Kaggle Competition 1 IFT 6390

Lucia Eve Berger, team name: Lucia Eve Berger

November 6, 2022

1 Introduction

The problem in this competition was developing a MNIST calculator. We were tasked with adding the two handwritten digits. We can understand the problem either as

- Jointly classifying the image together (28, 56) and finding their sum.
- Independently classifying the digits (Image1 as Feature 1 (28,28) , Image2 as Feature2 (28,28)) and then summing them to a final value.
- Regression problem based the pixel values (estimating their range), and estimating the relationship between the pixel values.

For modelling, I took the joint classifying approach. I used Logistic Regression, Support Vector Machine (SVM) and CNN modelling. To beat the first baseline, I used a Logistic Regression with feature normalisation and learning-rate variation.

For the other baselines, I used a SVM and then moved a deep learning technique of CNN modelling. With the SVM, I experimented with kernels and hyperparameters. With tweaking, I reached an acceptable accuracy. I received a 70 % accuracy on testset. With the CNN, I received a 98% on the testset. The high accuracy was achieved by feature design and model tuning.

2 Feature Design

2.0.1 Normalization, Padding

As a first step, I plotted the data to visualize it. The data can be transformed into image form with a np.reshape(56,28). I observed that this was a merging of two images across the x axis. The 56 dimension was two of the images merged together. I scanned the dataframe for missing values or "naan" values. I found this to be a cleaned dataset and did not observe strange characters except for the last column which I removed.

Our full training set is (50,000,56,28). For the linear methods, I saved a lot of compute by normalizing the data from [0, 1]. With this problem, we did not lose accuracy. With the logistic classifier and SVM, I experimented with removing the padding around the images. In other words, I experimented with removing values that were "0" across feature column and dropped these from consideration. For these linear algorithms, these "0" may not provide information and helped marginally.

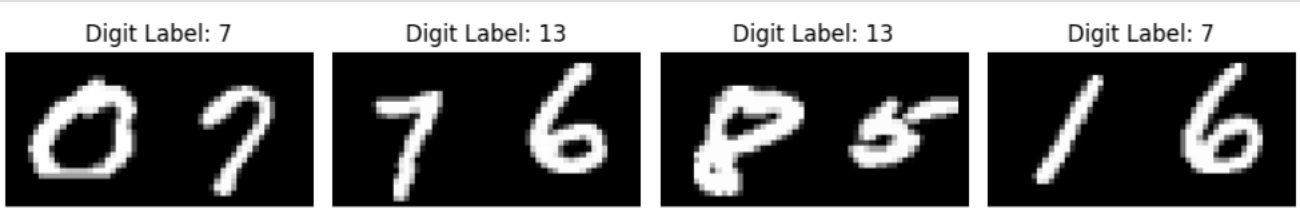

Figure 1: Visualizing our data

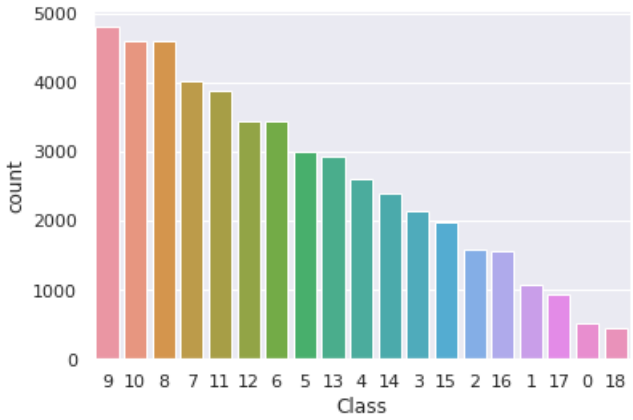

Figure 2: Frequency Count of our Training Labels

2.0.2 Greyscale

I transformed the images to greyscale.For this particular problem, I do not think the shading around the edges of the digits gives us **more** visual information. In more complex recognition problems, the shading or blur would be more necessary. I applied a rudimentary mapping, casting values under (0.5 to the black value, with values over 0.5 to white). Visually, I looked at the images following and I could not discern a difference.

