Fashion MNIST:

Hyper-parameter Optimization

Patrick Seitz
Computer Sciences Department
Indiana University South Bend
South Bend, USA
pseitz@iu.edu

Shreya Patel
Computer Sciences Department
Indiana University South Bend
South Bend, USA
spa4@iu.edu

Brock Goley
Computer Sciences Department
Indiana University South Bend
South Bend, USA
brogoley@iu.edu

Abstract* — *This paper is a project proposal on image classification on clothes where our group will be changing the hyperparameters of an already existing and licensed model so that we can try to better classify these images using our new features.

I. INTRODUCTION

Image classification is a significant problem in the field of computer vision and machine learning. It involves the process of categorizing digital images into different classes or categories based on their visual features. One of the popular applications of image classification is in the domain of fashion, where the classification of clothes is essential for online shopping, wardrobe organization, and fashion recommendation systems. In recent years, deep learning techniques have been widely used for image classification due to their high accuracy and robustness.

In this paper, we propose a project to improve the image classification of clothes using an existing and licensed deep learning model. We will be experimenting with different hyperparameters to enhance the performance of the model and try to achieve better accuracy in classifying clothes images. Our objective is to explore the impact of hyperparameters on the performance of the model and determine which combination of hyperparameters works best for classifying clothes images.

The paper is organized as follows. In the next section, we will provide a brief overview of related work on image classification of clothes. Then we will describe the dataset used for the experiments and the deep learning model architecture. We will then present the different hyperparameters that we will be modifying, and the experimental setup used for our project. Finally, we will present the results of our experiments and discuss the implications of our findings.

II. PROJECT PROPOSAL

The objective of this project is to improve the image classification of clothes using hyperparameter tuning. We aim to experiment with different hyperparameters and evaluate their impact on the performance of an existing deep learning model for classifying clothes images.

Our project creates a functional clothes classification program using a pre-existing licensed classification model that will be able to tell the differences between images of clothes. The model we will be using is called “Basic classification: Classify images of clothing” which can be found on tensorflow.org. This model uses the Fashion MNIST dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels).

We will specifically be changing the hyper parameters of the model. The list of hyperparameters we will be exploring is as follows, activations functions, batch sizes, dropout rates, number of epoch iterations, initializers, learning rates, number of neurons, and different optimization algorithms. Each one of these groups has several different values to choose from. For example, the hyperparameter activation function and number of epoch iterations. There are several different activation functions to choose from, such as relu , tanh, sigmoid, hard_sigmoid and linear. By choosing one of these activation functions, what would it mean for the construction of the model? As for the different set of epoch interactions to choose from, what would the model be capable of if we only used 5 epochs versus 50? These hyperparameters were just 2 out of 9 total categories we can manipulate to change the performance of our model. We will split the dataset into training, validation, and testing sets. We will use the training set to train the model with different combinations of hyperparameters. We will use the validation set to evaluate the performance of the model and select the best hyperparameters. Finally, we will test the performance of the model on the testing set.

In conclusion, we will learn about what combinations work better than others for performance, giving us further insight on the underlying structure of how an artificial neural network (ANN) operates.

III. DESIGN APPROACH

A. Project Design

Our design approach for this project was given to us in the form of instructions from the professor for all the undergraduate students of this class which was to create a project based on an already existing program that was licensed and freely available. Then our instructions were to specifically adjust the hyperparameters in the program and to document what changes those hyperparameters did to the results of our program. These changes including but not limited to a difference of percentages in regard to accuracy, loss, speed, efficiency, etc. With those instructions in mind our group chose a clothes classification program because of the various different iterations of the program we could have various references to start our project from.

Our approach was to spend the first week getting the ANN program onto our computers and having the program work properly for everyone in the group. Then for every consecutive week of the ongoing project we would attempt to change another hyperparameter of our project and document our results in a group-shared excel document as the documentation of our project. We also spent some time discussing what we found within our group. Finally, we ended up compiling all our data and findings together to form our conclusion on the project and which are the documentation of the changes that these various hyperparameters have on the results of our clothing classification program.

