Convolutional Neural Networks (CNNs / ConvNets)

Problem

- An image with the size of 32x32x3 (32 width, 32 height, 3 color channels).
- So a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly a fully-connected structure does not scale to larger images.
- Assume an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights.
- Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks

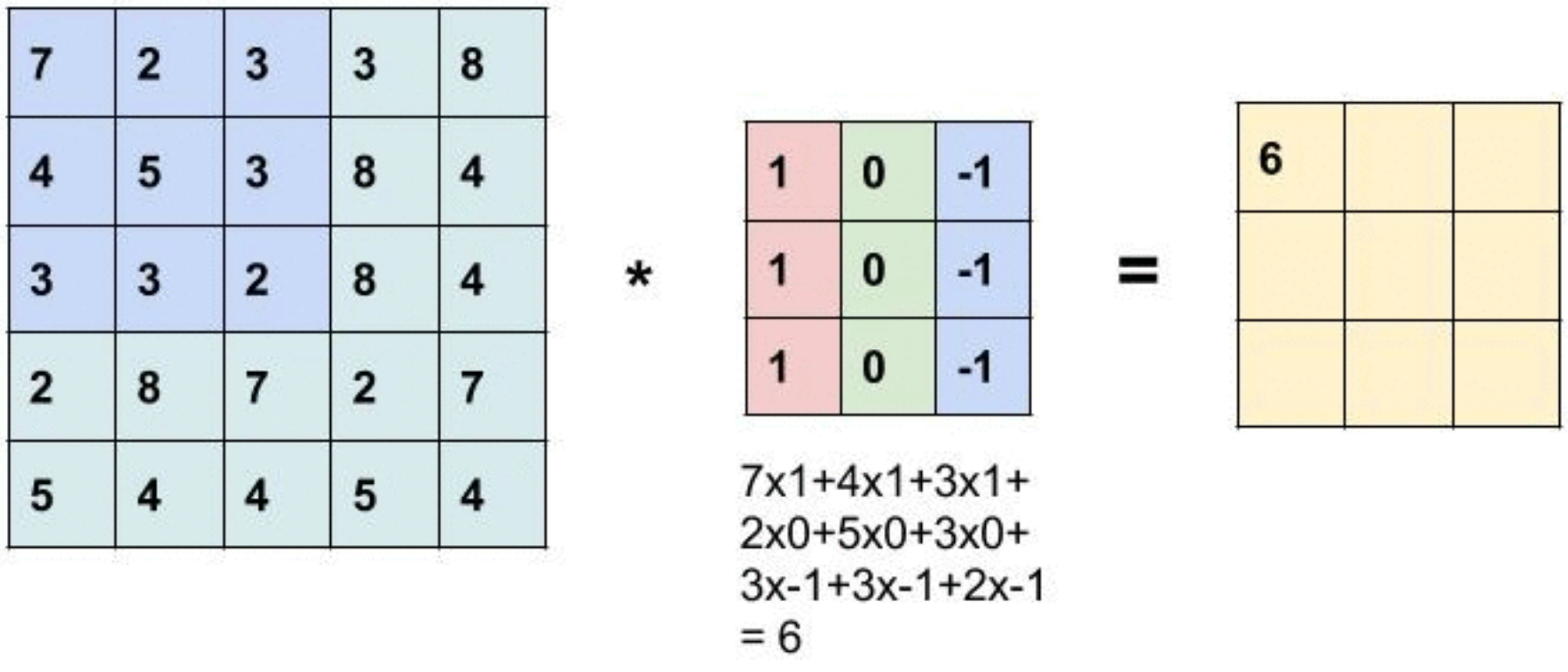
- Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way.
- Assume the input image has the dimension of 32x32x3 (width, height, depth respectively).
- In ConvNet, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.

Convolutional Layer (CONV) - 1

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels).
- During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.

Convolutional Layer (CONV) - 2

- As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.
- Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map.
- We will stack these activation maps along the depth dimension and produce the output volume.


7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6
7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

Convolutional Layer (CONV) - 3

- For example, suppose that the input volume has size 32x32x3.
- If the filter size is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter).

Stride

- Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data.
- Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video.
- For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

