AMATH 482/582: HOME WORK 5

SARAYU GUNDLAPALLI

Applied Math, University of Washington, Seattle, WA
saru07g@uw.edu

Abstract. This report presents an investigation into image classification using FCN and convolutional neural networks trained on the FashionMNIST dataset. The performance of FCNs with varying numbers of parameters (50K, 100K, and 200K weights) is evaluated in terms of accuracy. Convolutional neural networks (CNNs) are then explored as an alternative architecture, with CNN models constructed using up to 100K, 50K, 20K, and 10K weights. Hyperparameter tuning is performed for both FCN and CNN models to maximize test set accuracy. In addition to comparing classification performance across model types and sizes, the computational efficiency in terms of number of parameters and training time is analyzed. Overall, the study provides insights into the performance and efficiency of image classification with different neural network architectures.

1. Introduction and Overview

In this report, we explore the design and training of fully connected neural network (FCN) and convolutional neural network (CNN) models with different constraints on the number of weights.

First, we explore the performance of FCN models, with weights constrained up to 100K, for image classification on the FashionMNIST dataset[7], which consists of 28x28 grayscale images across 10 classes of clothing items. We perform hyperparameter tuning to achieve an FCN model with a testing classification accuracy above 88% on the testing set. Subsequently, we experiment by reducing the number of weights to create FCN 50K and doubling the number of weights to create FCN 200K. We analyze the accuracy of these models to understand the impact of weight constraints on performance.

Further, we extend our investigation to CNN models. We compare CNNs against classical fully-connected neural networks (FCNs) and analyze the effects of varying model sizes, defined by the number of learnable parameters. Specifically, we train and evaluate CNN models with approximately 100K, 50K, 20K, and 10K parameters.

In addition to predictive performance on test data, we also consider the computational efficiency of each model architecture in terms of training time requirements. By analyzing the trade-offs between model complexity, training time, and performance, we aim to provide insights into the suitability of different neural network architectures under varying constraints.

2. Theoretical Background

2.1. **Neural Networks:** Neural networks are a powerful machine learning model inspired by biological neural networks in the brain. They excel at tasks like image classification by learning intricate patterns in the input data during training. A neural network consists of layers of interconnected nodes, where each connection has an associated weight. The first layer is the input layer,

Date: March 4, 2024.

with one node for each input feature (e.g. 784 nodes for a 28x28 pixel image). The final layer is the output layer, with one node per class.

Between the input and output are one or more hidden layers that perform the learning and pattern recognition. Each node in a hidden layer calculates a weighted sum of the inputs from the previous layer, adds a bias term, and passes the result through an activation function. The network's weights and biases are iteratively adjusted during training to minimize a loss function between predicted and true outputs using optimization algorithms like gradient descent. This layered, weighted architecture allows neural nets to model highly complex, non-linear functions and extract relevant features from high-dimensional inputs like images.

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{1}$$

in the above equation, f() is the activation function, w_i is the weights, x_i is the input data, b is the biases, and y is the output.

2.1.1. ***Weights and Biases:*** The weights and biases develop how a neural network propels data flow forward through the network; this is called **forward propagation**. Once forward propagation is completed, the neural network will then refine connections using the errors that emerged in forward propagation. Then, the flow will reverse to go through layers and identify nodes and connections that require adjusting ; this is referred to as **backward propagation**.

2.2. **Convolutional Neural Network:** Convolutional neural networks (CNNs) are a specialized type of neural network architecture designed to efficiently process grid-like data such as images. CNNs use a series of layers, each of which detects different features of an input image[1].

2.2.1. ***Feature Map:*** A feature map is a two-dimensional array or grid of numbers resulting from the application of kernels to an input image or a previous layer's feature map. The process starts by sliding a filter designed to detect certain features over the input image, resulting in a feature map that highlights the presence of the detected features in the image. This feature map then serves as input for the next layer, enabling a CNN to gradually build a hierarchical representation of the image.

The key building blocks of CNNs are convolutional layers, pooling layers, and fully-connected layers.

2.2.2. ***Convolutional Layers:*** This layer uses a filter or kernel – a small matrix of weights – to move across the receptive field of an input image to detect the presence of specific features. Mathematically, the value at position (m, n) of the jth activation map in the ith convolutional layer can be computed as:

$$F[m,n] = I[m,n] * w[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} I[i,j]w[m-i,n-j]$$

2.2.3. ***Pooling Layers:*** The pooling layer aims to reduce the dimensionality of the input data while retaining critical information, thus improving the network's overall efficiency. This is typically achieved through decreasing the number of data points in the input.

2.2.4. ***Fully-Connected Layers:*** The fully connected layer integrates the various features extracted in the previous convolutional and pooling layers and maps them to specific classes or outcomes. Each input from the previous layer connects to each activation unit in the fully connected layer, enabling the CNN to simultaneously consider all features when making a final classification decision.

