In this assignment we explore the handwritten digit images from the MNIST dataset. We will split the dataset into 60,000 training and 10,000 testing sets. The neural network models will try to classify grayscale images of handwritten digits (28 pixels by 28 pixels), into their ten categories (0 to 9). We will visualize intermediate activations of each convolution (or hidden) layers with a set of learnable filters. Our neural network should learn filters that activate when they see some type of visual features, such as a pattern of some orientation (diagonal, horizontal, vertical edges), or eventually entire honeycomb-like patterns on higher layers of the network (Karpathy, n.d.).

According to Figure 1, the network transforms the digit image into representations that are increasingly different from the original image and increasingly informative about the final result. We can think of a deep network as a multistage information-distillation operation, where information goes through successive filters (or layers) and comes out increasingly purified (Chollet, 2018).

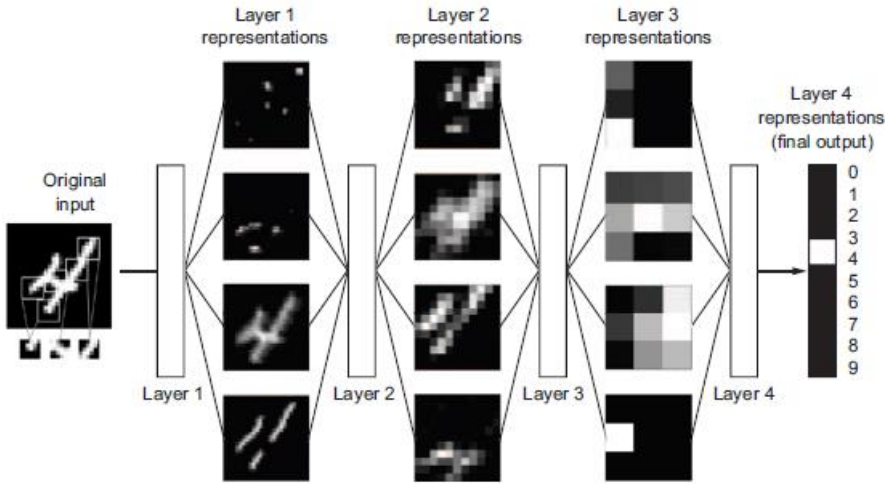
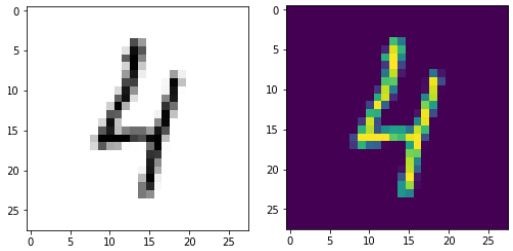


Figure 1

We can see how each filter has learned to activate optimally for different features of the image by minimizing an error (or cost) function. Images can be broken into local patterns such as edges, textures, and so on. A first convolution layer will learn small local patterns such as edges, a second convolution

layer will learn larger patterns made of the features of the first layers, and so on (Chollet, 2018). For example, the bottom picture of the 'layer 1' in Figure 1, appears to encode diagonal edges.



In Figure 2, let's look at another closest digit four from the testing set, which an image index number is 227.

Figure 2

Figure 3a and Figure 3b have the model summary and flow diagram; we can see that there are five layers of activations (conv2D, MaxPool, conv2D, MaxPool, and conv2D): three 2D convolution (or hidden) layers with ReLu activations and two pooling layers. The activation of the first convolution layer starts with 26 by 26 (height x width) pixels and 32 filters, and the last convolution layers end with 3 by 3 (height x width) pixels and 64 filters. We will try different filters (such as 1st out of 32 filters) of each layer and visualize them to understand what features can be learned. The filters are the neurons in the layers, which take input weights and output a value. For the input layer, the inputs are the pixel values, but as we go deeper in layers, the inputs are the outputs from the previous layer. Each filter encodes relatively independent features. A filter is moved across a layer, and the output is collected in the feature map.

