Computer Vision

Laxman Desai

June 15, 2021

Contents

1 MNIST Classification **2**
- 1.1 Model Layers 2
  - 1.1.1 Input Layer 2
  - 1.1.2 Convolution Layer 2
  - 1.1.3 Max Pooling Layer 2
  - 1.1.4 Flatten Layer 3
  - 1.1.5 Dropout Layer 3
  - 1.1.6 Softmax Dense Layer 3
- 1.2 Code 4
  - 1.2.1 Import modules: 4
  - 1.2.2 Load the dataset: 4
  - 1.2.3 Build the model: 4
  - 1.2.4 Train the model: 5

2 Optical Character Recognition **6**
- 2.1 Preprocessing 6
  - 2.1.1 Skew Correction 6
  - 2.1.2 Binarization 7

1 MNIST Classification

The MNIST (Modified National Institute of Standards and Technology) database is a large database of handwritten digits that is commonly used for training various image processing systems. It has a training set of 60,000 examples and a test set of 10,000 examples, and is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.
We will attempt to train models to classify the dataset in this section.

1.1 Model Layers

Keras Layers are the functional building blocks of Keras Models. Each layer is fed with input information, they process this information, do some computation and hence produce the output. Further, this output of one layer is fed to another layer as its input.

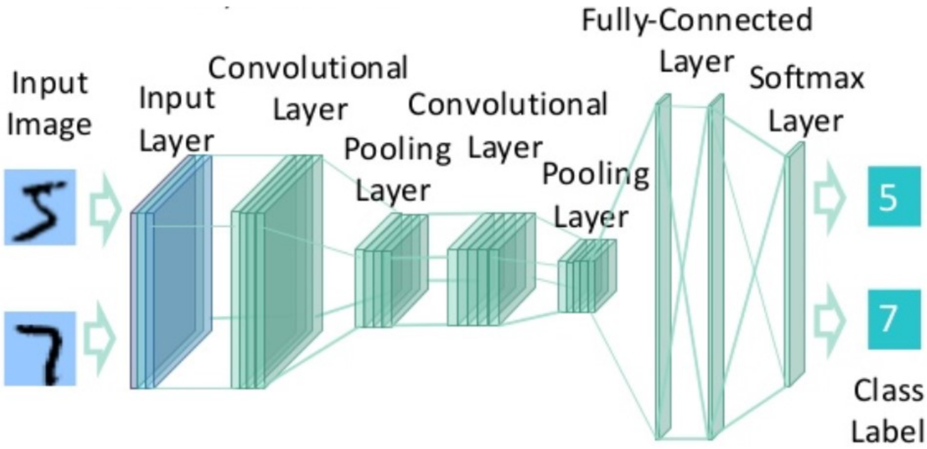

Figure 1: Intuitive overview

1.1.1 Input Layer

To understand the structure of the input data, Keras requires the shape of the input.The original input image is 28×28, so the `input_shape = (28,28,1)`, where 1 indicates that number of color channels is 1.

1.1.2 Convolution Layer

This layer creates a convolution kernel. It is convolved over a single input to produce a tensor of outputs. If you are using this layer as the first layer of your model, provide `input_shape` as the argument.

1.1.3 Max Pooling Layer

Maximum pooling is a pooling operation that calculates the largest value in each non overlapping patch of the feature map.
The results are down sampled (or pooled feature) maps that highlight the most present feature in the patch. This method has been found to work better in practice than the average pooling method for computer vision tasks like image classification.
Downsamples the input along its spatial dimensions (`height` and `width`) by taking the maximum value over an input window (of size defined by `pool_size`) for each channel of the input. The window is shifted by `strides` along each dimension.

1.1.4 Flatten Layer

Flattens the the input.

