Using Convolutional Networks on MNIST Dataset

By: Ethan McCartney & Michael Chen

Contents

1.0 Background

1.1 Introduction

The MNIST dataset is one of the most well known datasets, and is considered the hello world of deep learning. The dataset consists of thousands of handwritten digits represented by 28x28 pixel, black and white images. An example of the dataset can be seen below in figure 1.0.0. Typically, the goal is to design a model that is able to take any image of a 28x28 handwritten digit and correctly classify it as any 0-9 digit. This is also our goal, to tackle this problem, we implemented a 2d convolutional neural network utilizing tensorflow as the backend.

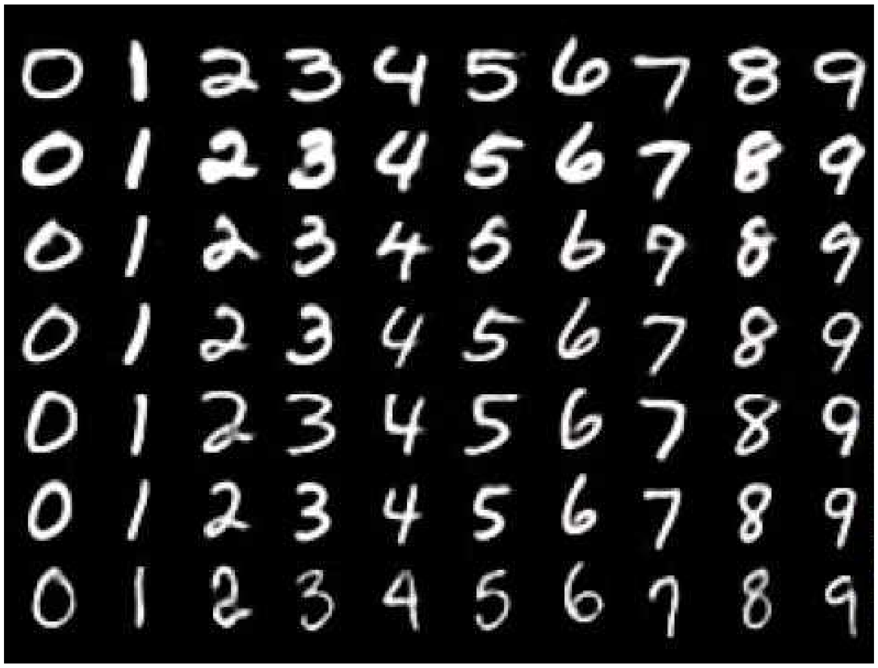

Figure 1.0.0 - MNIST dataset samples

1.2 Question Statement

Considering all of the background information, our goal is to complete the following tasks.

- Design and implement a deep learning model that is able to classify the MNIST dataset with reasonable accuracy.
- Show that our model has high predictive strength on unseen data.
- Test different methods to increase predictive performance such as data augmentation and dropout.

2.0 Methods

2.1 Model

The model we will be using is a convolutional deep neural network. The final network consists of stacked layers of convolutional layers, and pooling layers. The convolutional layers are what is use to extract "features" that the network will learn, and these features are extracted from the image into several feature maps, that are then passes into a pooling layer, which is used to down sample the feature map by essentially taking the max pixel channel value within a given area, and using that value as a single pixel. This process is repeated an arbitrary amount of times and is one of the hyperparameters of our model, until the final output layer is reached, where outputs are flattened, and sent through a standard fully connected feedforward network that outputs the final image classification output as a probability distribution array.

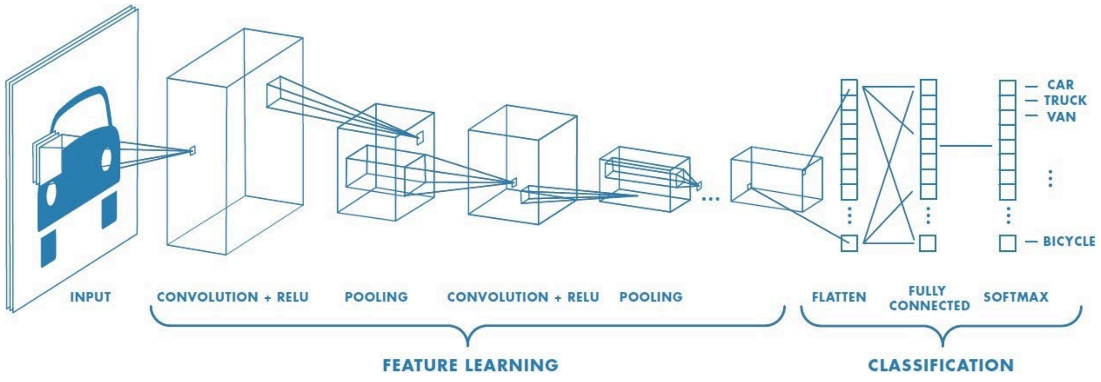

Figure 2.1.0 - Convolutional Neural Network Architecture (Saha, 2018)

