Project Report

Created	@December 1, 2024 9:58 PM
Created by	M Manmeet Singh

Dataset Exploration

First of all, I used the EMNIST Balanced dataset. It includes 47 classes which includes digits, uppercase letters, and lowercase letters. It is good for checking complicated models of handwritten recognition, but pretty soon I managed to understand that this dataset needed some advanced training techniques and far more computational power and knowledge than I currently have.

With some consideration, I decided to back off a bit and start with something more trivial like the MNIST dataset, which includes only 10 classes (digits 0 through 9). This was my choice for a reason-to be able to understand the very basics of deep learning in this area.

MNIST Dataset Overview

- Architecture: 70,000 training samples and 10,000 testing samples in 28×28 grayscale images.
- Preprocessing: I normalized the images in the range of -1 to 1 for better convergence of my model. I did the augmentation to generalize the results.
- That shift to MNIST freed my hands to lay the foundation skills and learning of the basics of building and training deep learning models.

Model Development

In this project, I implemented a CNN using PyTorch. Usually, CNNs do very well in image classification problems. I have tried to keep the architecture as simple as possible and yet powerful enough to suit the complexity of the MNIST dataset.

Model Architecture

- Convolutional Layers: Three layers with filters increasing stepwise from 32 to 128, all with a kernel size of 3×3.
- Batch normalization was included to make the training stable and converge fast. While MaxPooling reduced the dimensionality in the feature maps and extracted the most relevant features.
- Fully Connected Layers
    - These are followed by flattening the features, adding a dense layer of 128 neurons, and then an output layer of 10 neurons-all one for each digit.
    - Dropout: Used is dropout rate of 0.4 to reduce overfitting.
- I chose this architecture because it is a perfect balance between simplicity and power, hence a good match to learn and work with MNIST.

Hyperparameter

I tried experimenting with different hyperparameters to improve the performance of the model.

Learning Rate

First few tries with 0.01, then afterwards with a scheduler which will gradually reduce when the validation loss stopped improving.

Batch Size

I tried 32 and 64. A batch size of 32 performed a bit better on this dataset, giving a little higher validation accuracy.

Optimizer

I used Adam, then later changed to SGD with momentum because it generalized better and had less chance of exploding during training.

Epochs

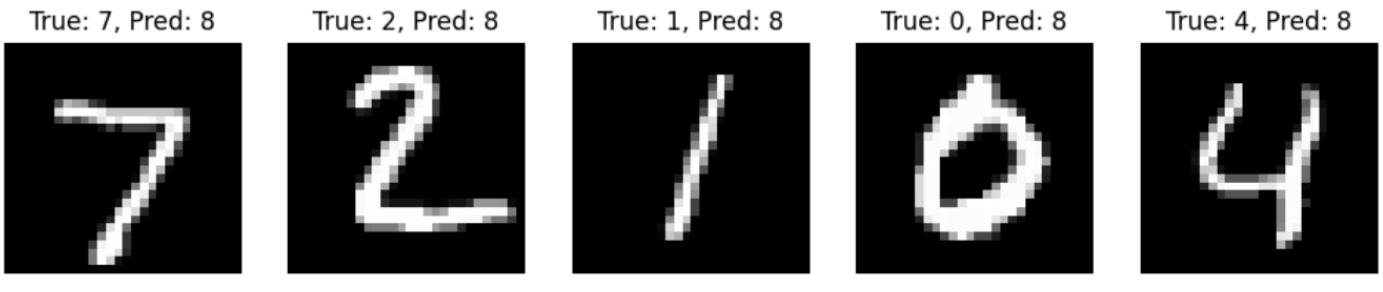
First, I trained it for 50 epochs, then went to 60 for fine-tuning, and got a good result.
These experiments showed me how different hyperparameters were changing the process of training and the work of a model.

Results Documentation

Performance

After training in 60 epochs, the model achieved a test accuracy of 98.4% in the MNIST dataset.
The loss, both for training and validation, generally went down with time, while the model stabilized at about 40 epochs.