B. Resources

- The dataset for this project was Fashion MNIST which is a dataset containing 70,000 grayscale images of clothing in 10 different categories. The one that we used on our project is linked and is found on GitHub.
- The downloads for the dataset found on GitHub through the Fashion MNIST link above:
    - The training set images was train-images-idx3-ubyte.gz
    - The training set labels was train-labels-idx1-ubyte.gz
    - The test set images was t10k-images-idx3-ubyte.gz
    - The test set labels was t10k-labels-idx1-ubyte.gz
- The python classification program which was the program we used to change the hyperparameters and run to record the results such as percentages of accuracy and loss. This program was called main.py.
- The libraries we used were TensorFlow, Keras, Matplotlib, and NumPy.

IV. FILES / FUNCTIONS

A. Programs

We used two programs in order to run tests on different sets of hyper-parameters. The two programs both hold the same structure of code with one program changing only one hyperparameter at a time. The two programs are called “fashion_MNIST_base” and ”fashion_MNIST_test”. The program “fashion_MNIST_base” holds the original code that was created by TensorFlow and distributed under the Apache version 2.0 license [7]. The code for the base program was kept unchanged in order to accurately compare the results of the test program to a known value. As for the program ”fashion_MNIST_test” this is where we performed all our tests. This program would systematically change one hyper-parameter after another and record the results that would be outputted.

B. main.py

For each program all that was contained was a main.py file. This file is what holds all of the code for the AI model.

C. Libraries

Libraries that were of importance were TensorFlow, Keras, and Matplotlib.

TensorFlow, as described by python documentation, is an open-source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices [8].

Keras, as described by Keras documentation, is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation [9].

Matplotlib, as described by their documentation, is a comprehensive library for creating static, animated, and interactive visualizations in Python [10].

D. Building the model

This is where we set up the basic building blocks of the neural network. These building blocks are layers. Layers extract representations from the data fed into them. Most of deep learning consists of chaining together simple layers. Most layers, such as tf.keras.layers.Dense, have parameters that are learned during training [7].

By using tf.keras.sequential we will be able to chain several layers together in order to form this basic building block. The method tf.keras.sequential is an array that takes in several arguments.

As said by the TensorFlow authors, The first layer in this network, tf.keras.layers.Flatten, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data. After the pixels are flattened, the network consists of a sequence of two tf.keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes [7].

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Fig. 1 Building the model

E. How the model will learn

Before we are ready to train the model we have to add a few extra settings. There settings are loss function, optimizer, and metrics.

The loss function measures how accurate the model is during training. Minimizing the loss function will help steer the model into the right direction to get the results you want.

The optimizer determines how the model is updated based on the data that it sees and the output from the loss function.

Lastly, the metrics is a way to monitor the training and testing. In our case the metric that is used is called 'accuracy', which measures the fraction of images that are correctly classified.

```
# How the model will learn
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Fig. 2 Model settings

F. Training the model

According to TensorFlow documentation training the neural network model requires the following steps:

1. Feed the training data to the model.
2. The model learns to associate images and labels.
3. You ask the model to make predictions about a test set.
4. Verify that the predictions match the labels from the test_labels array.

To start training, call the model.fit method, so called because it "fits" the model to the training data [7].

