The complete image classification pipeline can be formalized as follows:

* Our input is a training dataset that consists of *N* images, each labeled with one of 2 different classes.
* Then, we use this training set to train a classifier to learn what every one of the classes looks like.
* In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier.

Code 1:

Let’s get started with the code.

I started with loading keras and its various layers which will be required for building the model.

Code 2:

The next step was to build the model. This can be described in the following 3 steps.

1. I used two convolutional blocks comprised of convolutional and max-pooling layer. I have used relu as the activation function for the convolutional layer.
2. On top of it I used a flatten layer and followed it by two fully connected layers with relu and sigmoid as activation respectively.
3. I have used Adam as the optimizer and cross-entropy as the loss.

Code 3:

**Data Augumentation**

The practice of **Data Augumentation**is an effective way to increase the size of the training set. Augumenting the training examples allow the network to “see” more diversified, but still representative, datapoints during training.

The following code defines a set of augumentations for the training-set: *rotation*, *shift*, *shear*, *flip*, and *zoom*.

Whenever the dataset size is small, data augmentation should be used to create additional training data.

Also I created a data generator to get our data from our folders and into Keras in an automated way. Keras provides convenient python generator functions for this purpose.

Code 4:

Next I trained the model for 50 epochs with a batch size of 32.

Batch size is one of the most important hyperparameters to tune in deep learning. I prefer to use a larger batch size to train my models as it allows computational speedups from the parallelism of GPUs. However, it is well known that too large of a batch size will lead to poor generalization. On the one extreme, using a batch equal to the entire dataset guarantees convergence to the global optima of the objective function. However this is at the cost of slower convergence to that optima. On the other hand, using smaller batch sizes have been shown to have faster convergence to good results. This is intuitively explained by the fact that smaller batch sizes allow the model to start learning before having to see all the data. The downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima.Therefore it is often advised that one starts at a small batch size reaping the benefits of faster training dynamics and steadily grows the batch size through training.

Code 5:

Visualizing the loss and accuracy plots.

Code 6:

The model is able to reach 100% validation accuracy in 50 epochs.

Code 7:

## **Making new predictions from our trained model :**

The test\_image holds the image that needs to be tested on the CNN. Once we have the test image, we will prepare the image to be sent into the model by converting its resolution to 64x64 as the model only excepts that resolution. Then we are using predict() method on our classifier object to get the prediction. As the prediction will be in a binary form, we will be receiving either a 1 or 0, which will represent a dog or a cat respectively.