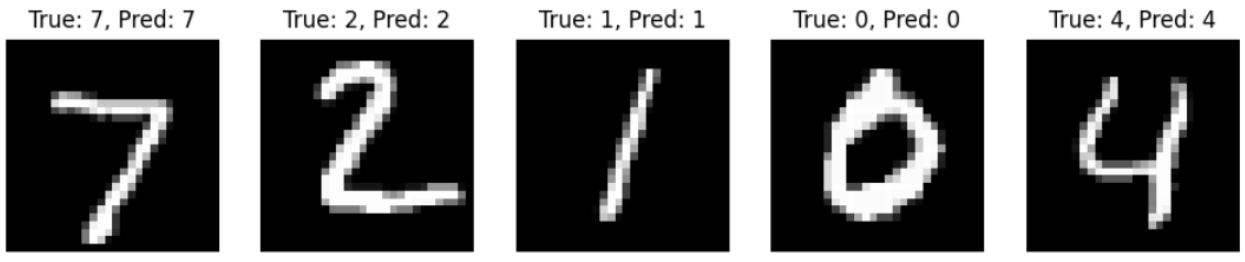
```
Epoch 17, Loss: 0.4874
Epoch 18, Loss: 0.4858
Epoch 19, Loss: 0.4865
Epoch 20, Loss: 0.4855
Model Trained and Saved
Testing a model
Accuracy of the model on the test images: 82.68184279979025%
Model Trained.
Time taken to train the model: 96.81692179838817 minutes
Testing user given input images
Predicted Class: 3
```

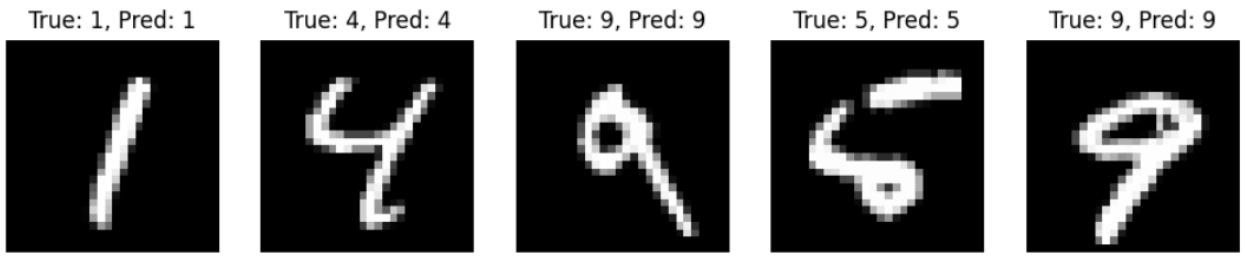



```
Epoch 10, Train Loss: 0.0105, Val Loss: 0.0336
Model Trained and Saved
Model Trained.
Time taken to train the model: 8.721180295944214 minutes
Testing a model
Accuracy: 99.08%
```

Key Findings

First, trying with EMNIST taught me to select an appropriate dataset considering my present skill level and resource availability. Then moving to MNIST helped me be focused and to learn and build a strong base. This is where the CNN worked well on MNIST, since this dataset is relatively simpler, and the model architecture went well with it.






```
Epoch 56, Train Loss: 0.0049, Val Loss: 0.0238
Epoch 57, Train Loss: 0.0049, Val Loss: 0.0238
Epoch 58, Train Loss: 0.0049, Val Loss: 0.0238
Epoch 59, Train Loss: 0.0049, Val Loss: 0.0238
Epoch 60, Train Loss: 0.0049, Val Loss: 0.0238
Model Trained and Saved
Model Trained. MNIST [0-9]
Time taken to train the model: 35.68084814945857 minutes
Accuracy: 99.35%
```

Applications:

This may serve in problems dealing with digit recognition, such as digitizing handwritten forms or postal codes. But for me, it's also a bridge to learn from simpler datasets to harder ones like the EMNIST dataset.

Future Enhancements:

In the future, I would like to experiment with deeper architectures like ResNet or EfficientNet to improve performance further. I would go ahead with reviewing the EMNIST dataset when I have the confidence and computational resources for the same. Elastic distortions in advanced data augmentation could add some robustness to it. This project has really taught me quite a lot from basic data processing, building the model, and optimization of training. One couldn't do better than starting with MNIST.

Note

If you are training mnist.py on your machine I would recommend using 10 epochs since it will take much short amount of time but doable performance to check the work I have done. Please check README.md file for more information and my git

Requirements

Python version 3.11

setuptools
torch
numpy==1.24.1
torchvision
opencv-python
matplotlib
pillow