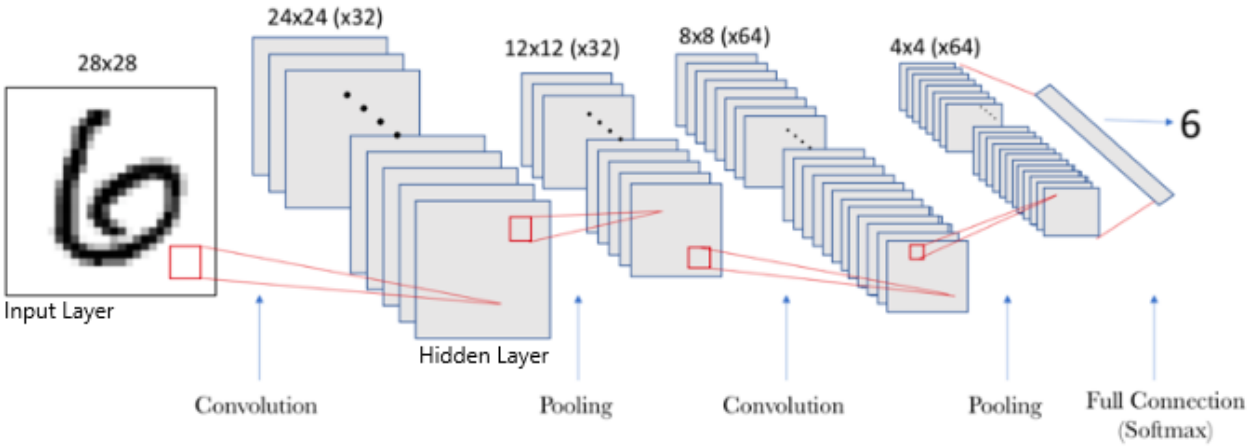


Figure 3a

In Figure 3a, the first convolutional layer receives a size input tensor (28, 28, 1) and generates a size output (24, 24, 32), where the level of hidden layers is composed of several filters. In our example, we propose 32 filters, where each filter is defined with a *W* matrix (filter or kernel size) of 3×3 and a bias b.

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(filters = 32,
                        kernel_size =(3, 3),
                        padding = 'same',
                        activation='relu',
                        input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(filters = 64,
                        kernel_size =(3, 3),
                        padding = 'same',
                        activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(filters = 64,
                        kernel_size =(3, 3),
                        padding = 'same',
                        activation='relu'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 7, 7, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

```
hidden_1 = tf.layers.conv2d(inputs = x_image,
                            filters = 32,
                            kernel_size=[3,3],
                            padding = 'same',
                            activation = 'relu')

pool_1 = tf.nn.max_pool(hidden_1,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding ='SAME')

hidden_2 = tf.layers.conv2d(inputs = pool_1,
                            filters = 64,
                            kernel_size=[3,3],
                            padding='same',
                            activation = 'relu')

pool_2 = tf.nn.max_pool(hidden_2,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding='SAME')

hidden_3 = tf.layers.conv2d(inputs = pool_2,
                            filters = 64,
                            kernel_size=[3,3],
                            padding='same',
                            activation = 'relu')

hidden_3 = tf.nn.dropout(hidden_3, keep_prob)
```

Figure 3b (left uses Keras, right does not use Keras)

Let's try visualizing the edges by trying various filters in Figure 4. The first layer activation of Figure 4.a, using 25th filter out of 32 filters, detects vertical and diagonal lines of the digit 4. The second and fourth layers of Figure 4.b and 4.c are max pooling, which downsamples the previous layers, and they looked zoomed-in images. The third layer activation of Figure 4.c is abstract and less visually interpretable. If we had 128x128 pixel sized photo of cat, this layer could have encoded higher-level concepts such as "cat

ear" or "cat eye." The last layer of Figure 4.e carries less information about the visual contents of the image, but more information related to the class of the image. We can see that the image is getting repeatedly transformed in each layer, and irrelevant information gets filtered, and useful information gets magnified and refined (Chollet, 2018).

a. 1st layer activation (None, 26, 26, 32) b. 2nd layer activation (None, 13, 13, 32) c. 3rd layer activation (None, 11, 11, 64) d. 4th layer activation (None, 5, 5, 64) e. 5th layer activation (None, 3, 3, 64)

