Fashionable Neural Networks

Matthew Dees, Julius Aguma, and James Cho
University of California, Irvine
{mdees, jaguma, chojh2}@uci.edu

***Abstract*—We trained a convolutional neural network and a feedforward neural network on the Fashion-MNIST dataset using the Keras API in the Tensorflow library. The goal of this project was to explore the differences in performance between the convolutional and feedforward neural networks on a robust image classification dataset. Moreover, we performed bootstrapping as a method of cross validation of our models' performances. We found that the convolutional neural network generally performed better than the feedforward neural network under our testing criteria and confirmed that our models were consistent enough to be used for further analysis.**

***Index Terms*—fashion-mnist, machine learning, neural networks**

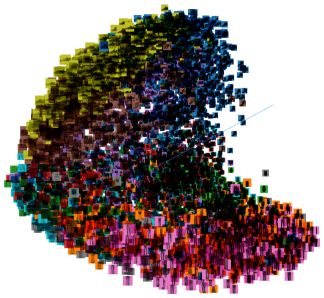

Fig. 1. Tensorboard visualization of the Fashion-MNIST dataset.

I. Introduction

Due to the growing popularity of computer vision and open-source machine learning tools, we decided to tackle an image classification problem by training two commonly used neural networks, convolutional (CNN) and feedforward (FNN), using the popular Python library Tensorflow. Of the choices provided, the Fashion-MNIST seemed to be the most robust image classification dataset, as it was designed as a drop-in replacement for the popular MNIST dataset. The Fashion-MINST dataset is supposed to be more challenging than the MNIST dataset, as the authors felt the original dataset was two easy to perform well on. Our project consists of a data visualization phase, modeling phase, and evaluation phase. In the data visualization phase we visualize the data in order to gain insights into potential modeling parameters and required normalizations. The modeling phase consists of the training of the CNN and FNN. The evaluation phase compares the performance of the two neural networks on the test data provided by the Fashion-MNIST [1] dataset. Due to the large amount of time required to train these neural networks, we decided to tackle the overfitting problem by adjusting the layers of the networks themselves, rather than by creating an ensemble of different networks. We measured the ratio of correct test image classifications to total number of test images as a way of measuring the strength of the each neural network. Utilizing the bootstrapping method and training the model with similar but different data, we confirmed our proposed models' consistency in performance.

II. Dataset

We decided to use the Fashion-MNIST dataset created by Zalando Research due to the excellent documentation on Github and plethora of utilities available within their Github repository. The dataset consists of a training set and validation set. The training set contains 60,000 images while the validation set contains 10,000 images. Each image is a 28×28 grayscale image (784 total pixels). The training set is of the form $X = Y$, where X is a $60,000 \times 784$ matrix and Y is a $60,000 \times 1$ matrix. Each row in X corresponds to a different image and each column corresponds to a pixel of that image (used as features). The Y matrix is $60,000 \times 1$ matrix where each value is a classification for the corresponding row (image) in the X matrix. Each classification is an integer value which maps to one of 10 different clothing classes.

To better understand our dataset, we decided to visualize it using Tensorboard [2] and Matplotlib [3]. Our implementation for visualizing the data is located in the **visualization.py** file in the top-level directory. We opted for a Principal Component Analysis (PCA) to get a better idea of what clusters may exist amongst the data. We chose PCA because the data has a relatively high number of features (784), which is not easy to visualize. With PCA, we have access to the features the preserve the most variance in the data, and may offer some insight into how it can be organized. When loaded into the Tensorboard visualization suite we were able to observe images like in Figure 1. Once colored, it was evident that the data actually formed reasonable clusters. This led us to believe that data was reasonably separable, so we *should* see fairly high classification rates with a model using the correct features. Ideally, we would learn which features these are using the neural networks. Using Matplotlib, we were also able to visualize a single image as shown in Figure 2. Upon visualizing this image, it was obvious that we should normalize the data. Since each pixel is a value between 0 and 255, we decided to normalize the data by dividing each pixel by 255. This proved extremely helpful in getting better results from our neural networks.

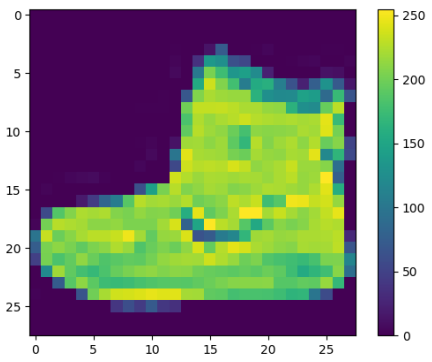

Fig. 2. Image 0 from the Fashion-MNIST dataset visualized using a color map.