2.0.3 Our Training Labels

Given the nature of this problem, we have a highly imbalanced class distribution. We can visualize the frequency distribution of the digits. As Figure 2 shows, we can see an imbalance of the digits with the two digit numbers generally being represented more often.For our SVM training, we can specified this is an imbalanced problem.

3 Algorithms

3.1 Logistic Regression

We can modify our traditional logistic regression algorithm to fit the multi-classifier problem. The main objective of our model is to find out the weights of the 19 classes. With the help of a hypothesis function, which is also called as sigmoid function, we calculate probabilities by giving some input data (known variables). Based on that, the model does the analysis to predict required classification. Learning rate can be tweaked to determine how we update the parameters of the weights. I initialized the weights with random values.

```
RMSprop
class            input    output     units    params     activ     label
=================================================================
Conv2D              28        26                1280      relu     Corel
MaxPooling2D        26        13                   0                Ma
Dropout             13        13                   0                Dr
Conv2D              13        11              295168      relu     Corel
MaxPooling2D        11         5                   0                Ma
Dropout              5         5                   0                Dr
Conv2D               5         3              590080      relu     Corel
MaxPooling2D         3         1                   0                Ma
Flatten              1      1280                   0                Fl
Dense             1280       256       256    327936      relu  De256rel
Dense              256        19        19      4883   softmax   De19sof
labelling this model as RMSprop:Corel|Ma|Dr|Corel|Ma|Dr|Corel|Ma|Fl|De256rel|De19sof
```

Figure 3: CNN Model Architecture

3.2 Support Vector Machine

Another model I used was a Support Vector Machine (SVM). A SVM is training algorithm that builds a model that assigns new examples to categorise and builds these decision-boundaries. As we've seen in class, the algorithm finds a hyperplane which maximizes the greatest possible margin between the hyperplane and any training set datapoint. In experimenting with this model, I tried a linear and a non-linear kernel as described in the Cross Validation Section.

3.3 Convolutional Neural Network

A model of interest for this task was a Convolutional Neural Network. This model fit well as it captures the internal representation of a two-dimensional image well. We have distinct control of the model via the various architectures. At first, I chose three convolutional layers with a final fully connected Dense Layer. The output function of the output layer was a softmax function with nineteen potential outputs. In Figure 3, we observe a depiction of the final model architecture. Note I experimented with four different architectures but will focus on my final architecture.

4 Methodology

4.1 Creating Validation Sets

To assess the generality of the learning, I divided the training set into a training and validation set. The validation set is our holdout set that the training algorithm does not see. This helps us assess quality of predictions on our validation set. I chose to hold-out 30% of the dataset. It gives a view of performance on the testset.

$$validation = StratifiedShuffleSplit(nsplits = 1, testsize = 0.30, randomstate = 46)$$

4.2 Cross-Validation

To fit this problem, I took 50%– 80% of the training data (statistical sample) to get a benchmark prediction metric before we do any extensive preprocessing of the data. I ran into processing limitations with the full dataset. As I was unfamiliar with some of the hyperparameters, I used a sklearn as a reference. I used a list of possible values to find near-optimal values for the

