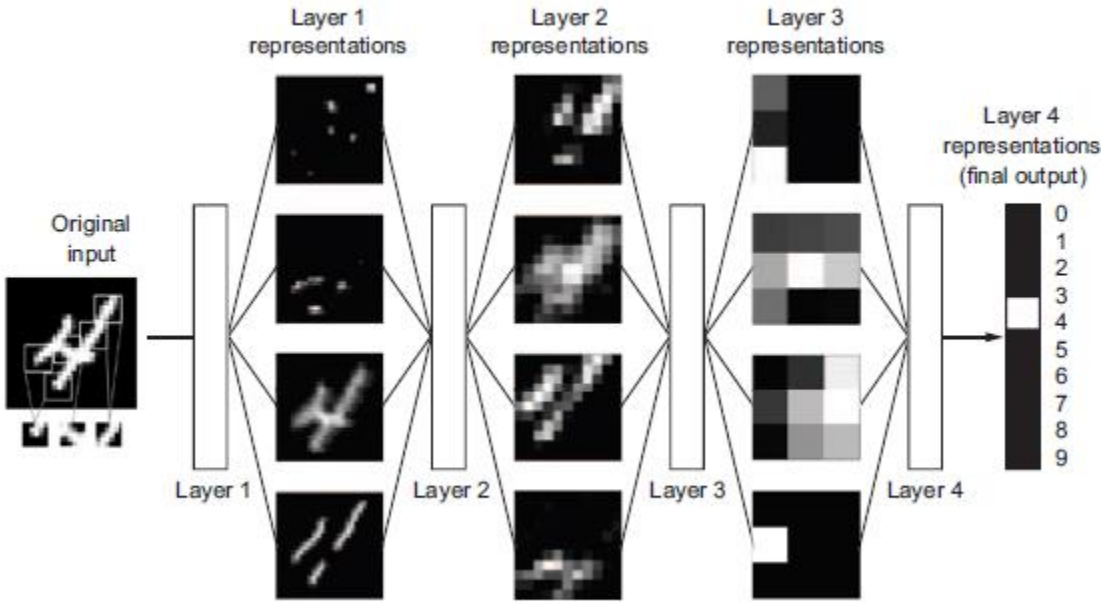
Alisher Siddikov

3/8/2020

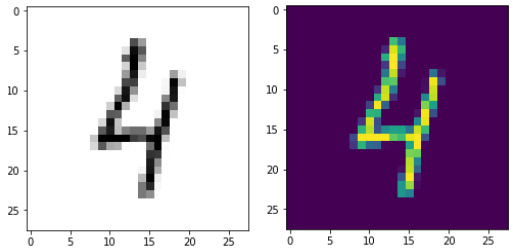
I would like to explore convnet architecture and visualize convnet intermediate activations using the MNSIT handwritten digit classification.

As we can see in the following image, 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).



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' appears to encode a diagonal edge detector.

Let's look at another closest digit 4 from the testing set, which is image index number 227.



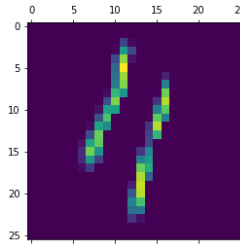
Here is the model summary; we can see that there are five activations. The activation of the first convolution layer starts with 26 by 26 (height x width) pixels and 32 channels, and the last convolution layers end with 3 by 3 (height x width) pixels and 64 channels. Each channel encodes relatively independent features.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_6 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

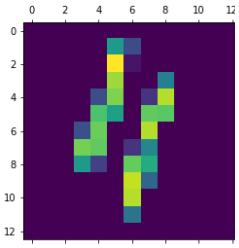
Let's try visualizing the edges by trying various channels. The 25th channel first layer activation detects vertical and diagonal lines. The second and fourth layers are max pooling, which downsamples the previous layers, and they looked zoomed-in images. The third layer activation is abstract and less visually interpretable. If we had 128 x 128 pixeled cat photo, this layer could have encoded higher-level concepts such as "cat ear" or "cat eye." The last layer 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.

