Handwritten Digit Recognition App using Neural Networks

Yuriy Sereda

March 29, 2022

1. Problem Identification

1.1 Context

Develop an accurate model for digit recognition and a GUI for digit handwriting and user feedback.

1.2 Criteria for success

The main criterion is the accuracy of handwritten digit recognition. Additional criterion is the ease of use of the GUI for digit handwriting.

1.3 Scope of solution space

Supervised learning methods will be used to handle this multi-label classification problem. We limit ourselves to sequential neural networks (NN), including convolutional ones (CNN). CNNs take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. A major advantage of CNNs is an independence from prior knowledge and human intervention in feature extraction.

1.4 Constraints within solution space

1) Images used for training may reflect handwriting styles that differ from that of a user. This problem will be solved by developing a GUI that allows user to not only handwrite digits but also supply a correct label in case of a misclassification.
2) Special care must be taken to ensure correct dimensions for input data, which will be solved by reshaping the input when necessary. E.g., non-convolutional NN requires a vector as its input, while CNN requires a matrix. Also, GUI window needs to be larger than 28×28 pixels for ease of drawing, so a rescaling of the GUI images will have to be done.
3) Colour map and colour inversion must be matched between MNIST dataset and the user input via GUI.
4) With too short model training (small number of epochs), an under-fitting may take place, while with too long training one may end up with an overfitting. Therefore, some initial search of a reasonable set of hyperparameters may be required.
5) CNN model training takes quite a long time, typically several hours.
6) Most tedious part is handwriting of the digits to assess accuracy and generate extra labelled images for model re-training.

1.5 Data sources

Data used for initial training of the neural networks were taken from the MNIST handwritten digit database. MNIST dataset contains 60,000 training images of handwritten digits from 0 to 9 and 10,000 images for testing. Images have black background and white-grey digits. Digits are represented as a 28×28 matrix where each cell contains grayscale pixel value from 0 (black) to

255 (white). There are 10 classes, one for each digit. A user can provide additional manually labelled images of digits using the GUI that was developed here.

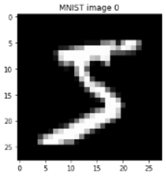
2. Data Wrangling: Collect, organize, define, and clean a relevant dataset

2.1 Load libraries, set plotting styles

Python libraries: numpy, pandas, matplotlib, keras, PIL, tkinter, win32gui.

2.2 Load data

The Keras library already contains some datasets and MNIST is one of them. So, we can easily import the dataset and start working with it. The 'mnist.load_data()' method returns us the training data, its labels as well as the testing data and its labels.



2.3 Preprocessing

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1), while for non-convolutional NN it is (60000,28^2,1).

2.4 Add manually labeled images

If some corrections were saved during previous digit recognitions, read them to add to the training dataset. Misclassified images were appended to a CSV file, and their manual labels were appended to a separate CSV file.

2.5 Scale predictors from 0 to 1 and one-hot encode labels

All predictors (28^2 pixels) were scaled from 0 to 1, and labels (integers 0 to 9) were one-hot encoded to be usable by the NN models.

3. Neural Network models in Keras

3.1 NN model

Hyperparameters of the sequential NN model were varied around the following reference values.

Layer	**Neurons**	**Activation**	**dropout**
Dense	50	ReLU	-
Dropout	-	-	**0**
Dense	**50**	ReLU	-
Dense	10	softmax	-

Varied hyperparameters are in bold. The model was ***compiled*** with the Adam optimizer and 'categorical_crossentropy' loss function. Reference value of a training parameter: epochs = 200. Minimal number of trials in recognizing the handwritten images was 200, often used value was 500, and the maximal one was 1150.

3.2 Convolutional Neural Networks

A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, - this is the reason why CNN works well for image classification problems. One image is distinguishable from another by its spatial structure. Areas close to each other are highly significant for an image.

