Find the "Largest" Digit: Comp 551-001 Project #4

Team name: Epochalypse
Patrick Nadeem Ward patrick.ward@mail.mcgill.ca 260629509
Daoud Piracha daoud.piracha@mail.mcgill.ca 260732867
Russell Wang russell.wang@mail.mcgill.ca 260517682

I. Introduction

Image classification is a popular area of research in the domain of Machine Learning. A standard entry point into this field is classification of handwritten digits using the MNIST dataset (LeCun et al. 1999).

Here, we tackle a variation of the MNIST dataset where the goal is now to classify the largest digit in an image containing multiple digits originating from the MNIST dataset. These digits have been randomly rotated and resized and stuck onto a randomly generated background.

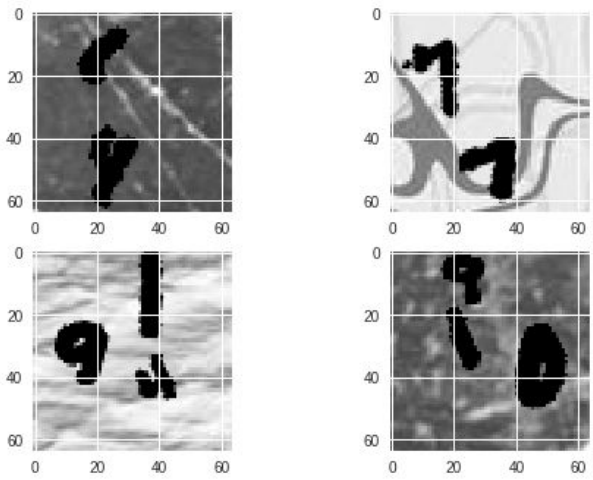

Figure 1: examples of the first four images from the dataset described above.

Since classical deep learning methods are largely successful on the MNIST dataset (Kussul, Ernst; Tatiana Baidyk, 2004) (Cires¸an, Dan; Ueli Meier; Jürgen Schmidhuber 2012), our approach involves trying to transform the data back, with minimal loss of information, to the original MNIST dataset. In doing so we would, theoretically, be recreating images from the original dataset (minus some noise due to the respective transformations). This allows us to pick the largest digit, since the digits are just rescaled versions of digits within MNIST. This reduces the problem from Multiple Object Classification, to simply handwritten digit recognition.

After reducing the problem, a convolutional neural network proved optimal since it exploits the structure of images and built representations for edges and higher level features from the pixel values. They have been shown to be highly successful in image classification applications (Cires¸an, Dan; Ueli Meier; Jürgen Schmidhuber 2012). This is compared to baseline models such as SVM and a multi-layer feed-forward neural network. A bagging approach is then implemented to boost results.

To evaluate our models we split our training data to generate our own training, validation and testing data. We perform hyper parameter tuning on validation set using 5-fold cross validation and report results on a held out test set. We further report our model's Kaggle accuracy score, evealuated on the provided test dataset. As expected, our convolutional neural network performed best with results around 94% when compared to our baseline support vector machine

classifier as well as our feed-forward neural network.

II. Feature Design

To reduce the problem to one closely resembling that of MNIST, we first had to do some preprocessing of the raw images. The goal was, similarly to MNIST, to feed to our model a single digit and have the model classify on that. That would mean that our preprocessing had to remove the extra complexity associated with this problem. To do so, the raw data required several transformations that would become our pipeline.

The first of the these steps involved removing the background noise in the images. Having noticed that our digits of interest are solid black, we used a hard threshold that would convert all image pixels of value less than 255 (black) to zero. This creates very "harsh" contours for the digits (**Figure 3:** the digits don't have a smooth contour) but we assumed the loss of information was acceptable. This removed the background and created a binary image.

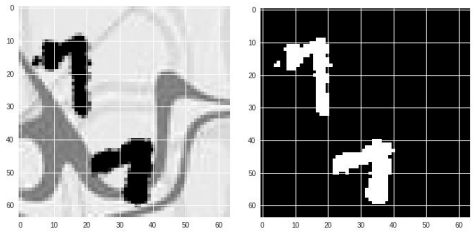

Figure 2: 3rd image in the dataset pre and post thresholding. Note: we flipped the images to have white on black background in order to easily integrated OpenCV functions.

Now, with such clear distinctions within the images, it is easy to extract the digits by using a rectangular bounding box approach. Each box was a mapping of the contour of an object, OpenCV provided simple functions for doing so. This allowed us to isolate upto 3 subimages from each image in the original dataset. Each sub image would contain one digit.

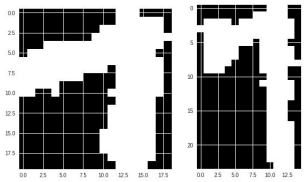

Figure 3: the two digits extracted from Figure 2.

Notice that the subimages are of different dimensions, we needed to standardize the dimensions for use as input into a model. Initially, we rescaled them by preserving the aspect ratio to 28x28 (exactly the same size as the MNIST dataset). However, further examination of the MNIST dataset led us to use a rescaling to a 20x20 pixel image and a further padding of 4 pixels on each side of the image. This is exactly the way the original MNIST dataset is structured (LeCun et al. 1999).

