2D CNN's for Fashion MNIST

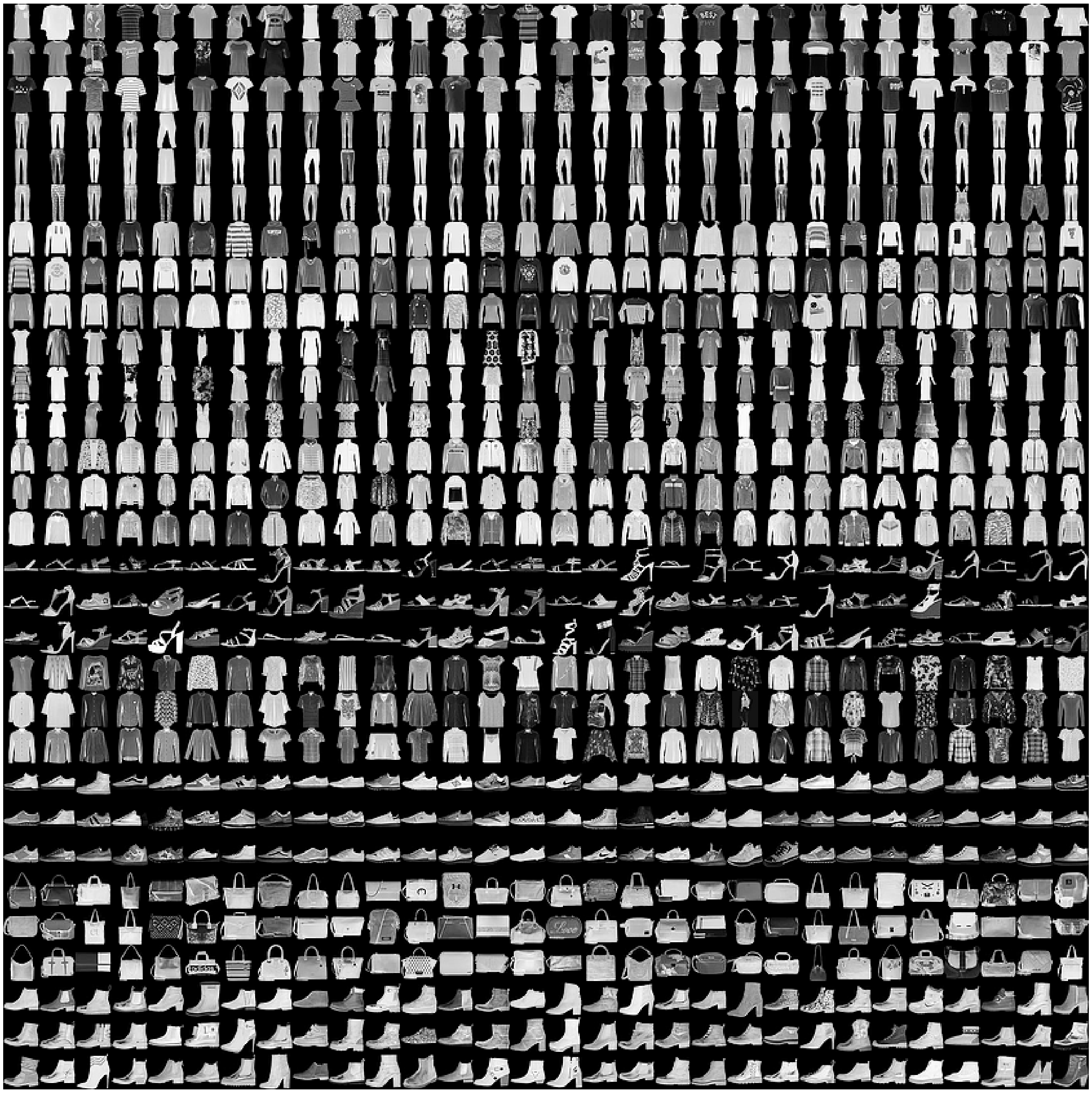
Herbart Hernandez - CSCI 167

Introduction

My simple report will be on a simple 2d Convolutional Neural Network, using the TensorFlow framework, on Google Collab for ease of access and implementation. The 2d CNN model aims to be a classification model of various different clothing items, and I will explore my findings here.