However, fully connected neural networks typically don't work well on large images, since they don't scale well with image size. Example: 32 * 32 * 3 image = 3072 weights; 200 * 200 * 3 = 120000 weights. This large number of parameters can quickly lead to overfitting. One could work with smaller version of images, but one would lose information.

The ***dropout layer*** is used to deactivate some of the neurons and while training, it reduces overfitting of the model. It randomly kills each neuron in layer of a training set with probability p, typically 0.5 (half of the neurons in a layer are dropped during the training). Therefore, the network cannot rely on activation of any set of hidden units, since they may be turned off at any time during training, and the model is forced to learn more general and more robust patterns from the data. We do not use dropout during validation of the data.

Convolution layers use a ***kernel*** (filter, or matrix), usually 3 x 3 or 5 x 5. Center element of the kernel is placed over the source pixel. The source pixel is then replaced with the sum of elementwise products in the kernel and corresponding nearby source pixels. Convolving the kernel over the image in all possible ways gives 2D ***activation map***. Convolving decreases the spatial size.

Max Pooling**:** Keep only a maximal value from each block, e.g., 2 x 2.

Hyperparameters of the sequential CNN model were varied around the following reference values.

Layer	Neurons	Activation	kernel_size / pool_size	dropout
Conv2D	32	ReLU	3, 3	-
Conv2D	64	ReLU	3, 3	-
MaxPooling2D	-	-	2, 2	-
Dropout	-	-	-	0.25
Flatten	-	-	-	-
Dense	256	ReLU	-	-
Dropout	256	ReLU	-	0.5
Dense	10	softmax	-	-

We ***compile*** the model with the Adadelta optimizer. However, beware of poor scaling of compute time with image size. Reference value of a training parameter: epochs = 200. The number of trials was 200 in all cases.

3.3 Choose / load / train / save the model

If the model is newly created or its hyperparameters updated or new training data added, we train and save the model. Otherwise, if returning to already studied model, we load it from an HDF5 file. The model.fit() function of Keras will start the ***training*** of the model. It takes the ***training data***, ***validation data***, ***epochs***, and ***batch size***. It takes some time to train the model. After training, we save the weights and model definition in the HDF5 file.

3.4. Evaluate the model

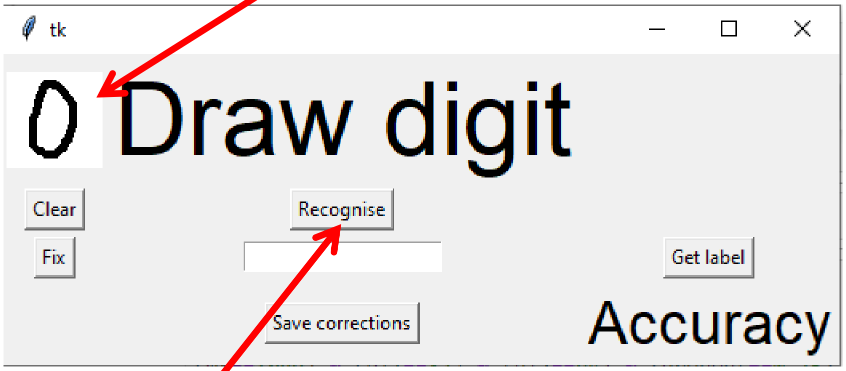
We have 10,000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the model; therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% of accuracy.