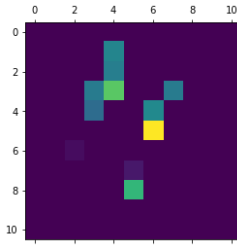
1st layer activation (None, 26, 26, 32)



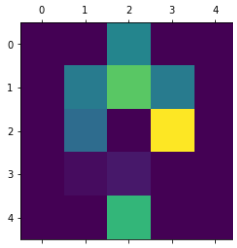
2nd layer activation (None, 13, 13, 32)



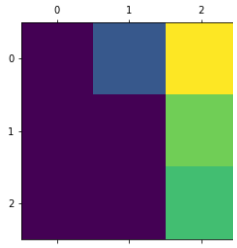
3rd layer activation (None, 11, 11, 64)



4th layer activation (None, 5, 5, 64)

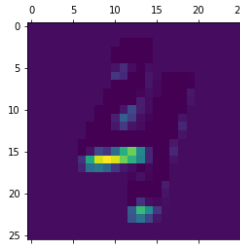


5th layer activation (None, 3, 3, 64)

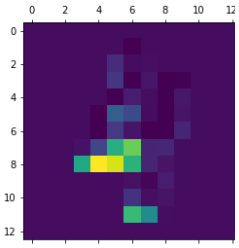


Let's try other channels and see what specific filters were learned by convolution layers. Channel number 7 of the first layer appeared to encode a horizontal edge, which was this filter learned by the first layer. This channel gives more emphasis on the horizontal edge. The last layer is blank because the pattern encoded by the filter isn't found in the input image.

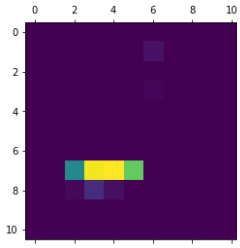
1st layer activation (None, 26, 26, 32)



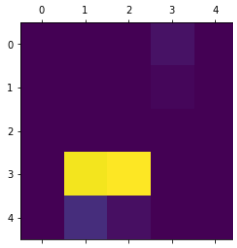
2nd layer activation (None, 13, 13, 32)



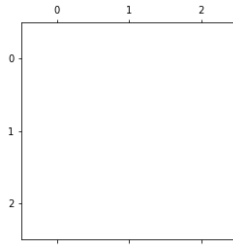
3rd layer activation (None, 11, 11, 64)



4th layer activation (None, 5, 5, 64)