Data

Fashion MNIST


Picture of fashion MNIST from https://github.com/zalandoresearch/fashion-mnist/blob/b2617bb6d3ffa2e429640350f613e3291e10b141/doc/img/fashion-mnist-sprite.png

I decided to use the Fashion MNIST for the main reason that it is a dataset already available in the TensorFlow framework, this ease of access made it easy to get things up and running.

```
import tensorflow as tf
from tensorflow.keras import layers, models

# load data, built in to tensorflow
(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.fashion_mnist.load_data()
```

The Fashion-MNIST dataset is meant to be a replacement for the MNIST data set, as the name implies. Its meant to replace the MNIST set by being a little more difficult than the original handwritten numbers, it instead using 10 different clothing items with many samples to test and train a model.

The training dataset has 60,000 images each image being a 28x28 grayscale image, along with every image a label with a number 0 - 9, each classifying the piece of clothing in its own category these are as follows.

- 0 - T-shirt/top
- 1 - Trouser
- 2 - Pullover
- 3 - Dress
- 4 - Coat
- 5 - Sandal
- 6 - Shirt
- 7 - Sneaker
- 8 - Bag
- 9 - Ankle Boot
  Every class has an equal amount of samples in our data set so we have 6,000 samples of every class for our model to be trained on

Loading the data comes with the 60,000 training images but also 10,000 testing images and labels for validation, witch is again one of the reasons I decided to go with MNIST Fashion being it would be an attainable, and accessible for me to use and create a model.

Normalization and Reshaping

The data did not need many changes and only 2 main things where done. The first thing that had to be dealt with is the pixel values witch range from 0 - 255. However when our model will process the images is is best that we normalize these pixel values to a number 0 - 1, if we normalize the data then the model should be more stable with the smaller number, if we had stuck to using 0 - 255, there could be instability as the large number may change drastically during backpropagation, working with smaller numbers makes changes smaller and stable as opposed to large and unstable changes.

```
# normalize pixel values to be between 0 and 1 grayscale more stable!
train_images = train_images / 255.0
test_images = test_images / 255.0
```

The second small change made was changing the dimensions of our training and testing images, a 2d CNN needs 4 dimension input being batch size(number of images), height, width, and channels fortunately we are only dealing with grayscale images meaning we have 1 channel being a value from 0 to 1 after normalization.

```
# reshape for CNN
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1)
```

Model Creation

For the tasks of image classification, and small resolution grayscale images small models where they obvious answer. Convolution Neural Networks are also the correct model architecture for image classification, the reason CNN's are effective compared to other models like Fully Connected Neural Networks, is because of the reuse of parameters for the entire image using Kernels, witch can capture features with varying complexity depending on various hyperparameters. Without getting unrealistically computable.

Therefore I tested 2 models, a 2D CNN with 1 convolution and 1 dense layer for the classification, and another with 2 convolutions and 2 dense layers.

Basic Model

```
modelBasic = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(10, activation='softmax')
])
```

This basic model has a convolution with 32 filters to capture features, using a 3x3 kernel window, ReLU activation function for non linearity, and the initial dimensions of the input 28x28 with 1 channel because the images are grayscale. We also have a MaxPooling function to reduce the computation costs but also keep our features in tact. Next we flatten our output so that we can use a Dense Layer for direct classification of our 10 labels in the data.

Standard Model

```
modelBasic = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu',
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu')
    Dense(10, activation='softmax')
])
```

This model with a little more complexity, has 2 convolutional layers witch allows for hierarchical feature learning, meaning the model may pickup on more complex features as receptive fields increase , we are also using max pooling again to reduce computations but retain strong features. After flattening the data we use a extra dense layer, with 64 neurons, witch may again lead to more complex relationships to be `understood by the model` and finally it has a final dense layer with SoftMax so that we can get values for our 10 labels and classes it may choose.

Both models were again based on the CNN architecture as it is efficient for the use of finding features and allowing a model to learn vision based problems, witch in this case is image classification.