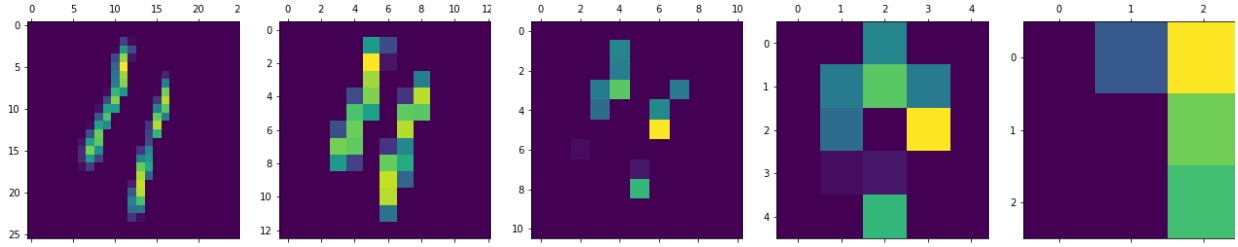


Figure 4

There are 32 filters in convolution layer 1 and 64 filters in convolution 2; we have just explored 25th filter of those layers. Let's try other filters such as filter 7th and see what specific patterns can be learned in Figure 5. Almost all layers of 7th filter appears to give more emphasis on the horizontal edges of the digit 4. However, the last layer in Figure 5 is blank because the pattern encoded by the filter isn't found in the input image.

a. 1st layer activation (None, 26, 26, 32) b. 2nd layer activation (None, 13, 13, 32) c. 3rd layer activation (None, 11, 11, 64) d. 4th layer activation (None, 5, 5, 64) e. 5th layer activation (None, 3, 3, 64)

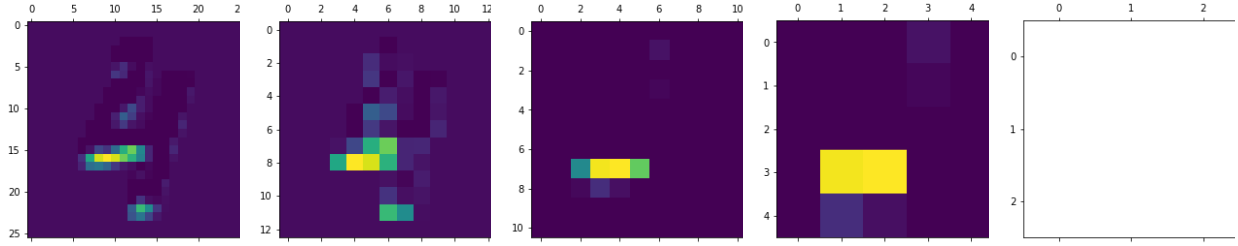
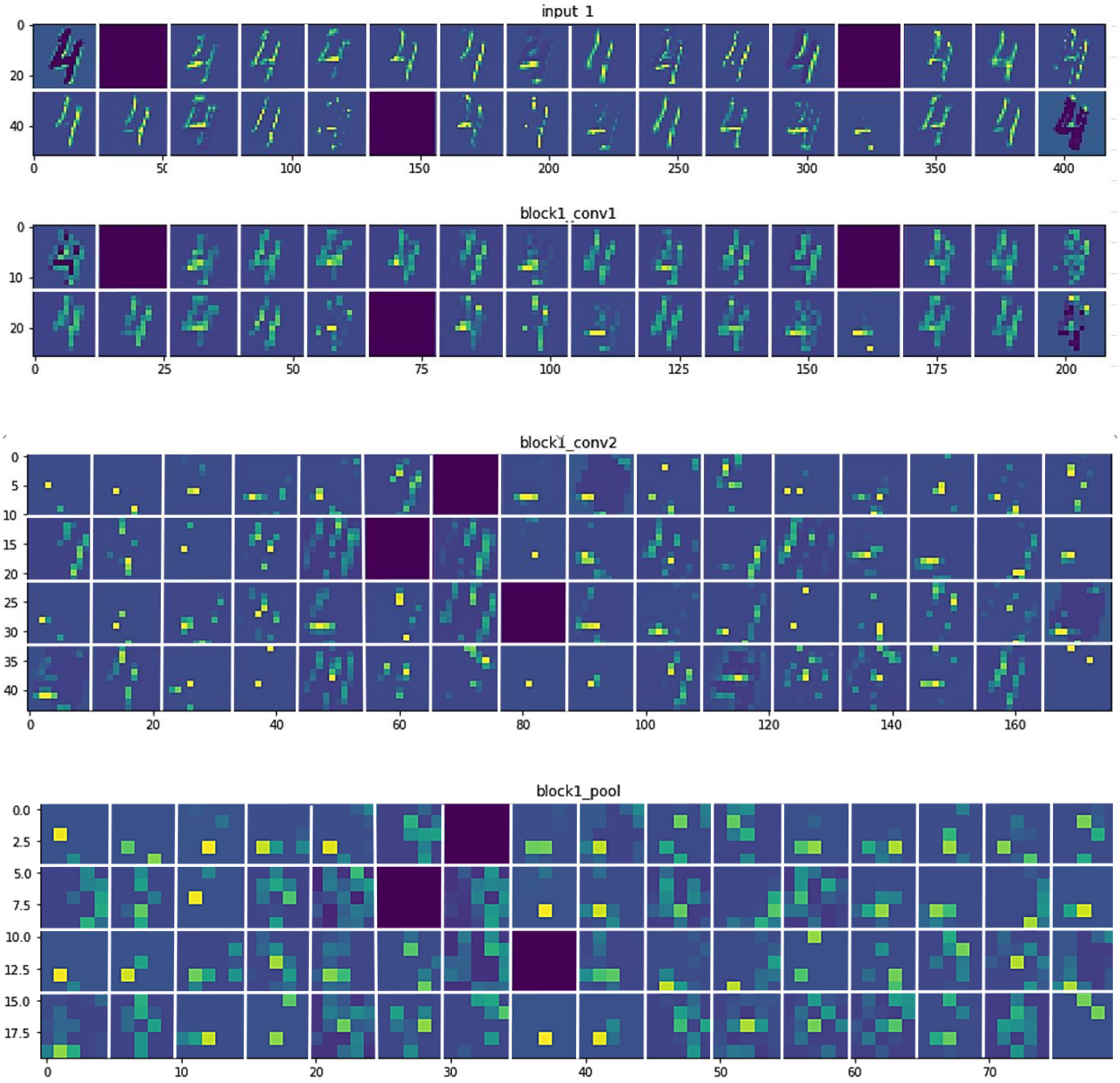


