**Gender Detection**

**Batch Normalization:** [**https://towardsdatascience.com/what-is-batch-normalization-46058b4f583**](https://towardsdatascience.com/what-is-batch-normalization-46058b4f583)

**Batch normalization standardizes the distribution of layer inputs to combat the internal covariance shift.**

**It basically normalizes the data, instead of having exponential shifts in the values in the dataset, normalization lowers the exponentiality and generalizes the data.**

**What is Internal Covariance Shift…?**

You might have heard a fancy word thrown around when talking about batch normalization, it is the **internal covariance shift**. Consider a network that learns a function that maps x to y. The internal covariance shift refers to the change in the distribution of the input x. If a change occurs, our network will not be efficient enough and can’t generalize. Thus, we will have to train all over.

Consider this example to understand the covariance shift. If we train a network to detect brown dog images and later you end applying this network to data with colored dog images it would not be able to perform well and we would have to train again. This change in the distribution of input is the covariance shift.

**What are the benefits?**

1. Train faster.
2. Use higher learning rates.
3. Parameter initialization is easier.
4. Makes activation functions viable by regulating the inputs to them.
5. Better results overall.
6. It adds noise which reduces overfitting with a regularization effect. Thus, make sure to use less dropout when you apply batch normalization as dropout itself adds noise.

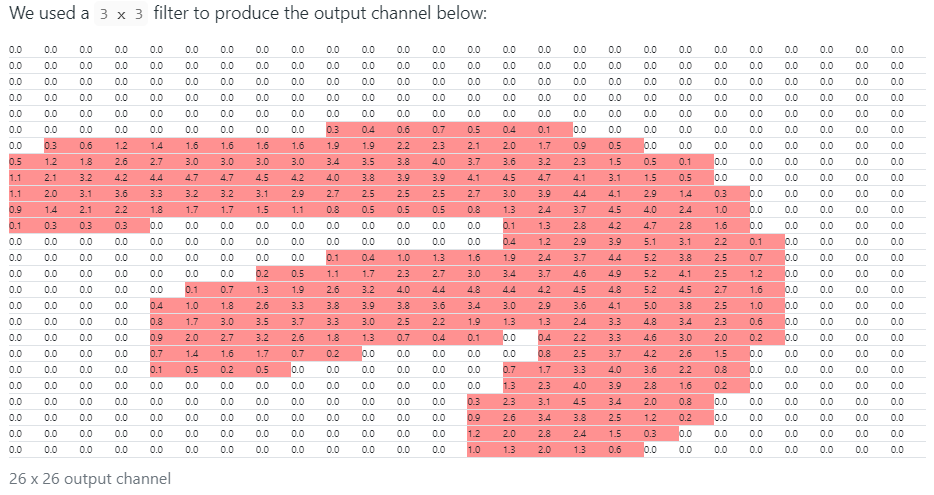
**Max Pooling:** [**https://deeplizard.com/learn/video/ZjM\_XQa5s6s**](https://deeplizard.com/learn/video/ZjM_XQa5s6s)

Max pooling is a type of operation that is typically added to CNNs following individual convolutional layers.

When added to a model, max pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer.

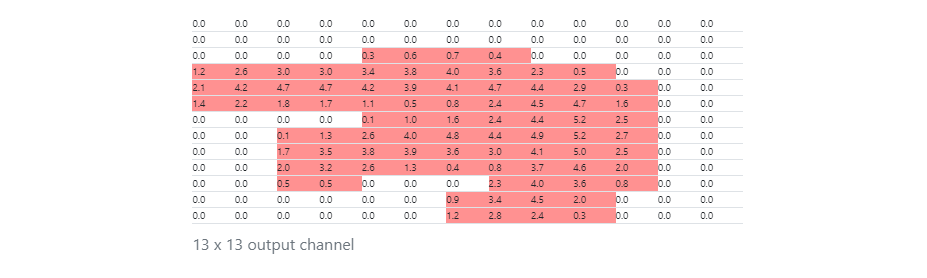
We've seen in our [post on CNNs](https://deeplizard.com/learn/video/YRhxdVk_sIs) that each convolutional layer has some number of filters that we define with a specified dimension and that these filters convolve our image input channels.

When a filter convolves a given input, it then gives us an output. This output is a matrix of pixels with the values that were computed during the convolutions that occurred on our image. We call these output channels.



As mentioned earlier, max pooling is added after a convolutional layer. This is the output from the convolution operation and is the input to the max pooling operation.

After the max pooling operation, we have the following output channel:



Max pooling works like this. We define some n x n region as a corresponding filter for the max pooling operation. We're going to use 2 x 2 in this example.

We define a stride, which determines how many pixels we want our filter to move as it slides across the image.

On the convolutional output, we take the first 2 x 2 region and calculate the max value from each value in the 2 x 2 block. This value is stored in the output channel, which makes up the full output from this max pooling operation.

We move over by the number of pixels that we defined our stride size to be. We're using 2 here, so we just slide over by 2, then do the same thing. We calculate the max value in the next 2 x 2 block, store it in the output, and then, go on our way sliding over by 2 again.

Once we reach the edge over on the far right, we then move down by 2 (because that's our stride size), and then we do the same exact thing of calculating the max value for the 2 x 2 blocks in this row.

We can think of these 2 x 2 blocks as pools of numbers, and since we're taking the max value from each pool, we can see where the name max pooling came from.