Hyperparameter Exploration

Hyperparameters are values used by the model that are not in control of the model, these values can change the performance of a model negatively if not correctly chosen, but may also increase performance if finding the optimal hyperparameters for a issue.

Model Choices

Convolutions

For the basic and standard model, convolutional layers with 32 filters for the initial layers where used, where the standard model used a second 64 filter convolution layer. The reason for the use of 32 filters, was inspired by many of the generic models that have been made for similar image problems. However what is standard and usual is the doubling of the filters at every hidden layer, and reducing as we get back to our output like was used for the dense layers in the standard model, when we double or half these parameters we are allowing our models to find more complex relationships between the data

Kernel size

For this model a 3x3 kernel (window) was used to find features in my images, while I could have used larger windows like 5x5 or 7x7, it would be more expensive, and while it may cover larger parts on an image for my case of 28x28 images 3x3 was sufficient as to not be overkill and more efficient for the problem.

Sub sampling

MaxPooling a 2x2 window was used for this model for both models, MaxPooling is a sub sampling technique that allows for us to reduce the information we are dealing for computational savings while being able to retain the actual needed information. Our simple 2d max pooling effectively halves the dimensions at every layer. For such a small and simple model 2x2 MaxPooling was sufficient.

Training Options

Optimizer

The ADAM optimizer was used in the compilation of the models, ADAM is a good starting base and since I was unsure I decided to stick to ADAM for its overall performance and quality, and again because our model is quite simple.

Loss

Doing some research online I found that the sparse-categorical-cross entropy loss function was appropriate for my application of image classification.

The standard model with deeper layers however was able to get a lower Loss Value indicating that the deeper model was in fact able to learn the relationships better even if slightly vs the basic model.

Epochs

I decided to use 8 Epochs for both of my models trainings, while I could have used more epochs for training, 8 was sufficient and was a good place to stop before overfitting and time of training issues started to show up, for this reason 8 epochs were used for the training of both models.

I had initially used 3 epochs and found the basic model to outperform the standard model witch I felt wasn't an accurate representation of the differences so then i decided to increase to 8 witch while it did bring diminishing returns it also conformed with what I expected from a model with a little more complexity.

Impact on Model

With these hyperparameters what we find is that the standard model does outperform the basic model even though just by a little bit with accuracy of standard being 0.9069/1 and basic model 0.9030/1, the model with more layers did outperform the basic model because it was able to find more features and relationships in the data.

However its important to note that the basic model has 54k parameters and the standard model has 121k parameters, roughly double of the basic model while outperforming the basic model only slightly, this tells us that for such a simple problem, like our Fashion MNIST classification, the basic model with less layers is sufficient for the task.