2.2 Convolutional Layer

The convolutional layer's purpose is to extract features out of the provided image/input and create feature maps. The convolutional layer does this by having a matrix of learnable parameters called the kernel that goes across the image row by row, and takes the dot product between one channel of the input image and the kernel matrix, which produces a resultant matrix that represents the feature extraction of that section, an illustration of this can be seen in figure 2.2.0. Once the kernel has processed the entire image, all of these resultant matrices are concatenated together to produce the resultant feature map. As for the implementation for this layer, we utilized the tensorflow 2d convolutional layer `` `tf.keras.layers.Conv2D()` ``.

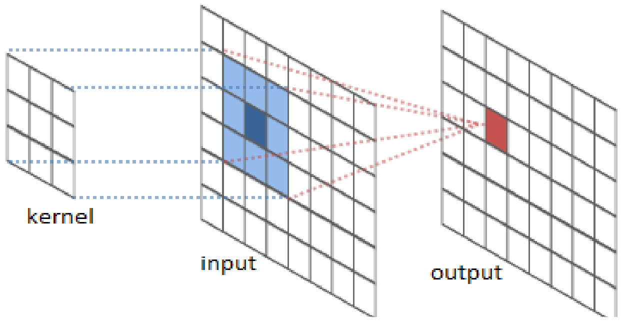

Figure 2.2.0: Convolutional Layer kernel Example (Colah, 2014)

2.3 Pooling Layer

Once the input has run through the convolutional layer, the output of the convolutional layer has produced several feature maps that are effectively several images that represent different feature extractions of the input to the convolutional layer. The purpose of the pooling layer is to downsample these feature maps to a smaller size. The pooling layer accomplishes this in a similar way as the convolutional layer does, in that it also views the input area by area and takes the max pixel value in the region, and uses the max value as the output pixel value, where in the output each $n \times n$ area in the original image is represented by a single pixel, thereby reducing the size by n^2 for each feature map (where n represents the length/width of the area of the input that gets reduced to one pixel). An illustration of this can be seen in figure 2.3.1. As for the implementation for this layer, we utilized the tensorflow max pooling layer `` `tf.keras.layers.MaxPooling2D((2, 2))` ``.

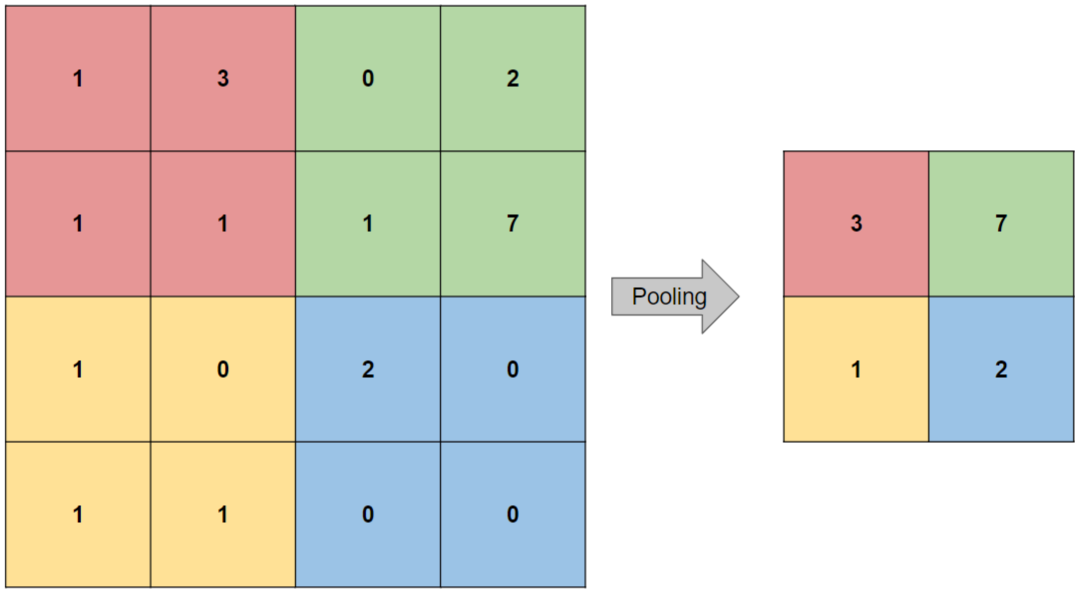

Figure 2.3.1: Pooling Down sampling Example ($n = 2$)

3.0 Results

After training the completed model for 10 epochs over the MNIST data set and a 80/20 test/validation data split, our model started at an accuracy of 89.85% and after training it achieved a 97.46% on test data accuracy showing very strong generative performance. And we also reached 97.41% validation accuracy showing very strong predictive performance. The training accuracy by epoch can be seen in figure 3.0.1 below.