4. Create GUI to predict digits

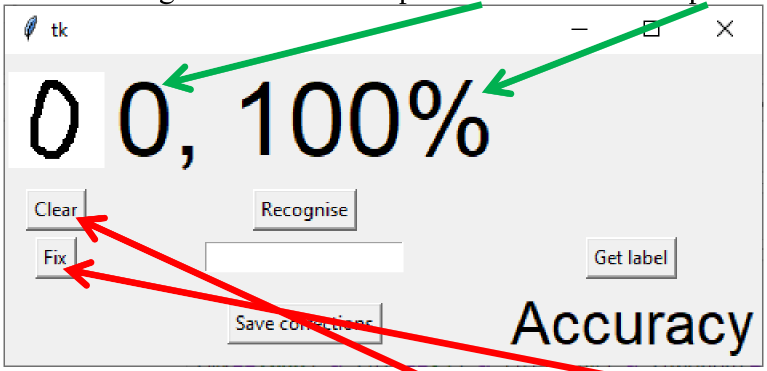
Here an interactive window (GUI) is created where one can draw a digit using mouse and get a prediction of what this digit is. The GUI is implemented using Tkinter library that comes in the Python standard library. The `App` class is responsible for building the GUI for our app. It has a canvas where one can draw by capturing the mouse event. Functions are triggered by pushing control buttons: button 'Clear' clears canvas and button 'Recognise' activates the function `predict_digit()` to recognize the digit. This function takes the image as input and then uses the trained model to predict the digit. The predicted label and its probability percentage are displayed.

Instructions for GUI:

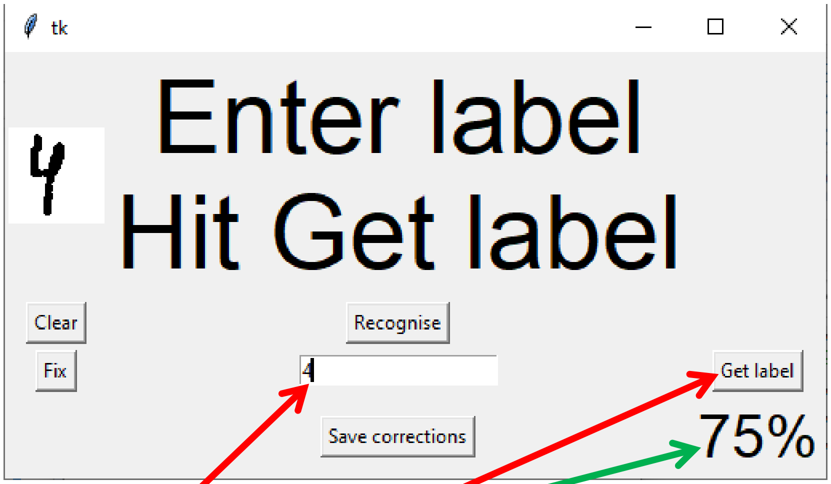
1. Start by drawing a digit in the canvas window (60 x 60 pixels).



2. Press "Recognise" and see the predicted class and its probability.



3. If classified correctly, press "Clear" and go to Step 1.
4. Otherwise, if the handwritten digit is misclassified, press "Fix" button to add the image to train set, which will be concatenated with the MNIST data before next training cycle.



5. Type in correct label and press "Get label" button to add the label to train set.
6. Press "Save corrections" button to save manually labelled images and labels, and clean memory.
7. Accuracy % is displayed on the bottom right.

