Outfit Classification

Jainy Patel, Shiv Patel

Intro

Outfit Prediction Problem:

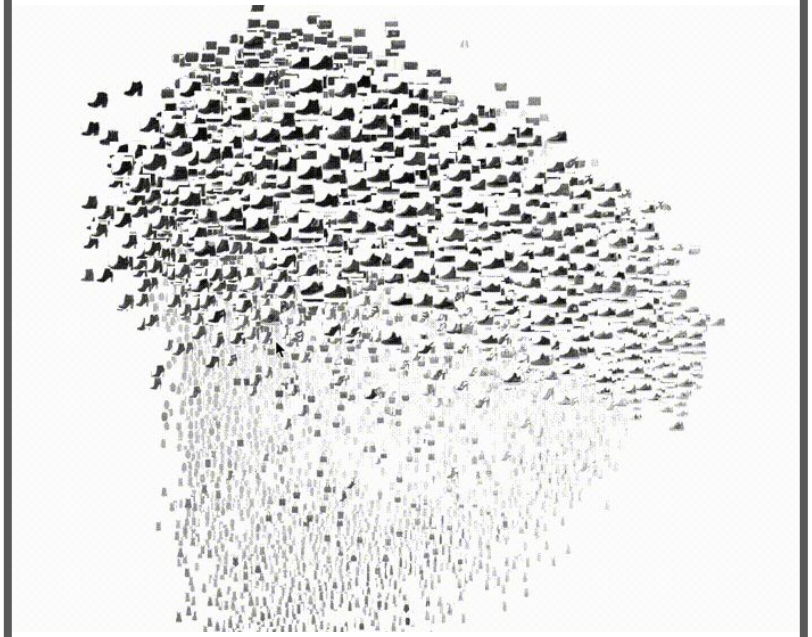
- How to efficiently classify then select various outfits?

Significance:

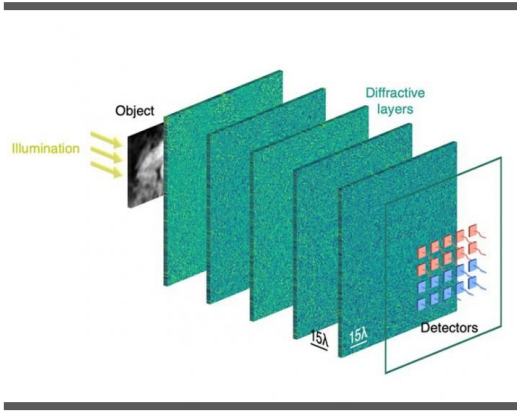
- We wanted to develop an image classification algorithm that could separate parts of outfits into different classes.
- We used the Fashion-MNIST dataset, which provided us 60,000 testing and 10,000 training images.
- Fashion-MNIST is meant to be a replacement for the MNIST dataset, which explains the similarities between the structures of the datasets.

Dataset

- Imported using the tensorflow datasets library
- 60,000 training images, with 10,000 testing images, all standardized to be 28x28 and grayscale, with pixel values ranging from 0 to 255.
- Classification labels ranged from “shirts” to “shoes” to “accessories” with 10 classes in total.



Related Work



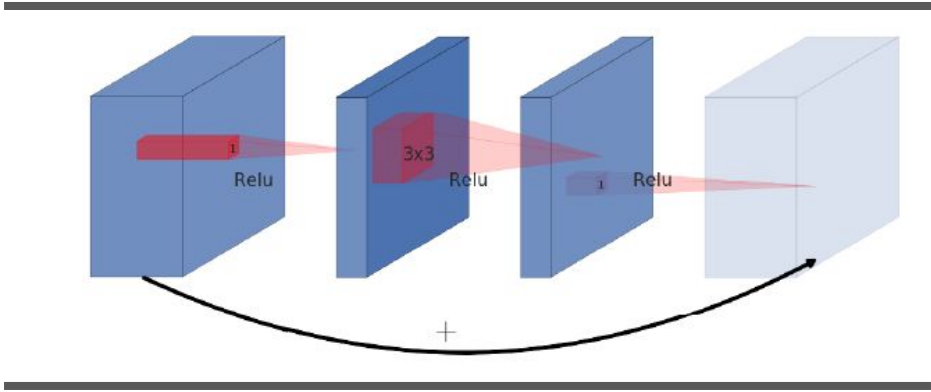
- Initial research work has been done by Kashif Rasul and Han Xiao from Zalando Research group.
  - They sought to create a new dataset that would be a drop-in replacement for the original MNIST dataset, due to difficulty concerns regarding the original dataset.
  - Their purpose of creating a new separate 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.
- An 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”.