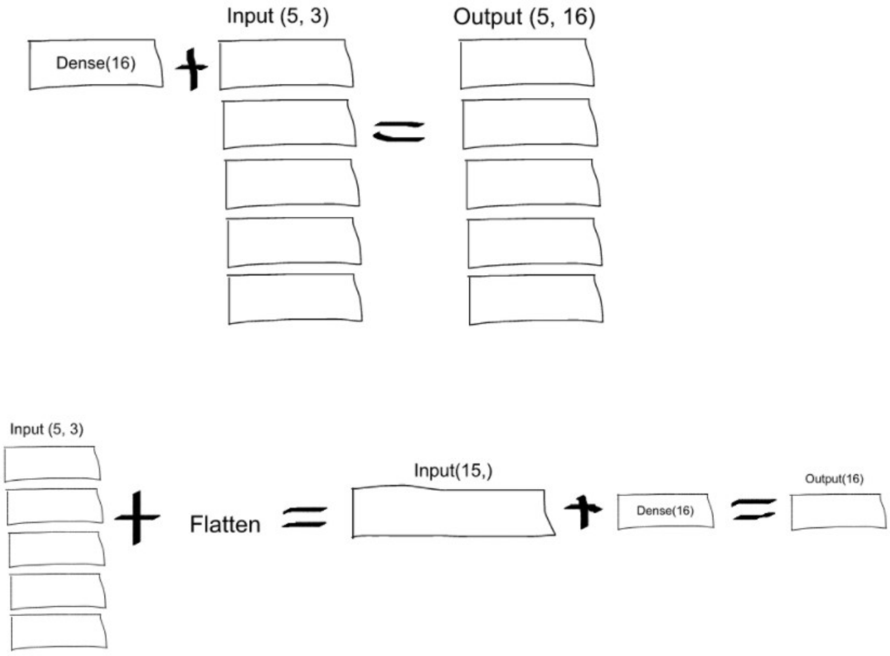

Figure 2: Reason for the use of a flatten layer.

1.1.5 Dropout Layer

You can implement dropout by added Dropout layers into our network architecture. The Dropout layer randomly sets input units to 0 with a given frequency. It prevents overfitting by helping you dropping-out user-defined hyperparameters.

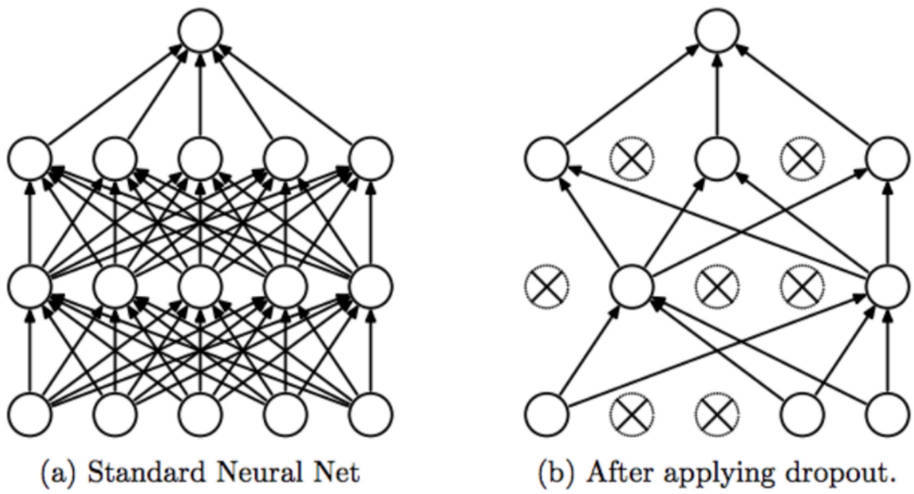

Figure 3: Dropout in action. Crossed units on the right have been dropped.

1.1.6 Softmax Dense Layer

The dense layer is one of the core layers. It is a standard neural network layer. It is helpful to produce output in the desired form.

1.2 Code

1.2.1 Import modules:

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

1.2.2 Load the dataset:

```
# Import the MNIST dataset
from tensorflow.keras.datasets import mnist

# Split data between training and testing sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Scale images from [0, 255] to the [0.0, 1.0] range
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Add an extra dimention for the layers
# Layers = 1 for grayscale images
# Layers = 3 for RGB / HSV images

X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
```

1.2.3 Build the model:

```
model = keras.Sequential(
  [
    keras.Input(shape=input_shape),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation="softmax"),
  ]
)

model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010

Total params: 34,826, Trainable params: 34,826, Non-trainable params: 0

1.2.4 Train the model:

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam",
  metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
  validation_split=0.1)
```

2 Optical Character Recognition

We will try and build a model that can break CAPTCHAs. What's a CAPTCHA? It's basically a the test to determine if the user is a bot or a human. We will first attempt at breaking text based CAPTCHAs.

Stages:

1. Data acquisition
2. Preprocessing
3. Building a model
4. Training the model
5. Testing the model

2.1 Preprocessing

We cannot input an image directly for the OCR system. Some pre-processing has to be done on the image so that it becomes easy for the OCR model to recognize the information in the image.

Preprocessing includes the following:

1. Skew Correction
2. Removing Lines (Horizontal & Vertical)
3. Building a model
4. Testing

2.1.1 Skew Correction

Image obtained from the previous stage may not be correctly oriented, It may be aligned at any angle. So we need to perform skew correction to make sure that the image forwarded to subsequent stages is correctly oriented.

