September 28). *Fashion-MNIST: Year in review*. Han Xiao Tech Blog - Deep Learning, NLP, AI. https://hanxiao.io/2018/09/28/Fashion-MNIST-Year-In-Review/

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *ArXiv:1708.07747 [Cs, Stat]*. http://arxiv.org/abs/1708.07747