Date:20/5/2006

# Implementing Neural Network Implementations provided by PyTorch

Report Details:

Number of Words (excluding Bibliography): 2187

Personnel details:

Writer: Shikhar Srivastava (Intern)
Mentor: Sumanth Pichika (Senior Tech Lead, TCTSL-Corporate)
Management: Sandeep Gupta (Deputy General Manager, TCTSL-Corporate)

# Overview:

Objective: To compare open-source/publicly available/freeware implementations of CNNs for image recognition

Tools: The following tools were used during the production of this report:

1. PyTorch (with PyTorch's TorchVision)
2. Cuda (for device agnostic code)
3. Numpy
4. Matplotlib (for plotting and visualisation.
5. Jupyter Notebook (run locally)
6. Python(Language)

# Implementation:

## Neural Networks:

The program tests three neural networks, which build up from a fully linear model, which is implemented the same as in the LinModel class in net_tools. It then builds up, first by adding a ReLU function, then adding a Convolution function and then building a CNN using the Tiny-VGG architecture on CNN Explainer [1].

This only compares models through accuracy with testing and plots it on a graph. The testing procedure is standardised for each neural net and uses the same dataset.

## Data:

The Dataset used here is a dataset called FashionMNIST, which is available through TorchVision. It is an MNIST-style dataset which has the following attributes:

1. Image Structure: Each image is a black and white image of an article of clothing, structured as a 28x28 pixel grid. Each image is associated with a label (formatted as str).
2. Dataset Structure: There are 70000 images. All images in are labelled with exactly one of ten classes. The Dataset widely has two categories – testing and training data. Training data has 60000 images and training data has 10000 images.
3. Classes: Each image is labelled with a class, all of which are categories of clothing, such as Ankle boot, Coat, Pullover etc.

## Training loop:

Each iteration (or epoch) of the training loop works by pulling each image from the training set and following these steps:

1. Converting image into a [1, 28, 28] sized tensor and passing through the neural net class to get an output. This returns a predicted label, which the *train()* function then compares to the actual label using *accuracy_fn()*. It records the percentage of labels predicted correctly and adding the percentages to a list after going through the entire training dataset. These predictions are used to train the model.
2. Every epoch, the program runs a *test()* function by feeding all 10000 test images (distinct from the training images) and records the percentage of correct predictions, adding to a list but does not use these images to train the model.

The reason the training and testing data is kept separate is to ensure that the model does not actually recognise the images used for testing. This makes the testing more accurately evaluate the performance of the model when used to predict the label of an image that it has not used to train and has never seen before or has seen before but not recorded any data or 'learning' from.

### Convolution Neural Network (CNN):

Convolution Neural Networks (or CNNs or ConvNets) are Deep Learning Neural networks that implement 'convolution blocks', or sequences of mathematical functions which code runs through, by passing data across nodes in these predefined convolution blocks. Code blocks usually work with several functions – usually Convolution, ReLU (rectified linear unit) and Pooling.

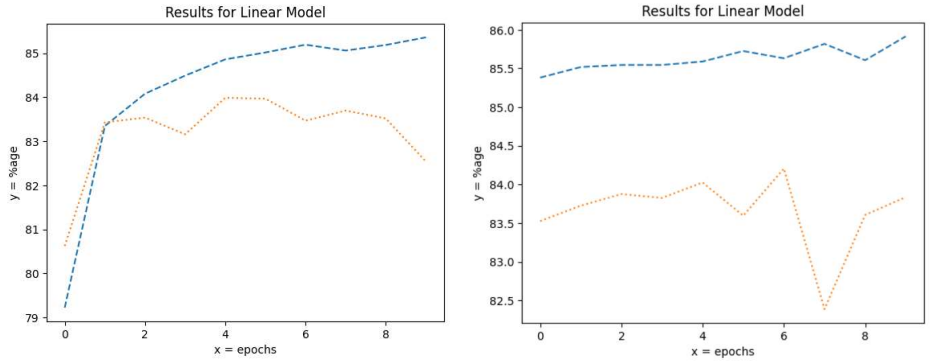
## Result:

### Random Guess Model:

While it is not shown here, randomly guessing the labels for data (over a large enough dataset), should return a near 10% accuracy. This is because there are 10 classes and pictures which are about evenly divided between them – statistically, there is a one in ten chance that random guesses are correct.