Figure 5

Let's extract and plot every filter and all the activations in the network for the same digit image.

The first layer (input 1) of Figure 6 is almost retaining the full shape of the digit 4, although there are several filters that are not activated and left as blank because the pattern encoded by the filter isn't found in the input image. At that stage, the activations retain almost all of the information present in the initial picture. As we go higher-up, the activations become increasingly abstract and less visually interpretable while filtering out irrelevant visual details.



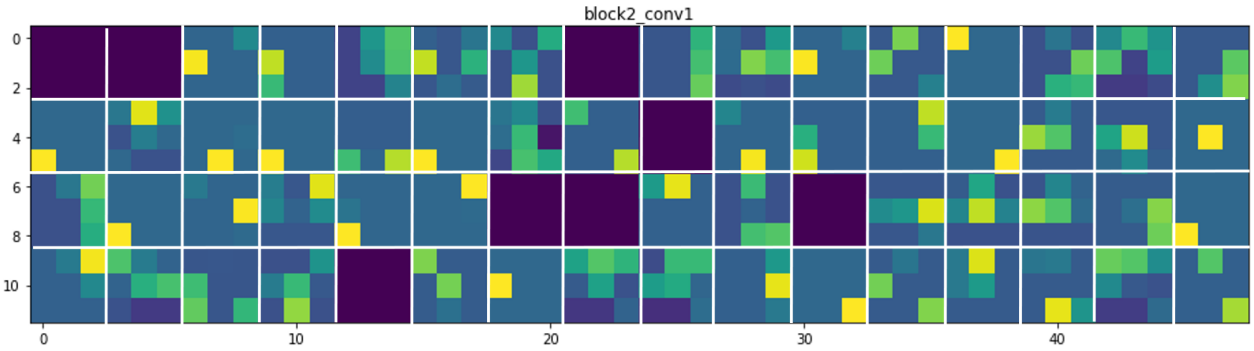


Figure 6

We have tried 32, 64, 64 filter sizes. What does happen if we reduce the filter sizes to a smaller number such as 3, 6, 12?

In previous examples, I used the Keras package, and we had seen the code in Figure 3, which is the left-side code. Let's run the code without using Keras to understand the filters further. That code is found in Figure 3, which is the right-side code. Both codes give us similar results; however, without Keras, we encode filters with better images. I visualized 3 convolution layers only, and I did not include max pool visuals.

We can see how filters activate the neurons of the first convolutional layer with 3 filters only (instead of 32) in Figure 7. We can see how each filter has learned to activate optimally for different features of the image.

