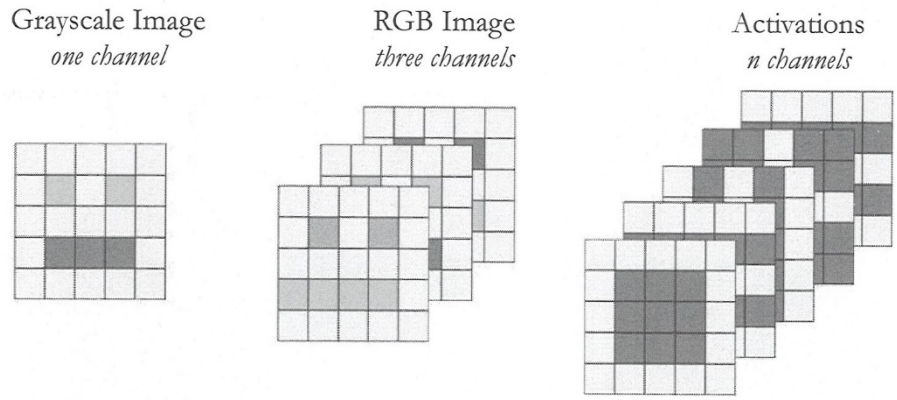
# Activations

Neural networks are made of layers that take information from the previous layer, manipulate it, and pass the result to the next layer. In the case of CNNs, information is passed between layers as a collection of 2-D arrays. Each of these arrays can be visualized as a grayscale image, and together can be thought of as a set of features used to represent the original image. You can extract these features with the activations function.

```
features = activations(net, im, layer)
```

net is the pretrained network, im is an input image, and layer is the layer to extract the features from. The output is a 3-D numeric array, where the third dimension is often called a *channel*.

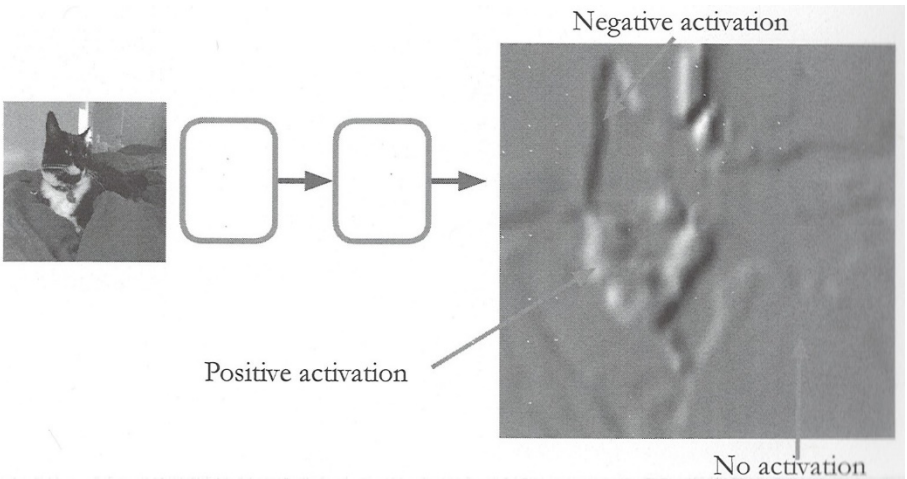
Many images have multiple channels. For example, RGB image have three(red, green, and blue).




There is one output channel for each filter in a convolution layer. A convolution layer can have hundreds of filters, so each layer can create hundreds of channels. You can visualize each channel as a grayscale image.

A white pixel indicates that the channel is positively activated at that position.

A black pixel is where the channel activated negatively.



# Deeper Layer Activations

Most convolutional neural networks learn to detect features like color and edges in their first convolutional layer. In deeper convolutional layer, the network learns to detect more complicated features. Later layers build up their features by combining features of earlier layers.

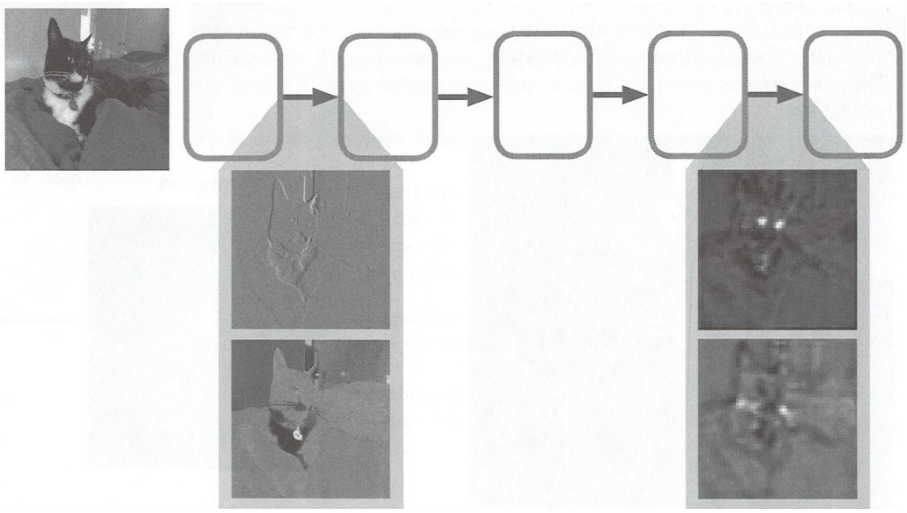
A deep network like AlexNet has never been told to learn specific features, but it often finds recognizable features. AlexNet often detects eyes in an image of a face. During training, the network learned that eyes are a useful feature to distinguish between classes of images.

