Image Classification using Deep Neural Networks — A beginner friendly approach using TensorFlow

Abdellatif Abdelfattah Follow

Jul 28, 2017 · 7 min read

tl;dr

We will build a deep neural network that can recognize images with an accuracy of 78.4% while explaining the techniques used throughout the process.

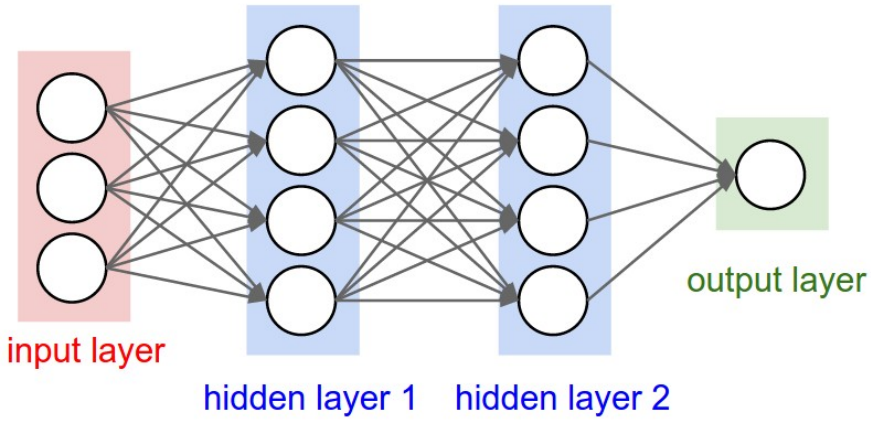
Introduction

Recent advances in deep learning made tasks such as Image and speech recognition possible.

> *Deep Learning: A subset of Machine Learning Algorithms that is very good at recognizing patterns but typically requires a large number of data.*

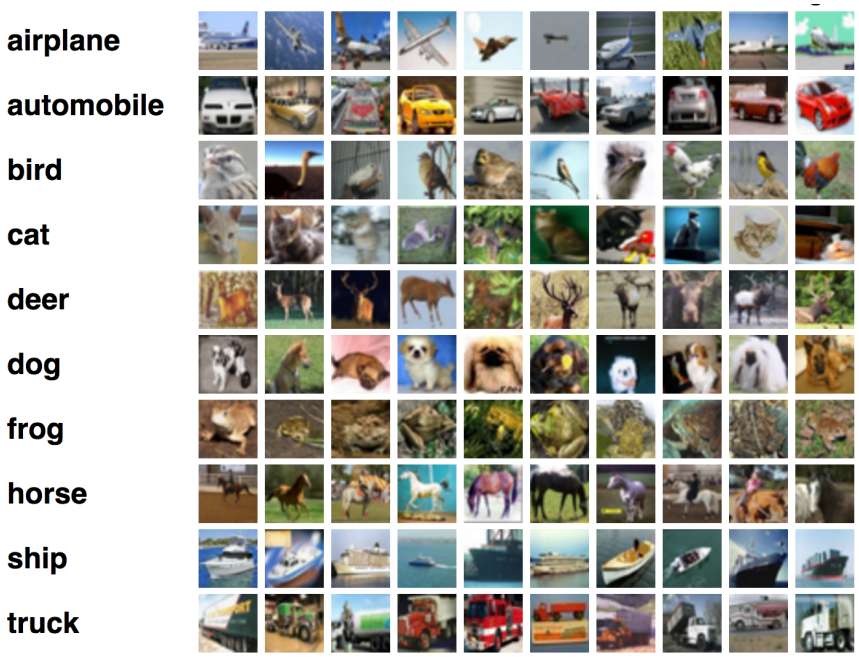
Deep learning excels in recognizing objects in images as it's implemented using 3 or more layers of artificial neural networks where each layer is responsible for extracting one or more feature of the image (more on that later).