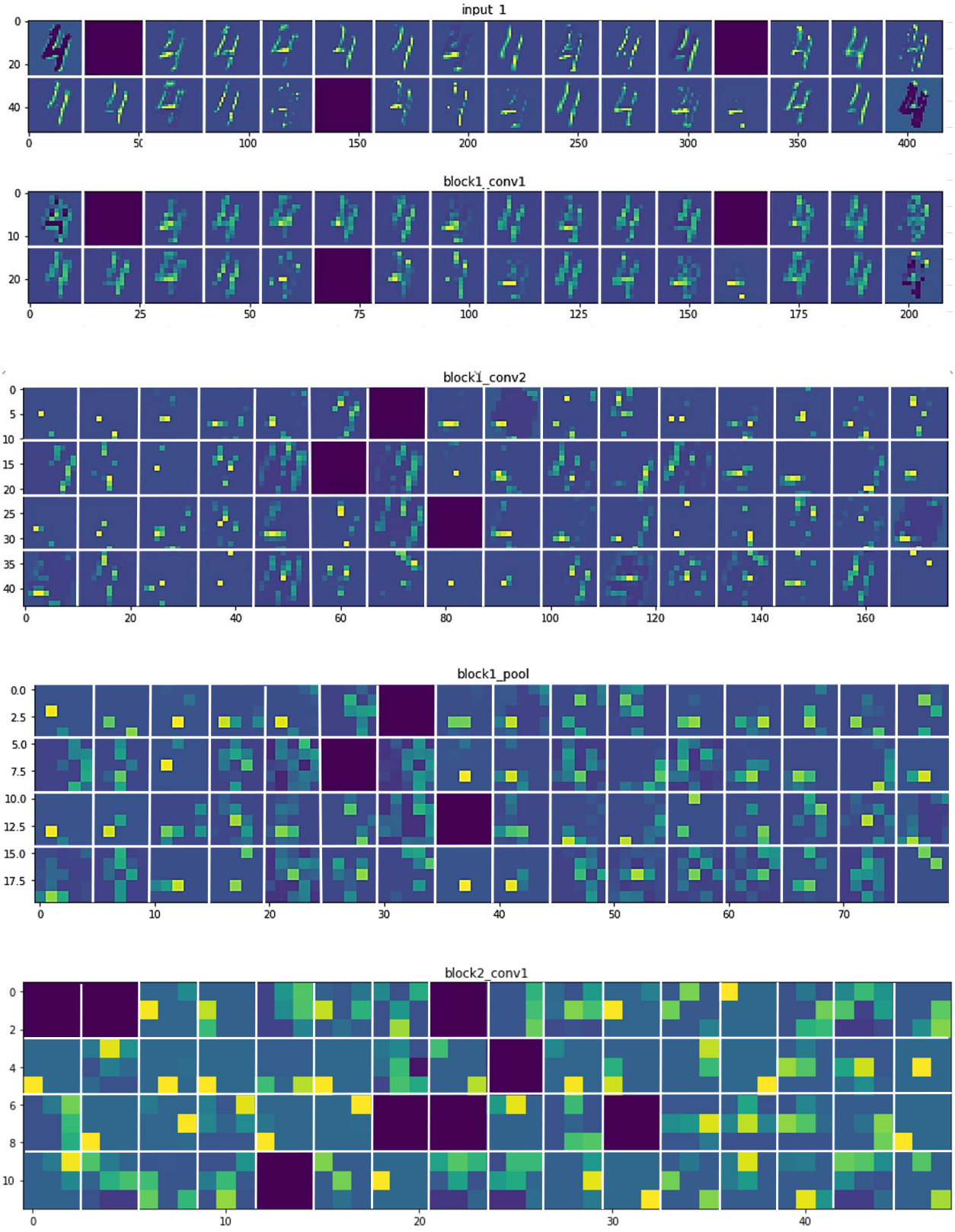


5th layer activation (None, 3, 3, 64)



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

The first layer acts as a collection of various edge detectors. As we go higher-up, the activations become increasingly abstract and less visually interpretable while filtering out irrelevant visual details.


input_1
block1_conv1
block1_conv2
block1_pool
block2_conv1

Let's explore the filter activations without using Keras package. I have three convolutions (or hidden) layers with ReLu activations, two pooling layers, and a fully connected output layer with softmax activation. I tweaked the number of convolutional filters at each layer. We will visualize filters to understand what features can be learned by tweaking the number of filters.

```
hidden_1 = tf.layers.conv2d(inputs = x_image,
                      activation = 'relu',
                      filters = 3,
                      kernel_size=[3,3],
                      padding = 'same')
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,
                      activation = 'relu',
                      filters = 6,
                      kernel_size=[3,3],
                      padding='same')
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,
                      activation = 'relu',
                      filters = 12,
                      kernel_size=[3,3],
                      padding='same')
hidden_3 = tf.nn.dropout(hidden_3, keep_prob)
```

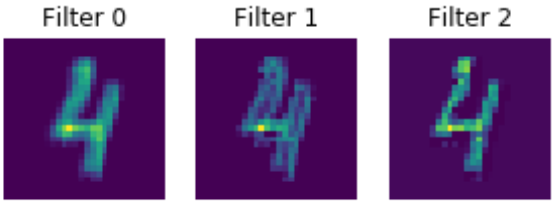
Here, each filter capable of extracting different features from the image. Conv2D parameter, filters determines the number of kernels to convolve with the input volume. Each of these operations produces a 2D activation map.

- Convolution/hidden layer 1: learned a total of 3 filters. Max pooling is then used to reduce the spatial dimensions of the output volume.

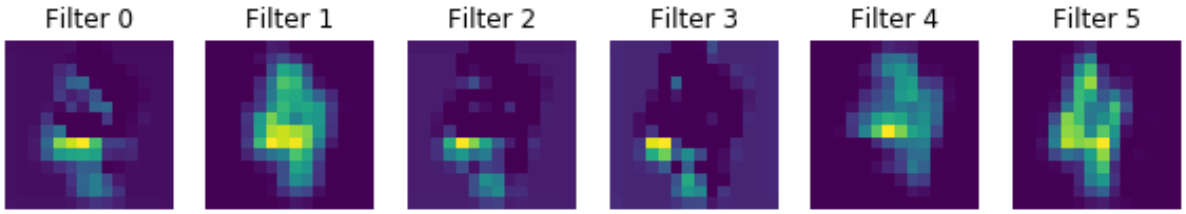
- Convolution/hidden layer 2: learned a total of 3*2=6 filters. Again, max pooling is used to reduce the spatial dimensions.
- The final Conv2D: layer learned 6*2=12 filters.