The above steps allowed us to process the raw images into a set of standardized subimages of digits.

The next set of steps involves finding a way of choosing the correct subimage within the above mentioned set. We considered 3 approaches, largest by area of bounding box (i.e. the total cumulative number of pixels), largest by axes and finally largest by pixel count.

The largest by axes approach chooses the "largest" subimage by comparing the larger axis of each of the subimages.

Finally we, implemented a comparison of the pixel area occupied by each digit in its subimage. Assuming that the digit after our rescaling was approximately equivalent to that of an MNIST digit, we proceeded to examine the ratio of the pixel count pre rescaling, to that of the digit post. This allowed us to to find the largest digit by scale factor.

As a result of rescaling however, we introduced pixel of different grey scale values (not just 0 or 255) to our images. Since each of these low intensity pixels contained information about the respective digit's pixel area, we decided not to

ignore this information. We countered this problem by dividing all pixel values by 255 to normalize and then took a weighted sum of pixel values to calculate the true area occupied by the digits in the subimage.

We also noted that a common problem with bounding box approaches arises when two digits touch. This causes an abnormally large bounding box which is unable to isolate a single digit. We dealt with this problem by removing all 178 such images from the training dataset of 50000 which amounts to 0.36% of the data, deemed a bearable loss.

Since the maximum possible length of any of the rescale images could be 20 and the highest scaling factor is 1.2, accounting for rotation, we get: $20 * 1.2 * \sqrt{2} = 36\ px$ as largest acceptable axis length (sqrt(2) is the diagonal length of a pixel). We therefore removed all bounding boxes with either axes larger than 36 px.

All of the above preprocessing steps formed a pipeline that allowed us to significantly reduced the complexity of our problem. We are left with a newly created dataset that consists of only one 28x28 pixel digit and a label, exactly the same format as the MNIST dataset.

III. Algorithms

The learning algorithms used were a support vector machine model, which was used as a baseline, a multi-layer feed-forward neural network, transfer learning using CNNs and several variations of a vanilla convolutional neural network (CNN). We also tried to implement an ensemble of 3 different convolutional neural network architectures to increase test performance using "committee" (majority) voting. This is a popular industry practice that increases prediction power (Rokach, L. 2010), compared to a single classifier.

We also experimented with a CNN model that stops prematurely (right before the softmax layer) and then we would use a SVM to classify the abstraction that the CNN created. The CNN's was used for feature generation, and the SVM for final classification. This was shown to increase accuracy and results, compared to feeding raw images to the SVM (Athiwaratkun and Kang 2015).

for the CNN model, we tried several architectures, starting with a vanilla one-hidden-layer convolutional neural network and subsequently adding layers and complexity to adjust results. We found that a larger kernel size and increased stride with maxpool improved results. We believe this may allow the first layers to form better low level representations of the digits. This was to account for thicker digits and noise in transformations compared to MNIST.

We then implemented grid search over the hyperparameters which were thought to be relevant: the number of layers, the number of neurons per layer, the number of epochs, the dropout rate, the backpropagation optimization function as well as the activation function. However, grid search proved to be too computationally expensive so the task of choosing hyperparameters was reduced to a greedy approach to optimize for each hyperparameter while keeping the others fixed (Bergstra et al. 2011).

For the SVM algorithm the hyperparameters tested were the kernel (linear vs radial basis function 'rbf').

We also attempted Transfer Learning (Bengio, 2012) to solve the given problem. We trained the model first on the MNIST dataset, then after freezing some layers we trained the model on our own data. We tuned between 5 and 10 epochs for each dataset. We froze the feature generating lower convolutional layers and kept the fully connected layers trainable. We also tried approaches with the lowest one or two layers trainable but kept the rest of the the feature generating layers frozen. Both

approaches gave comparable results to a vanilla CNN on the dataset.

IV. Methodology

The convolutional neural networks were implemented using Keras with a Tensorflow as the backend. Scikit-learn wrapper functions were used to perform hyperparameter grid search optimization and cross validation on our model.

To regularize the models, we assumed the data to be independent and identically distributed. This allowed us to implement 5-fold cross validation on on a split of our training data.

To do hyperparameter tuning, we split the data into training (80%) and validation (20%), with the kaggle competition testing data used as our test set.

The raw dataset was pipelined through different variations of preprocessing datasets. Variations implemented different strategies for picking the largest digit, were used for evaluation.

Finally, in an attempt to increase the testing accuracy for the competition, an ensemble, majority voting, approach was used to increase testing performance.

V. Results

A summary of our results compared with other methods are listed in **Table 1**.

The best performing convolutional neural network had 2 convolutional layers of 32 and 64 neurons respectively with a kernel size of 3x3 and 2x2 respectively. It also had one fully connected layer with 128 neurons and was trained for 16 epochs. the batch size was 64 and the dropout rate was set to 0.3 after each hidden layer, each of which uses ReLU activation functions. The softmax activation function was used for the output. The Adadelta backpropagation optimizer was used, with a binary cross entropy loss function.