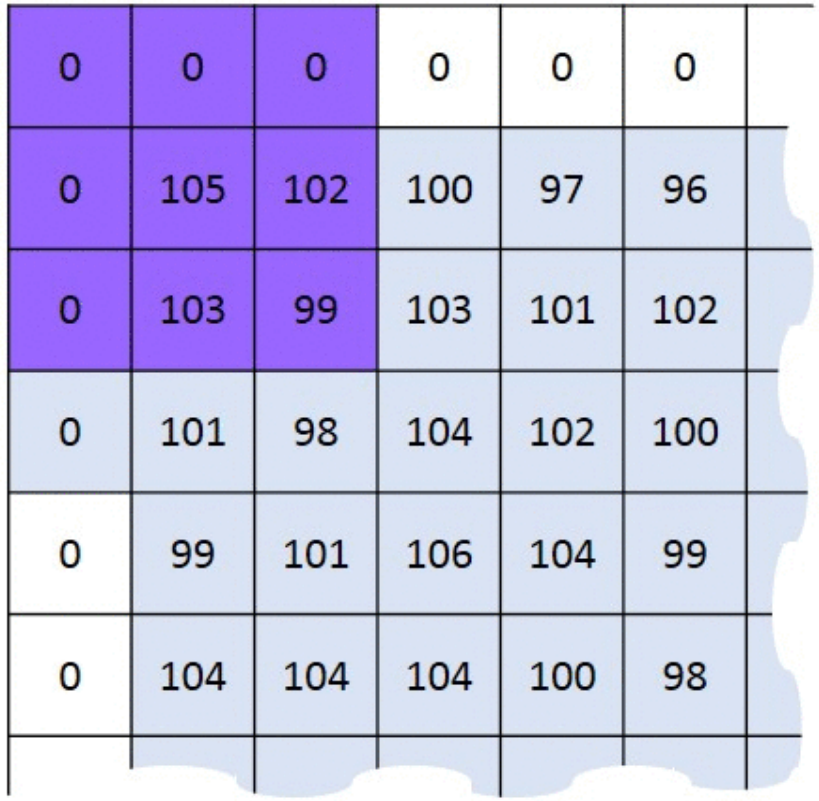
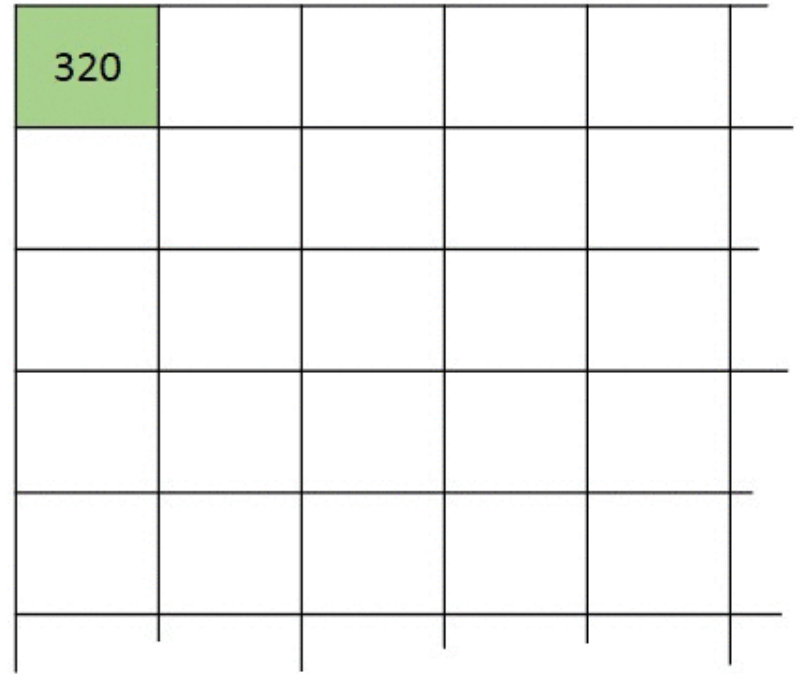


Image Matrix

Kernel Matrix

0	-1	0
-1	5	-1
0	-1	0



Output Matrix

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 \quad = 320$$

Convolution with horizontal and vertical strides = 1

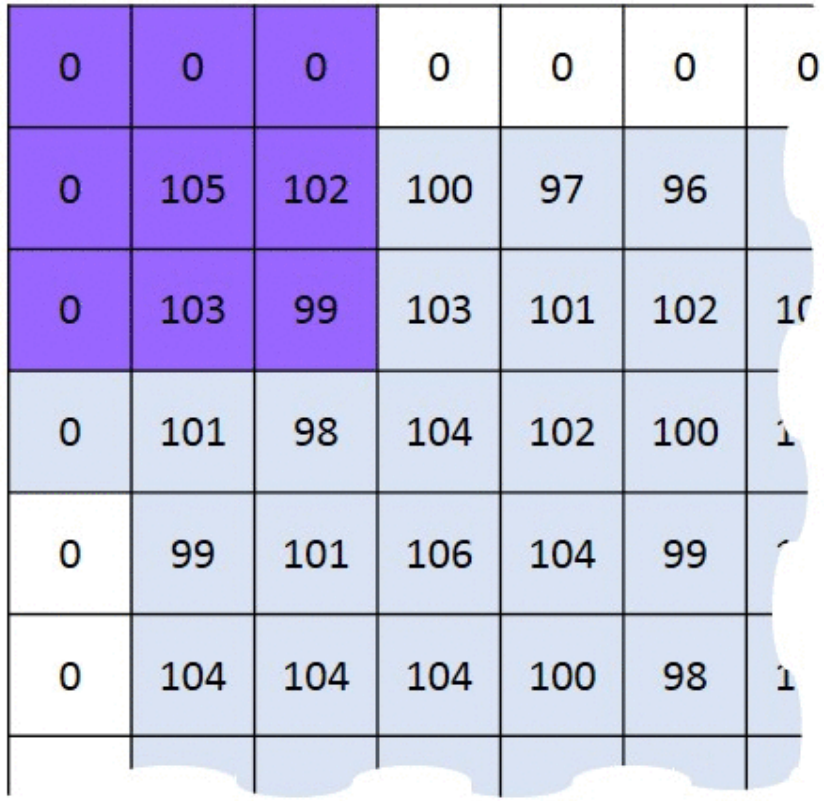
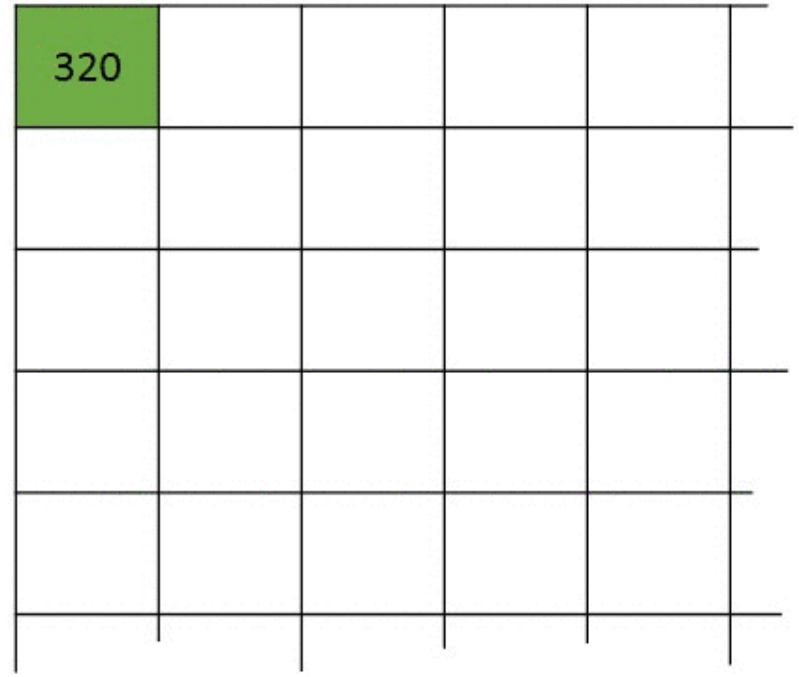