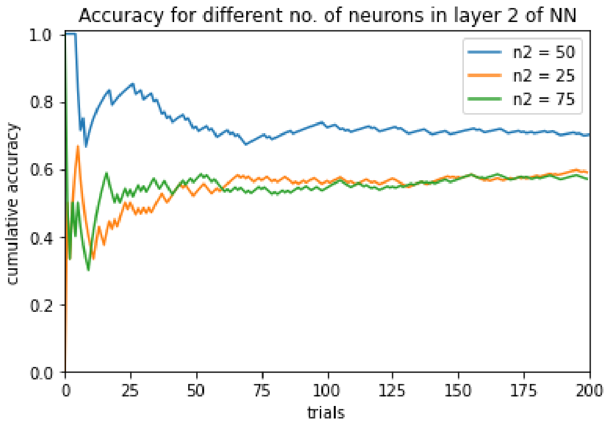
5. Results

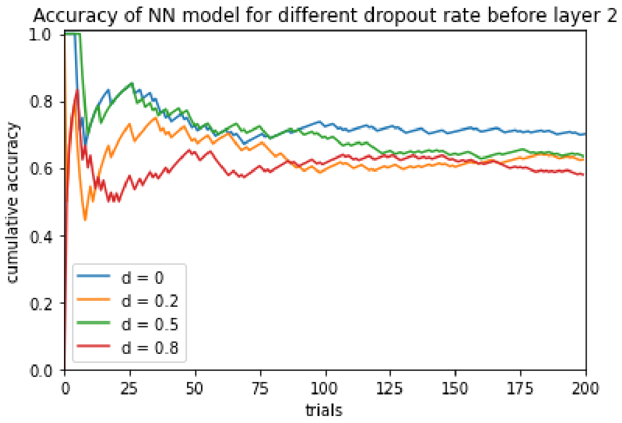
The recognition of handwritten digits was tested by applying the model to 10,000 test images and in the actual test by handwriting the digits in the GUI window. We iterated over the digits from 0 to 9 until accuracy curve has reduced its fluctuations. During the development of GUI, 78 misclassified digits were encountered, which were not saved. After implementing the user feedback in the GUI, all misclassified images and their correct labels were saved in two CSV files (all images in one file and all labels in a separate file).

5.1 NN models

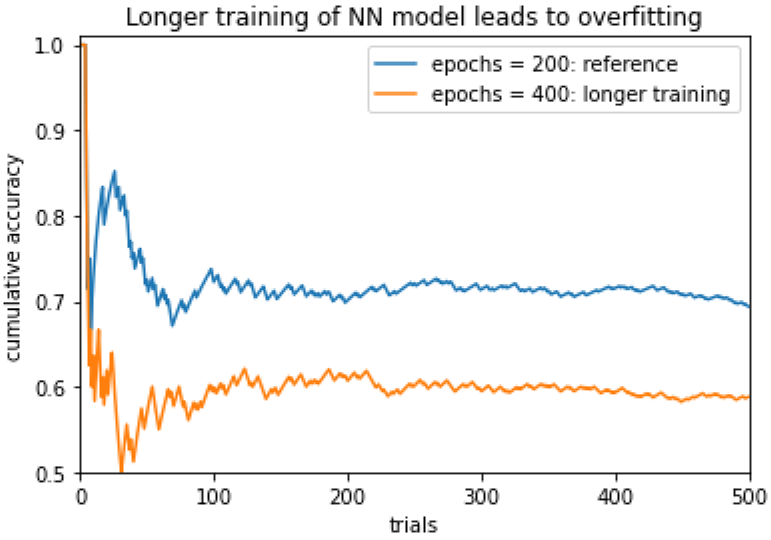
The following hyperparameters were optimized: n_2 – number of neurons in layer 2 of NN model. Testing a few values of n_2 shows that the optimal number of neurons in layer 2 is between 25 and 75, unless there are several maxima in accuracy.

The effect of dropout is negative: accuracy decreases for the three tested values dropout = {0.2, 0.5, 0.8}.