Model	Pre	Raw
CNN	93.0	11.0
Feed-Forward NN	-	-
CNN Ensemble	94.0	-
Pre-trained MNIST CNN	65.2	-
Vanilla SVM	56.7	44.2
Random	10.0	10.0

Table 1: Results of our CNN models against the baseline methods evaluated on left out test set after cross validation and hyperparameter tuning. "Pre" refers to the data after preprocessing and "Raw" refers to the raw data as provided.

We notice that there is a clear overall trend which shows that scores differ drastically between the raw vs preprocessed datasets. This would indicate that the models on their own can't handle the additional complexity of the problem of "choosing" the largest image and then classifying versus just being given an image to classify with the preprocessed data. This helps to justify our preprocessing and feature design mentioned previously. By simplifying the required task, classification benefits immensely.

VI. Discussion

Having initially preprocessed the raw images into images that match exactly with the dimensions and structure as those of the MNIST dataset, we thought that using a model trained on MNIST would therefore give performance results similar to those obtain on actual the MNIST data. However, a pre-trained MNIST model performance was substantially worse with an accuracy of 65.2%. This, we assumed, to be due to the fact that images are, rotated, and have harsher, extrapolated contours and hence don't make the MNIST dataset perfectly. The intuitive next step was to simply

train a CNN on our own data. However, our reported accuracy was lower when compared to a benchmark such as MNIST. We reported an accuracy of 94.2% whereas accuracies of 99.79% have been achieved on the MNIST dataset (Cires¸an, Dan; Ueli Meier; Jürgen Schmidhuber 2012). We believe this drop in accuracies may have arisen from some assumptions in preprocessing and noise in the data which made it different when compared to MNIST. A point to be noted was that all digits in the given dataset were much thicker than those in MNIST, this caused our classifiers to misclassify for instance a skinny 8 as a 1.

With respect to the different preprocessing methods used, we got higher accuracies with the choosing the largest sub image based on pixel count ratio vs largest bounding box area. The largest by area approach, was insufficient since it didn't really capture the idea of "largest". For example a thick 9 would have more pixels that a thin 1 even though the 1 was obviously larger. Similar problems arose when we considered the digit with the largest axis. Digits occupy a two dimensional area and different areas are scaled differently with respect to the spatial geometry of the digit itself.

To counter this, a largest scaled digit approach was therefore adopted where we now measured the idea of "largest" by directly estimating the pixel area by each digit. However, here we do assume that our rescaled subimage is similar to the original MNIST images used to generate the dataset. While both are apparently similar, we acknowledge that inconsistencies may exist.

Furthermore, since there was only one label per image but multiple digits within an image, manual examination was necessary to give an estimate to how accurately we were choosing the correct digit from within the set of possible choices. We sampled and examined over 200 images and found that we wrongly chose the largest digit, using the scaling factor approach, roughly 3% of the time. Assuming that the dataset is independent and identically distributed, we then generalized this to say to say that our approach only chose the wrong digit 3% of the time. This was deemed acceptable.
We initially thought of first classifying the digit using transfer learning from MNIST weights and then computing scale factor. We hoped to better account for this by comparing to a norm array generated from MNIST. We generated this norm array counting the average pixel counts in the MNIST dataset for each digit. However, we only had labels for the largest digits and would have to solely rely on transfer learning accuracies(~94%) without sufficient validation data. We believe after further optimization and more data, this is an area to be explored in the future.

We also believe there may be room for further improving transfer learning by implementing all practices listed by (Benjio, 2012.) . We would all want to try more optimization algorithms and hyper parameter tuning using additional algorithmic approaches (Bergstra et al. 2011). Finally we would like to attempt transfer learning using deeper networks and weights from VGG ImageNet, InceptionNet.

VII. Statement of Contribution

Both Patrick Nadeem Ward and Daoud Piracha contributed equally with respect quantity of code and the writing of the report. Russell Wang tried to the implement a neural network from scratch. We hereby state that all the work presented in this report is that of the authors.

VIII. References

Kussul, Ernst; Tatiana Baidyk 2004. "*Improved method of handwritten digit recognition tested on MNIST database*."

LeCun et al. (1999): "*The MNIST Dataset Of Handwritten Digits (Images)"*.

LeCun, Yann; Corinna Cortes; Christopher J.C. Burges.(2013) "*MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*".

Cires¸an, Dan; Ueli Meier; Jürgen Schmidhuber (2012). *"Multi-column deep neural networks for image classification"*. 2012 IEEE Conference on Computer Vision and Pattern Recognition.

Rokach, L. (2010). *"Ensemble-based classifiers"*. *Artificial Intelligence Review.*

Athiwaratkun, B.; Kang, K.; (2015). "*Feature Representation in Convolutional Neural Networks"*.

Bergstra, J., Bardenet, R., Bengio, Y., and Kegl, B. (2011). *"Algorithms for hyper-parameter optimization"*.

Bengio, Y.; (2012). "*Deep Learning of Representations for Unsupervised and Transfer Learning."*