Assignment 1 Report

Problem Statement

In this assignment we will be addressing the task of *Multinomial Classification* of handwritten digits from the famous MNIST dataset.
The MNIST dataset consists of 60,000 training images and 10,000 test images. Our classes are the digits 0-9.
You are required to build 2 models to solve this problem:

1. A simple using fully connected layers.
2. A model using Convolutional Neural Network (CNN) before applying the simple model you should observe the difference in performance between the 2 models.

Requirements

Building 2 neural network models to be applied to MNIST digit classification.

Simple Model

A neural network consisting of 2 fully connected layers and apply this to the digit classification task, Our network will ultimately output a probability distribution over the 10 digit classes (0-9).

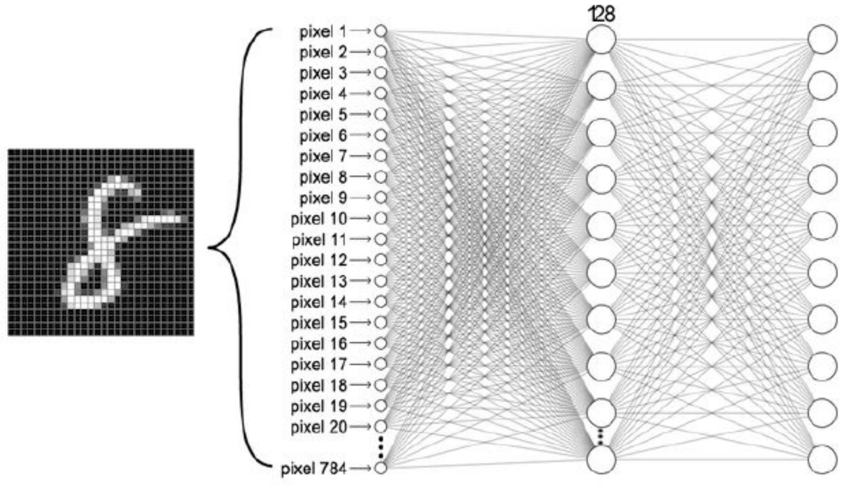


Figure 1: Simple Model

CNN Model

A CNN composed of two convolutional layers and pooling layers, followed by two fully connected layers, and ultimately output a probability distribution over the 10 digit classes (0-9).