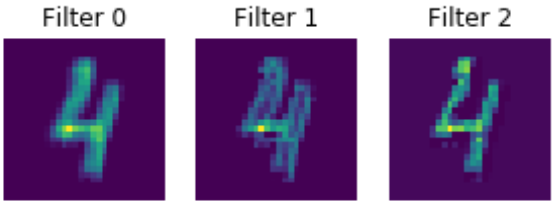


Figure 7

And here is how it activates the neurons of the second convolutional layer with 6 filters in Figure 8. This layer, which is in between, learned more filters than the previous layers but fewer filters than the layers closer to the output.

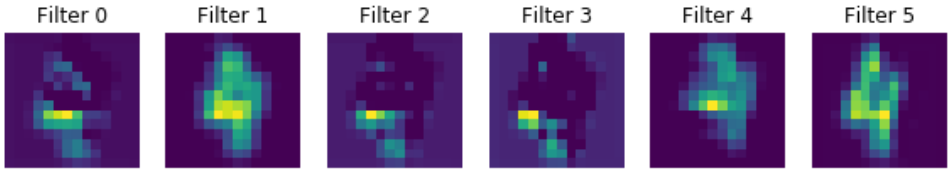


Figure 8

And here is how it activates the neurons of the third convolutional layer with 12 filters in Figure 9.

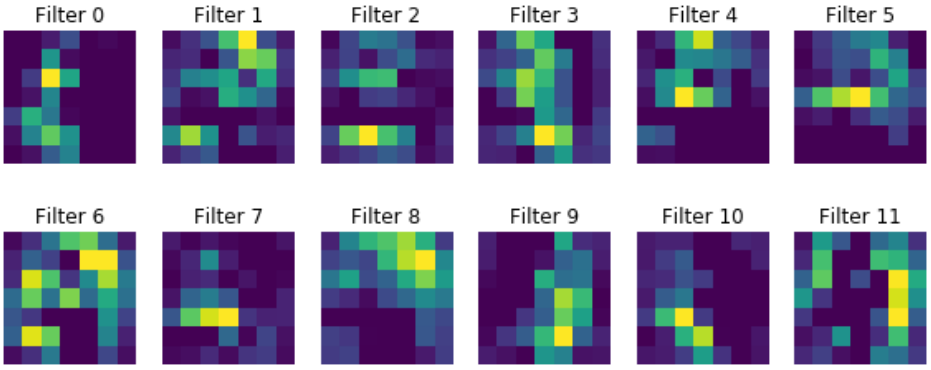


Figure 9

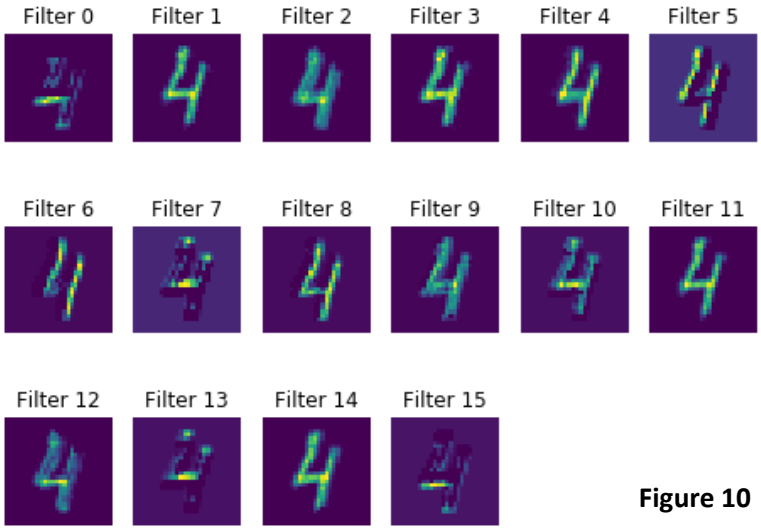


Figure 10

With fewer filters, we do not see a clear pattern. Let's increase filters from 3, 6, 9 to 16, 32, 64, respectively. After increasing the filter sizes, the model test and train accuracy increased drastically. Now the first convolutional layer has 16 filters in Figure 10. We can see that the filter 6 detected vertical signals while filters 13 and 15 detected horizontal lines.

The second convolutional layer has 32 filters in Figure 11. We can see all the filters detecting edges, diagonals, horizontals, verticals, and other patterns.