2.3. **Loss function:** The Loss function is a method of evaluating how well your algorithm is modeling your dataset. Here we use the cross-entropy loss function:

$$L(\hat{y}, y) = -(ylog\hat{y} + (1-y)log(1-\hat{y}))$$

3. Algorithm Implementation and Development

(1) *Hyperparameter Tuning for FCN 100K:*
   - Define the FCN model architecture using the provided FCN class, ensuring that the total number of parameters does not exceed 100K.
   - Perform hyperparameter tuning by adjusting parameters such as learning rate, batch size, number of hidden layers, and neurons per layer to maximize testing accuracy
   - Train the FCN model with the tuned hyperparameters.
   - Evaluate the testing classification accuracy of the trained FCN model.

(2) *FCN Variants (200K and 50K):*
   - Create FCN variants with 50K and 200K weights by reducing or doubling the number of weights in the FCN 100K model/ reducing or increasing the hyperparameters.
   - Train these FCN variants similarly to the FCN 100K model.

(3) *CNN 100K Model Training and Hyperparameter Tuning:*
   - Define the CNN model architecture with 2 convolutional, pooling, and 2 fully connected layers with a constraint of 100K weights.
   - Perform hyperparameter tuning similar to Task 1, but for CNN-specific hyperparameters such as kernel size, number of filters, and pooling size.
   - Train the CNN model on the FashionMNIST dataset.
   - Evaluate and report the model's testing classification accuracy.

(4) *CNN Variants (50K, 20K, and 10K Models):*
   - Reduce the number of weights in the CNN 100K model to create CNN variants with 50K, 20K, and 10K weights.
   - Train these CNN variants similarly to the CNN 100K model.
   - Compare the testing classification accuracy of these CNN variants with all FCN and CNN models.
   - Compute efficiency metrics, in terms of the number of weights and training time, for each model variant.

Packages used:

- NumPy[2]
- scikit-learn [5] for mathematical calculations
- Matplotlib[3] for visualizations.
- Pytorch[4] for building deep learning models.
- seaborn[6] for data visualisation.

4. Computational Results

We train a FCN model with $<$ 100K weights, and perform hyperparameter tuning to achieve a model whose testing classification accuracy is above 88% on the testing set. With the following parameters [LR = 0,003, Epoch = 15]1, we achieve a testing accuracy of 88.7%. Further, we train FCN models with reduced and doubled number of weights1.

We see that even though there are significant variations in the number of weights (100K, 200K, and 50K), the testing and validation accuracies across the models are relatively close to each other1. There is a slight overfitting observed in all models because the models might be capturing some noise or specific patterns present only in the training data. These results suggest that for the FashionMNIST dataset, the FCN's performance is relatively insensitive to the number of parameters within the range explored (50K to 200K weights). If the specific architecture design is too simple or too complex, changing the number of weights might not have a significant impact.

FCN			
Weights	HL dimensions	Testing Accuracy	Validation Accuracy
100K	100, 64	88.7%	89.266%
200K	200, 184	88.27%	89.966%
50K	54, 54	87.92%	89.11%

Table 1. Hyper-parameter tuning for FCN of different weights and testing accuracy> 88%

We implement and train a CNN model with convolutional, pooling, and FC layers with up to 100K weights.

With a test accuracy of 90.09%, the CNN 100K model outperforms all the FCN variants in terms of predictive performance on the FashionMNIST dataset. Compared to the best FCN (100K model at 88.7% test accuracy), the CNN achieves an improvement of around 1.4% on the test set. Moreover, the CNN exhibits significantly reduced overfitting compared to the FCNs.

The superior performance of the CNN can be attributed to its ability to automatically learn visual features directly from raw pixels in a hierarchical manner. While FCNs can still model the task reasonably well with enough capacity, CNNs are more efficient at leveraging the inherent properties of natural images.

CNN				
Weights	output channels of cv1/2	neurons in fc	Test acc.	Val. acc.
100K	8, 16	120	90.09%	91.0%
50K	8, 16	60	89.52%	90.533%
20K	4, 8	45	87.44%	88.033%
10K	4, 8	22	87.87%	88.533%

Table 2. Hyper-parameter tuning: CNN of different weights + test accuracies

We reduce the number of weights in the CNN 100K model to create CNN 50K, CNN 20K, and CNN 10K, training these models similarly to CNN 100K. We achieve this by adjusting the number of output channels in the convolutional layers and the number of neurons in the fully connected layers.

It is seen that even with just 50K weights, the CNN 50K model maintains competitive performance with 89.52% test accuracy. This is only marginally lower than the CNN 100K, but still outperforms the best FCN model. As we go down further to 20K weights in the CNN 20K model, there is a more noticeable drop in accuracy to 87.44% on the test set. However, this compact model still matches or exceeds the performance of the FCN variants like the 200K and 50K FCN2.