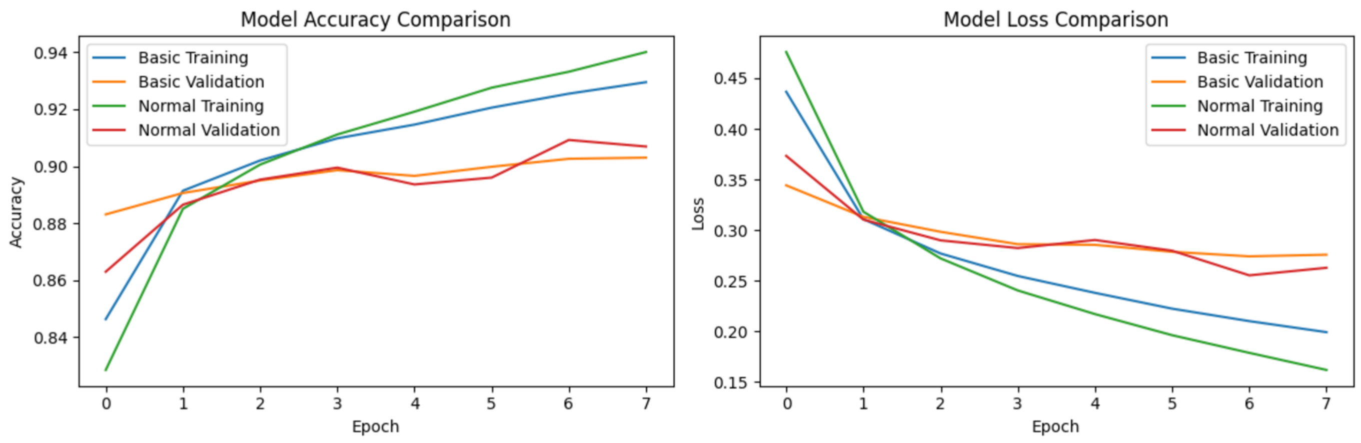
Results and Takeaways

Model performance

After 8 epochs the accuracy of the models are as follows

- Basic Model Accuracy: 90.3%
- Standard (deeper) Model Accuracy: 90.69%
  Deeper model was able to map the relationship better, but for the amount of resources it used (roughly 2x parameters vs basic model) it is clear that the model is too complex and too expensive for such a simple task, whereby the basic model was able to handle the problem at practically the same performance while being half as computationally expensive. This reinforces the idea that models highly depend and should be evaluated for the task at hand, in my case the simple problem of Fashion MNIST was very simple that a simple model could map the relationship well.

Accuracy and Loss



The left graph tells us that both models where effective at the classification problem getting up to the 90s range, telling us both models are accurate 90% of the time, We also notice that after the 3rd Epoch a gap starts to form from the validation and training data accuracy, this tells us that overfitting may be taking effect as if the graph continues validation accuracy may become worse.

The right graph models the loss of the models against the number of epochs. Both models drop to .34 loss after the first epoch and then slowly reduce there after. The increasing gap similar to the first graph again may indicate that overfitting may be occurring, as the training loss reduces the actual new data performance degrades witch indicates overfitting.

The 3 to 4 Epoch range appears to be the optimal Epoch range for our simple classification task, anything more seems to return diminishing gains while our cost to performance reduces heavily.

Strengths

1. Both models could effectively classify the proper clothing to the proper label it belongs to with 90% accuracy.
2. Both Models are relatively speaking very small and lightweight models, but with diminishing performance from deeper models we know that a basic 1 convolution layer network is able to effectively work for Fashion MNIST problem.
3. If different data is used more layers may be built onto these simple models as they are able to pass this MNIST test witch is used to just get models running and checking if they will work

Limitations

1. Because the problem is simple, the model is simple, recall the input for these models are 28x28 greyscale images, witch is very very limited, in input size and dimensionality of the data complex data will not handles well by out simple CNN's.
2. The use of the ReLU activation may limit more complex features should different inputs be used, The problem with this model is on the fact that it is very simple and not complex.

Real-World Applications

1. Though simple the model could be very useful for retailers backend systems, where the model may categorize and put labels on clothing items that come through the retailer.
2. The model can be expanded and used by Consumers at home to identify their wardrobes and create a database that recognizes all of their clothing's witch can be valuable and recommend other items to use.
3. This model may be possibly expanded and used in scanning clothing for defects, if the data was different for example and was one style of shirt, a model like this would quickly find defective clothing and mark it or let a factory know.

Future Improvements

This model is very simple but proves to be a good starting point for the possible creating of more complex models. Some possible improvements are as follows:

1. Different data can be used as in Data Augmentation where images could be stretched, change perspectives, flipped, mirrored etc. this would allow the model to understand items with more caveats.
2. Different Hyperparameters may change performance for the better while my experimentation was limited more time could have been spent testing the hyperparameters and optimizing performance
3. Deeper CNN's may be built by using more layers, to map hierarchal features, witch would allow for the use of much more complex relationship mapping and a more useful model.

Conclusion

This implementation of a CNN for Fashion MNSIT classification tells us how CNN's are the proper architecture for machine learning learning visual tasks. Comparing a simple model to a deeper model also tells us the importance of choosing a model architecture and complexity that make sense for a problem at hand. We should try to use simple models for simple problems, and build upon these simple models to map more and more complex problems. However for our case we find that a simple 1 convolution model is able to solve and successfully classify many of the clothing items.