For example, learning to identify eyes could help the network distinguish between a leopard and a leopard print rug.

It can also be useful to view all activations at once using the imtile function. Each activation can take any value, so you should normalize the activations before you display them with the rescale function. You can then display all the activations from one layer.

```
normalized = rescale(features);

imshow(imtile(normalized))
```



# Following Activations in the Network Architecture

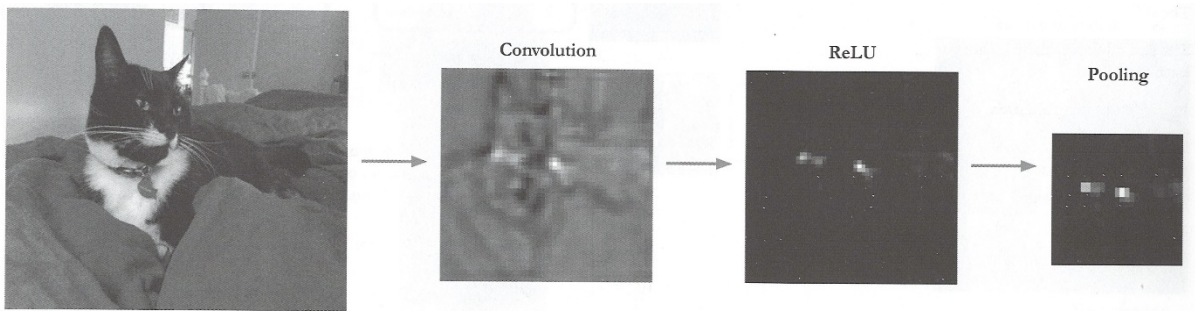
You have looked at specific activations at the first and last convolutional layers in AlexNet.

You can also track the evolution of an image through the network to see how the image is modified at each layer.

In the first convolution layer, the input image is still mostly recognizable. As the image progresses through the network, it will look less like the input. This is because the network learns a smaller set of features to represent the original image.

Rectified linear unit (ReLU) layers apply a threshold to an input such that every value less than zero is set to zero. You will see that any negative activations from a convolution layer will be set to zero after passing through the ReLU layer.

Max pooling layers perform downsampling by dividing the input into rectangular pooling regions and computing the maximum of each region. This cause large activations to be more pronounced after passing through the pooling layer.



# Feature Extraction for Machine Learning

It is difficult to perform deep learning on a computer without a GPU because of the long training time. An alternative is to use the activations from pretrained networks as image features. You may then use traditional machine learning methods to classify these features.

## Machine Learning

Traditional machine learning is a broad term that encompasses many kinds of algorithms. These algorithms learn from predictor variables. Instead of using an entire image as the training data for a model, you need a set of features. Features can be anything that accurately describes your data set. For example, features to describe dogs could be coat colors, pattern, and size.

## Image Features in Pretrained Networks

CNNs learn to extract useful features while learning how to classify image data. As you have seen, the early read an input image and extract features. Then, fully connected layers classify these features.

## Extracting Features

With deep learning, you can use the activations function to extract features. These features can be used as the predictor variables for machine learning.

The rest of the machine learning workflow is very similar to that of deep learning:

1. Extract training features.

2. Train model.

3. Extract test features.

4. Predict test features.

5. Evaluate model.

You can get activations from an entire datastore of images at once. The activations need to ne stored as rows to train a machine learning model. This makes each row correspond to one image, and each column one feature. You can set the OutputAs option to accomplish this.

```
Features = activations (net, images, layer, …
  'OutputAs', 'rows')
```

Extract features from the training and testing images stored in the datastore. You can get features from any layer in a network. For AlexNet, the fc7 layer is often used.

# Applying Traditional Machine Learning

You will use the extracted features and the training image labels to create a *k*-nearest neighbors model. One way to categorize images is to classify a new image based on the pairwise distance between its features and the nearest image features. Here you have to think of the features as being a point in space. The *k*-nearest neighbors (*k*-NN) algorithm uses the majority vote from *k* neighboring points to determine the class of the unlabeled data. You can train a *k*-nearest neighbors classifier using the fitcknn function

```
classifier = fitcknn (trainfeatures, labels)
```

where trainfeatures is the output matrix from the activations function on the training data, and labels is from the labels property of the image datastore. Your features are now predictor variables and your image labels are your response variable.

```
pedLabels = predict (classifier, testfeatures)
```

As with deep networks, you can evaluate the performance of a machine learning model using a confusion matrix.

```
confusionchart (known, predicted)
```