> *Neural Network: A computational model that works in a similar way to the neurons in the human brain. Each neuron takes an input, performs some operations then passes the output to the following neuron.*



Neural Network representation (src)

We're going to teach the computer to recognize images and classify them into one of these 10 categories:



To do so, we first need to teach the computer how a cat, a dog, a bird, etc. look like before it being able to recognize a new object. The more cats the computer sees, the better it gets in recognizing cats. This is known as supervised learning. We can carry this task by labeling the images, the computer will start recognizing patterns present in cat pictures that are absent from other ones and will start building its own cognition.

We're going to use Python and TensorFlow to write the program. TensorFlow is an open source deep learning framework created by Google that gives developers granular control over each neuron (known as a "node" in TensorFlow) so you can adjust the weights and achieve optimal performance. TensorFlow has many built-in libraries (few of which we'll be using for image classification) and has an amazing community, so you'll be able to find open source implementations for virtually any deep learning topic.

So let's do it—let's teach the computer to classify images!

Machine learning for Images

Computers are able to perform computations on numbers and is unable to interpret images in the way that we do. We have to somehow convert the images to numbers for the computer to understand.

There are two common ways to do this in Image Processing:

1. Using Greyscale:

The image will be converted to greyscale (range of gray shades from white to black) the computer will assign each pixel a value based on how dark it is. All the numbers are put into an array and the computer does computations on that array. This is how the number 8 is seen on using Greyscale:

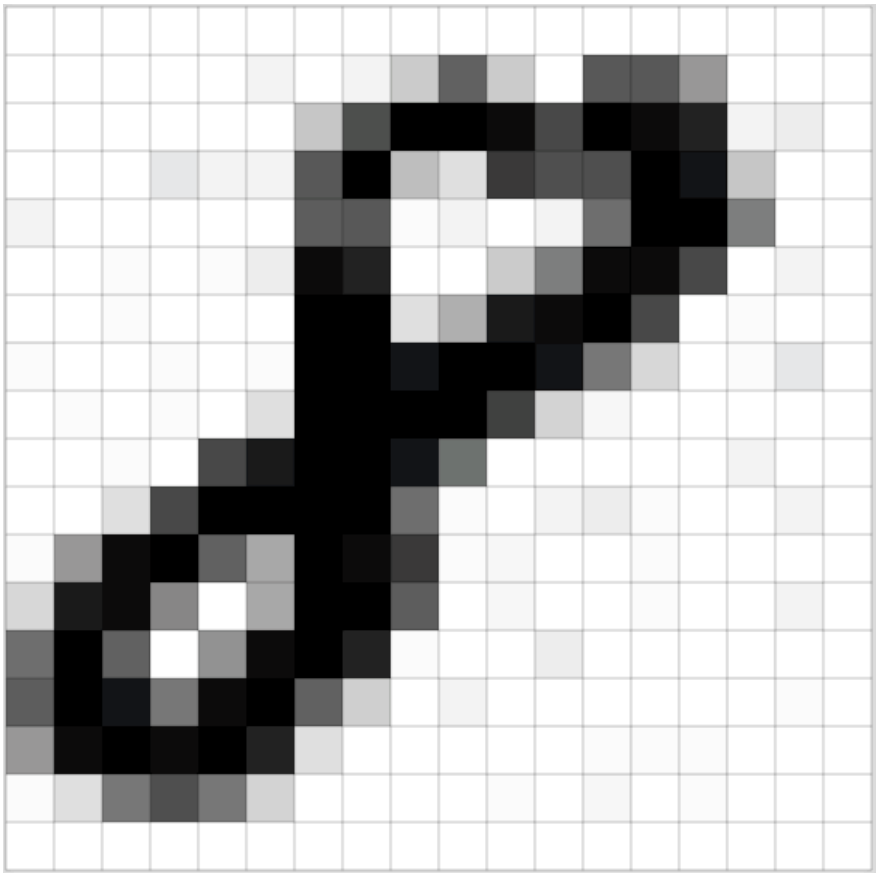


Image converted to numbers (src)

We then feed the resulting array into the computer:

=


```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```


Resulting Array (src)

2. Using RGB Values:

Colors could be represented as RGB values (a combination of red, green and blue ranging from 0 to 255). Computers could then extract the RGB value of each pixel and put the result in an array for interpretation.

When the computer interprets a new image, it will convert the image to an array by using the same technique, which then compares the patterns of numbers against the already-known objects. The computer then allots confidence scores for each class. The class with the highest confidence score is usually the predicted one.

CNNs in a Nutshell

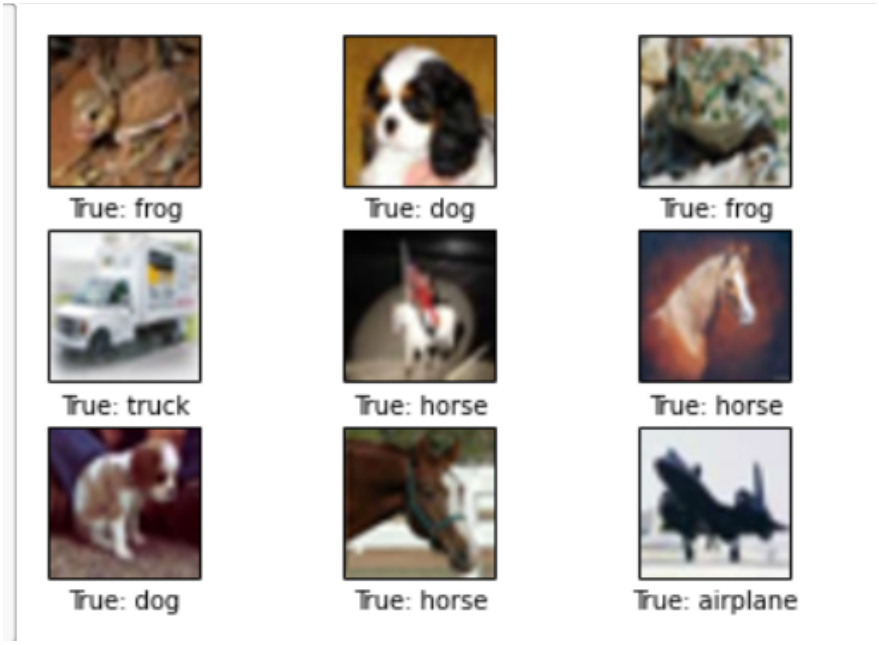
One of the most popular techniques used in improving the accuracy of image classification is Convolutional Neural Networks (CNNs for short).

> *Convolutional Neural Network: A special type Neural Networks that works in the same way of a regular neural network except that it has a convolution layer at the beginning*

Instead of feeding the entire image as an array of numbers, the image is broken up into a number of tiles, the machine then tries to predict what each tile is. Finally, the computer tries to predict what's in the picture based on the prediction of all the tiles. This allows the computer to parallelize the operations and detect the object regardless of where it is located in the image.

Dataset

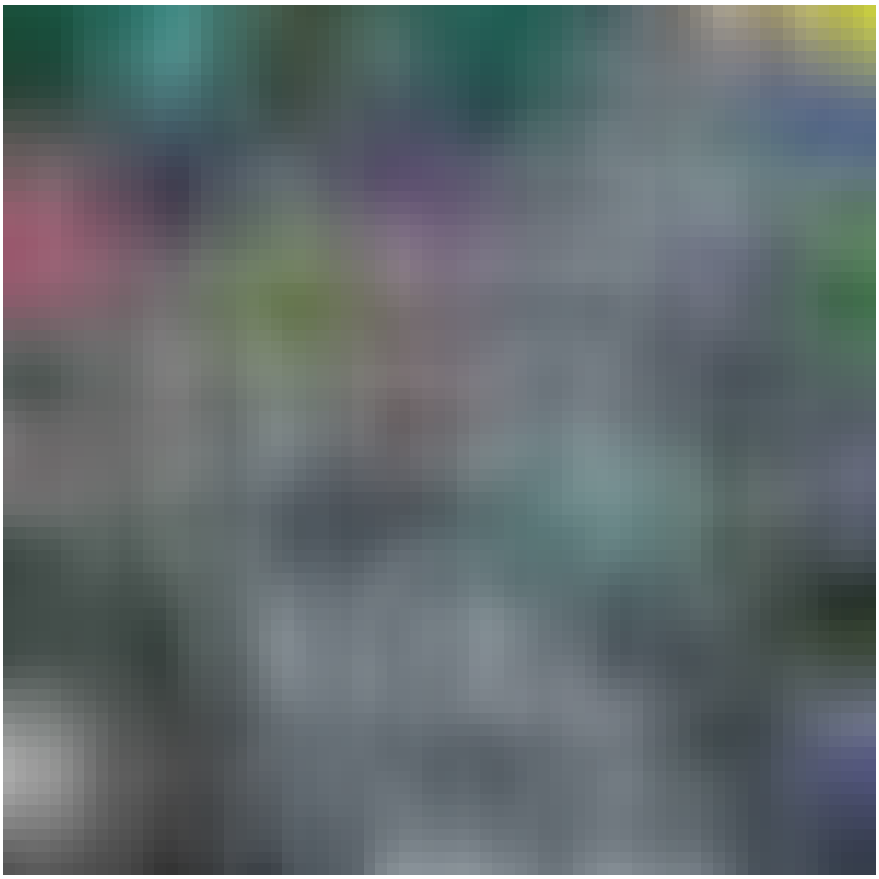
We decided to use the CIFAR-10 dataset that consists of 60,000 images sized 32 x 32 pixels. The dataset contains 10 classes that are mutually exclusive (do not overlap)with each class containing 6,000 images. The images are small, clearly labelled and have no noise which makes the dataset ideal for this task with considerably much less pre-processing. Here are few pictures taken from the dataset:



Step 1: Pre-processing

First, we need to add a little bit of variance to the data since the images from the dataset are very organized and contain little to no noise. We're going to artificially add noise using a Python library named imgaug. We're going to do a random combination of the following to the images:

- Crop parts of the image
- Flip image horizontally
- Adjust hue, contrast and saturation



Here's how the picture could look like with a combination of the above effects

Here's how the Python code for pre-processing images looks like:

```
def pre_process_image(image, training):
    # This function takes a single image as input,
    # and a boolean whether to build the training or testing
graph.

    if training:
        # For training, add the following to the TensorFlow
graph.

        # Randomly crop the input image.
        image = tf.random_crop(image, size=
[img_size_cropped, img_size_cropped, num_channels])

        # Randomly flip the image horizontally.
        image = tf.image.random_flip_left_right(image)

        # Randomly adjust hue, contrast and saturation.
        image = tf.image.random_hue(image, max_delta=0.05)
        image = tf.image.random_contrast(image, lower=0.3,
upper=1.0)
        image = tf.image.random_brightness(image,
max_delta=0.2)
```

```
        image = tf.image.random_saturation(image, lower=0.0,
upper=2.0)

        # Some of these functions may overflow and result in
pixel
        # values beyond the [0, 1] range. It is unclear from
the
        # documentation of TensorFlow 0.10.0rc0 whether this
is
        # intended. A simple solution is to limit the range.

        # Limit the image pixels between [0, 1] in case of
overflow.
        image = tf.minimum(image, 1.0)
        image = tf.maximum(image, 0.0)
    else:
        # For training, add the following to the TensorFlow
graph.

        # Crop the input image around the centre so it is
the same
        # size as images that are randomly cropped during
training.
        image =
tf.image.resize_image_with_crop_or_pad(image,

target_height=img_size_cropped,

target_width=img_size_cropped)

