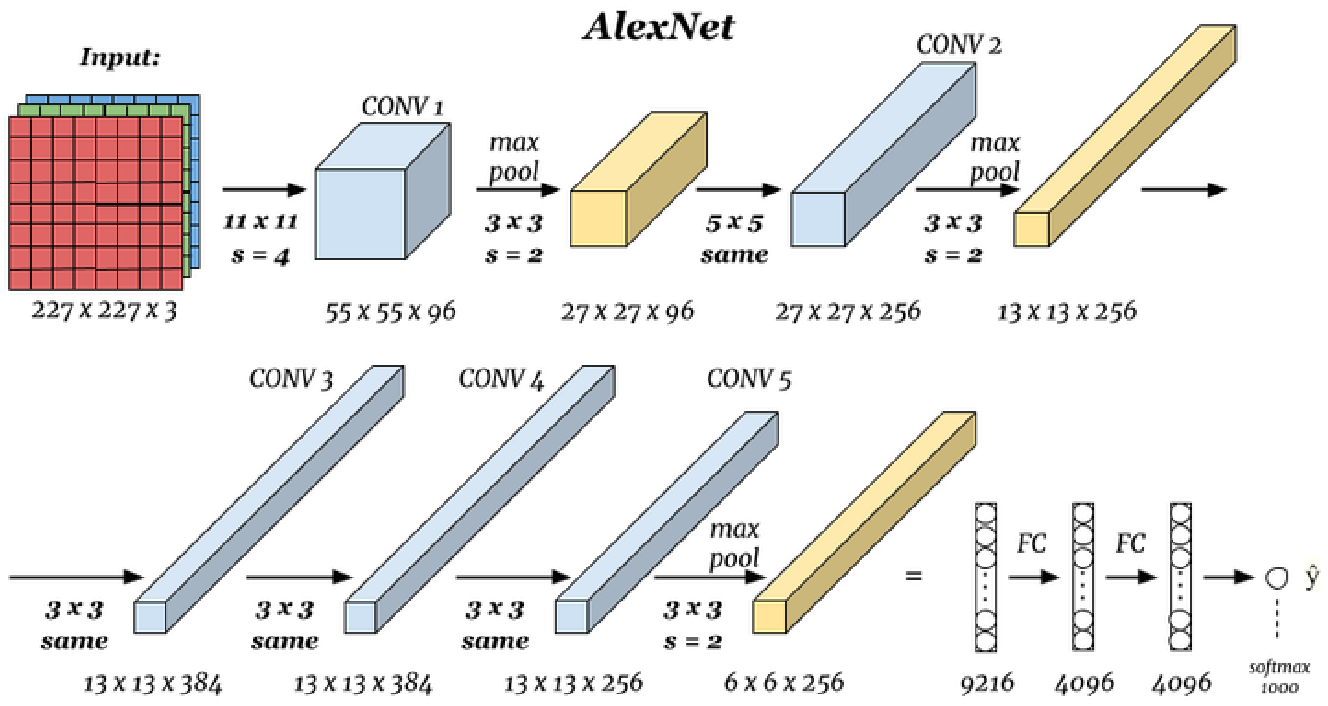
Alexnet Architecture (CNN)

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor.

