NUMBER CLASSIFIER

Rushabh Tike (1800104)

Harish Devathraj (1800866)

The Number Classifying System

The goal: Take an input image (28x28 pixels) of a handwritten single digit (0–9) and classify the image as the appropriate digit. Number classifying system is the working of a machine to train itself for recognizing the digits from different sources like images and in different real-world scenarios for online handwriting recognition on computer tablets or system, recognize number plates of vehicles, processing bank cheque amounts, numeric entries in forms filled up by hand.

Application

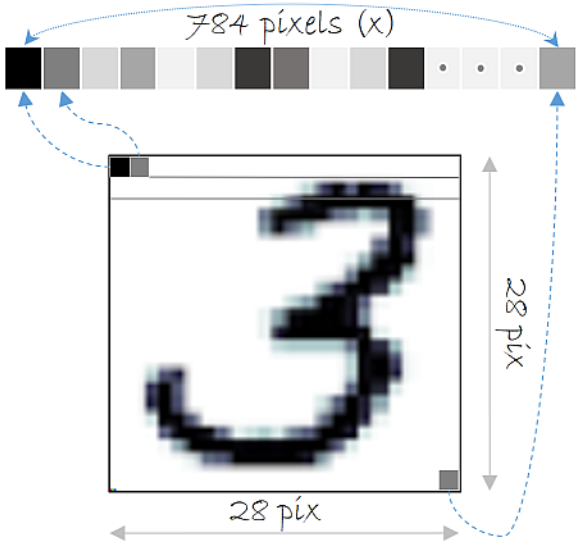
When we consider or think of some of the real-world examples or scenarios where this handwritten guessing might be useful: 1) The US Postal Service processes 493.4 million pieces of mail per day, and 1% of that workload is 4.9 million pieces of mail. Accurate automation can prevent postal workers from individually handling and examining millions of parcels each day. But reading entire address would not be that easy as recognizing each handwritten digit because it may require very complex models.

But it will be the first step for any such huge task which can lead us towards the goal.

Problems with the Handwritten digits

The handwritten digits are not always of same dimensions, as they can differ from writing of individual to individual for example the size or width and orientation so the general issue would be while grouping the digits because of the likeness between digits.

This image consists of digit 3 sloppily written and rendered at an extremely low resolution of 28 x 28 pixels.



But our brain has no trouble recognizing it as a three. We should take a moment to appreciate the Brain how it can do task effortlessly. The light-sensitive cells in our eyes that are firing will help us recognize other images as their own distinct ideas.

Various methods to recognize handwritten digits

1. Fully Connected Multi-Layer Neural Network

They can classify handwritten digits by using one or more hidden units. The fully connected multilayer NN extracts features based on entire spatial domain of image so the number of parameters required may be very high. Which makes difficult to work with complex datasets.

2. K-nearest neighbor

This is also one of capable methods to classify images, it uses Euclidean distance between the input to classify. These have higher error rate compared to above method. The main idea about this classifier is that it does not require training time nor input from the programmer in terms of designing the model. The major issue with this method is it requires too much memory and classification time. It works on raw data rather than the feature vectors.

3. Convolutional neural networks

The building block of CNN is the convolutional layer. CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images. It takes 28 by 28 matrix of neurons as an input and make connections in small and localized regions of input image rather than connecting every pixel in input image in first layers to next.

Since CNNs eliminate the need for manual feature extraction, one doesn't need to select features required to classify the images,

Specialty about neural networks

Artificial neural networks have the exceptional capability of deriving meaning from complex data. The information processing system is the key element of Neural networks' structure. It also has a capability to work with incomplete data and which even supports parallel processing. Neural networks have simple learning algorithm compared to some other models such as for example Bayesian model. They even have distributed memory

Disadvantages

- Sometimes neural networks can be slow.
- It requires tons and tons of data to be well trained.
- Requires more computation time compared to linear regression to learn things.
- Are sometimes prone to overfitting the data.

The idea of this project was to create an application which could classify numbers drawn by users on a gui. This is an extension of the MNIST handwritten digit classification problem.

We have used the MNIST dataset, keras deep learning library to train the model and used pygame and tkinter to create a gui for the user to draw the numbers on.

Implementation:

1. Data preparation

We used pandas to load the training and testing datasets. Since the pixel values were grey scale between 0 and 255, we normalized the values by dividing each value by 255.

In Keras, the layers used for a two-dimensional convolution network expect pixel values with dimensions [pixels][height][width]. Hence, we had to use the reshape function to convert pixel values to the required format.

The output variable is an integer from 0 to 9, hence we had to use to_categorical() from keras to convert class vector(integers) to binary values.

Apart from the images in the MNIST dataset, we produced a lot more images by randomly rotating, scaling and shifting the images.

2. Model

The first question that we needed to answer was that do we really need a complex CNN to create a model with almost 100% accuracy.