III. Neural Network Configuration

Our neural networks shared many of the same characteristics, as the only thing that needed to change between the CNN and FNN were the hidden layers between the input and output layers.

A. Activation Function

Within these hidden layers we used the ReLU [4] activation function.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

The ReLU function works well because it is not expensive to compute, it doesn't contain any plateaus (so convergence is relatively quick), and it is sparsely activated. Sparse activation is a good thing in this scenario because it will increase the chance of a node not firing, and hence simulating a dropout effect to reduce overfitting.

B. Loss Function

We decided to use the categorical cross-entropy loss function within our neural networks.

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} \log \hat{y}_{ij})$$

We wanted to use a loss function that returned a low value when the true value matched the value guessed by the model and a high value when the guess was not correct. In the categorical cross-entropy loss function there will only be contribution from a given term y_i when $y_i \approx 1$, indicating a classification chosen by the algorithm. This classification will be correct when $\hat{y_i} \approx 1$, which will cause the overall term to approach zero (since $\log 1 = 0$). This loss function works very well for classification problems with multiple classes in general.

C. Optimization Algorithm

For our optimization algorithm we decided to go with Adam [5] because we wanted to try out one of the newer gradient descent variants produced by modern research. Adam was published in ICLR in 2015, and showed very promising training rates which we hoped to leverage in our project. The Adam algorithm can be used in Tensorflow applications through `tensorflow.keras.optimizers.Adam`.

D. Metric

To evaluate the testing and training performance of our neural networks we decided to use the typical classification accuracy formula:

$$A = \frac{1}{M}\sum_{i=0}^{M}(y_i = \hat{y_i})$$

In the above formula, M is the number of images in the training/test set that is being evaluated. The condition inside of the summation is used as shorthand to indicate the number of correct guesses. Essentially, this formula represents the number of correct guesses over the total number of items. We used this metric because we were familiar with it from class and there was support for it in Tensorflow.

IV. Convolutional Neural Network

A. Overview

Our Convolution Neural Network(CNN) architecture is inspired by ImageNet, the very popular architecture that achieved state-of-the-art results at the time of it's publication [6]. However, we use convolutional layers with r many kernels of size 3×3 or 5×5, where $r = 32, 64, 128$. The network also has k hidden layers whose sizes are varied for experimental purposes. To avoid overfitting, increase training speed, and maintain stability, we employ dropout, max pooling, and batch normalization. These are explained in detail below.

B. Dropout

To reduce overfitting, we employed google's regulation method termed dropout [7]; where the most redundunt units of each layer are deleted from the neural network with a propability of $1 - p$. Our value of p ranged between 0.2 and 0.5, with 0.4 giving the best results. In addition, dropout also helped increase the speed of training because the network does not have to train all data and initial input nodes.

C. Max Pooling

We also employed max pooling layers to further reduce overfitting by reducing the dimensionality of input and convolutional layers. Max pooling is a sample-based discretization process where the dimension of an $n \times n$ matrix is down sampled to allow for assumptions to be made about the features that are cut out. This process abstracts input or layers down to their basic internal representation [8]. To apply max pooling, we used a 2×2 max filter after the 3×3 convolutional layers.

D. Batch Normalization

By scaling the activations, we normalize the data in batches using means and variances. Batch normalization has been proven to make neural networks faster, stable, and more efficient. The method works by fixing the mean and variance of a batch C of data with size m to fixed values using the formulae below.

$$\mu_C = \frac{1}{m}\sum_{i=1}^{m} x_i$$

, and

$$\sigma_C^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_C)^2$$

V. Feedforward Neural Network

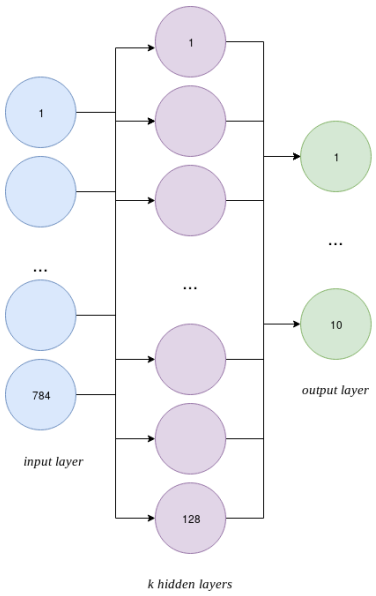

Fig. 3. Diagram of our feedforward neural network design. The singular input layer contains 784 data points. There are k hidden layers of 128 nodes each, and one output layer consisting of 10 data points