Image Matrix

Kernel Matrix

0	-1	0
-1	5	-1
0	-1	0



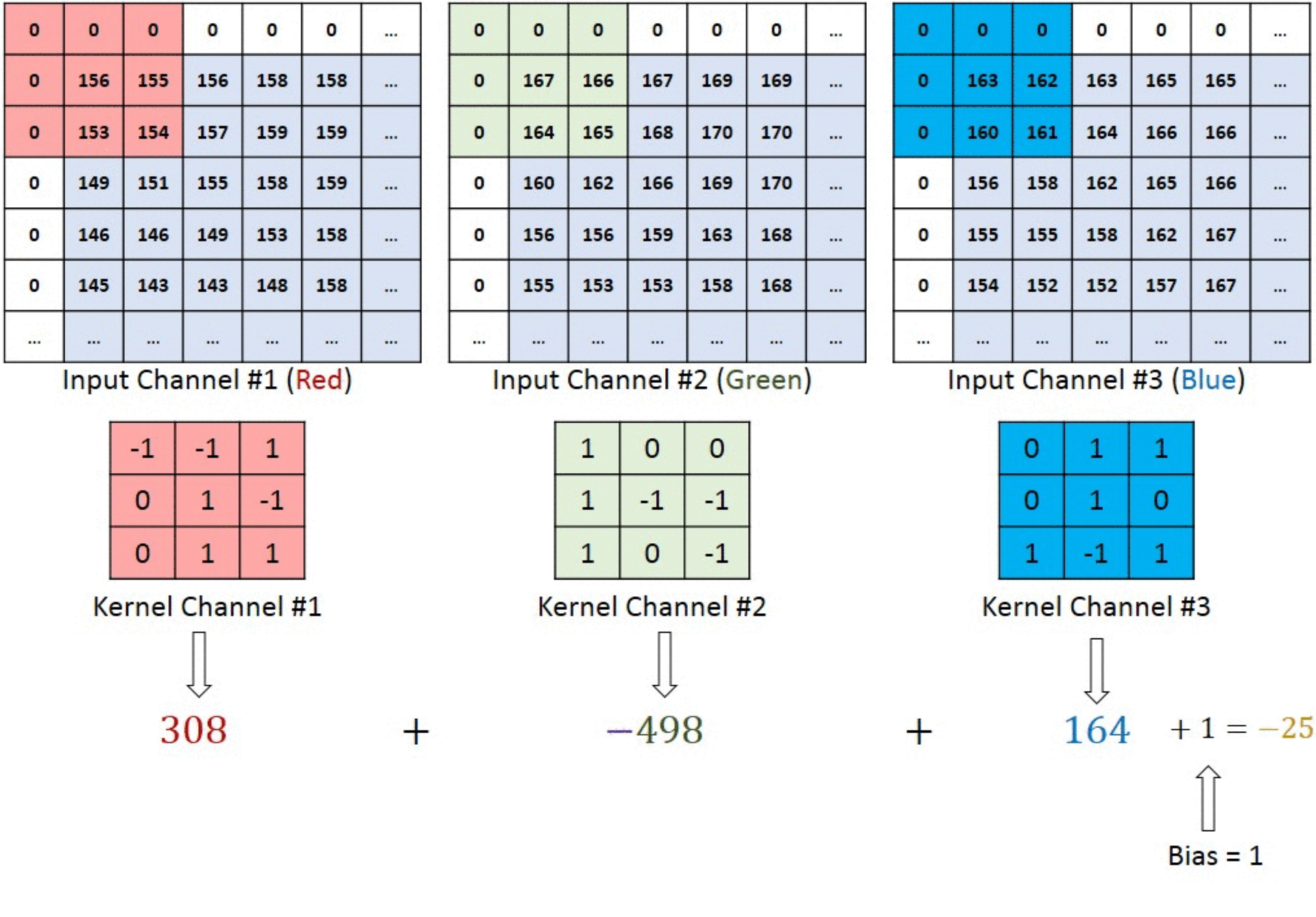
Output Matrix

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 \quad = 320$$

Convolution with horizontal and vertical strides = 2

Padding

- Let’s think about a scenario: What happens when you apply three 5 x 5 x 3 filters to a 32 x 32 x 3 input volume? The output volume would be 28 x 28 x 3.
- Notice that the spatial dimensions decrease. As we keep applying conv layers, the size of the volume will decrease faster than we would like.
- In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features.
- Let’s say we want to apply the same conv layer but we want the output volume to remain 32 x 32 x 3. To do this, we can apply a zero padding of size *n* to that layer. Zero padding pads the input volume with zeros around the border. If we think about a zero padding of 2, then this would result in a 36 x 36 x 3 input volume.


