Visualizing Convolutional Layers

The opaqueness of neural network models makes it difficult to explain why a specific decision or prediction was made. However, convolutional neural networks, which are intended for image data, may be less inscrutable due to their structure and function.

These models use small linear filters, which produce activation maps (or feature maps), both of which can be visualized. By visualizing the filters, such as line detectors, within a learned convolutional neural network, we may gain insight into the model's workings.

Additionally, analyzing the feature maps resulting from the filters applied to input images and prior layer feature maps can provide insight into the model's internal representation of a specific input at a given point.

In this project, we will explore both of these visualization approaches for convolutional neural networks.

Pre-fit ResNet Model

We need a model to visualize.

Instead of fitting a model from scratch, we can use a pre-fit prior state-of-the-art image classification model.

Keras provides many examples of well-performing image classification models developed by different research groups for the ImageNet Large Scale Visual Recognition Challenge, or ILSVRC. One example is the ResNet50V2 model.

We can load and summarize the model with just a few lines of code.

How to Visualize Filters

Perhaps the simplest visualization to perform is to plot the learned filters directly.

The weights in neural networks, when referring to learned filters, have a two-dimensional structure that gives them a spatial relationship to each other. Therefore, these weights can be represented as two-dimensional images, which is a meaningful way to plot them.

The first step is to review the filters in the model, to see what we have to work with.

The model summary printed in the previous section summarizes the output shape of each layer, e.g. the shape of the resulting feature maps. It does not give any idea of the shape of the filters (weights) in the network, only the total number of weights per layer.

We can access all of the layers of the model via the *model.layers* property.

Each layer has a *layer.name* property, where the convolutional layers have a naming convolution like *block#_#_conv#*, where the '*#*' is an integer. Therefore, we can check the name of each layer and skip any that don't contain the string '*_conv*'.

```
conv1_conv (7, 7, 3, 64)
conv2_block1_1_conv (1, 1, 64, 64)
conv2_block1_2_conv (3, 3, 64, 64)
conv2_block1_0_conv (1, 1, 64, 256)
conv2_block1_3_conv (1, 1, 64, 256)
conv2_block2_1_conv (1, 1, 256, 64)
conv2_block2_2_conv (3, 3, 64, 64)
conv2_block2_3_conv (1, 1, 64, 256)
conv2_block3_1_conv (1, 1, 256, 64)
conv2_block3_2_conv (3, 3, 64, 64)
conv2_block3_3_conv (1, 1, 64, 256)
conv3_block1_1_conv (1, 1, 256, 128)
conv3_block1_2_conv (3, 3, 128, 128)
conv3_block1_0_conv (1, 1, 256, 512)
conv3_block1_3_conv (1, 1, 128, 512)
conv3_block2_1_conv (1, 1, 512, 128)
conv3_block2_2_conv (3, 3, 128, 128)
conv3_block2_3_conv (1, 1, 128, 512)
conv3_block3_1_conv (1, 1, 512, 128)
conv3_block3_2_conv (3, 3, 128, 128)
conv3_block3_3_conv (1, 1, 128, 512)
conv3_block4_1_conv (1, 1, 512, 128)
conv3_block4_2_conv (3, 3, 128, 128)
conv3_block4_3_conv (1, 1, 128, 512)
```

An architectural concern with a convolutional neural network is that the depth of a filter must match the depth of the input for the filter (e.g. the number of channels).

We can see that for the input image with three channels for red, green and blue, that each filter has a depth of three (here we are working with a channel-last format). We could visualize one filter as a plot with three images, one for each channel.