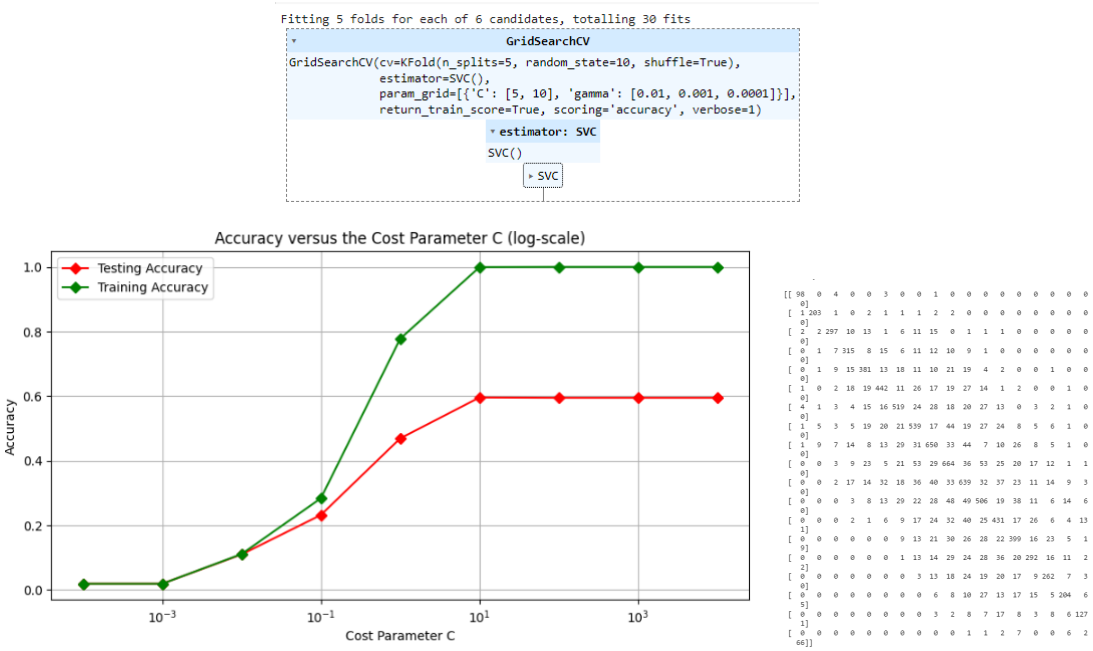

Figure 4: SVM Cross Validation with Confusion Matrix

hyperparameters of a machine learning model. I used confusion matrices to see where the model was making mistakes during this cross-validation.

When using the SVM method, I applied grid search to test the kernel and cost. In particular, I looked at *which kernel* we should use. I tested across a linear or non-linear kernel. I found higher performance with non-linear, rbf kernels.

I also conducted a search on cost (C). In this context, cost defines the weight of how much samples inside the margin contribute to the overall error. I found an optimum at 10, as shown in Figure 4.

The best observed I observed was gamma="scale", cost = 10, class weight = "balanced" and kernel: rbf. These were determined by *searching* the hyperparameter space for these optimums. By setting the class weight to balanced, we allow the algorithm (sklearn) to find the imbalance and automatically set it. I hypothesize these hyperparameters suit as we have many classes. We may need a hard margin in differentiating them. We may struggle to get over 80% as the SVM may be less effective on noisier datasets with overlapping classes.

4.3 Fighting CNN Overfit!

As I trained the model, originally, I encountered overfit. In other words, the model was performing well on the training set but not generalizing behaviour to the validation set. In the CNN architecture, I redesigned my architecture with the following which I explain in my results section.

- Added several dropout layers, specifically between the Convolutional Layers
- Added early stopping
- Increased the sizes of my filters

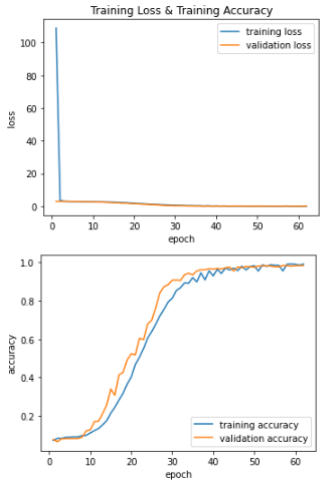

Figure 5: Training of the Tuned CNN

- Adjusted learning rates

5 Results

As we observe on the table and graph, the tuned CNN provided us the best results. With this model, we beat all three baselines. We also can see a healthy training and validation loss. While I set the model to train for 100 epochs, with early stopping, the training concluded at approximately 60 epochs where the validation loss was not decreasing.

