APSC 258: Lab 5 Manual

Andre Cox
Scott Halston

July 22, 2022

Contents

Chapter 1

Introduction

1.1 Introducing the Convolution Layer

In our last lab, you got to work with dense layers. You may have noticed that the neural network was not very effective. In this lab, we will introduce the concept of Convolution layers. To explain why we use these layers we must first revisit the idea of a dense layer. You can think of a dense layer as a layer of neurons. The neurons in a dense layer are simply a weighted sum of the inputs. When training the neural network these weights get adjusted until the network is able to predict the correct output. This is a great way to predict simple things such as the average grade a student may achieve based on class attendance and the amount of homework they have done. However, they become less effective as the number and complexity of the inputs increase. For instance, let's say we have an image of our track that we feed into some dense layers. These dense layers may be able to predict the steering angle of the car however they may get confused depending on the different features of the image. This is because these layers represent the image in 1 dimension. Let me give you an example.

Figure 1.1: A 1D representation of an image

Can you tell what this above image is? Probably not easily. This is essentially what our neural network uses to learn with respect to dense layers. We need a way to represent the image in a 2-dimensional space.

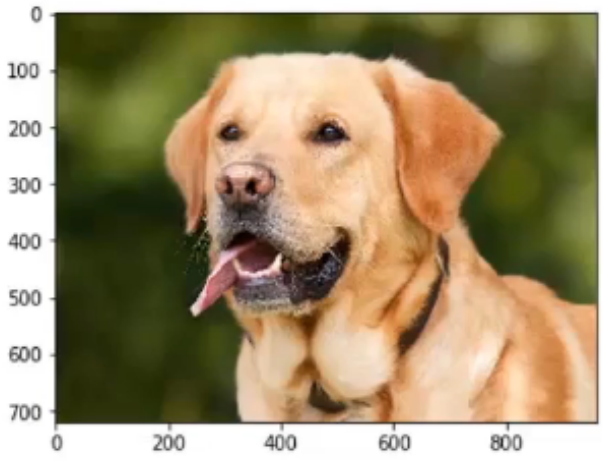

Figure 1.2: A 2D representation of an image

You can see that the image is now represented in a 2-dimensional space and is now much easier to understand. This is why convolution layers are so useful. In summary for your understanding, dense layers essentially view data in a 1-dimensional array, whereas convolution layers view data in a 2-dimensional array

1.2 More on Convolution Layers

Convolution layers are useful for feature extraction. For instance, if we were running facial recognition we would use multiple layers that would each progressively extract features with more and more detail. An example of this is shown below.

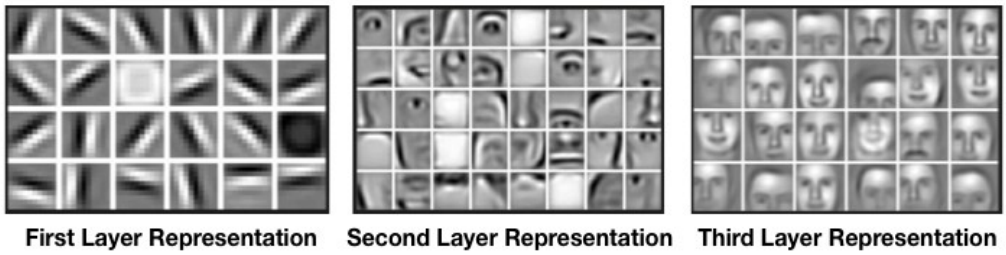

Figure 1.3: Extracting features from an image

After we have extracted our features, we can use a Flatten layer to turn the features back into a 1-dimensional vector. This can then be used as input to a dense layer. A representation of this is shown below. In the code shown below, there is also a pooling layer. Pooling layers just reduce the size of the image. This is useful for increasing the efficiency of the network.

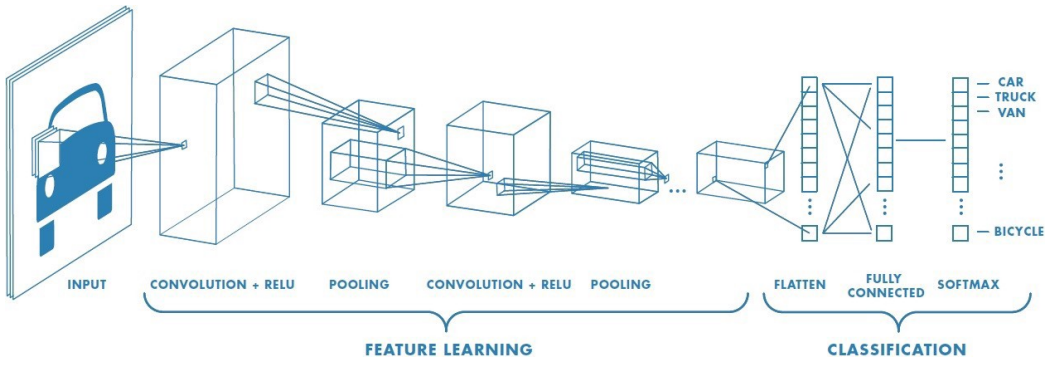

Figure 1.4: Representation of a CNN

Chapter 2

Using the Convolution Layer

Now that we have a better understanding of the convolution layer we can use it in our neural network.

2.1 Creating a Convolution Layer

Below is a simple example of a convolutional neural network. For your understanding, all of the parameters are explained within the code. It is highly recommended to go through this code on your own to better your understanding of convolution layers and neural networks.

```
from keras.layers import Conv2D, MaxPooling2D, Dense Sequential,
    InputLayer, Flatten
# a sequential model is a model that is made up of layers
model = Sequential()
# the input layer is the first layer in the model
model.add(InputLayer(input_shape=(100, 66, 1)))

# the convolution layer
# the different parameters are:
# the number of output filters: 32
# the kernel size: (3, 3) specifying the height and width of the 2D
    convolution window.
# the activation function: ReLU the type of activation function that
    is used in the convolution layer.
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))

# the pooling layer
# the pooling layer is used to reduce the size of the image.
# the pool size: (2, 2) specifying the height and width of the 2D
    pooling window.
model.add(MaxPooling2D(pool_size=(2, 2)))

# the flatten layer
# the flatten layer is used to turn the output of the convolution
    layer into a 1 dimensional vector.
model.add(Flatten())

# our dense layers, used to do the classification
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='relu'))
```

Chapter 3

Applying the Convolution Layer

Now that you have a good grasp of the convolution layer we can apply it to our neural network. Open the model from your previous lab and add try experimenting by adding the convolution layer to the model. See if multiple convolution layers function better than a single one. Once you are done with this and are happy with your model you can save the model and download it. Then you can run the model with the following code. "python runNetwork.py -m yourConvolutionModel.h5"

Roadblock: You may have noticed that your training model has a very low mean squared error and that your testing model has a very high mean squared error. This issue is caused by overfitting. Overfitting is when the model is too complex and cannot generalize to new data. This is a problem that we will solve in the next lab.