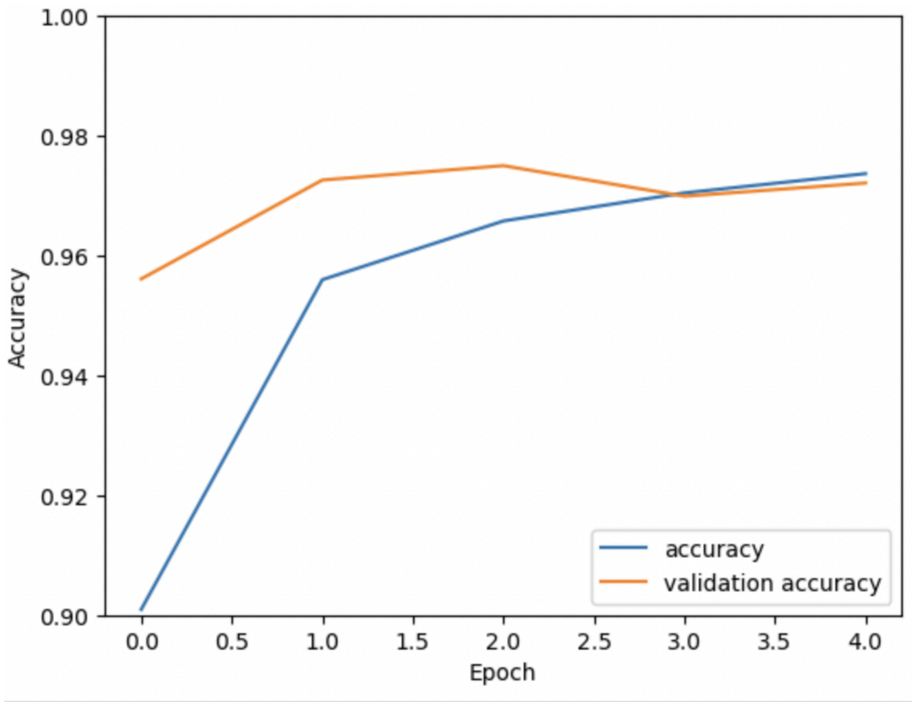

Figure 3.0.1: Training accuracy vs. epoch

Following this, we attempted to try to increase the predictive performance and validation accuracy by making the model more robust by utilizing dropout and data augmentation methods. First we will deploy a gaussian blur to make the model more robust. Advantages of using gaussian blur are that it removes low intensity edges and smooths out the training image to allow for greater predictive accuracy for images it has never seen before. This is deployed in the preprocessing stage along with other changes. We also introduce slight tilt, horizontally and vertically shifts in the images as part of the data augmentation segment. By changing these parameters of the image we can allow the model to be able to handle images it has never seen before.

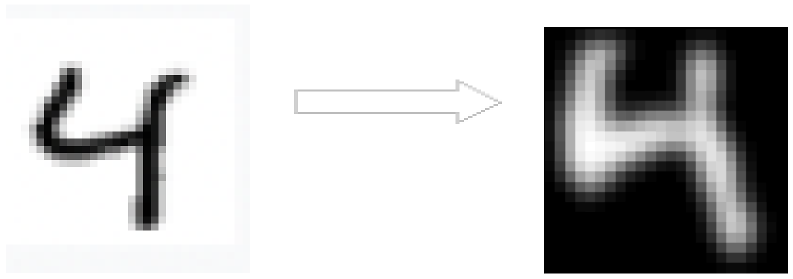

Figure 3.0.2: Data augmentation sample

After deploying the preprocessed dataset with gaussian blur and data augmentation we now have even better performance, we have reached 98.14% accuracy with data augmentation and a dropout layer. Not only is it better in terms of performance, it will also be more robust in handling data the model has never seen before.