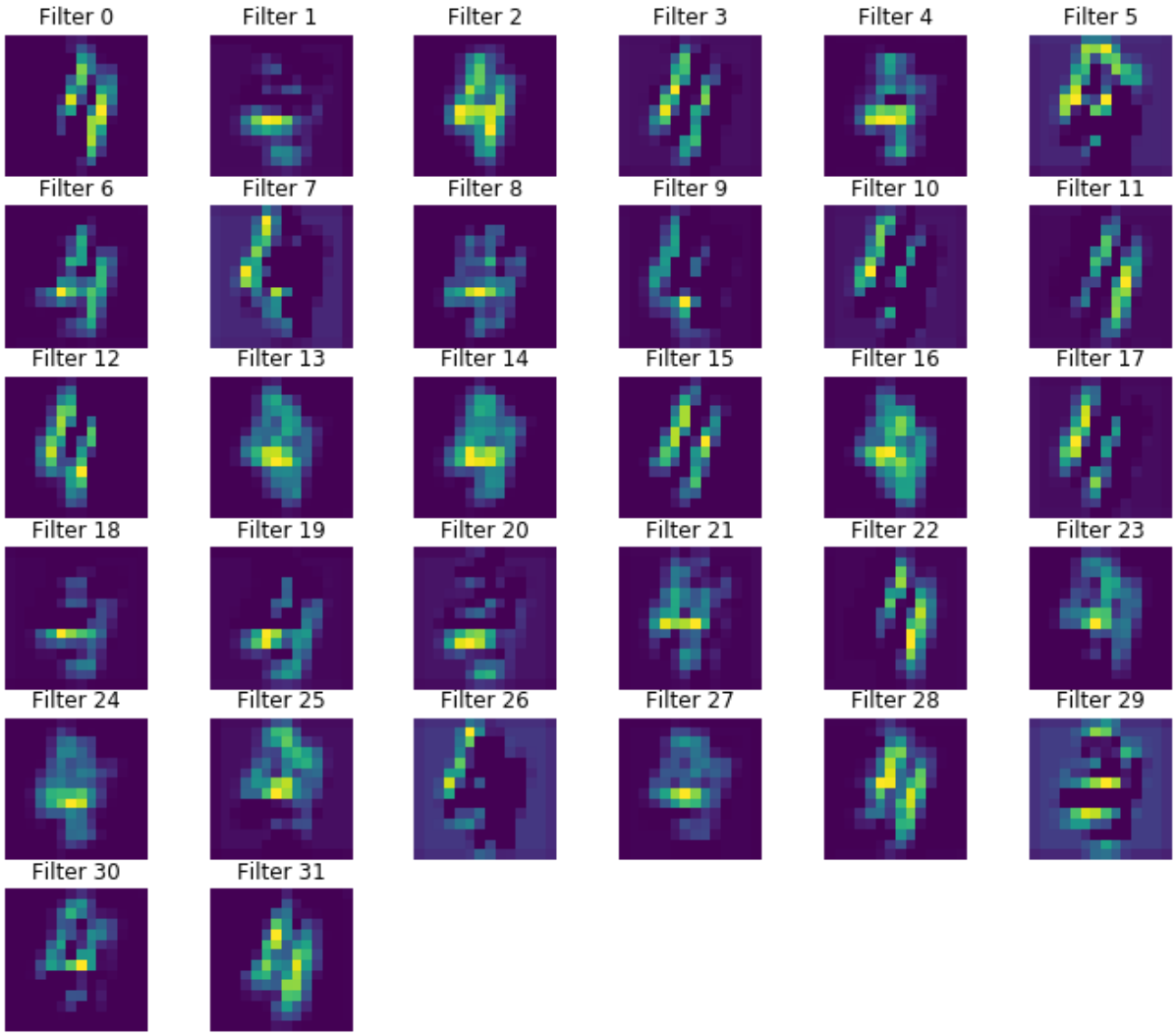


Figure 11

However, when it comes to the last layer, it is difficult to extract meaningful patterns of those filters in Figure 12.