Results			
Method	Hyperparameters	Val. Accuracy	Kaggle Test Accuracy
Logisitic Regression	*Batch Size:10, 1D input, fixed learning rate, Epochs: 20.*	0.26	0.22
SVM	*Linear Kernel*	0.67	0.6897
SVM Tuned	*RBF (non-linear), Cost=10, Gamma=scaled, Classes Imbalanced.*	0.724	0.725
CNN Untuned	*3 layers, varied learning rate, Epochs: 100.*	0.9702	Did not run here
CNN Tuned	*3 layers, varied learning rate, dropout, early-stopping, larger filters, Epochs: 60.*	0.989	0.981

5.1 Discussion of CNN Results

I experimented with the structure of the CNN. The dropout layers discouraged memorization of data by introducing increased randomness. In our CNN architecture, these layers randomly disable neurons and their corresponding connections. I found this helped a lot (reducing validation accuracy by 2%).

As we learned in class, if we train the model for too many epochs, we might saturate our learning. Adding early stopping allowed the validation loss be the indicator of when we should stop rather than arbitrary stopping point. As my model was not very deep (only three convolutional layers), a hyper-parameter I tweaked was the filter size. Specifically, I increased the filter size deeper (2nd, 3rd layer) inside of the network. I did this to capture more complex patterns (edges, turns) in the digits. There were a larger combinations of patterns to capture. This was an effort to to capture as many combinations as possible (i.e for more complex digits like 8, or 3).

For the optimising function, I selected the Root Mean Squared Propagation. I paired this with a variable learning rate, scheduled to decrease. Varying the learning rate allowed the model to learn move slowly and then faster with each step size at each iteration while moving toward a minimum of a loss function. As we saw in class, updating this value changes the model in response to the estimated error each time the model weights are updated. It is difficult to tune this, so using a shifting learning rate (larger to smaller), allowed us more flexibility over the epochs.

5.2 Examining the Model's Mistakes

While the CNN performed well overall, there were some errors the model made on the validation dataset. In Figure 6, we can see the model has some problems with the curves on 6,4, 9 and 7s. In the last example, we can see an incomplete digit (6).

6 Discussion:

While I was satisfied with reaching the 98 % accuracy on our testset, I would like to propose some improvements to my work.

- **Feature Design** Using Logistic Regression to classify the features independently. I could have harnessed a simpler classifier had I split the input data up into two features. See Figure 7.
- **Data Augmentation:** Use more robust data (transformations of data where model makes mistakes (8, or 3))
- **Harness a pretrained model** (Large CNN or other) and do model tuning or adaptation with our training data.
- **Increase Cross Validation** I would like to try other modelling techniques (k-folds).

7 Statement of Contribution

I hereby state that all the work presented in this report is that of the author.

8 References:

- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://medium.com/analytics-vidhya/logistic-regression-from-scratch-multi-classification-with-onevsall-