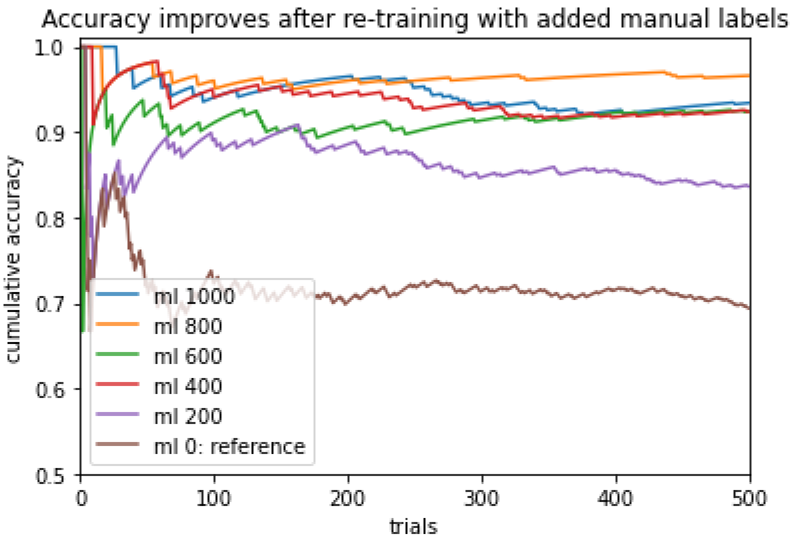




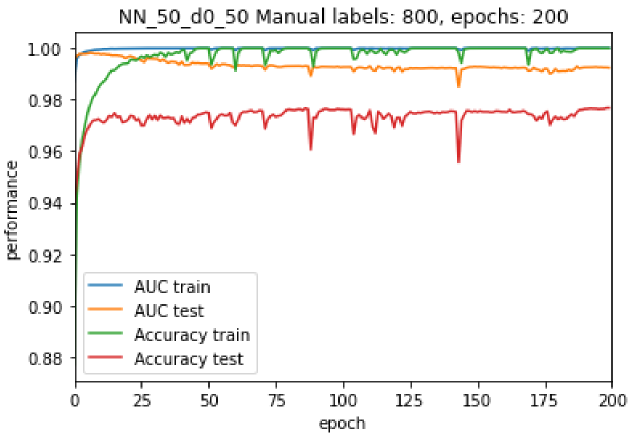
Longer training (epochs = 400) led to overfitting: accuracy has dropped down to 0.588.

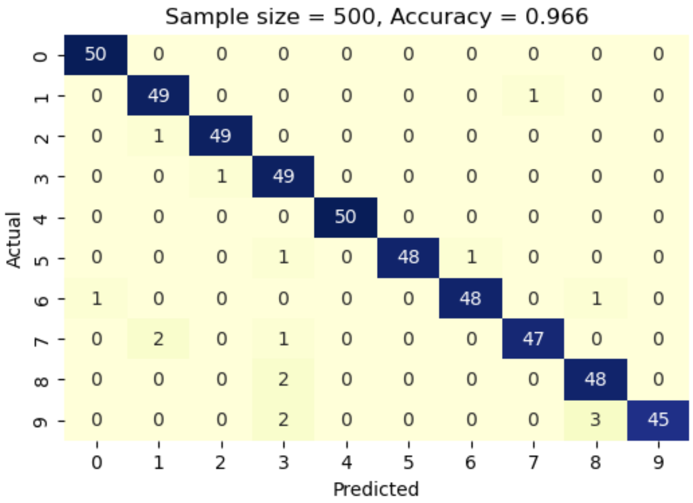


The highest accuracy was achieved by adding manually labeled images and re-training the model with these data added to the MNIST database. The more data added, the higher the accuracy is. However, there is no difference in accuracy between 400 and 600 manual labels added, since all 200 additional labels were generated during testing of lower-accuracy models, and not the model with 400 manual labels. The number of added labeled images is indicated in the legend following the ‘ml’. Same situation with no useful labels added to 800 manual labels among extra 200 labels leads to a slight degradation in accuracy, which drops from 0.966 for 800 manual labels down to 0.934 (after 500 trials).