```
model.fit(train_images, train_labels, epochs=10)
```

Fig. 3 Training the model

G. Evaluating accuracy

The accuracy on the test dataset is actually a little lower than the accuracy on the training dataset, it turns out. Overfitting is shown by the discrepancy between training accuracy and test accuracy. When a machine learning model outperforms itself on novel, untrained inputs, it is said to have overfitted. An overfitted model "memorizes" the noise and specifics in the training dataset to the extent that it has a detrimental effect on the model's performance on the fresh data [7]. We can use the model.evaluate method in order to evaluate the model's accuracy.

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

Fig. 4 Evaluating models accuracy

V. Testing Strategy

A. Our Testing Strategy

As explained previously we have two programs both with the same structure of code. One is the base program (control) and the other is our test program. We figured out the average results of our base program. These results will be used as a basis to compare our new data against. For our test program we systematically change hyper-parameters one by one and run it to view the output. Once we have our output we log it into a spreadsheet to compare. We tested several hyper-parameters such as dropout, learning rate, optimizers, epochs, batch sizes, and initializers. For each hyper-parameter there was often multiple choices to choose from for each. For every test we ran we tested a total of three times in order to calculate an average. Once our values was found we logged them into our spreadsheet to be evaluated.

VI. Review

A. Results

The results that we acquired were to be expected for some and others were interesting to say the least. Starting with the simplest to understand conceptually is the number of epochs. For the control we start of by using a standard of 10 epochs. From there we then tested 25, 50, 75, and 100. Our results were as follows,

Epochs:	10
Loss:	0.3386
Accuracy:	0.8857

Epochs: 25

Loss: 0.3678
Accuracy: 0.8909

Epochs: 50
Loss: 0.4925
Accuracy: 0.8871

Epochs: 75
Loss: 0.6601
Accuracy: 0.8854

Epochs: 100
Loss: 0.7934
Accuracy: 0.8862

We can begin to see as the number of epochs go up so does the loss function. With our testing it seems to not have affected the accuracy.

The next hyper-parameter we tested was dropout. Dropout is supposed to minimize overfitting. These are the results we obtained,

Dropout: (base)
Loss: 0.3386
Accuracy: 0.8857

Dropout: (0.001)
Loss: 0.356
Accuracy: 0.873

Dropout: (0.3)
Loss: 0.3403
Accuracy: 0.8837

Dropout: (0.8)
Loss: 0.681
Accuracy: 0.8527

Dropout: (0.999)
Loss: 2.3028
Accuracy: 0.1088

We can see that as the dropout rate increases so does the loss function. We can also see that the accuracy also decreases by a significant margin. One can conjecture that having a small dropout rate is better for the model's performance. This makes sense as when the dropout rate meets a certain threshold the model will not be able to fit properly, thus affecting the performance of the model.

The next hyper-parameter we changed was learning rate. The learning rate is a hyper-parameter that controls how frequently the network's weights are updated by the optimization algorithm during training. It regulates how quickly and how much weight is adjusted in response to calculated loss or inaccuracy. The data we collected for learning rate is as follows,

Learning rate: (base)
Loss: 0.3386
Accuracy: 0.8857

Learning rate: (0.001)
Loss: 0.3425
Accuracy: 0.8779

Learning rate: (0.1)
Loss: 2.1829
Accuracy: 0.1497

Learning rate: (1)
Loss: 2.3865
Accuracy: 0.1001

Learning rate: (5)
Loss: 3.0267
Accuracy: 0.1

Our results make sense as when it comes to choosing a learning rate, selecting a proper value is essential. When an improper rate is chosen (higher values) in our case 0.1, 1, and 5 the training algorithm will fail, or overshoot the optimal solution. On the other hand, when too small of a value is chosen the algorithm can become slow and stuck on computing certain solutions. It seems that the learning rate of 0.001 was close to the base learning rate.

Continually, we also tested the hyper-parameter called optimizer. These are a set of built-in algorithms given by TensorFlow that can be chosen. Each optimizer is called by name and once called sets a variety of variables for the model. For example, reference *fig. 5*, which is the optimizer 'Adam' used in the base configuration. The list of these optimizers are long, each one containing multiple variables with their own list of documentation. We will not go into long detail on these optimizers, but focus on a few key ones evaluating their performance. For further reading and documentation please refer to the official TensorFlow API documentation for the optimizers [11]. That being said our data that we collected is as follows,

Optimizer: Adam (base)
Loss: 0.3386
Accuracy: 0.8857

Optimizer: (SGD)
Loss: 0.4096
Accuracy: 0.8555

Optimizer: (RMSprop)
Loss: 0.4293

Accuracy: 0.879

Optimizer: (Adadelta)
Loss: 0.9184
Accuracy: 0.6859