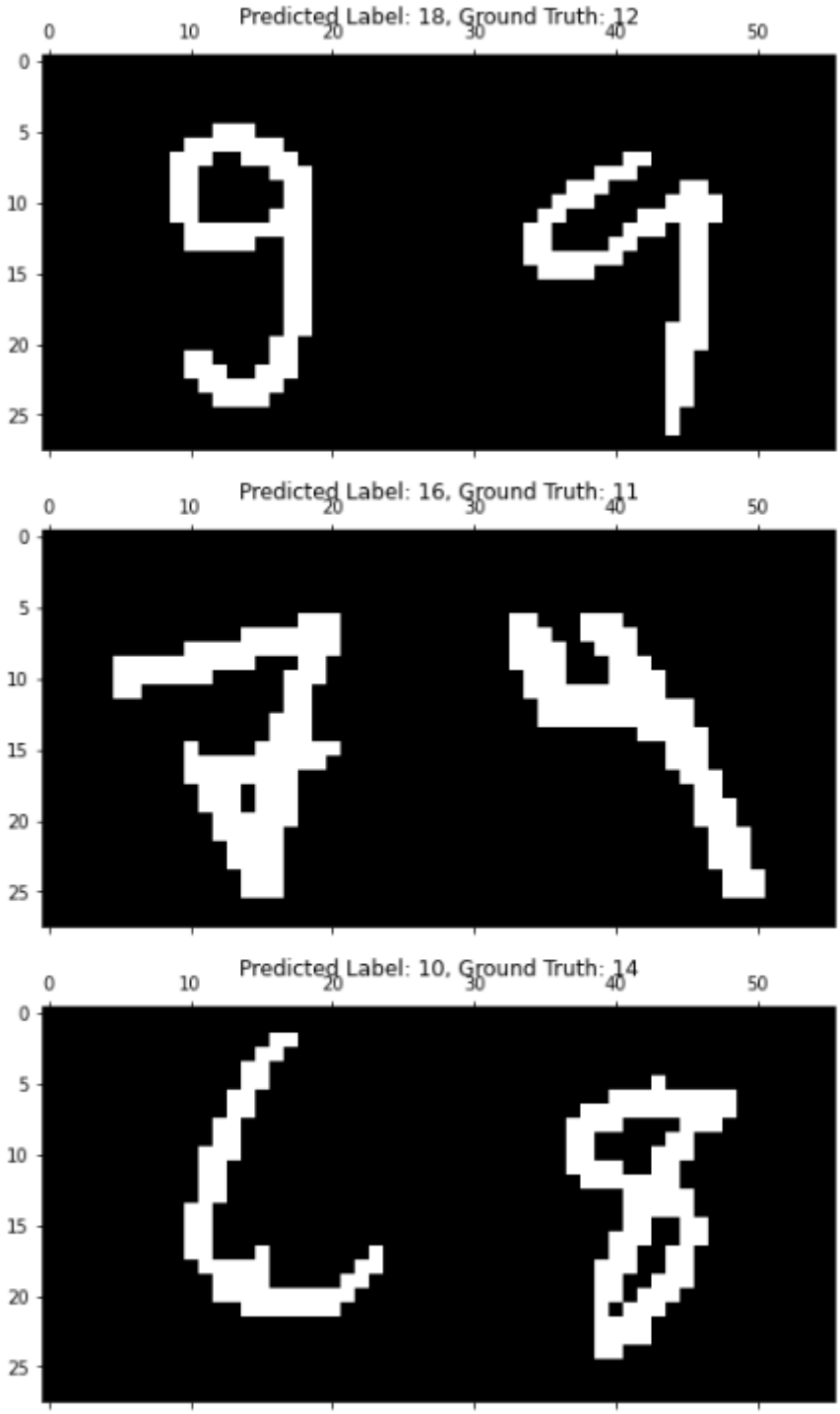

Figure 6: Looking at Some of the Model Mistakes

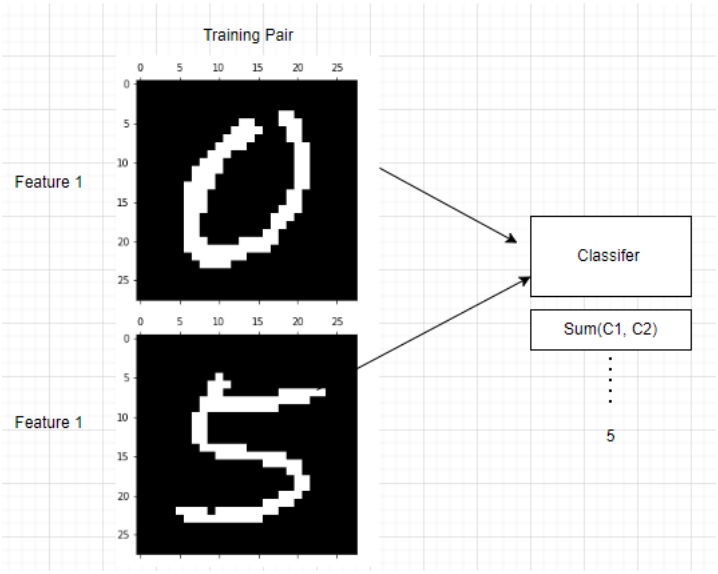

Figure 7: Improved Feature Design

- https://github.com/yawen-d/Logistic-Regression-on-MNIST-with-NumPy-from-Scratch
- https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
- https://stats.stackexchange.com/questions/225409/what-does-the-cost-c-parameter-mean-in-svm
- https://keras.io/api/optimizers/rmsprop/
- https://medium.com/analytics-vidhya/logistic-regression-from-scratch-multi-classification-with-onevsall-d5c2acf0c37c