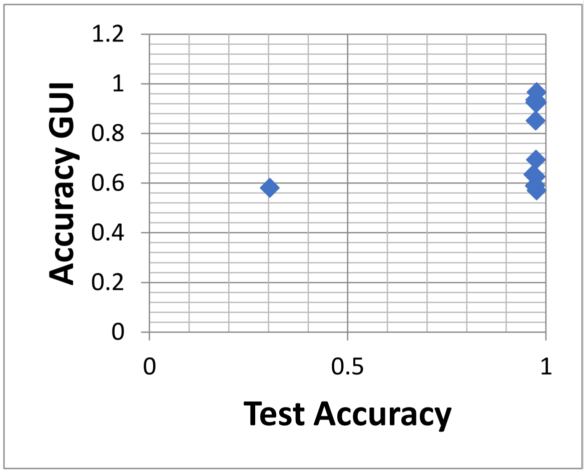
The best model is NN with the reference parameter set (50 neurons in both hidden layers, no dropout, 200 training epochs) and 800 manually labeled digits added for model training. The **accuracy** after 500 trials is **0.966**. Train-test accuracy in the process of training and the confusion matrix are shown below.

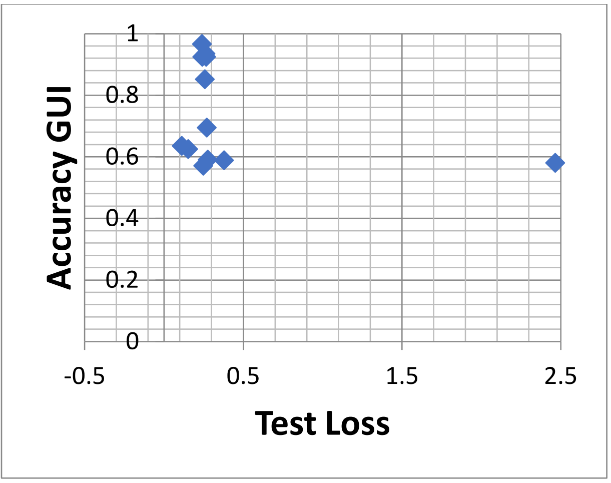




	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	49	0	0	0	0	0	1	0	0
2	0	1	49	0	0	0	0	0	0	0
3	0	0	1	49	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	1	0	48	1	0	0	0
6	1	0	0	0	0	0	48	0	1	0
7	0	2	0	1	0	0	0	47	0	0
8	0	0	0	2	0	0	0	0	48	0
9	0	0	0	2	0	0	0	0	3	45

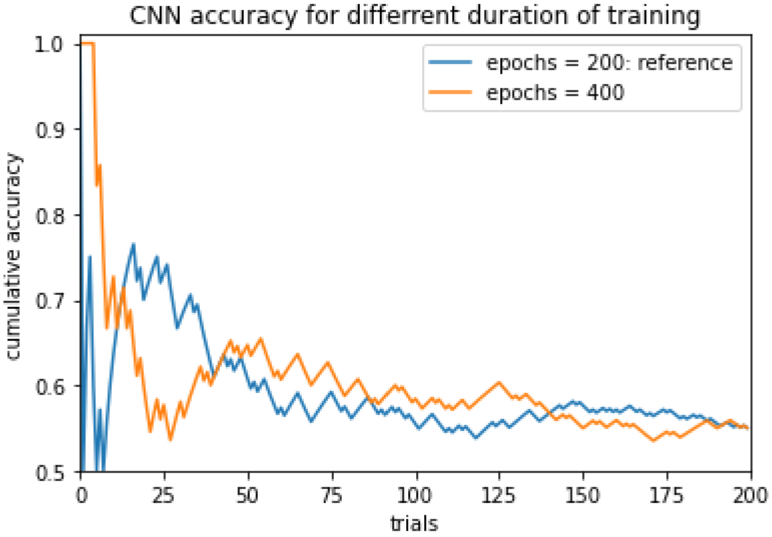
It was observed that test accuracy and loss are not reliable metrics of actual performance due to the lack of association with the actual accuracy, as illustrated in the two figures below. As an example, after adding 200 manually labeled images, test accuracy has slightly decreased (from 0.9752 to 0.9741), while the actual performance of digit recognition of the images drawn in the GUI window has strongly increased.