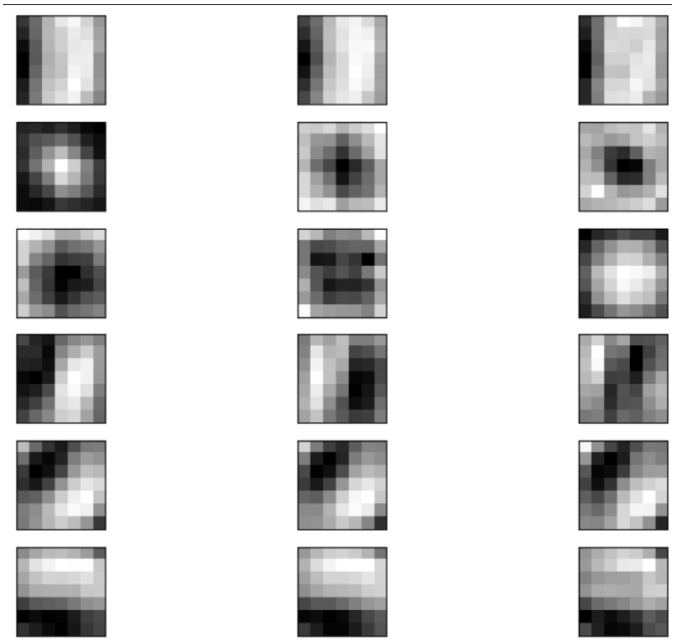
We can retrieve the filters from the first layer.

The weight values will likely be small positive and negative values centered around 0.0.

We can normalize their values to the range 0-1 to make them easy to visualize.

Now we can enumerate the first six filters out of the 64 in the block and plot each of the three channels of each filter.

We use the matplotlib library and plot each filter as a new row of subplots, and each filter channel or depth as a new column.



We have displayed only the first 6 filters. One row for each filter and one column for each channel.

The dark squares indicate small or inhibitory weights and the light squares represent large or excitatory weights.

The intuition of some filters is reported below (row-column):

- (1-1, 1-2, 1-3): We can see that the filters on the first row detect a gradient from light in the right to dark in the left.
- (2-1, 3-3): We see that the filters detect a gradient from light in the center to dark in the corners.
- (2-2, 2-3, 3-1): We can see that the filters detect a gradient from dark in the center to light in the corners.
- (4-1, 5-1, 5-2, 5-3): The filters detect a gradient from dark in the top left to light in the bottom right.
- (4-2): The filters detect a gradient from light in the top left to dark in the bottom right.
- (6-1, 6-2, 6-3): The filters detect a gradient from dark in the bottom to light in the top.

Although we have a visualization, we only see the first six of the 64 filters in the first convolutional layer. Visualizing all 64 filters in one image is feasible.

Sadly, this does not scale; if we wish to start looking at filters in the second convolutional layer, we can see that again we have 64 filters, but each has 64 channels to match the input feature maps. To see all 64 channels in a row for all 64 filters would require (64×64) 4,096 subplots in which it may be challenging to see any detail.

How to Visualize Feature Maps

The activation maps, called feature maps, capture the result of applying the filters to input, such as the input image or another feature map.

The idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps. The expectation would be that the feature maps close to the input detect small or fine-grained detail, whereas feature maps close to the output of the model capture more general features.

In order to explore the visualization of feature maps, we need input for the ResNet50V2 model that can be used to create activations. We will use a simple photograph of a bird.

The picture of the bird is in the "./img" directory with the filename '*bird.jpg*'.

Next, we need a clearer idea of the shape of the feature maps output by each of the convolutional layers and the layer index number so that we can retrieve the appropriate layer output.

```
2 conv1_conv (None, 112, 112, 64)
11 conv2_block1_2_conv (None, 56, 56, 64)
23 conv2_block2_2_conv (None, 56, 56, 64)
34 conv2_block3_2_conv (None, 28, 28, 64)
46 conv3_block1_2_conv (None, 28, 28, 128)
58 conv3_block2_2_conv (None, 28, 28, 128)
69 conv3_block3_2_conv (None, 28, 28, 128)
80 conv3_block4_2_conv (None, 14, 14, 128)
92 conv4_block1_2_conv (None, 14, 14, 256)
104 conv4_block2_2_conv (None, 14, 14, 256)
115 conv4_block3_2_conv (None, 14, 14, 256)
126 conv4_block4_2_conv (None, 14, 14, 256)
137 conv4_block5_2_conv (None, 14, 14, 256)
148 conv4_block6_2_conv (None, 7, 7, 256)
160 conv5_block1_2_conv (None, 7, 7, 512)
172 conv5_block2_2_conv (None, 7, 7, 512)
183 conv5_block3_2_conv (None, 7, 7, 512)
```

We see the same output shapes as we saw in the model summary, but in this case only for the convolutional layers (7*7 and 3*3 Filters).