Notice at as our output spatial volume is decreasing our number of filters learned is increasing — this is a common practice in designing CNN architectures. I chose the same kernel size (3, 3), default strides value (1, 1) value, and padding = 'same' for all Conv2D layers.

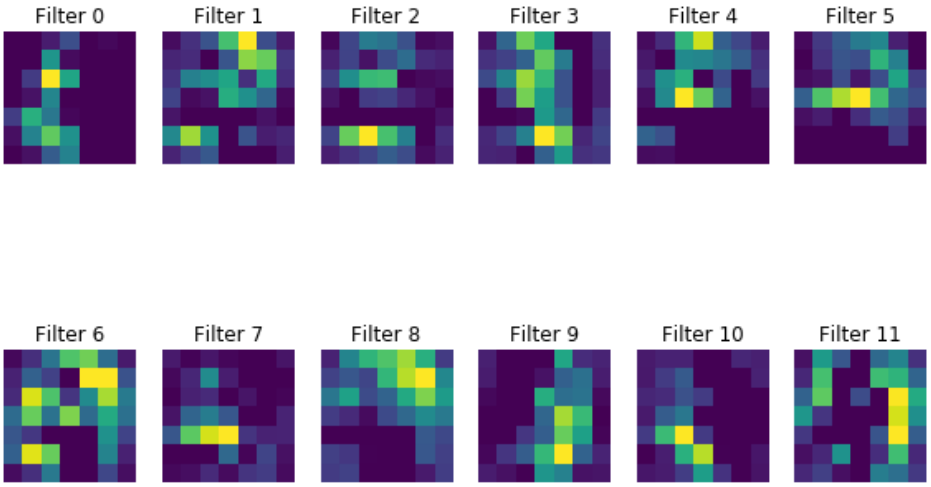
We can see how the digit image-4 activates the neurons of the first convolutional layer. We can see how each filter has learned to activate optimally for different features of the image.



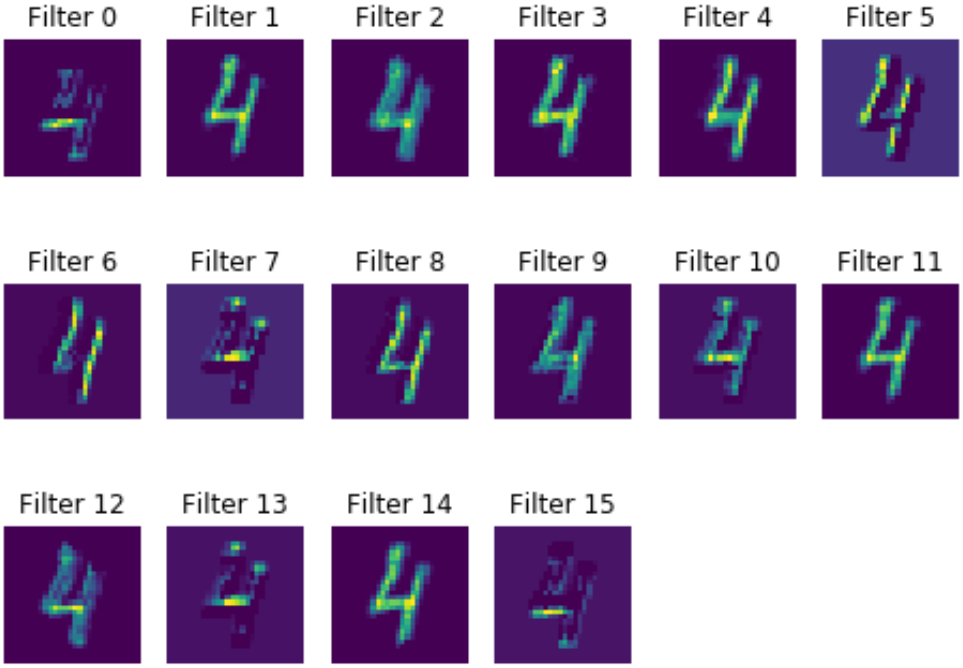
And here is how it activates the neurons of the second convolutional layer. This layer, which is in between, learned more filters than the previous layers but fewer filters than the layers closer to the output.