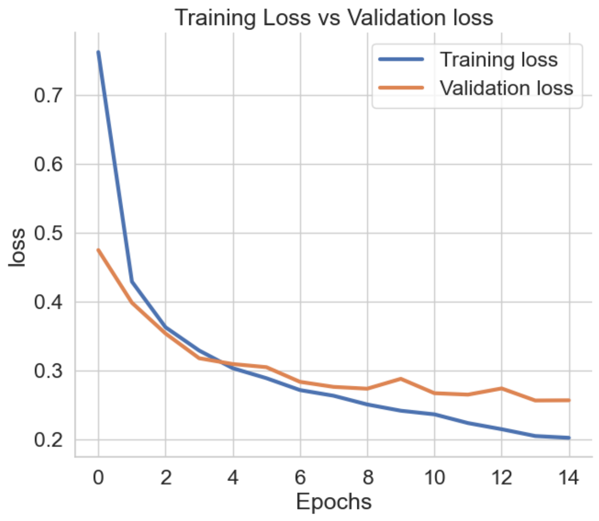
We also notice that the overfitting reduces significantly as we go from the larger CNN 100K model down to the more compact CNN 10K variant1. This might be because CNNs with limited parameters have far fewer degrees of freedom to overfit to the training data compared to larger models or FCN models.

4.1. **Efficiency:** In terms of computational efficiency, the CNNs demonstrate a clear advantage by achieving superior or comparable accuracy to FCNs using far fewer parameters.

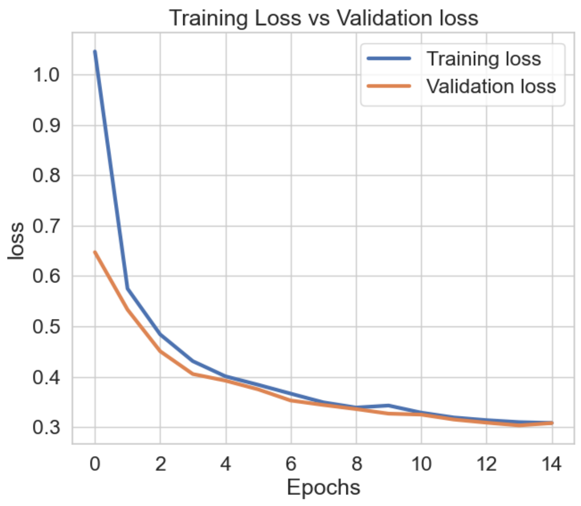
For FCN, the 50K FCN emerges as the most efficient, followed by the 100K and then the 200K model. This aligns with expectation, as larger FCN models have more parameters to train, increasing computational overhead without proportional accuracy gains3.

For CNN, the CNN 10K achieves the highest efficiency score. While the larger CNN 100K achieved the highest overall accuracy, the CNN 10K model obtained a still very respectable test accuracy using just 10,000 weights. Moreover, it took only 279 seconds to train, compared to 666 seconds for the CNN 100K3. In comparison, the FCN architectures exhibit relatively poor efficiency. Despite taking similar training times, they require many more weights to attain the same level of predictive performance as CNNs.

In scenarios where marginal accuracy improvements are not critical, deploying highly efficient compact CNNs like the 10K variant can provide substantial computational savings in terms of memory footprint, inference times, energy usage etc.



(A) for CNN: 100K



(B) for CNN: 10K

FIGURE 1. Comparison of over-fitting (Training loss vs Validation loss) as number of weights are reduced in the CNN model.

Weights	Total training time	Efficiency (weights/time)
FCN: 100K	237.7151 s	360.1536 s
FCN: 200K	273.5891 s	715.7958 s
FCN: 50K	239.1837 s	191.9445 s
CNN: 100K	666.1259 s	145.1046 s
CNN: 50K	629.5047 s	77.7722 s
CNN: 20K	482.5087 s	38.3018 s
CNN: 10K	279.2758 s	32.9853 s

TABLE 3. Efficiency of all the models, in terms of the number of weights and training time.

5. Summary and Conclusions

This study investigated the performance of FCN and convolutional neural network (CNN) architectures for image classification on the FashionMNIST dataset. Several FCN models were trained with varying numbers of parameters (50K, 100K, and 200K weights), achieving a peak test accuracy of 88.7% with the 100K model after hyperparameter tuning.

CNN models were then explored as an alternative, with network variants constructed using up to approximately 100K, 50K, 20K, and 10K weights. The best CNN model with 100K weights obtained a test accuracy of 90.09%, outperforming the FCN architectures. Decreasing the model capacity generally led to a very slight degradation in performance, with the 10K CNN still managing to achieve a reasonably high accuracy of 88.039%. In terms of computational efficiency, CNNs demonstrated a clear advantage over FCNs by achieving higher accuracy with fewer parameters.

Overall, the results indicate that CNNs are well-suited for the image classification task due to their ability to automatically learn rich hierarchical representations in a data-efficient manner.

Acknowledgments

The author is thankful for the suggestions/advice about the algorithm implementation from peers in AMATH 482/582. The author is thankful to the Teaching Assistants, Saba Heravi and Juan Felipe Osorio Ramírez, for their invaluable time and fruitful discussions about setting up the model. Further, the author is thankful to Prof. Eli Shlizerman for useful discussions about the course content.

References

[1] R. Awati. What is convolutional neural network?, Sep 2022.

[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

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

[4] J. Kutz. *Methods for Integrating Dynamics of Complex Systems and Big Data*. Oxford, 2013.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] M. L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[7] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.