Note: We did not consider filters with size 1*1.

We can use this information and design a new model that is a subset of the layers in the full ResNet50V2 model. The model would have the same input layer as the original model, but the output would be the output of a given convolutional layer, which we know would be the activation of the layer or the feature map.

After loading the ResNet50V2 model, we can define a new model that outputs a feature map from the first convolutional layer (index 2).

Making a prediction with this model will give the feature map for the first convolutional layer for a given provided input image.

After defining the model, we need to load the bird image with the size expected by the model, in this case, 224×224.

Next, the image PIL object needs to be converted to a NumPy array of pixel data and expanded from a 3D array to a 4D array with the dimensions of [*samples, rows, cols, channels*], where we only have one sample.

The pixel values then need to be scaled appropriately for the ResNet50V2 model.

We are now ready to get the feature map. We can do this easy by calling the *model.predict()* function and passing in the prepared single image.

We know the result will be a feature map with 224x224x64. We can plot all 64 two-dimensional images as an 8×8 square of images.

Note: this model is much smaller than the ResNet50V2 model, but still uses the same weights (filters) in the first convolutional layer as the ResNet50V2 model.

Next, a figure is created that shows all 64 feature maps as subplots.

We can see that the result of applying the filters in the first convolutional layer is a lot of versions of the bird image with different features highlighted.

Some filters have separated the edges of the image. Some filters distinguish the horizontal edges and others the vertical edges of the image.

For example, some highlight lines, other focus on the background or the foreground.

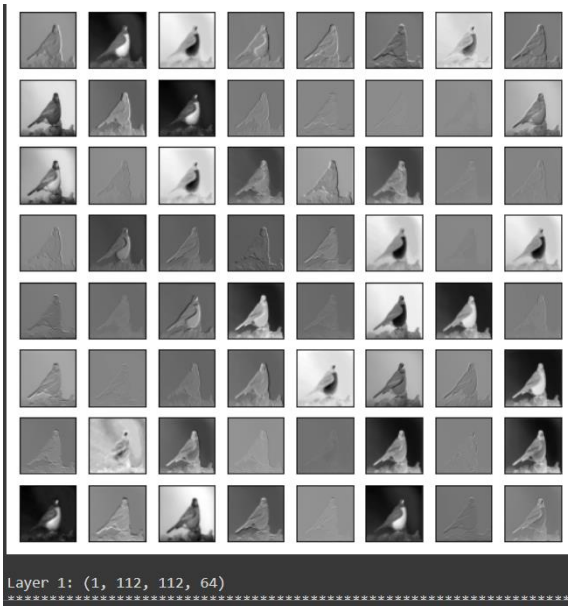
This is an interesting result and generally matches our expectation. We could update the example to plot the feature maps from the output of other specific convolutional layers.

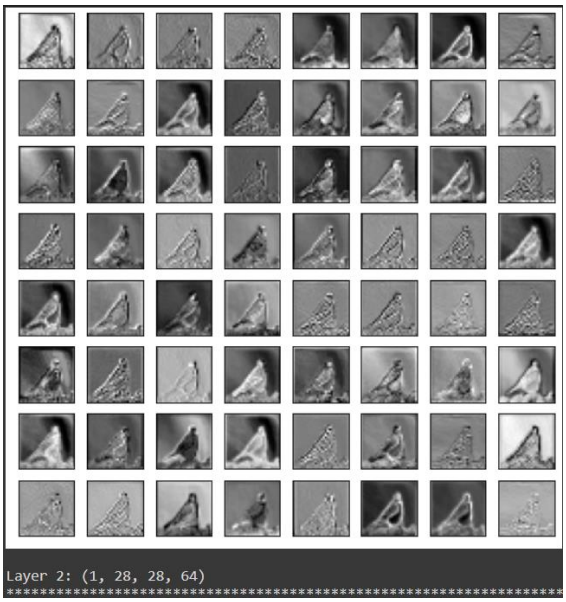
Another approach would be to collect feature maps output from each block of the model in a single pass, then create an image of each.