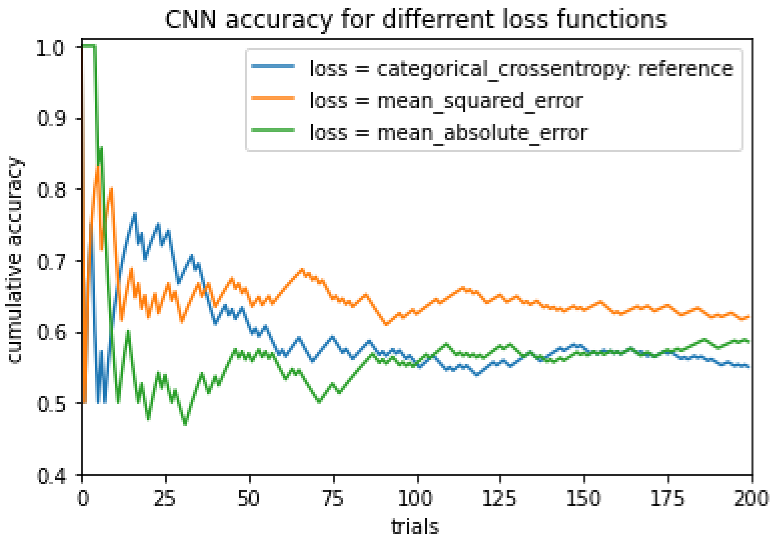


5.2 CNN models

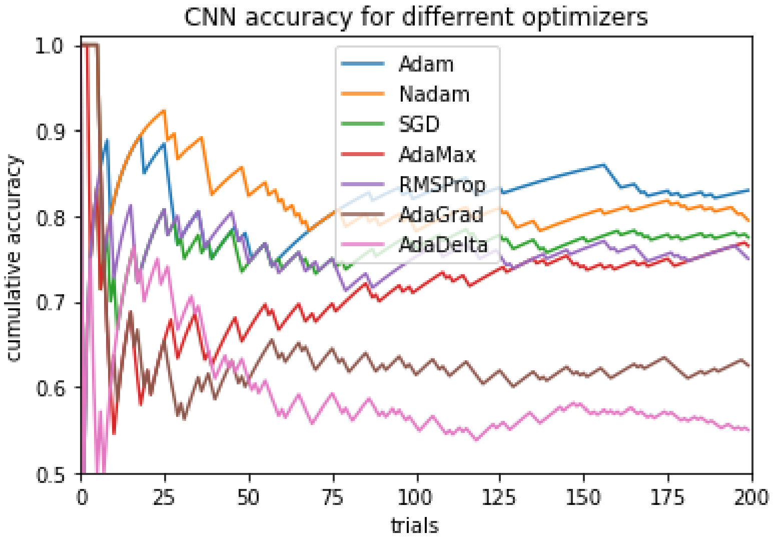
Training CNN model longer (400 epochs) did not lead to a change in accuracy, which stayed at 0.55 (after 200 trials).