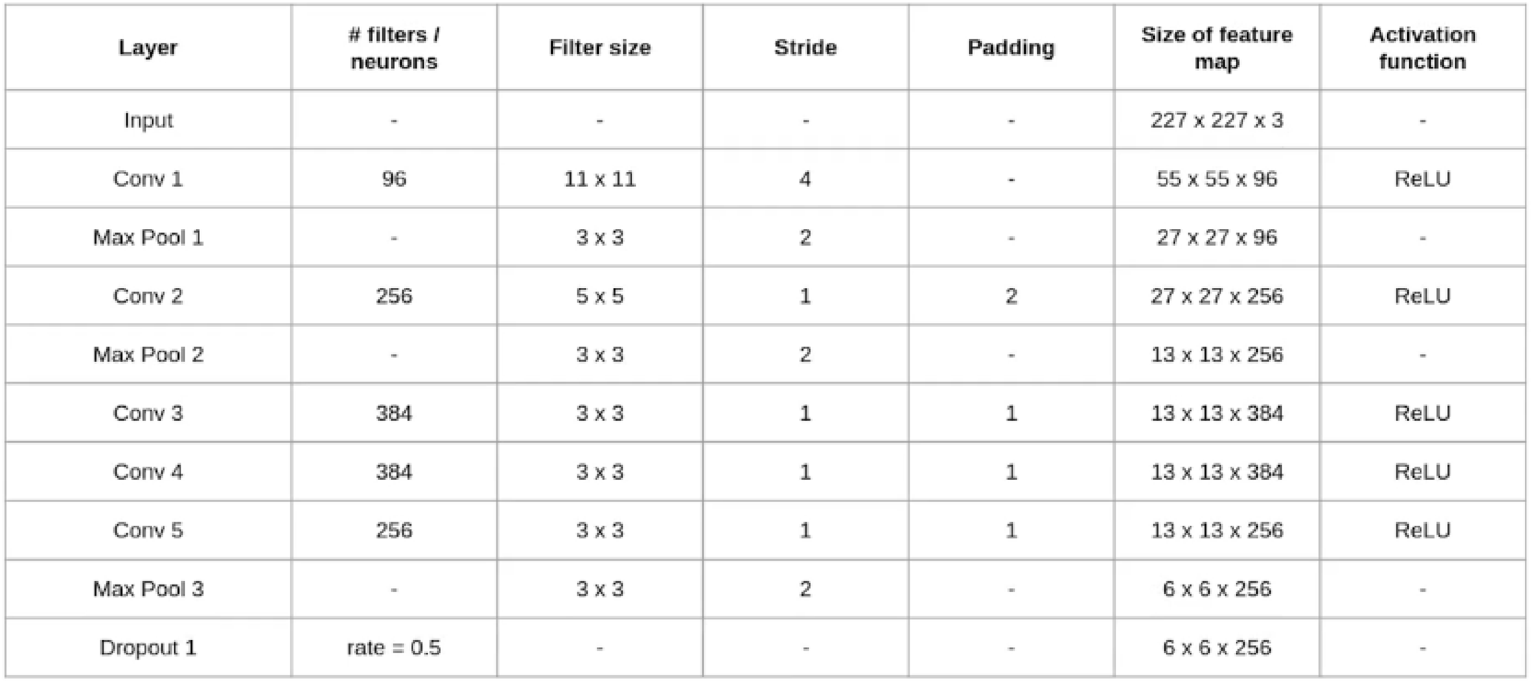
The Alexnet has eight layers with learnable parameters. The model consists of five layers with a combination of max pooling followed by 3 fully connected layers and they use Relu activation in each of these layers except the output layer.



They found out that using the relu as an activation function accelerated the speed of the training process by almost six times. They also used the dropout layers, that prevented their model from overfitting. Further, the model is trained on the Imagenet dataset. The Imagenet dataset has almost 14 million images across a thousand classes.

One thing to note here, since Alexnet is a deep architecture, the authors introduced padding to prevent the size of the feature maps from reducing drastically. The input to this model is the images of size 227X227X3.

One thing to note here, since Alexnet is a deep architecture, the authors introduced padding to prevent the size of the feature maps from reducing drastically. The input to this model is the images of size 227X227X3.



Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Convolution and Maxpooling Layers Then we apply the first convolution layer with 96 filters of size 11X11 with stride 4. The activation function used in this layer is relu. The output feature map is 55X55X96.

"In case, you are unaware of how to calculate the output size of a convolution layer :

```
output= ((Input-filter size)/ stride)+1
```

Also, the number of filters becomes the channel in the output feature map.

Next, we have the first Maxpooling layer, of size 3X3 and stride 2. Then we get the resulting feature map with the size 27X27X96.

After this, we apply the second convolution operation. This time the filter size is reduced to 5X5 and we have 256 such filters. The stride is 1 and padding 2. The activation function used is again relu. Now the output size we get is 27X27X256.

Again we applied a max-pooling layer of size 3X3 with stride 2. The resulting feature map is of shape 13X13X256.