```
tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name='Adam',
    **kwargs
)
```

Fig. 5 Adam optimizer

After running our test, we can see Adadelta was not a good set of values for our particular model. The same can be said for the other optimizers, but on a smaller scale. The base optimizer that is used, which came pre-set in the original code, was the best optimizer for the model.

In conclusion, these were the key hyper-parameters that we found to have major effects on the performance of the Fashion MNIST model. We were unable to find a new set of hyper-parameters that were able to perform better than the base hyper-parameters. This being the case, it goes to show the amount of work and expertise the TensorFlow team has when developing this model. We may not have been able to find a better combination of hyper-parameters, but we have indeed learned more about the interactions these have on the performance of the model when changing them even in the slightest. We also gained a deeper understanding of the TensorFlow library. Having to read the API documentation and other documentation of the different methods have given us deeper insight on the innerworkings of a neural network.

B. *Challenges: Easy*

What we found easy was finding an idea that was interesting for us to work on. AI is a very broad field, there is something for everyone. We also found it easy to run tests on the model once we figured out what hyper-parameters to change.

C. *Challenges: Hard*

There were a few things that we found hard during this project. However, as they may been hard, it was definitely a good learning experience as we now can apply what we have learned to future projects going forward. The first thing that was hard was finding a model that was simple enough for us to learn from, but was still challenging. There was many topics to choose from, but we could only find a couple models meeting the proper requirements. The second thing that was difficult was interpreting the documentation on the hyper-parameters that we would encounter. Once we had an understanding we could then proceed to test out model. The last thing that was challenging was actually interpreting what the results meant. As said earlier, the base configuration was the version that was able to get the best results. We were unable to find a better version. After the tedious process of understanding the hyper-parameters and comparing results be began to see the correlations between the changing of these values. Once this was realized we were able to then see the bigger picture when it comes to setting hyper-parameters.

In the end we gained a solid understanding of the importance of choosing hyper-parameters and what it means on the performance of a model. With this knowledge we can then go forward and apply it to future endeavors, with the hopes of taking another step into a deeper understanding on how neural networks operate.

References

[1] J. Li, W. Shi, and D. Yang, “Clothing Image Classification with a Dragonfly Algorithm Optimised Online Sequential Extreme Learning Machine,” FIBRES & TEXTILES in Eastern Europe, vol. 29, no. 3(147), pp. 91–96, Jun. 2021, doi: 10.5604/01.3001.0014.7793.

[2] O. Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge,” Int J Comput Vis, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[3] Y. Seo and K. Shin, “Hierarchical convolutional neural networks for fashion image classification,” Expert Systems with Applications, vol. 116, pp. 328–339, Feb. 2019, doi: 10.1016/j.eswa.2018.09.022.

[4] A. Vijayaraj et al., “Deep Learning Image Classification for Fashion Design,” Wireless Communications and Mobile Computing, vol. 2022, p. e7549397, Jun. 2022, doi: 10.1155/2022/7549397.

[5] W. Wang, Y. Xu, J. Shen, and S.-C. Zhu, “Attentive Fashion Grammar Network for Fashion Landmark Detection and Clothing Category Classification,” 2018, pp. 4271–4280. Accessed: Mar. 09, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Wang_Attentive_Fashion_Grammar_CVPR_2018_paper.html

[6] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.” arXiv, Sep. 15, 2017. doi: 10.48550/arXiv.1708.07747.

[7] “Google colaboratory,” Google Colab. [Online]. Available: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb.

[8] “Find, install and publish python packages with the python package index,” PyPI. [Online]. Available: https://pypi.org/.

[9] K. Team, “Simple. flexible. powerful.,” Keras. [Online]. Available: https://keras.io/.

[10] “Visualization with python,” Matplotlib. [Online]. Available: https://matplotlib.org/.

[11] “Module: Tf.keras.optimizers :   tensorflow V2.12.0,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers.