AMATH 482/582: HOMEWORK 5

HAI VAN LE

Applied Mathematics Department, University of Washington, Seattle, WA
levh@uw.edu

Abstract. For this project, we are interested in training a Convolutional Neural Networks (CNN) to classify the images in FashionMNIST data set which has been used extensively in training machine learning models. Here, we design a CNN model and evaluate its accuracy at different number of weights and compare its performance against the Fully Connected Deep Neural Networks model that we built in the previous project.

1. Introduction and Overview

The FashionMNIST[2] database (Fashion Modified National Institute of Standards and Technology database) is a dataset of Zalando's article images—hosting a training set of 60,000 examples and a test set of 10,000 examples. Similar to the MNIST dataset that we examined in homework 3, each example in FashionMNIST is also a 28x28 grayscale image, associated with a label from 10 classes. Since MNIST has been extensively and overly used for training machine learning, FashionMNIST was published to replace its predecessor for benchmarking machine learning algorithms. According to the developers, FashionMNIST shares the same structure of training and testing splits as MNIST. Thus, we trained our classifiers using the training set while the test set is only used for validation/evaluation of our classifiers.

2. Theoretical Background

2.1. **Deep Neural Network.** Machine learning is a complex subject that is best explained through visualization such as a deep neural network as seen in Figure 1[1]. According to IBM, a neural network makes decisions in a similar way to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions. A neural network consists of layers of nodes or artificial neurons which comprise an input layer, one or more hidden layers, and an output layer are the layers of nodes, or artificial neurons. Every node has a weight and threshold attached to it and is connected to other nodes. For a 28x28 grayscale image, there are $28 \times 28 = 784$ inputs with values between 0 and 255. Training data is essential for neural networks to learn and become more accurate over time. Once we have determined the baseline configuration or set of parameters that trains in reasonable time and results in reasonable training loss curve and testing accuracy, we can apply the neural network model to quickly classify and cluster data.

2.2. **Convolutional Neural Networks.** Convolutional neural networks 3D data to for image classification and object recognition tasks. Figure 2 is a visualization[2] of the CNN model structure. They have three main types of layers, which are:

Date: March 9, 2024.

[1]https://www.ibm.com/topics/neural-networks

[2]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

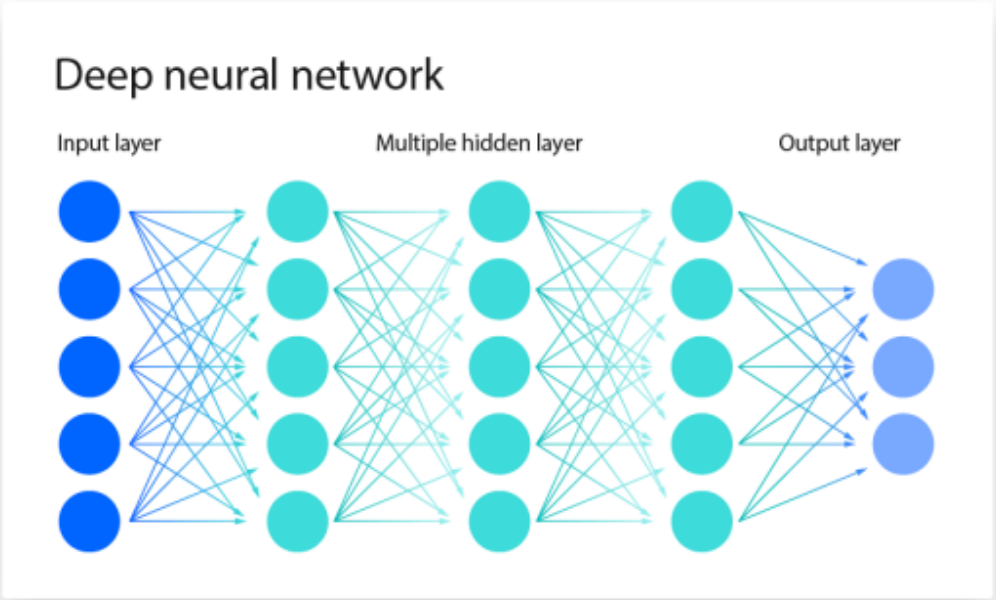

FIGURE 1. Deep Neural Network

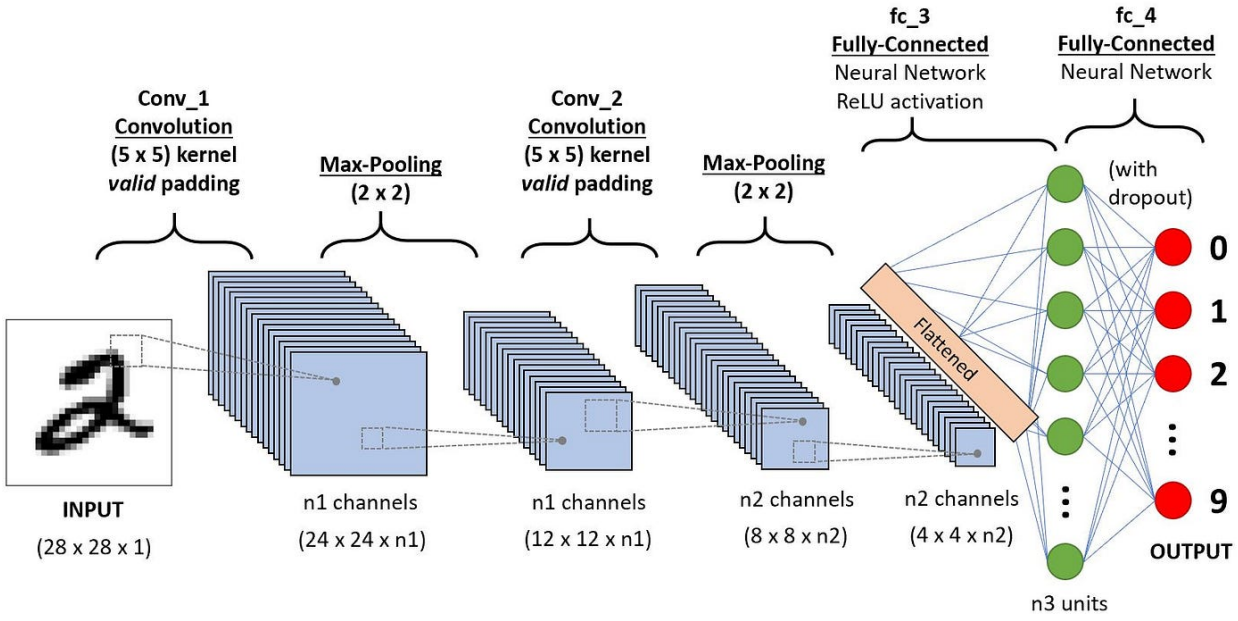

FIGURE 2. Convolutional Neural Network

- Convolutional layer: The majority of computation in a CNN takes place in this layer. Three things are needed for a convolutional layer: a feature map, a filter, and input data. As the inputs are images composed of a 3D matrix of pixels, the three dimensions of the input—height, width, and depth—will match the RGB values in an image. Additionally,

we have a feature detector—also referred to as a kernel or filter—that will scan the image's receptive fields to determine whether the feature is there. In this project, we also implemented another convolutional layer. Thus, the CNN's structure may become hierarchical because the pixels in the receptive fields of earlier layers are visible to the layers that come after them[1].

- Pooling layer: In this project, we implemented a max pooling layer for our CNN model. According to IBM, as the filter moves across the input, max pooling layer selects the pixel with the maximum value to send to the output array.
- Fully-connected (FC) layer: In this layer, every output layer node is directly connected to every other layer node. Using the features that were retrieved from the earlier convolutional layers and their various filters, FC layer can classify the data.

3. Algorithm Implementation and Development

This project deploys different library interfaces such as torchvision to load the data, *matplotlib.pyplot* - a plotting library for the Python programming language and its numerical mathematics extension NumPy. torchvision has useful transformations (e.g. normalization) and loads data as Pytorch tensors. Specifically, the training data is split into Training and Validation datasets and loaded into DataLoader as batches. We defined a DataLoader that allows for easy iteration over the batches and samples within it. Batches are useful for setting when weights updates will occur. We then reused the best-performing FCN model with parameters from previous project and defined a new CNN model using convolutional, max pooling and FC layers. Then, we wanted to test the performance of the new CNN model at different number of weights against the performance of the best-performing FCN model. We built CNN model using the skeleton from the PyTorchLab/PyTorchCNN notebook.