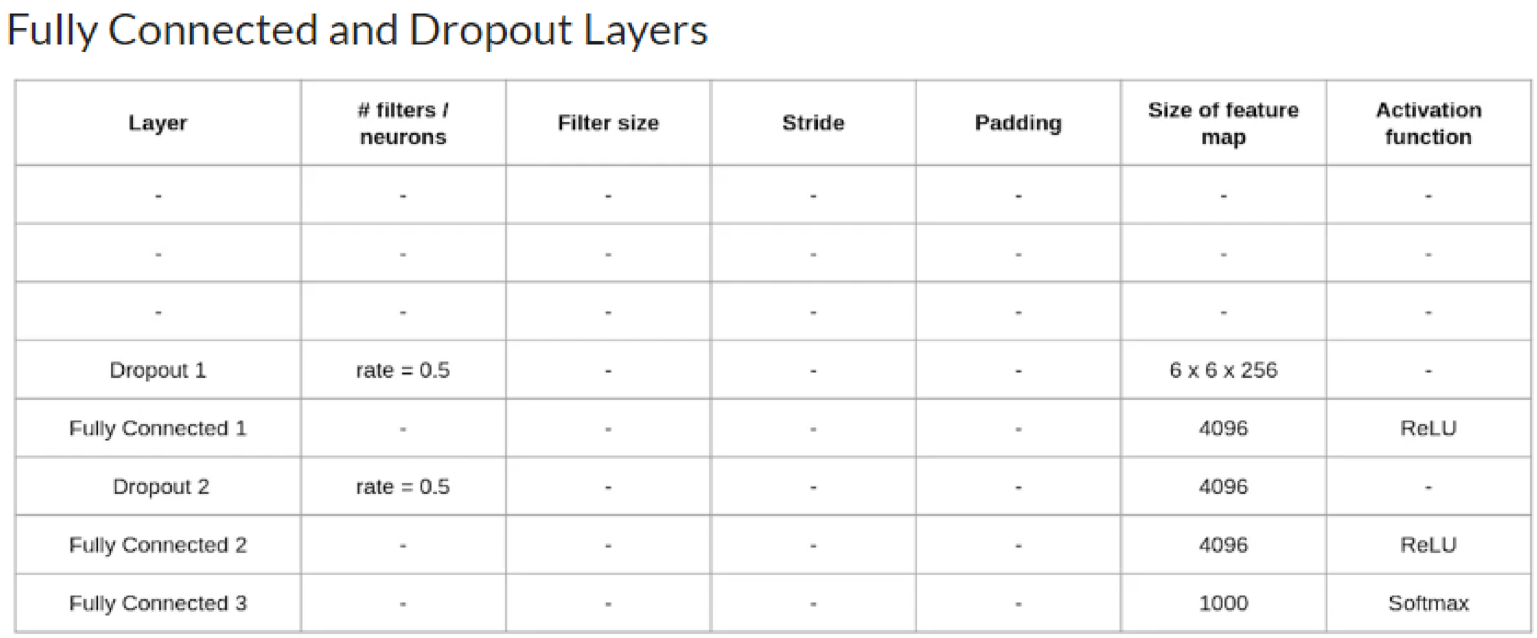
Now we apply the third convolution operation with 384 filters of size 3X3 stride 1 and also padding 1. Again the activation function used is relu. The output feature map is of shape 13X13X384.

Then we have the fourth convolution operation with 384 filters of size 3X3. The stride along with the padding is 1. On top of that activation function used is relu. Now the output size remains unchanged i.e 13X13X384.

After this, we have the final convolution layer of size 3X3 with 256 such filters. The stride and padding are set to one also the activation function is relu. The resulting feature map is of shape 13X13X256.

So if you look at the architecture till now, the number of filters is increasing as we are going deeper. Hence it is extracting more features as we move deeper into the architecture. Also, the filter size is reducing, which means the initial filter was larger and as we go ahead the filter size is decreasing, resulting in a decrease in the feature map shape.

Next, we apply the third max-pooling layer of size 3X3 and stride 2. Resulting in the feature map of the shape 6X6X256.



Fully Connected and Dropout Layers

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
-	-	-	-	-	-	-
-	-	-	-	-	-	-
-	-	-	-	-	-	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

After this, we have our first dropout layer. The drop-out rate is set to be 0.5.

Then we have the first fully connected layer with a relu activation function. The size of the output is 4096. Next comes another dropout layer with the dropout rate fixed at 0.5.

This followed by a second fully connected layer with 4096 neurons and relu activation.