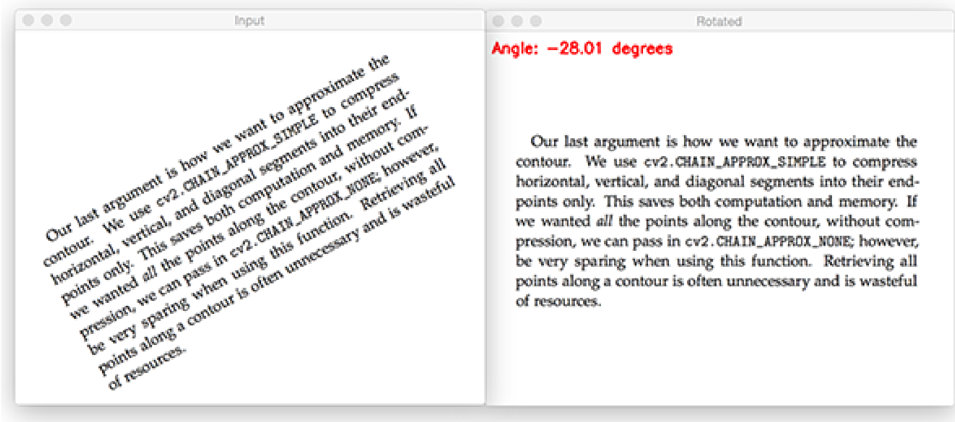

Figure 4: Picture caption

2.1.2 Binarization

That is, converting a coloured image to a black and white binary image. In practice, there exists an intermediate grayscale image.

$$\text{Coloured image} \rightarrow \text{Grayscale image} \rightarrow \text{Binary image}$$

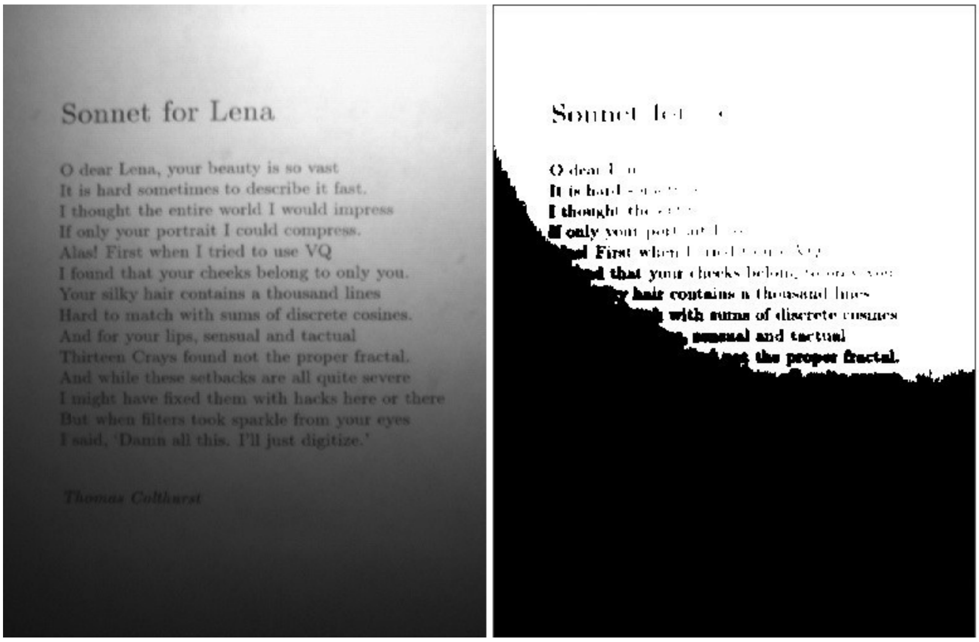

Figure 5: Binarization using a threshold on the image captured under non-uniform lighting.

Converting an RGB image to a grayscale image is easy. To a computer, a black pixel has a value of 0 and a white pixel has a value of 255. Converting a grayscale to B&W requires one to set a threshold value. If the pixel value is greater than the threshold, it is considered as a white pixel, else its as a black pixel. This naive strategy fails when lighting conditions are not uniform in the image. Here we can use Otsu's method or adaptive thresholding.

Otsu's Binarization: Since we are working with bimodal images, Otsu's algorithm tries to find a threshold value (t) which minimizes the weighted within-class variance given by the relation:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

where,

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

Python code:

```python
import cv2 as cv

# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
```