Our feedforward neural network (FNN) took a 784×1 vector as input, representing the entire 28×28 image. Each pixel was normalized using the following function as determined in the data visualization stage:

$$norm(x_i) = \frac{x_i}{255}$$

We thought the best way to fairly represent the feedforward neural network was to vary the capacity (variable k in Figure 9), so in our experiments we tested with k values of 1, 6, and 12. In testing these values we held the number of epochs at 20 to keep the training time low. With more time and better hardware we would have liked to increase the number of epochs to around 80.

VI. Results

A. Convolutional Neural Network

Results from three runs with the CNN can be seen in Figure 4. Our best test accuracy was 0.9125 with four hidden layers. To test the CNN we configured it as mentioned in section IV, and fit the neural network using 15 epochs. We used a small number of epochs to keep the training times reasonable. See Python file **experiments.py** for the code used to generate the graphs.

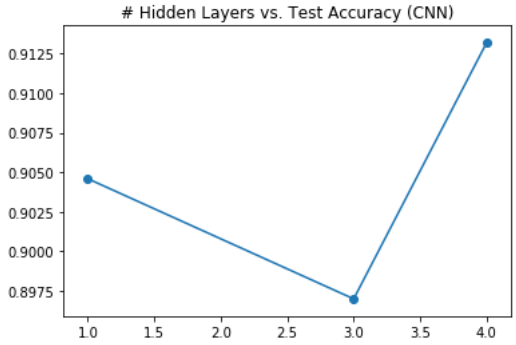

Fig. 4. Evaluation of the convolutional neural network with varying numbers of hidden layers.

B. Feedforward Neural Network

Results from testing the feedforward neural network(FNN) can observed in section 5. Our best performance came from six hidden layers. We suspect this is because the neural network becomes too complex with more hidden layers, and is not complex enough with fewer layers. With lots of hidden layers neural networks typically begin to learn the noise of training data, and thus do not perform as well on unknown test data. Our evaluation confirmed our feeling about neural network complexity and what we learned in class. Code for the FNN experiments can be found in **experiments.py** as well.

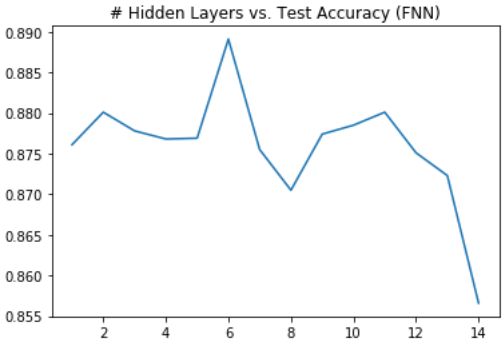

Fig. 5. Evaluation of the feedforward neural network with varying numbers of hidden layers.

VII. Evaluation & Validation

A. Bootstrapping Architecture & Motivation

As expected, the convolutional neural network model's accuracy and, especially, the feedforward neural network model's accuracy jumps around depending on the number of hidden layers (K). Moreover, we were unable to achieve an accuracy higher than 0.9 for the feedforward neural network regardless of the number of hidden layers. Conclusively, we became unsure whether our performance scores were a true representation of the power of our neural network models or whether the results happened by chance. Therefore, we became motivated to perform cross-validation to assess the validity of our results. Although the provided sample is huge enough to be recognized as an appropriate representation of the population, we decided to bootstrap sample the provided data with replacement. We chose a hidden layer of 4 and 6 for the convolutional neural network and feedforward neural network respectively. Furthermore, we recorded our models' performance scores on each generated bootstrap sample and computed 95% confidence intervals to diagnose whether we could achieve higher accuracy and to confirm our models' consistency in performance.