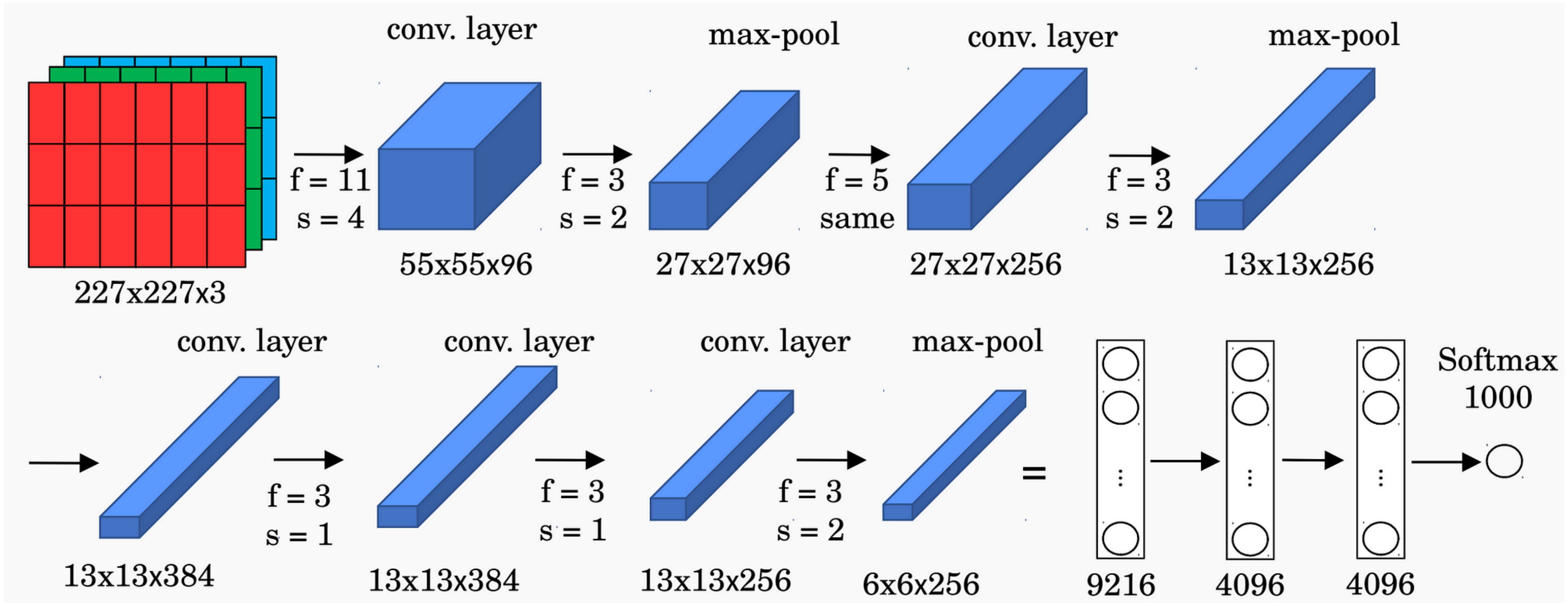
Finally, we have the last fully connected layer or output layer with 1000 neurons as we have 10000 classes in the data set. The activation function used at this layer is Softmax.

This is the architecture of the Alexnet model. It has a total of 62.3 million learnable parameters.

End Notes :

To quickly summarize the architecture that we have seen in this article.

- It has 8 layers with learnable parameters.
- The input to the Model is RGB images.
- It has 5 convolution layers with a combination of max-pooling layers.
- Then it has 3 fully connected layers.
- The activation function used in all layers is Relu.
- It used two Dropout layers.
- The activation function used in the output layer is Softmax.
- The total number of parameters in this architecture is 62.3 million.



Code: Python code to implement AlexNet for object classification

In [ ]:

```
model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(filters = 96, input_shape = (224, 224, 3),
                        kernel_size = (11, 11), strides = (4, 4),
                        padding = 'valid'))
model.add(Activation('relu'))
# Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2),
                        strides = (2, 2), padding = 'valid'))
# Batch Normalisation
model.add(BatchNormalization())

# 2nd Convolutional Layer
model.add(Conv2D(filters = 256, kernel_size = (11, 11),
                        strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2),
                        padding = 'valid'))
# Batch Normalisation
model.add(BatchNormalization())

# 3rd Convolutional Layer
model.add(Conv2D(filters = 384, kernel_size = (3, 3),
                        strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 4th Convolutional Layer
model.add(Conv2D(filters = 384, kernel_size = (3, 3),
                        strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 5th Convolutional Layer
model.add(Conv2D(filters = 256, kernel_size = (3, 3),
                        strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2),
                        padding = 'valid'))
# Batch Normalisation
model.add(BatchNormalization())

# Flattening
```

```
model.add(Flatten())

# 1st Dense Layer
model.add(Dense(4096, input_shape = (224*224*3, )))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# 2nd Dense Layer
model.add(Dense(4096))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# Output Softmax Layer
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Created by Sanjay Panchal

Follow on LinkedIn