0 0 0 0 0 0 ...
0 156 155 156 158 158 ...
0 153 154 157 159 159 ...
0 149 151 155 158 159 ...
0 146 146 149 153 158 ...
0 145 143 143 148 158 ...
... ...
Input Channel #1 (Red)
0 0 0 0 0 0 ...
0 167 166 167 169 169 ...
0 164 165 168 170 170 ...
0 160 162 166 169 170 ...
0 156 156 159 163 168 ...
0 155 153 153 158 168 ...
... ...
Input Channel #2 (Green)
0 0 0 0 0 0 ...
0 163 162 163 165 165 ...
0 160 161 164 166 166 ...
0 156 158 162 165 166 ...
0 155 155 158 162 167 ...
0 154 152 152 157 167 ...
... ...
Input Channel #3 (Blue)
-1 -1 1
0 1 -1
0 1 1
Kernel Channel #1
1 0 0
1 -1 -1
1 0 -1
Kernel Channel #2
0 1 1
0 1 0
1 -1 1
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25 ... ...
... ...
... ...
... ...
... ...

What is the Dimension of Output?

- We can compute the spatial size of the output volume as a function of:
  - Input volume size (W)
  - The receptive field size (filter size) of the Conv Layer neurons (F)
  - The stride with which they are applied (S)
  - The amount of zero padding used (P) on the border
- The dimension of the output is given by:

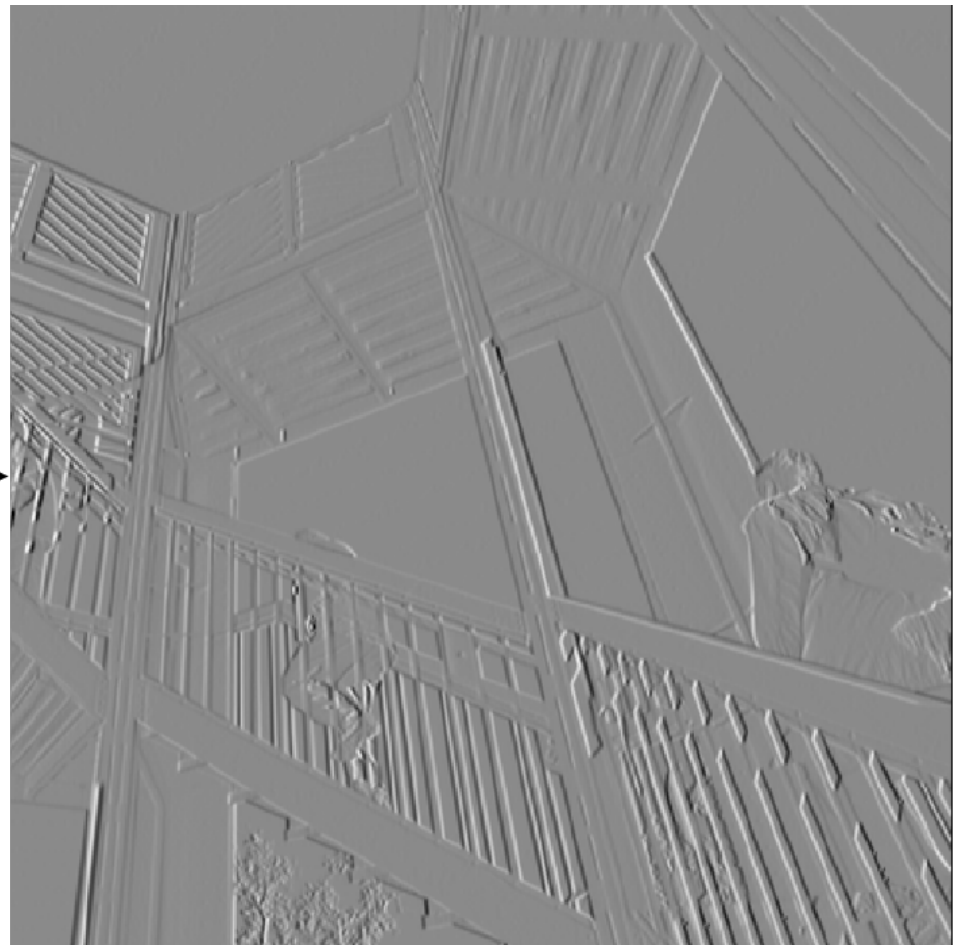
$$\left(\frac{W - F + 2P}{S}\right) + 1$$

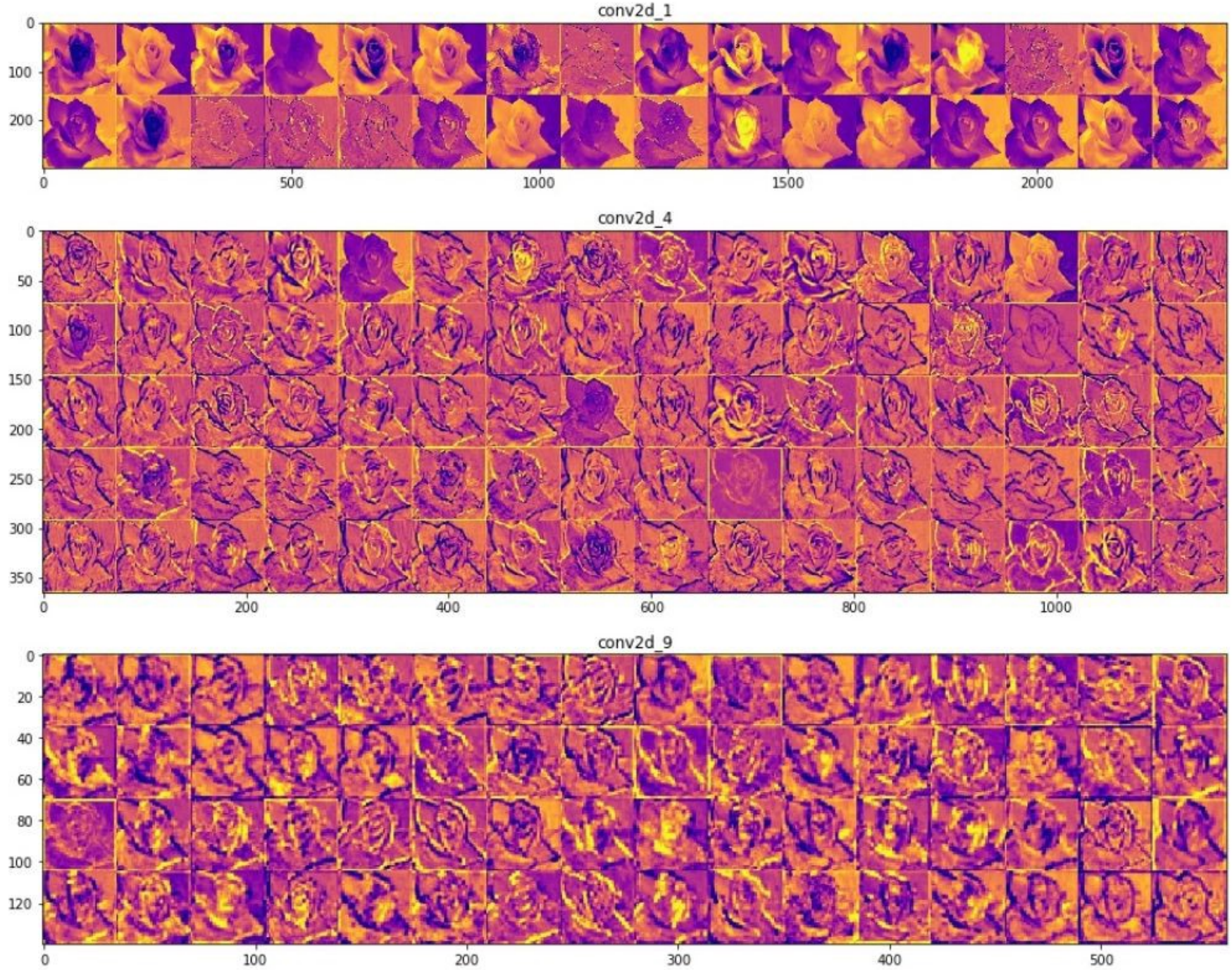
Summary of a Conv Layer

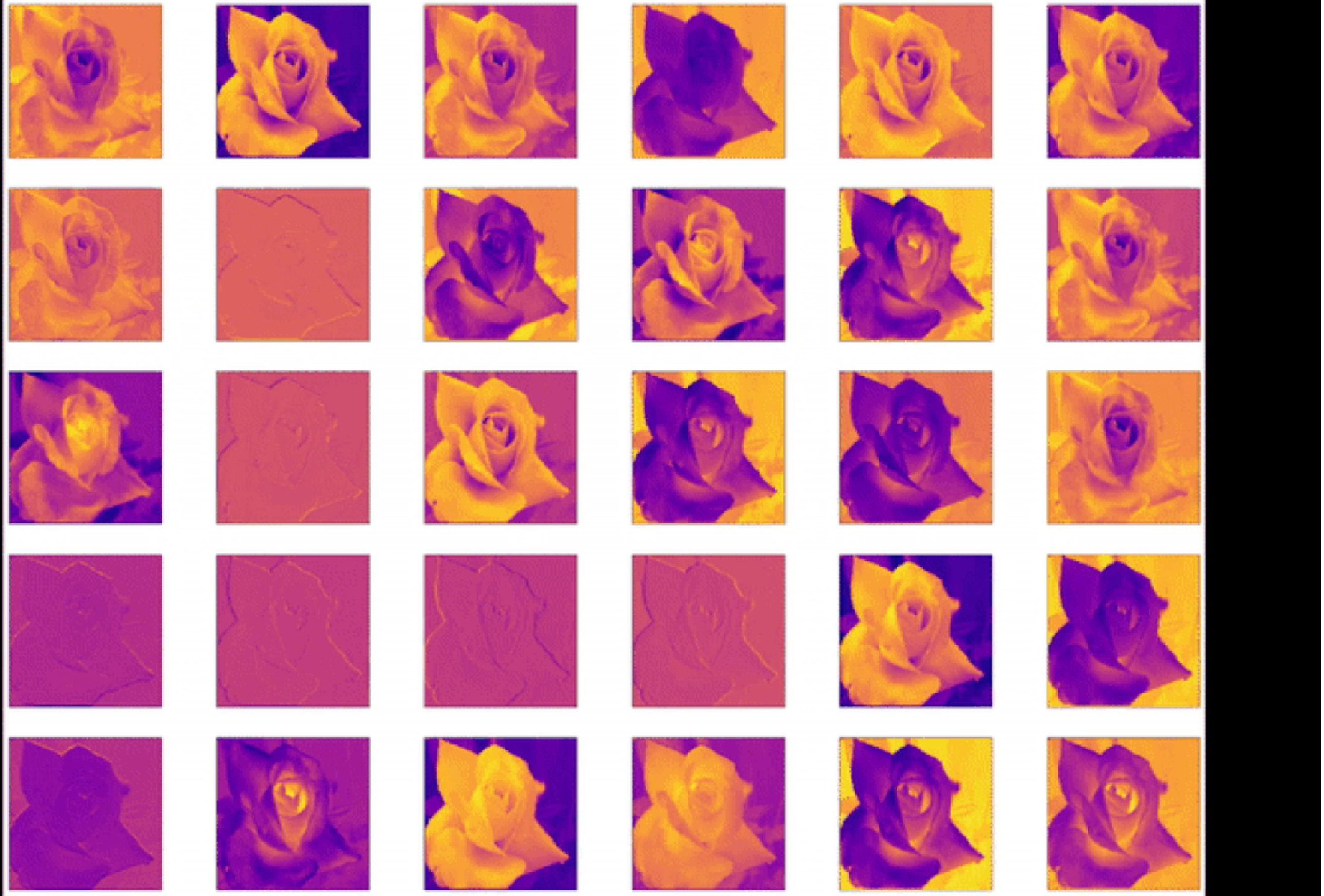
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - Their spatial extent (size of the filter) F,
  - The stride S,
  - The amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \left(\frac{W_1 - F + 2P}{S}\right) + 1$
  - $H_2 = \left(\frac{H_1 - F + 2P}{S}\right) + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F.F.D_1$ weights per filter, for a total of $(F.F.D_1).K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

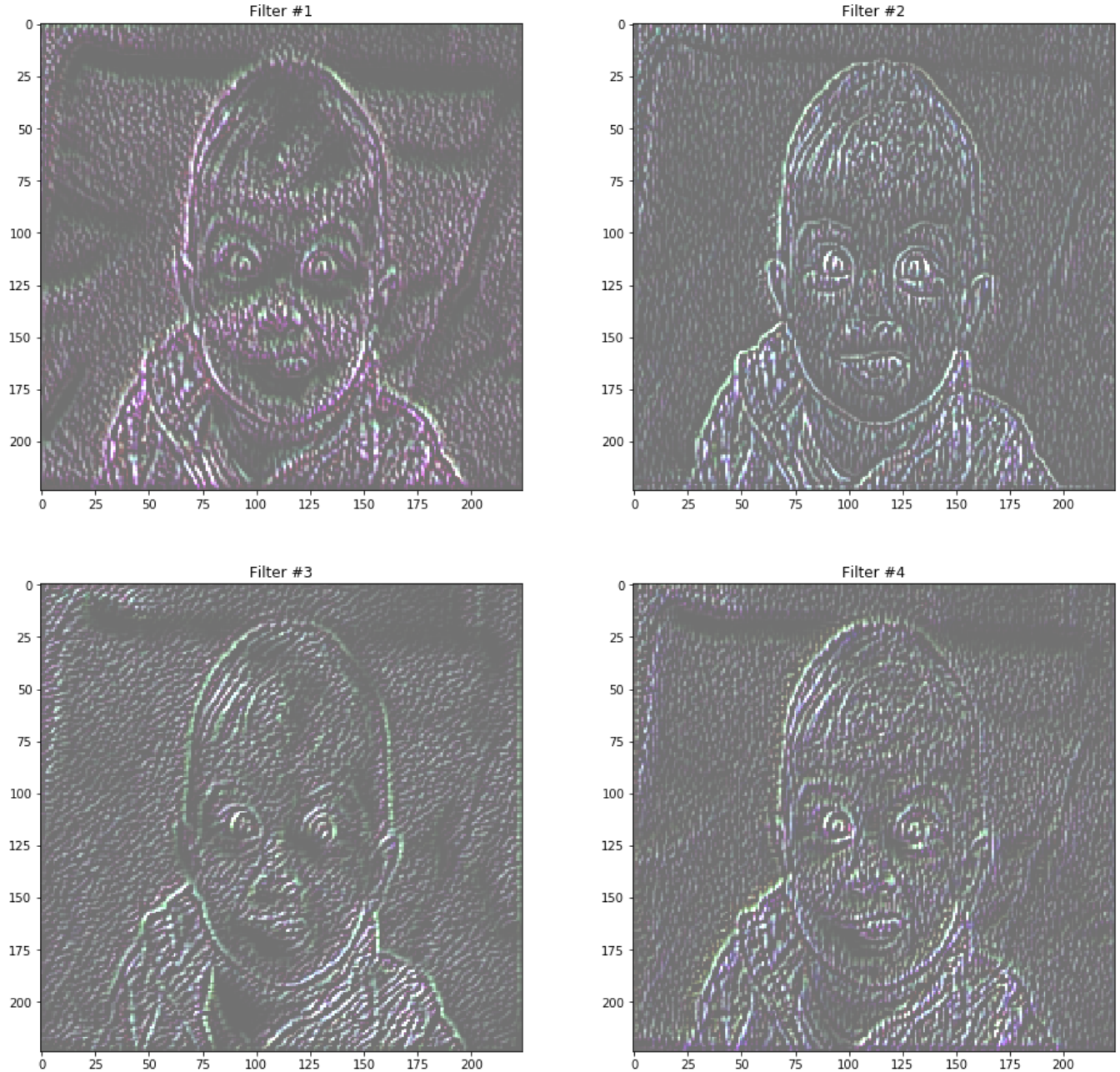
Horizontal Sobel kernel



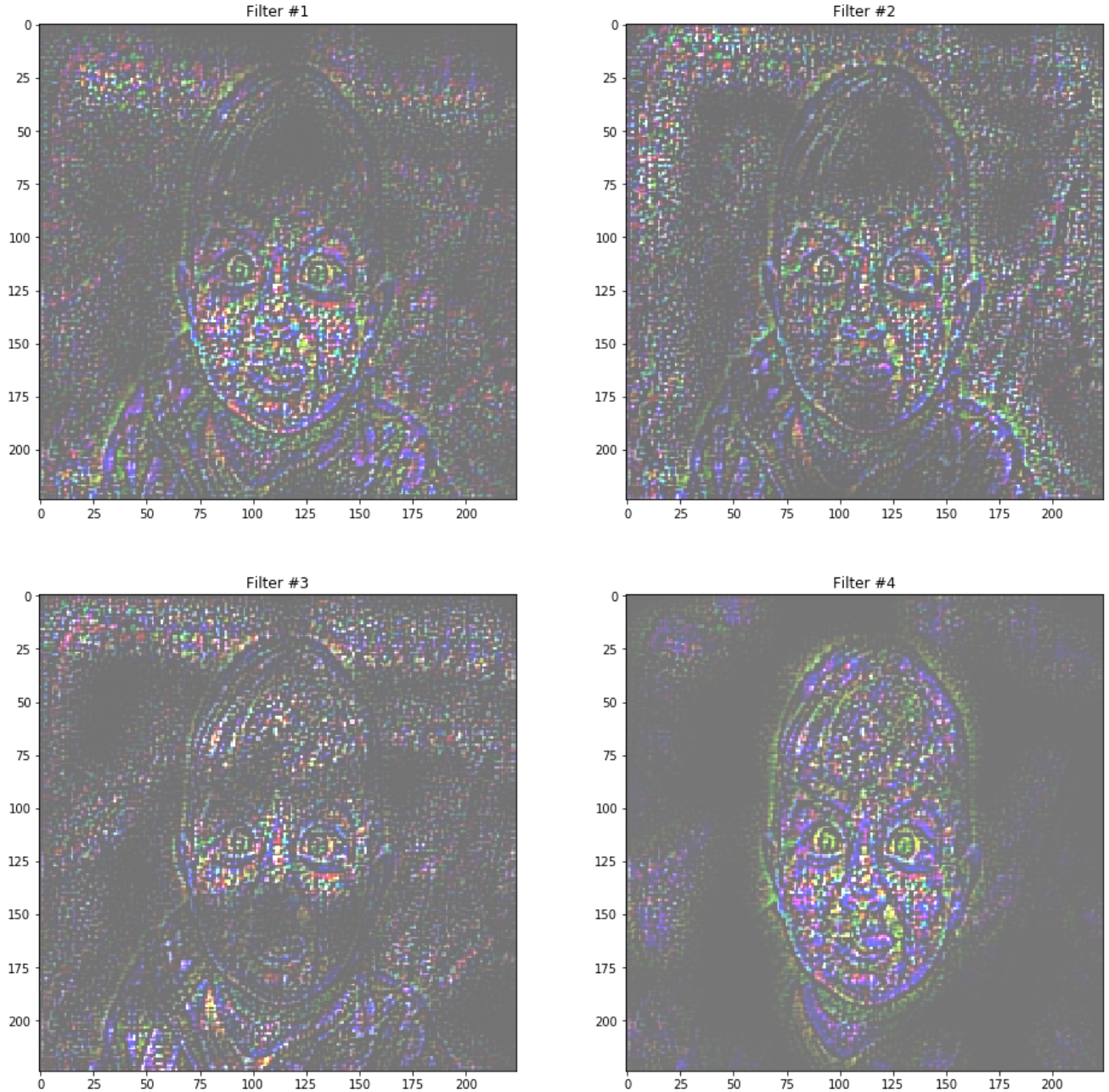

conv2d_1
0
100
200
0
500
1000
1500
2000
conv2d_4
0
50
100
150
200
250
300
350
0
200
400
600
800
1000
conv2d_9
0
20
40
60
80
100
120
0
100
200
300
400
500



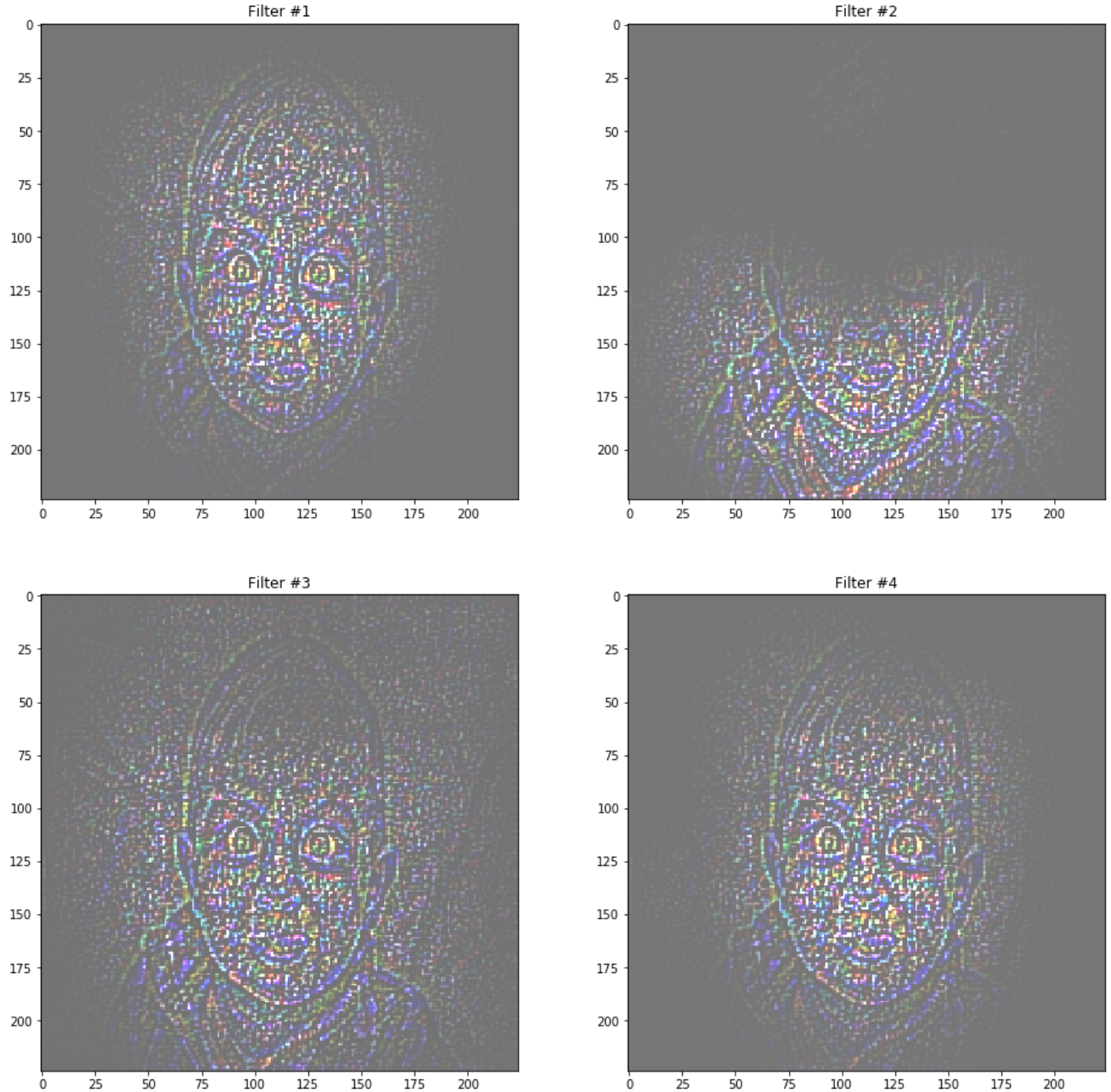
block2_conv1



block3_conv3



block5_conv1



Visualization Goal

- By visualizing the output from different convolution layers in this manner, the most crucial thing that you will notice is that the layers that are deeper in the network visualize more training data specific features, while the earlier layers tend to visualize general patterns like edges, texture, background etc.

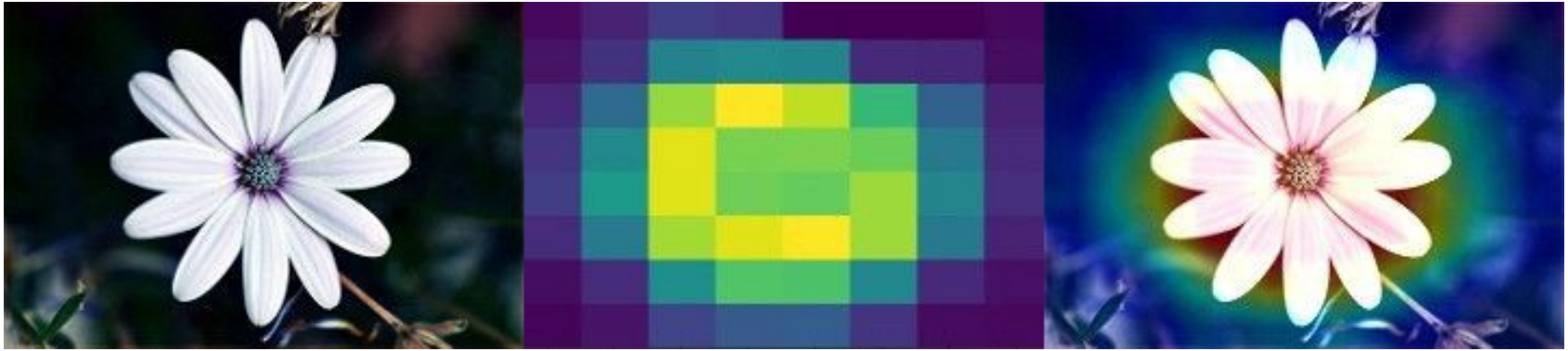
Visualizing Heatmaps of class activations

- While predicting the class labels for images, sometimes your model will predict wrong label for your class, i.e. the probability of the right label will not be maximum. In cases such as these, it will be helpful if you could visualize which parts of the image is your convnet looking at and deducing the class labels.

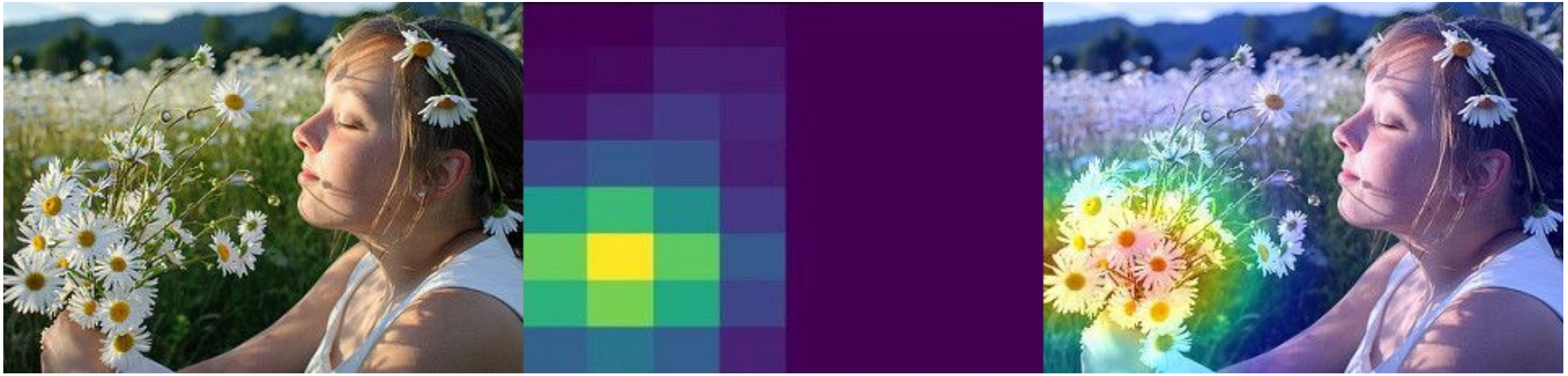
Class Activation Map

- The general category of such techniques is called **Class Activation Map** (CAM) visualization. One of the techniques of using CAM is to producing heatmaps of class activations over input images. A class activation heatmap is a 2D grid of scores associated with a particular output class, computed for every location for an input image, indicating how important is each location is with respect to that output class.

- What the heatmap is trying to tell us is the locations in the image which are important for that particular layer to classify it as the target class, which is **Daisy** in this case**.**



- What the heatmap is trying to tell us is the locations in the image which are important for that particular layer to classify it as the target class, which is **Daisy** in this case**.**



- What the heatmap is trying to tell us is the locations in the image which are important for that particular layer to classify it as the target class, which is **Daisy** in this case**.**