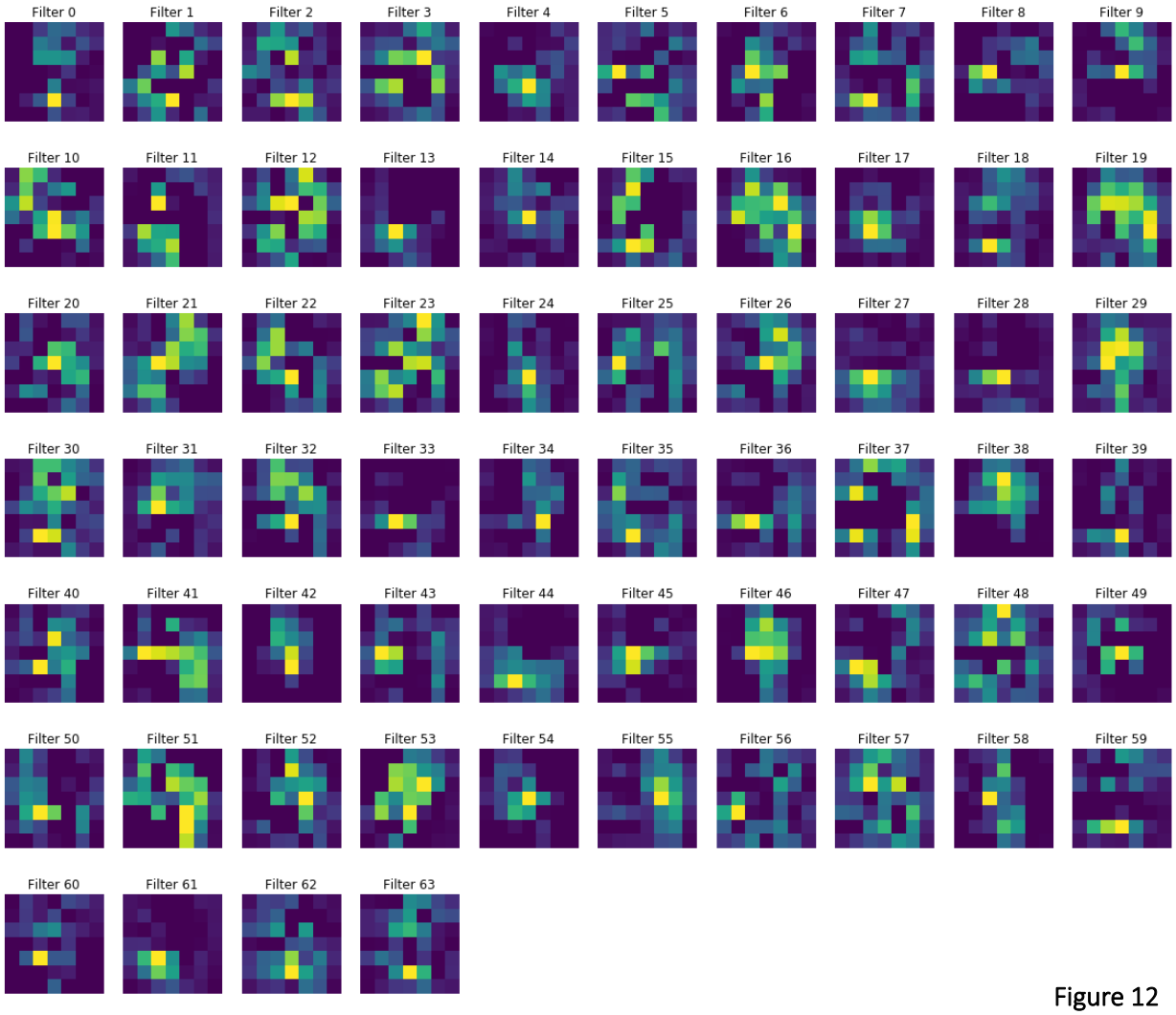


Figure 12

Summary

In this paper, we explored various filter sizes by using Keras and not using it. We went through each layer to understand how filter activations detected visually. Our neural network learned filters that activate when they see some type of visual features, such as a pattern of some orientation (diagonal, horizontal, vertical edges), or eventually entire honeycomb-like patterns on higher layers of the network. The earlier convolution or hidden layers acted as a collection of various edge and pattern detectors; also, the activations were still retaining almost all of the information present in the initial picture. However, the filter activations of the later layers became increasingly abstract and less visually interpretable.

References

Chollet, F. (2018). *Deep Learning with Python.* Manning. Retrieved March 01, 2020

Karpathy, A. (n.d.). *Convolutional Neural Networks (CNNs / ConvNets)*. (Stanford) Retrieved March 08, 2020, from http://cs231n.github.io/convolutional-networks/#conv