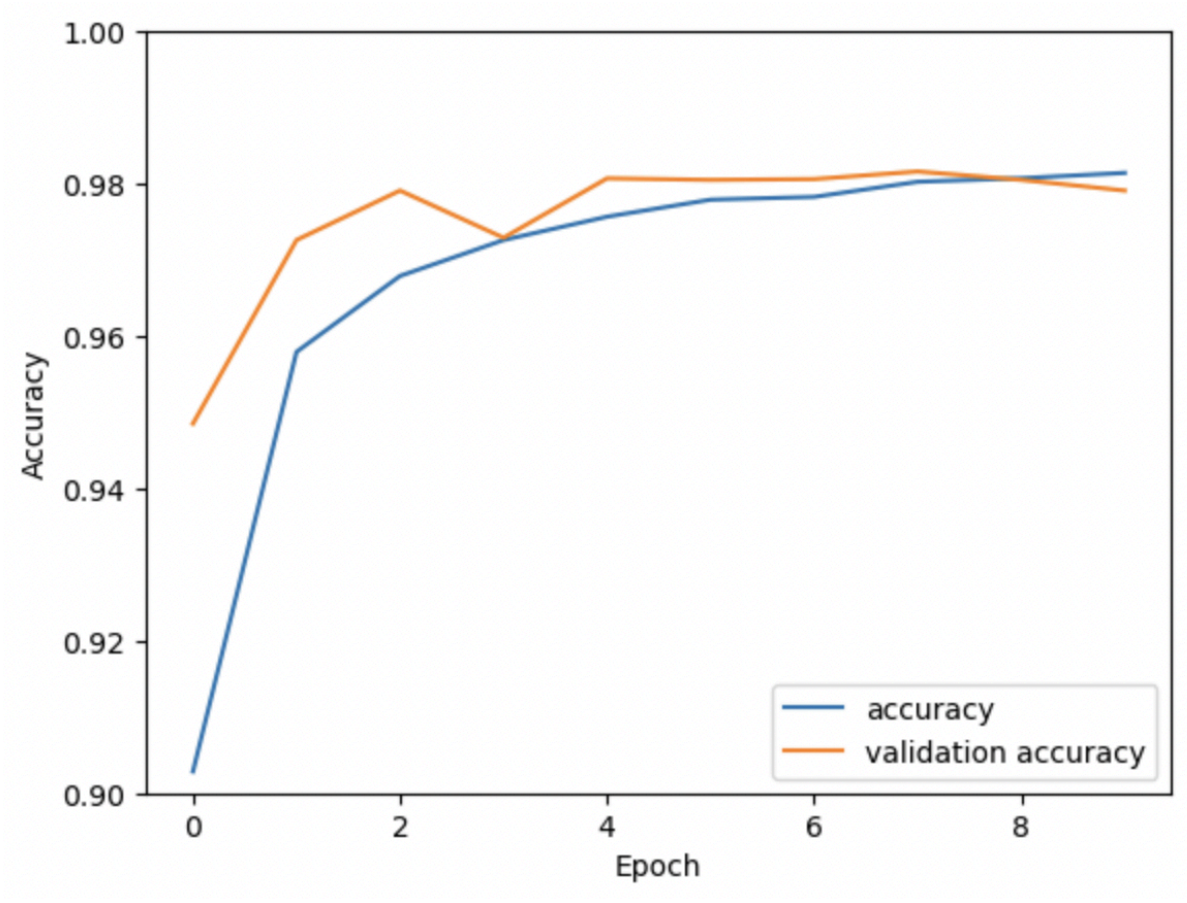

Figure 3.0.3: Training accuracy vs. epoch with dropout

4.0 Final Notes

In this experiment, we have explored how to use convolutional neural networks to identify handwritten numbers. We used multiple convolution and pooling layers to extract features and create smaller feature maps to get the most crucial information. In order to make the model more robust/handle new data we added data augmentation techniques as well as dropout layers. Through gaussian noise, shifting the images and dropout layer we were able to achieve a model that can handle different types of hand-written numbers effectively. Our model was able to correctly predict the hand-written number with 98% accuracy after data augmentation. We broke up the train/test split into 80/20 to allow the model adequate number of test data. This is a part of the changes to ensure the model is robust and can handle different data. In conclusion we were able to fulfill our goal of constructing a CNN to accurately predict hand-written numbers. In the future we could possibly change to add more convolution layers or modify the activation functions, or even potentially change the structure of our model.

5.0 Citations

"Understanding Convolutions." Understanding Convolutions - Colah's Blog, colah.github.io/posts/2014-07-Understanding-Convolutions/. Accessed 24 Apr. 2024.

Saha, Sumit. "A Comprehensive Guide to Convolutional Neural Networks - the Eli5 Way." Medium, Towards Data Science, 16 Nov. 2022, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

"What Are Convolutional Neural Networks?" IBM, IBM, www.ibm.com/topics/convolutional-neural-networks. Accessed 24 Apr. 2024.

Python | Image blurring using OpenCV. (2019, April 17). GeeksforGeeks. https://www.geeksforgeeks.org/python-image-blurring-using-opencv/#