The first step to answer that question for to build a very simple neural network with a single hidden layer

This model is a simple neural network with one hidden layer. A rectifier function was used for the hidden layer. We had used softmax activation function in the output layer to turn the outputs into probability like values and allow one class of the 10 to be selected as the model's output prediction. We also used

categorical_crossentropy as the loss function and ADAM gradient algorithm to learn the weights.

```
#first model
model = Sequential()
model.add(Dense(num_samples,input_dim=num_samples, kernel_initializer='normal', activation='relu'))
model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train2, Y_train2, validation_data=(X_val2, Y_val2),epochs=10,  verbose=2)
scores=model.evaluate(X_val2, Y_val2, verbose=0)
print("NN Error: %.2f%%" % (100 - scores[1] * 100))
```

We used the test dataset to evaluate the model and we a classification error of 2.34%.

We could improve this result by using convolutional neural network.

Second model we built was a simple convolutional neural network

CNN are more complex than multi-layer perceptrons. We built a model with a simple CNN architecture to get better results.

The first hidden layer was a convolutional layer. It had 32 feature maps and the kernel size of 3. We also used a rectifier function in this layer. This was the input layer which took images as input.

The next layer was a regularization layer using dropout. It was configured to randomly exclude 40% of neurons in the layer to reduce overfitting.

The next layer was a flattening layer. It converted the matrix data to a vector, which allows the output to be processed by standard fully connected layers.

```
#secondmodel
model=Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.fit(X_train2, Y_train2, validation_data=(X_val2, Y_val2),epochs=10,  verbose=2)
scores=model.evaluate(X_val2, Y_val2, verbose=0)
print("CNN Error: %.2f%%" % (100 - scores[1] * 100))
```

We evaluated this model in the same way we evaluated the first model. This model gave a classification error of 2.29%.

The third model we built was a deeper convolutional neural network with more layers

Since the second model did not give us desirable result, we had to improve it to get better results.

We had to change the model configuration. We started by more convolutional layers with different filter sizes. We also added a Batch Normalization layer, which had the effect of changing the distribution of the output of the layer, specifically by standardizing the outputs. Which also enabled us to stabilize and accelerate the learning process.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(128, kernel_size=4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
```

The second stage of building a better CNN model was to decrease the learning rate of the model. We used LearningRateScheduler for that. We also used fit_generator() to fit this model using the images we created using ImageDataGenerator.

This model gave us a classification error of 0.40%.

3. GUI

The final part of the implementation was to create a gui for user to draw on. We would use the final model created above to classify numbers drawn by the user. The gui was created using pygame and tkinter.

We used a get_pressed() function in pygame to track the number drawn by the user. The pixels which were clicked by the user were colored black and the remaining pixels were colored white.

After the user has finished drawing, the guess method will use the trained model to classify the number.

Figure 1 Drawing Surface

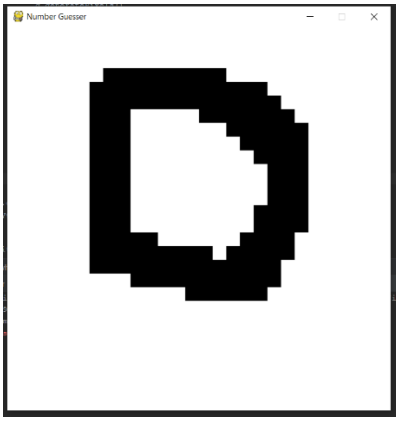


Figure 2 Number drawn by User

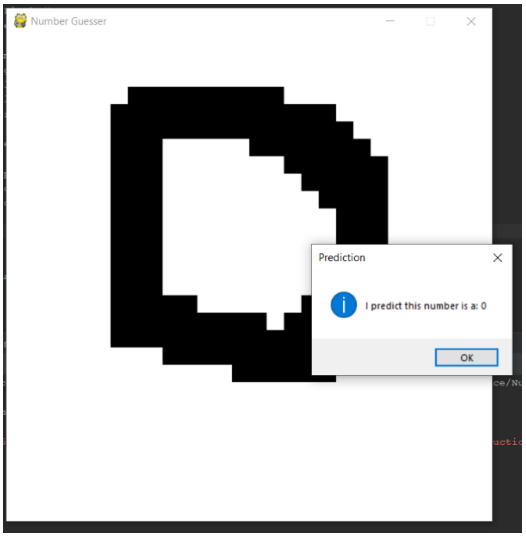


Figure 3 Number classified

Conclusion

We can observe that out of the 3 models we created, the deeper CNN had the lowest classification error. We assembled and trained CNN model to correctly classify images of numbers. We found out that the accuracy of the model depends on the number of epochs in order to detect potential overfitting. Usually 10 epochs are enough for successfully training the model, we found that 45 epochs increased the accuracy of our model (but it was a very minute improvement).

We also found out that the accuracy of model can be increased by using an ensemble of 2 or more CNNs, but it will also increase the training time significantly and would increase accuracy of the model by around 0.1%

References

- Yann LeCun, Leon Bottou,Yoshua Bengio and Patrick Haffner , paper on "Gradient Based Learning Applied to Document Recognition" , Proc of the IEEE , NOVEMBER 1998
- Xuan Yang , Jing Pu paper on "Mdig:Multi-digit Recognition using Convolutional Neural Network on Mobile
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- http://www.datamind.cz/cz/vam-na-miru/umela-inteligence-a-strojove-uceni-ai-machine-learning
- https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/d/80879v00_Deep_Learning_ebook.pdf
- http://cs231n.github.io/convolutional-networks/