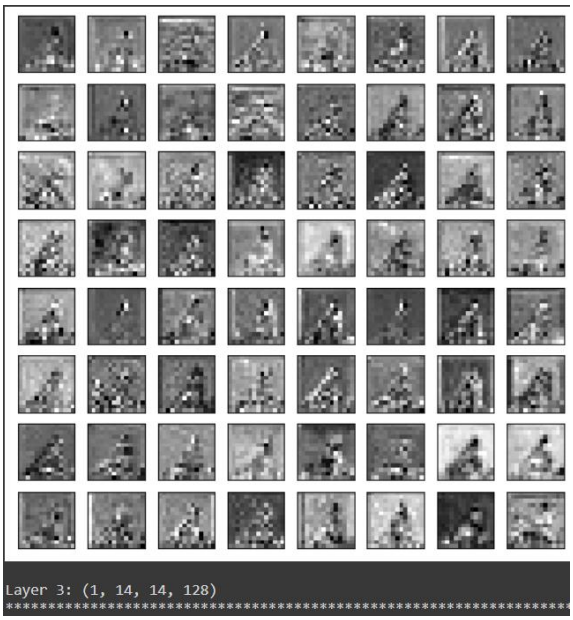
There are five main blocks in the model (e.g. conv1, conv2, etc.). The layer indexes of the last convolutional layer in each block are [2, 34, 80, 148, 183].

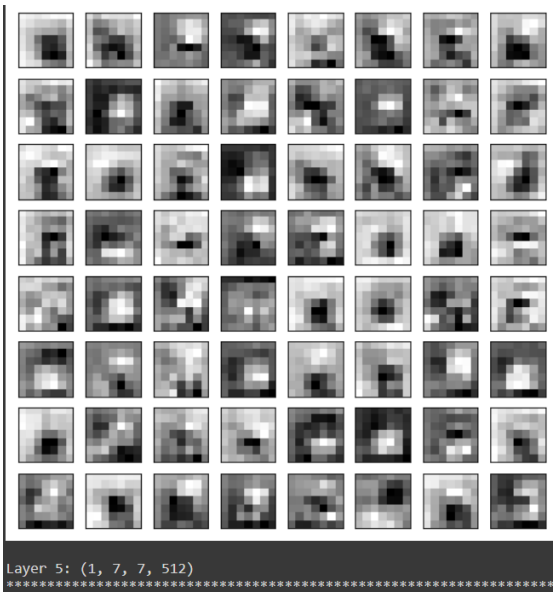
We can define a new model that has multiple outputs, one feature map output for each of the last convolutional layer in each block.

Running the example results in five plots showing the feature maps from the five main blocks of the ResNet50V2 model.









We know that the number of feature maps (e.g. depth or number of channels) in deeper layers is much more than 64, such as 256 or 512. Nevertheless, we can cap the number of feature maps visualized at 64 for consistency.

The output of 64 convolutional filters for 5 layers is shown.

We can see that the feature maps closer to the input of the model capture a lot of fine detail in the image and that as we progress deeper into the model, the feature maps show less and less detail.

This pattern was to be expected, as the model abstracts the features from the image into more general concepts that can be used to make a classification.

Although it is not clear from the final image that the model saw a bird, we generally lose the ability to interpret these deeper feature maps but some of the filter outputs are clearly focused on the bird part.