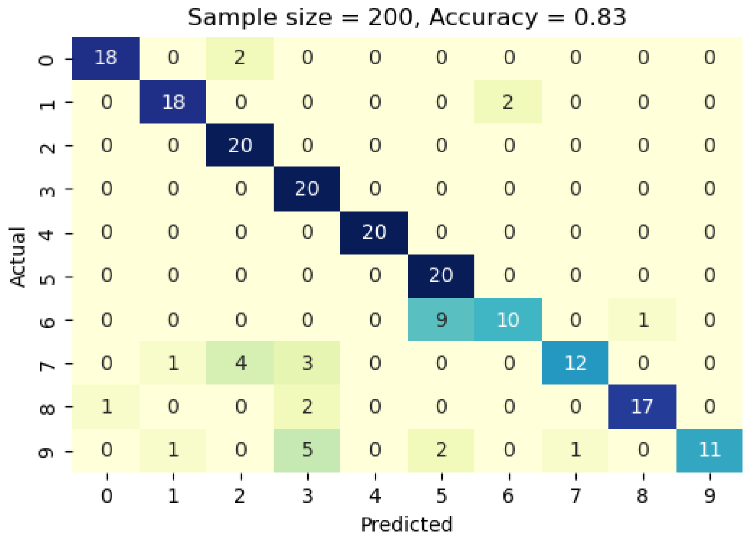


Different loss functions were tested, including user-defined mean absolute error function. Some improvement in accuracy was achieved, as measured after 200 trials. The ranking of the loss functions together with corresponding accuracies is: ‘categorical_crossentropy’ (0.55), ‘mean_absolute_error’ (0.585), and ‘mean_squared_error’ (0.62).



When testing optimizers, it was found that 'Adam' gives much higher accuracy of **0.83** than 'Adadelta' with the accuracy of 0.55. Other optimizers have accuracy values between the above two, except for 'Ftrl' optimizer, which could not train within 200 epochs and always predicted digit 1, and thus had an accuracy of 0.1.





Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	18	0	2	0	0	0	0	0	0	0
1	0	18	0	0	0	0	2	0	0	0
2	0	0	20	0	0	0	0	0	0	0
3	0	0	0	20	0	0	0	0	0	0
4	0	0	0	0	20	0	0	0	0	0
5	0	0	0	0	0	20	0	0	0	0
6	0	0	0	0	0	9	10	0	1	0
7	0	1	4	3	0	0	0	12	0	0
8	1	0	0	2	0	0	0	0	17	0
9	0	1	0	5	0	2	0	1	0	11

6. Conclusions

1. It is well worth saving misclassified digit images and their correct labels for re-training of the model. This can be seen from the following table of accuracy of handwritten digit recognition after 500 trials.

Manual labels	**Accuracy**
0	0.694
200	0.836
400	0.924
600	0.924
800	**0.966**
1000	0.934

Recommendation 1: Start from the saved model with the best accuracy, test your handwritten digit recognition using GUI, provide correct label for each misclassified image and save these images and labels using GUI. Re-train the model with the added manually labelled images. Repeat the above steps of testing and adding manual labels until performance no longer improves.

2. After adding 200 manually labeled images, test accuracy has slightly decreased (from 0.9752 to 0.9741), while the actual performance of digit recognition of the images drawn in the GUI window has strongly increased. Thus, training accuracy is not a reliable metric of actual performance.
3. NN model is prone to overfitting: the accuracy for 400 training epochs is significantly lower than for 200 epochs when not using manual labels.

Recommendation 2: Keep the number of epochs below 400 in NN model.

4. Accuracy is sensitive to the number of neurons in the 2nd layer: 50 neurons have a higher accuracy of 0.694 (after 500 trials) compared to 25 neurons with accuracy of 0.59 and 75

neurons with accuracy of 0.57 (after 200 trials). If there is one maximum of accuracy versus the number of neurons, then we have localized the lower and upper boundary for this hyperparameter.

5. Dropout between the two layers of NN model lowers the accuracy.

Recommendation 3: Avoid using dropout after the first layer in NN model.

Recommendation 4: First focus on hyperparameters of NN model, which is much faster to train than CNN model. However, keep training the CNN models in the background.

Recommendation 5: When adding more manual labels, generate them using the model that is being upgraded, and not some other lower-accuracy model.

7. Future Directions

- Compare the accuracy of recognition of original hand writer whose manual labels were used to re-train the NN model to the accuracy values for another user with the same models.
- Narrow down the optimal ranges of all hyperparameters within current NN and CNN configurations, including number of neurons in each hidden layer and dropout rates after each layer, loss function, optimizer, and activation functions.
- Test different numbers of layers.
- Minimize the number of control buttons in the GUI to simplify and speed up the process of dealing with misclassified digits. For instance, try to implement input of correct label via hitting Enter key, which would allow one to eliminate the button 'Get label'.
- Enable creation of new classes for letter symbols. This would require switching from integer to string variables for the labels.