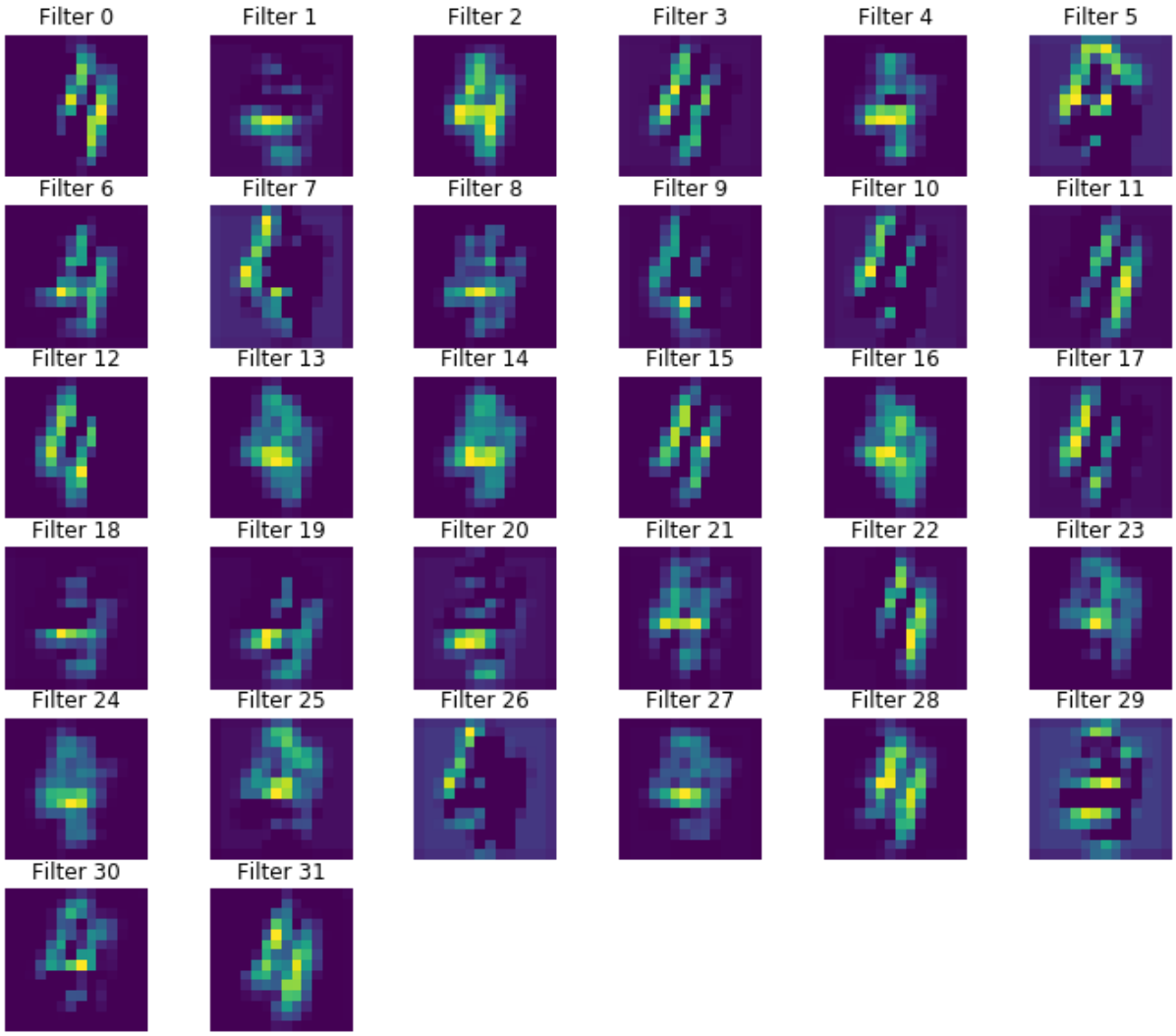
And here is how it activates the neurons of the third convolutional layer.



