Convolutional Neural Network

Step 1: Load and Preprocess the Data

Our first step was to load the data. This one is very simple, just taking the csv file and collecting the three numbers on their own, as well as concatenated to see the whole image. I also used the given plotImg function to show the image using matplotlib.

Step 2: Build the Odd/Even Classifier

The purpose of the build_odd_even_classifier function is to create a neural network to classify digits as either even or odd. This makes it binary. It has a single output neuron with a sigmoid activation function, so there is either a 0 or 1 representing even or odd. It also has the binary_crossentropy loss function which is appropriate for binary classification.

Step 3: Build the Digit Classifier

The purpose of the build_digit_classifier function is to create a neural network to classify digits themselves. This means all digits from 0-9, necessitating a categorical classification function which is suitable for multi class classifications because there is more than two categories (compared to the odd/even classifier).

These two classifiers use keras to build the neural networks. These are also specifically convolutional neural networks which bode well with image processing and helped with the classification of digits. These both can be changed to apply to more pixels as well.

Step 4: Training the Models

This step is where I call the classifier functions. Firstly, I load the data and classify each label as even or odd. My first classifier looks at the first label and determines if it is odd or even only. This way, the 0,2,4,6,8 are grouped together in the correct classification. Similarly, the odd numbers are also classified together. Moreover, I split the data into training and validation sets so that I can train the data with accuracy and validity. I also spent a while testing different ratios of training vs validation sets, and I concluded that 0.2 was the best percentage of validation set as it returned the most accurate predictions. Next was the digit classification. I split the data once again into training and validation sets, and specifically trained the neural network for the bottom two images shown. I also have 5 epochs or times the data runs through the network. I chose both the number of epochs and ratio of training to validation sets based on trial and error which produced the best results. **Step 5: Predictions**

This is the final step where I put the predictions of the labels into arrays. This section just takes the images and sends them into the neural network models and places the outputs of the neural networks into corresponding arrays. And finally, the arrays are printed out along with the accuracy of the prediction and it also outputs all labels that were labelled wrong for potential tweaking of the program.

Step 6: Other Thoughts

We did not have any other iterations of these models, but we chose this one based on simplicity and efficiency. I also found that with this model, the ratio of training to validation or the number of epochs did not affect the result a lot unless the percentage of validation sets or epochs was absurdly high (80% and 15). This was probably due to the already high number of training data points (10,000). For this reason, we kept the validation set percentage lower to allow for more training.

To evaluate the performance of this model, we will use metrics such as accuracy and precision. During the training process, we observed the potential problems with the neural network and its prediction. Certain numbers were much more difficult to

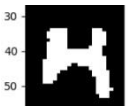
differentiate between: 1 vs 7, oddly shaped 6's and 4's, bolded numbers, and different types of 4's like 4 vs 4 with an open top. These are definitely because of inconsistencies between the number and their counterparts or similarities in the pixel structures when rendered. In the future, we could solve this by upscaling or downscaling the images. This will help a lot specifically with the bolded numbers.

The simplicity of this model makes it suitable for practical applications where quick and reliable number classification is required.

Step 7: Optimizations

Some more improvements of this model include:

1. Fine-tuning the model: Explore more complex algorithms to enhance accuracy and help with the edge cases stated before.
2. Allow the program to loop more: This would allow for a higher accuracy with a much longer run time, but this may be valued in specific scenarios where speed is not an important factor.
3. More hyperparameter testing: This lets us find better solutions and accuracies by increasing the number of epochs and ratios of training to validations. There is an optimal number of for the epochs and ratio leaving the program running for under 30 minutes of training, however, there will always be the problem of overfitting which must also be avoided.
4. Explainability: Some tools can be used to provide insight into how the model makes predictions. This would allow for further tweaking and refining to improve the model's accuracy.
5. Multi-Variable Learning: If the model was able to recognize different fonts or styles of handwriting, this would drastically improve the model's effectiveness delivering a greater accuracy.



In our tests, we found a consistent prediction performance of 75-85%. This is quite reasonable given that some numbers such as 4 looked like H's. I am quite comfortable with this prediction accuracy because although it may struggle with edge cases, it performs well with most of the set and does not overfit to accommodate the edge cases.

Step 8: Chosen Optimization to Implement

The biggest optimization that could have been implemented with more time is checking the individualized performance of the odd/even classifier and the digit classifier. Preferably, the odd/even should have a high accuracy because when it does not know what the correct answer 100% should be, it only has a ½ chance of being wrong, whereas the other classification has a 1/10 change.

I chose this one because:

1. The implementation should be feasible with the knowledge I have now, and it should not take too long
2. The numbers that are similar such as 1 and 7 are both in the same classification
3. By performing this optimization, the whole program will be able to be broken down easier
4. This would improve the modular performance, so that if in a future project I will use an odd/even classifier, this classifier will run as optimally as possible

Step 9: Practical Applications

As I finish this project, I see many potential use cases of an AI that can classify images. One such example is banking systems. This can be used to classify images of numbers, letters, and signatures all of which are handwritten and can be completely different. With the previously mentioned optimization techniques, it can become a very reliable software to expedite the process of reading cheques or deposit slips. On a similar train of thought, any document with handwritten notes on it can be identified and read by a computer.