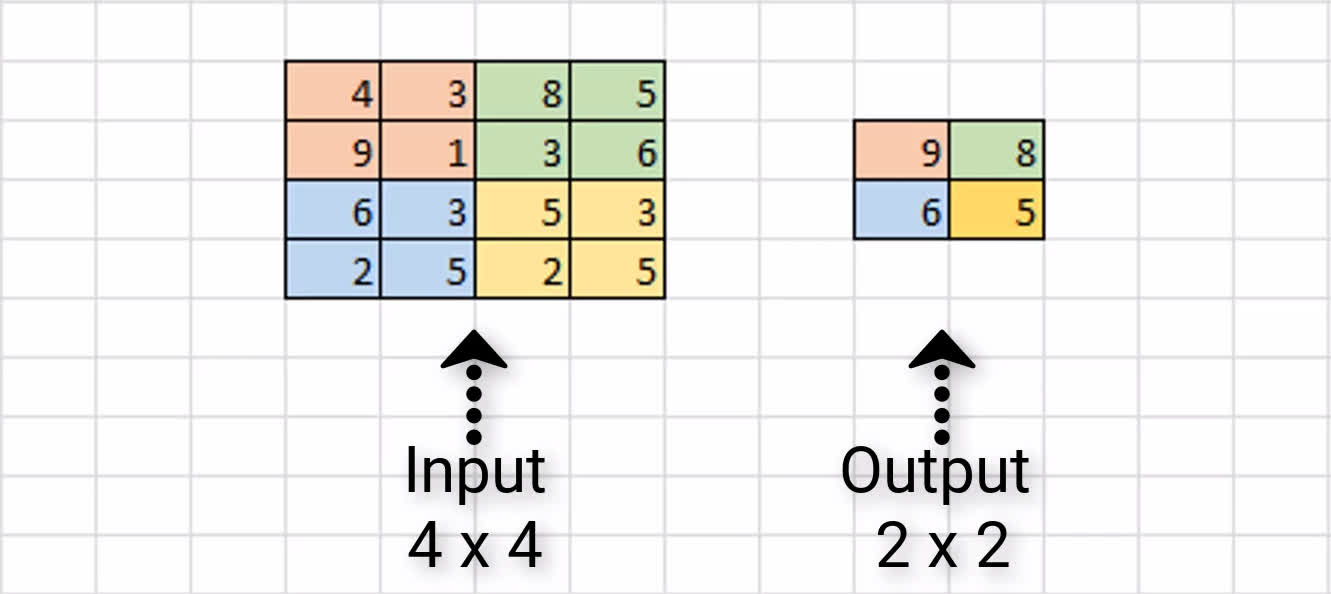
This process is carried out for the entire image, and when we're finished, we get the new representation of the image, the output channel.

In this example, our [convolution](https://deeplizard.com/resource/pavq7noze2) operation output is 26 x 26 in size. After performing max pooling, we can see the dimension of this image was reduced by a factor of 2 and is now 13 x 13.

Just to make sure we fully understand this operation, we're going to quickly look at a scaled down example that may be simpler to visualize.

#### **Scaled Down Example**

Suppose we have the following:



We have some sample input of size 4 x 4, and we're assuming that we have a 2 x 2 filter size with a stride of 2 to do max pooling on this input channel.

Our first 2 x 2 region is in orange, and we can see the max value of this region is 9, and so we store that over in the output channel.

Next, we slide over by 2 pixels, and we see the max value in the green region is 8. As a result, we store the value over in the output channel.

Since we've reached the edge, we now move back over to the far left, and go down by 2 pixels. Here, the max value in the blue region is 6, and we store that here in our output channel.

Finally, we move to the right by 2, and see the max value of the yellow region is 5. We store this value in our output channel.

This completes the process of max pooling on this sample 4 x 4 input channel, and the resulting output channel is this 2 x 2 block. As a result, we can see that our input dimensions were again reduced by a factor of two.

### **Why Use Max Pooling?**

There are a couple of reasons why adding max pooling to our network may be helpful.

#### **Reducing Computational Load**

Since max pooling is reducing the resolution of the given output of a convolutional layer, the network will be looking at larger areas of the image at a time going forward, which reduces the number of parameters in the network and consequently reduces computational load.

#### **Reducing Overfitting**

Additionally, max pooling may also help to reduce overfitting. The intuition for why max pooling works is that, for a particular image, our network will be looking to extract some particular features.

Maybe, it's trying to identify numbers from the MNIST dataset, and so it's looking for edges, and curves, and circles, and such. From the output of the convolutional layer, we can think of the higher valued pixels as being the ones that are the most activated.

**With max pooling, as we're going over each region from the convolutional output, we're able to pick out the most activated pixels and preserve these high values going forward while discarding the lower valued pixels that are not as activated.**

Just to mention quickly before going forward, there are other types of pooling that follow the exact same process we've just gone through, except for that it does some other operation on the regions rather than finding the max value.

#### **Average Pooling**

For example, average pooling is another type of pooling, and that's where you take the average value from each region rather than the max.

We use dropout so that the model doesn’t overfit i.e., learn the train dataset very well and cannot generalize the output.

Dropout specifies the % of neurons we would want to keep inactive.

So here we have specified **model.add(Dropout(0.25))**, i.e., 25% of the neurons will be inactive for this layer.