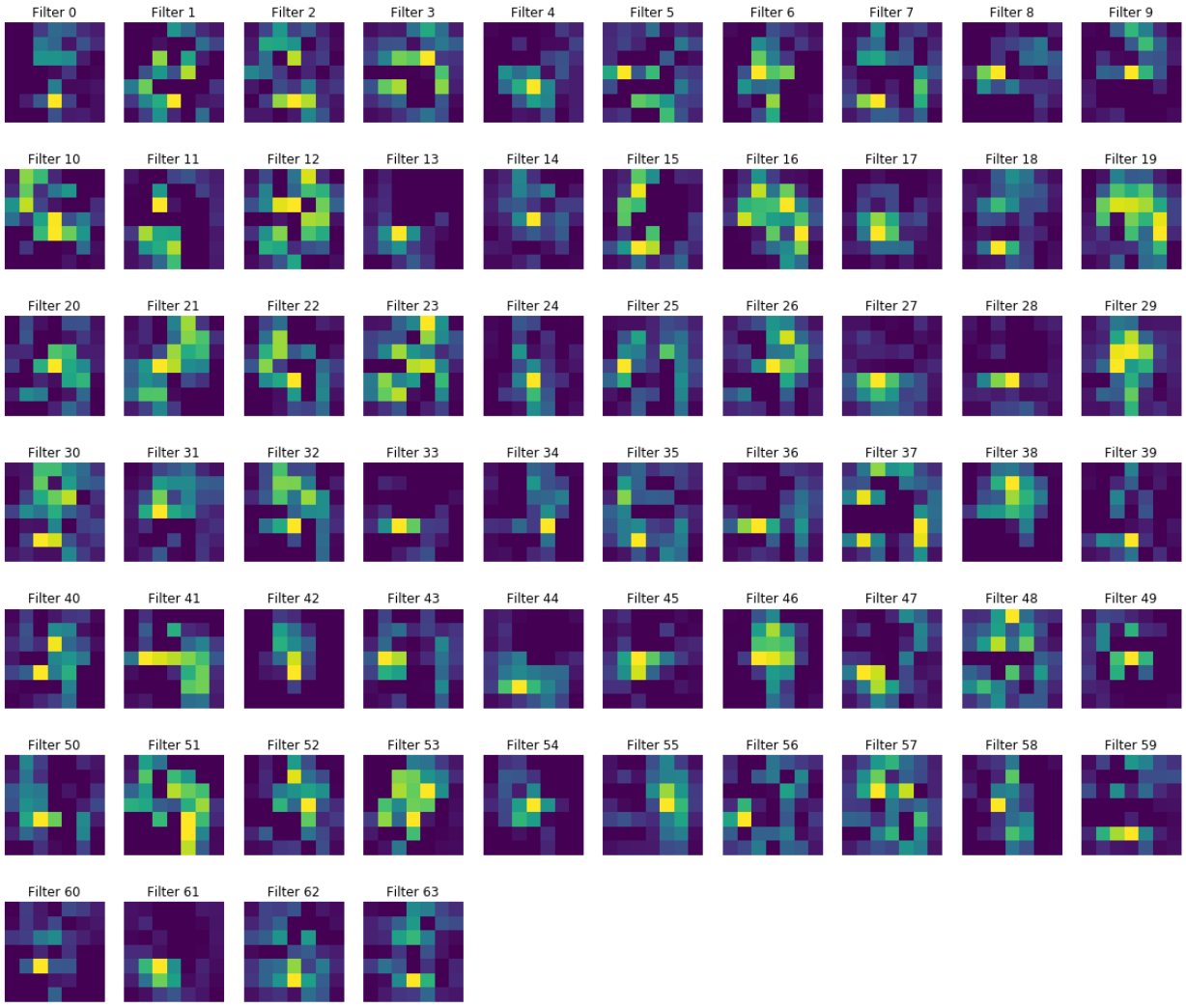
When I increased the filters from 3, 6, 9 to 16, 32, 64, respectively, the model test and train accuracy increased drastically. Now the first convolutional layer has 16 filters. We can see that the filter 6 detected vertical signals while filters 13 and 15 detected horizontal lines.



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



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



Summary

In this assignment, we explored channel by using Keras and TensorFlow by using filters to see how each layer detects what pattern. We went through each layer to understand what it was detecting. The earlier layer 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 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