B. Results

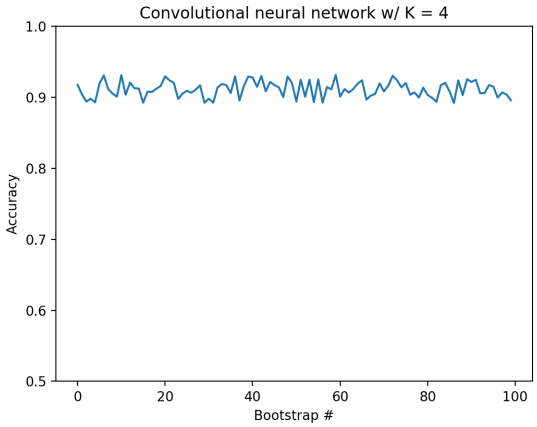

Fig. 6. cNN model (K = 4) Performance score (accuracy) for each bootstrap

C. Comments

With the generated bootstrap samples, we have trained our convolutional neural network (K = 4) model and feedforward neural network (K = 6) model, computed each bootstrap samples' performance, and constructed 95% confidence intervals of the performance scores for further diagnosis. Observing our initial performance scores lying in between the computed confidence intervals, we have confirmed our models' consistency. Further enhancing our argument of convolutional neural network models being a better choice for our given data, we note that the spread of the fNN model's performance score was wider than the spread of the cNN model's performance score.

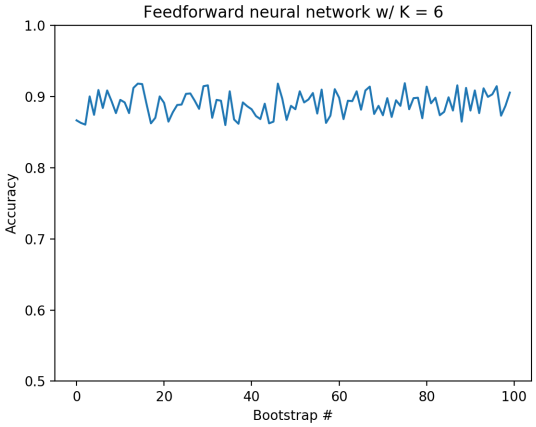

Fig. 7. fNN model (K = 6) Performance score (accuracy) for each bootstrap

Convolutional Neural Network Model with 4 Hidden Layers			
Minimum	Maximum	95% CI lower bound	95% CI upper bound
0.892	0.9315999999999957	0.9091439216757118	0.9135360783242839

Fig. 8. cNN (K = 4) Performance score (accuracy) minimum, maximum values and 95% confidence intervals of the bootstrap samples

Feedforward Neural Network Model with 4 Hidden Layers			
Minimum	Maximum	95% CI lower bound	95% CI upper bound
0.86	0.911800000000044	0.8863639285886439	0.8929640714113608

Fig. 9. fNN (K = 6) Performance score (accuracy) minimum, maximum values and 95% confidence intervals of the bootstrap samples

Bootstrapping provided us insight of the performance and validity of our models but heavily required computing power, since the original sample was decently large enough. We had to run Tensorflow in a GPU [9] (RTX 2080 TI) environment and it still took us quite a bit of time to bootstrap 100 samples (especially for the cNN model). Considering the industry rule of thumb of 1,000 - 10,000 samples, our choice of number of samples is small. However, we can already witness the spread of the performance scores for both models is not alerting and safely still claim our models' power and consistency has been validated.

VIII. Conclusion

In conclusion, we found that the convolutional neural network with four hidden layers gave us the best perfomance on the 10,000 test images. We were not necessarily surprised by this result because similar findings can be found on the Fashion-MNIST benchmarking website. Although, it was both exciting and rewarding to implement the neural networks for ourselves using the Tensorflow third party library and to corroborate the results of others.

IX. Contributions

A. Matt

Matt was responsible for organizing the Github repository, performing the data visualization phase (visualization.py), re-running experiments for the report, tuning hyperparameters for

the FNN and CNN (experiments.ipynb), and for writing the majority of the report.

B. Julius

Julius was responsible for writing the first iteration of the experiment code (cNN.ipynb and fNN.ipynb) and writing preliminary results to the report.

C. James

James was responsible for implementing bootstrapping (cNNwBootStrap.py fNNwBootStrap.py) by incorporating TensorFlow GPU settings to validate our results and contributing supplementary knowledge to the report.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[2] Tensorboard. https://github.com/tensorflow/tensorboard.

[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[4] Vinod Nair and Geoffrey E. Hinton. *Rectified linear units improve restricted boltzmann machines*. Omnipress, Jun 2010.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, Dec 2014.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[8] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] Tensorflow gpu. https://www.tensorflow.org/install/gpu.