4. Computational Results

For task 1-2, we modified the FCN model from previous project so that it satisfied the constraint of using up to 100,000 weights and achieve a testing accuracy of 88% or more. We retained the key parameters of the best-performing configuration for FCN including Adam optimizer and Xavier Normal initialization. Our modified configuration is:

- Input dimension: 784
- Output dimension: 10
- Number of hidden layers: 2
- Neurons in each hidden layer: [100, 128]
- Learning rate: 0.1
- Number of epochs: 15
- Optimizier: Adam
- Initialization: Xavier Normal

We calculated the number of weights using *sum(p.numel() for p in model.parameters())*. With the above configuration, the number of weights is 92718 and testing accuracy is 88.43% with a training time of 36 seconds. We performed hyperparameter tuning: decreasing the neurons in each hidden layer to [64,32] to reduce the number of weights to 52,650 and testing accuracy decreased to 86.99% with a training time of 35 seconds. Then, we increased number of weights to 209,514 by increasing the number of neurons in the first hidden layer to 256 and got a testing accuracy of 88.26% with a training time of 37 seconds. For task 3-4, we designed a CNN model that has two convolutional layers, a max pooling layer and two fully-connected layers. At 103,018 weights, with paramaters:

- input dimension: 28
- output dimension: 10
- number of hidden layers : 1

- dimension of the hidden layer: 128
- learning rate: 0.001
- number of epochs: 10
- Training time: 1 minute 22 seconds

our CNN model has a testing accuracy of 90.05%. At 52,138 weights, with paramaters:

- input dimension: 28
- output dimension: 10
- number of hidden layers : 1
- dimension of the hidden layer: 64
- learning rate: 0.001
- number of epochs: 10
- Training time: 1 minute 45 seconds

our CNN model has a testing accuracy of 88.14%. At 20,954 weights, with parameters: with paramaters:

- input dimension: 28
- output dimension: 10
- number of hidden layers : 1
- dimension of the hidden layer: 32
- learning rate: 0.001
- number of epochs: 10
- Training time: 1 minute 34 seconds

our CNN model has a testing accuracy of 89.41%. Finally, at 13,978 weights, with parameters: with paramaters:

- input dimension: 28
- output dimension: 10
- number of hidden layers : 1
- dimension of the hidden layer: 16
- learning rate: 0.001
- number of epochs: 10
- Training time: 1 minute 24 seconds

our CNN model has a testing accuracy of 86.93%.

5. Summary and Conclusions

This project attempted to build a CNN model and compares its performance against the FCN model that was trained in previous project to differentiate images from the FashionMNIST dataset. This project tested the accuracy of each model on the testing dataset at different number of weights and found that the best performer at 90.05% testing accuracy is the CNN model with the following configuration:

- input dimension: 28
- output dimension: 10
- number of hidden layers : 1
- dimension of the hidden layer: 128
- learning rate: 0.001
- number of epochs: 10

The summary of performance of different models is in table 1.

TABLE 1. Performance Comparison

Model	**Number of Weights**	**Training Time**	**Testing Accuracy**
CNN	100K	1 min 22 sec	90.05%
CNN	50 K	1 min 45 sec	88.14%
CNN	20 K	1 min 34 sec	89.41%
CNN	10 K	1 min 24 sec	86.93%
FCN	100K	36 sec	88.43%
FCN	50 K	35 sec	86.99%
FCN	200 K	37 sec	88.26%

ACKNOWLEDGEMENTS

The author is thankful to Prof. Eli for useful discussions about the SVD algorithm and Python libraries. She is also thankful to the TAs who answered all questions regarding class materials and resources.

REFERENCES

[1] IBM. Convolutional Neural Networks. `https://www.ibm.com/topics/convolutional-neural-networks`, Accessed: 2024. Accessed on: March 9, 2024.

[2] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.