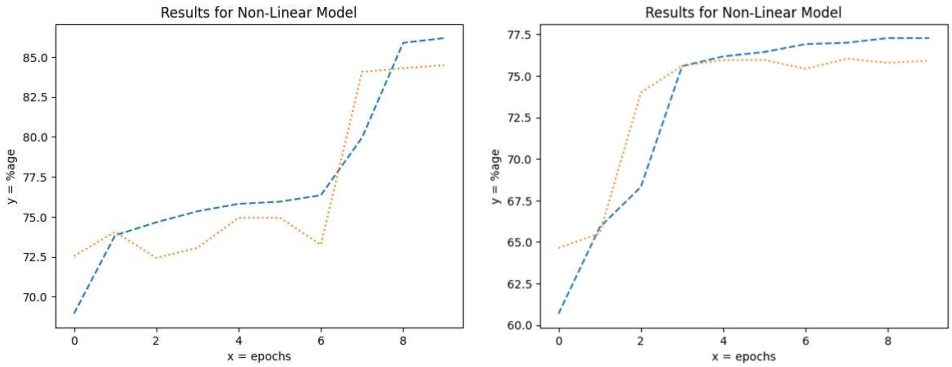
### Linear Model:

The Linear Model after 10 epochs returns between 79% and 89-90% accuracy on testing, and between 80-81% and 84% accuracy on the testing dataset. There is a lot of variation in these recordings and in some cases, the peak of the testing dataset accuracy comes near the 5th or 6th epoch. This indicates some type of flaw in the model and that these peaks are coming more from randomness than actual improvement.

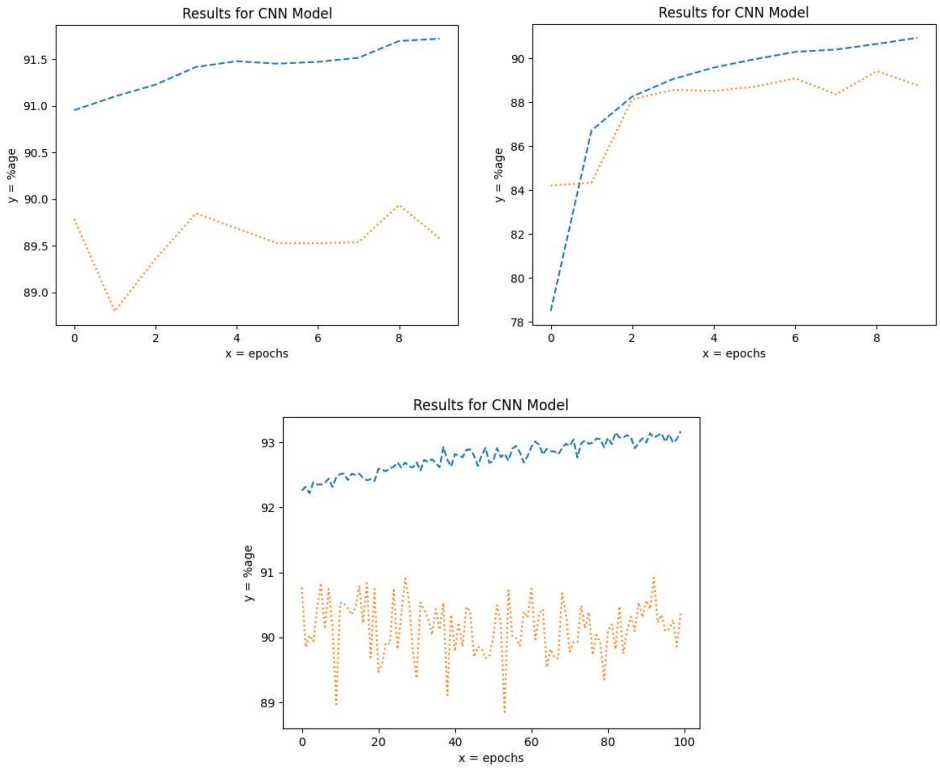



### ReLU Model:

The addition of a ReLU function layer seems to add some consistency to the model (perhaps removing more of the random element). In this case, none of the accuracy graphs show any significant drop in the accuracy with more epochs. There is also a much lower gap between the training graph and the testing graph – this may be due to randomness, but the reproducibility of this trait indicates that it is because the addition of the ReLU layer caused the model to be better at making predictions and at learning. This does not, however, show a significant increase in the accuracy at 10 epochs.



## CNN Model:




The addition of the Convolution and Pooling layers to replicate the tiny-VGG architecture seems to increase the accuracy of the training predictions to between 78% and 93% (usually actually starting ~90-91%), and that of the testing predictions between 84% and 91%. However, this architecture reintroduces a gap between training and testing accuracy graphs – in some cases, going up to 4%. However, in other cases, it also gets very low, which indicates a large amount of randomness on top of a well-trained, well-functioning model.

# Bibliography

[1] J. Wang, R. Turko, O. Shaikh, N. Das, H. Park, F. Hohman, P. Chau and M. K. Kahng, "https://poloclub.github.io/cnn-explainer/#article-relu," [Online]. Available: https://poloclub.github.io/cnn-explainer/#article-relu. [Accessed 22 May 2024].