Expected Results

We expected to create a classification algorithm that would be able to detect different pieces of clothing in isolation. We were able to accomplish this by implementing and testing different classifiers.

For expected results, we believe testing this dataset out on ResNet-50 will show higher accuracy when compared with ResNet-18 due to training data in an increased number of layers. Speaking with numbers, we expect to have a difference of at least 5% in testing accuracy when comparing both of these models.

Approach



- Use a set of 60,000 training and 10,000 testing images and labels.
- All images were standardized to be 28x28 pixels, and grayscale to match the original MNIST dataset.
- Implementing ResNet-50, implemented using tensorflow
  - Utilized the Bottleneck Architecture to simplify runtime
  - Applied Max Pooling and Average Pooling when necessary
  - Used Softmax to compute loss based on Average Pooling
  - We then classify each image to 1 of 10 classes.
- Based off the results from each epoch, save the training and testing accuracy in an array
  - Compute an average of that to understand accuracy results
- Test this out with various number of epochs

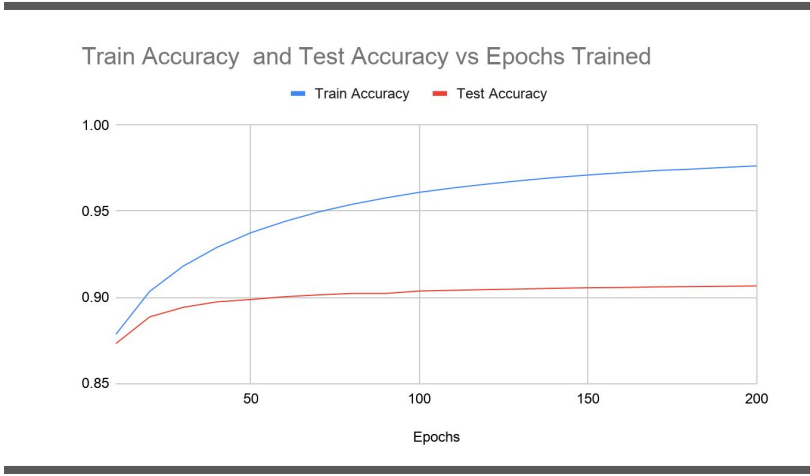
Issues

The largest issue we dealt with was resource management. We don’t have local computers with GPUs so we had to resort to cloud solutions. This greatly impacted us as we had an extra step of troubleshooting when things didn’t go as planned. We also had to troubleshoot finicky things with Google’s solutions like resource allocation or resource limits.

Another large issue we faced was documentation. We realized that some libraries are heavily favored by the community over others, and due to that, documentation and help was better found. We ended up using the TensorFlow library and its functions to implement our classifier.

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.



Out 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. Epoch tuning made more of a difference on this, so we focused more efforts on epoch training and tuning.

We wanted to reach approximately 90% accuracy and were able to reach this. We did not meet our goals of implementing other classifiers due to time constraints.

Future Work

2 Week Plan:

With 2 weeks of additional time, we would continue our process of comparing classifiers, seeing which implementations give us the most lightweight and efficient process, this would help us with our 1 semester plan of creating a mobile application.

2 Months Plan:

With 2 months of additional time, we would add new features to the model including color and pattern detection. We would also allow the classification algorithm to then create “outfits,” combining pieces of each class to create an aesthetically pleasing combination.

1 Semester Plan:

With a semester’s worth of additional time, we would combine the 2 week and 2 month progress and deploy a mobile application that would be able to scan an inventory a user’s closet into different categories, and then create outfits for the user to wear. The user would be able to “like” or “dislike” patterns, colors, or entire outfits, which would influence the weights of how the algorithm suggests outfits.