    return image
```

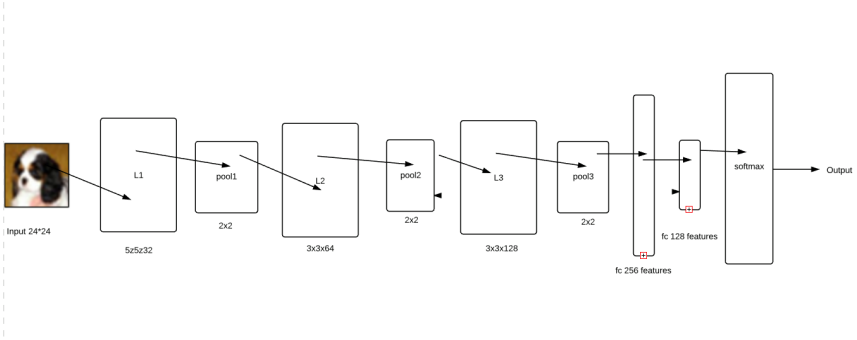
Step 2: Splitting our dataset

It takes a long time to calculate the gradient of the model using the entirety of a large dataset . We therefore will use a small batch of images during each iteration of the optimizer. The batch size is normally 32 or 64—we'll use 64 since we have fairly a large number of images . The dataset is then divided into training set containing 50,000 images, and test set containing 10,000 images.

```
train_batch_size = 64


def random_batch():
    # Number of images in the training-set.
    num_images = len(images_train)

    # Create a random index.
    idx = np.random.choice(num_images,
                           size=train_batch_size,
```

```
                                replace=False)

    # Use the random index to select random images and
labels.
    x_batch = images_train[idx, :, :, :]
    y_batch = labels_train[idx, :]

    return x_batch, y_batch
```

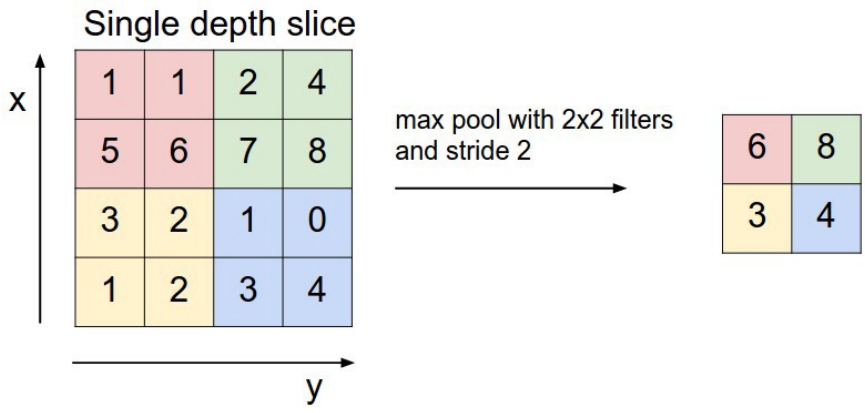
Step 3: Building a Convolutional Neural Network



Neural Network Architecture

Now that we're done pre-processing and splitting our dataset we can start implementing our neural network. We're going to have 3 convolution layers with 2 x 2 max-pooling.

> *Max-pooling: A technique used to reduce the dimensions of an image by taking the maximum pixel value of a grid. This also helps reduce overfitting and makes the model more generic. The example below show how 2 x 2 max pooling works*



2 x 2 max-pooling (src)

After that, we add 2 fully connected layers. Since the input of fully connected layers should be two dimensional, and the output of convolution layer is four dimensional, we need a flattening layer between them.

At the very end of the fully connected layers is a softmax layer.

The parameters identification is a challenging task since we have so many parameters to be adjusted. Hence, we read a lot of resources and tried to figure out a way to do it. However, it seems that it is based on experience. So we found a structure made by Alex Krizhevsky, who used this structure and won the champion of ImageNet LSVRC-2010. Since our job is much simpler that his work, so we only used 3 convolutional layers and maintained a gradient between each of them.

Results

Now that we have a trained neural network, we can use it! We then made the computer interpret 10,000 unknown images and got an accuracy of 78.4% (7844 / 10000). What's interesting is that the incorrect predictions look pretty close to what the computer thought it is.

Sample of the incorrect predictions

Take the picture in the top right for example, the picture looks like a cat sitting on a truck, it's reasonable enough for the computer to predict it as a truck.

Conclusion

We were able to build an artificial convolutional neural network that can recognize images with an accuracy of 78% using TensorFlow. We did so by pre-processing the images to make the model more generic, split the dataset into a number of batches and finally build and train the model.