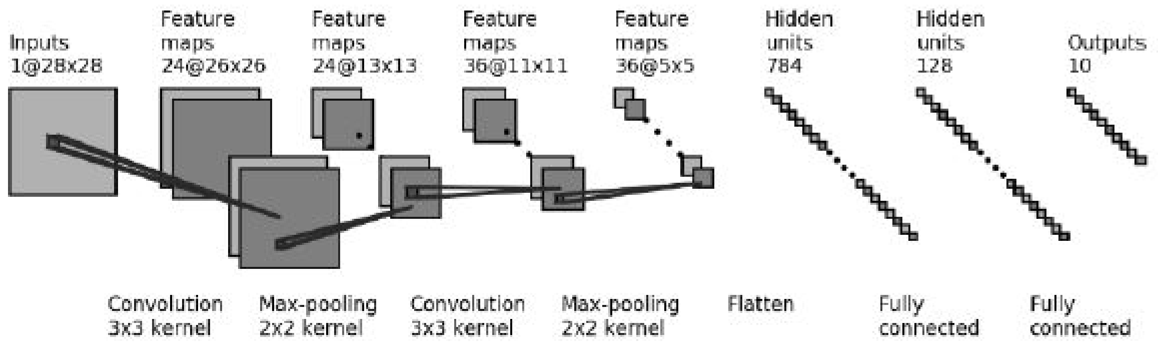


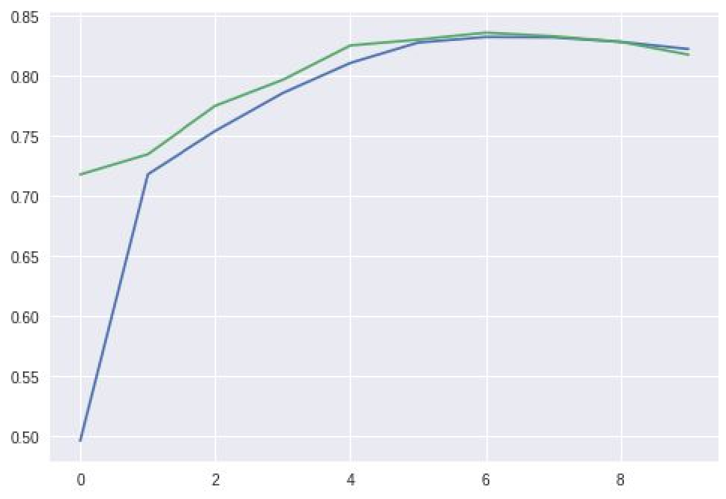
Figure 2: CNN Model

Implementation Results

Simple Model Results

- `loss: 0.2523 - acc: 0.9304 - val_loss: 0.2651 - val_acc: 0.9283`
- `Test Loss: 0.27227048588246106`
  `Test Accuracy 0.9241999983787537`

Simple Model Performance Graph



Hint:

- Green curve: Validation Accuracy
- Blue curve : Training Accuracy
- X-axis: Epochs Number
- Y-axis: Accuracy Value

CNN Model Info & Hyper-Parameters

- Training Running time: about 1 min.
- Optimizer: Adamax
- Loss Function: 'sparse_categorical_crossentropy'
- Optimizer Learning rate = 0.002
- Optimizer beta_1 = 0.999
- Optimizer beta_2 = 0.999
- Model Training Batch Size = 160 *“Made a significant better results”*
- Model Training Epochs = 10

CNN Model Tuning Results

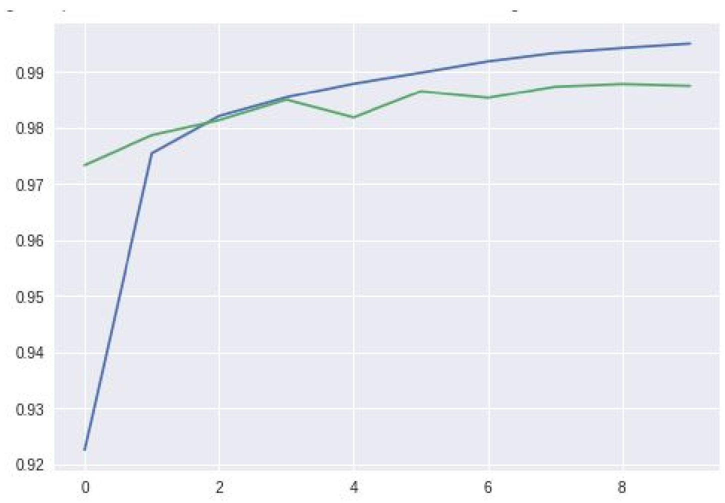
- Choosing **Adamax** as optimizer and tuning the parameter **beta_2** to the value 0.999 made a significant difference in the results.
- Tuning **Training Batch Size** to the value 160 made the results much more better.
- Choosing **sparse categorical crossentropy** as a loss function was the best .

CNNModel Results

-
  ```
  loss: 0.2523 - acc: 0.9304 - val_loss: 0.2651 - val_acc: 0.9283
  ```

-
  ```
  Test Loss: 0.27227048588246106
  Test Accuracy 0.9241999983787537
  ```

Cnn Model Performance Graph



Hint:

- Green curve: Validation Accuracy
- Blue curve : Training Accuracy
- X-axis: Epochs Number
- Y-axis: Accuracy Value

CNN Model Info & Hyper-Parameters

- Training Running time: about 10 mins.
- Optimizer: Adamax
- Loss Function: 'sparse_categorical_crossentropy'
- Optimizer Learning rate = 0.002
- Optimizer beta_1 = 0.999
- Optimizer beta_2 = 0.999
- Model Training Batch Size = 160 *“Made a significant better results”*
- Model Training Epochs = 10

CNN Model Tuning Results

- Choosing **Adamax** as optimizer and tuning the parameter **beta_2** to the value 0.999 made a significant difference in the results.
- Tuning **Training Batch Size** to the value 160 made the results much more better.
- Choosing **sparse categorical crossentropy** as a loss function was the best .

CNN Model Results using Relu

- `loss: 0.0160 - acc: 0.9950 - val_loss: 0.0466 - val_acc: 0.9875`

-
  ```
  Test Loss: 0.044627650970781904
  Test Accuracy 0.9886000156402588
  ```

CNN Model Results using tanh

- `loss: 0.0124 - acc: 0.9957 - val_loss: 0.0619 - val_acc: 0.9867`

-
  ```
  Test Loss: 0.05630356174340413
  Test Accuracy 0.9860000014305115
  ```