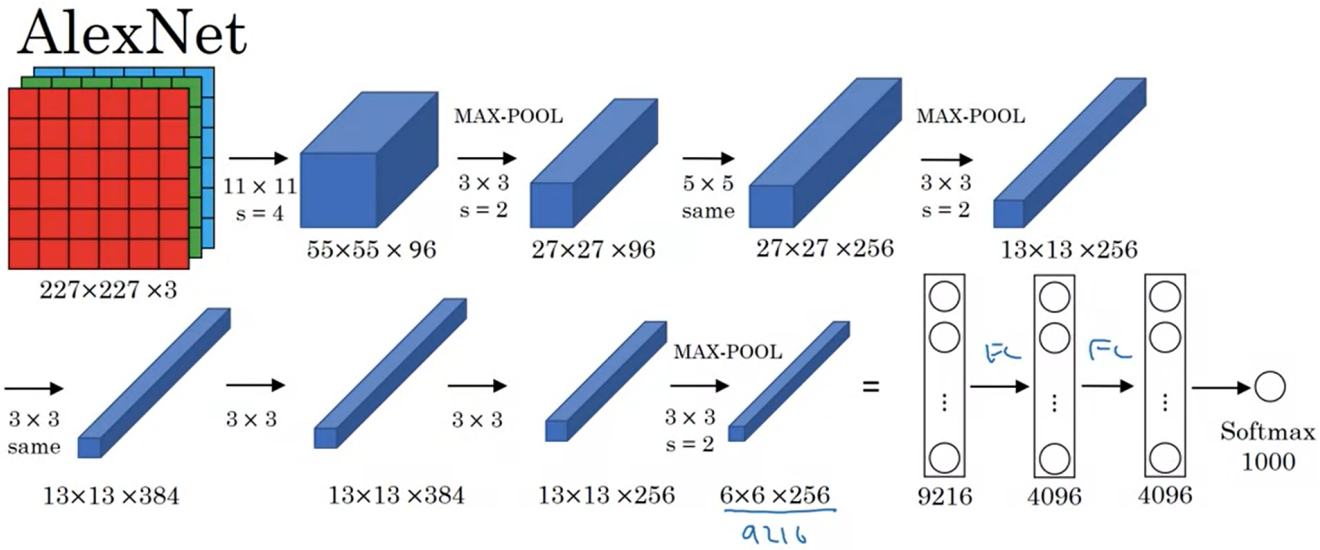
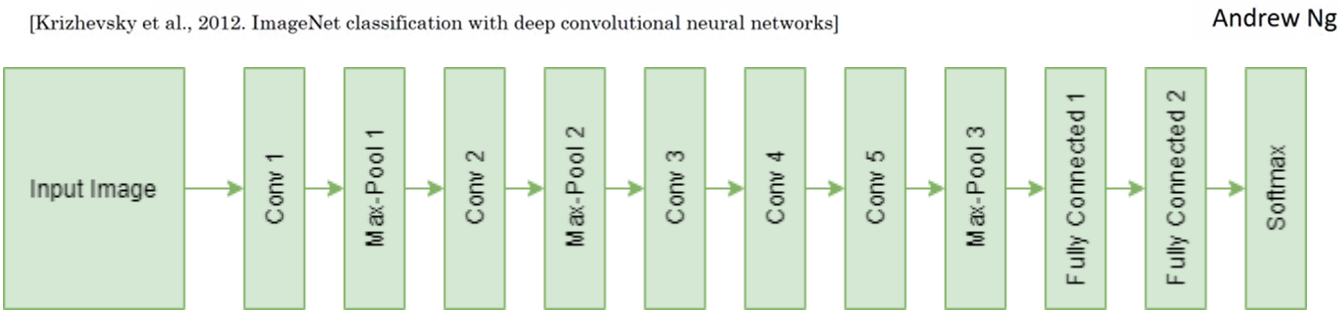
# ALEXNET

**The most important features of the AlexNet paper are:**

- As the model had to train 60 million parameters (which is quite a lot), it was prone to overfitting. According to the paper, the usage of Dropout and Data Augmentation significantly helped in reducing overfitting. The first and second fully connected layers in the architecture thus used a dropout of 0.5 for the purpose. Artificially increasing the number of images through data augmentation helped in the expansion of the dataset dynamically during runtime, which helped the model generalize better.
- Another distinct factor was using the ReLU activation function instead of tanh or sigmoid, which resulted in faster training times (a decrease in training time by 6 times). Deep Learning Networks usually employ ReLU non-linearity to achieve faster training times as the others start saturating when they hit higher activation values.

- AlexNet is a Classic type of Convolutional Neural Network, and it came into existence after the 2012 ImageNet challenge. The network architecture is given below :




## Model Explanation :

- The Input to this model have the dimensions 227x227x3 follwed by a Convolutional Layer with 96 filters of 11x11 dimensions and having a ‘same’ padding and a stride of 4. The resulting output dimensions are given as :

```
floor(((n + 2*padding - filter)/stride) + 1 ) * floor(((n + 2*padding – filter)/stride) + 1)
```

***Note : This formula is for square input with height = width = n***

- Explaining the first Layer with input 227x227x3 and Convolutional layer with 96 filters of 11x11 , 'valid' padding and stride = 4 , output dims will be

= floor(((227 + 0–11)/4) + 1) * floor(((227 + 0–11)/4) + 1)

= floor((216/4) + 1) * floor((216/4) + 1)

= floor(54 + 1) * floor(54 + 1)

= 55 * 55

- Since number of filters = 96 , thus output of first Layer is : 55x55x96
- Continuing we have the MaxPooling layer (3, 3) with the stride of 2,making the output size decrease to 27x27x96, followed by another Convolutional Layer with 256, (5,5) filters and 'same' padding, that is, the output height and width are retained as the previous layer thus output from this layer is 27x27x256.
- Next we have the MaxPooling again ,reducing the size to 13x13x256. Another Convolutional Operation with 384, (3,3) filters having same padding is applied twice giving the output as 13x13x384, followed by another Convulutional Layer with 256 , (3,3) filters and same padding resulting in 13x13x256 output.
- This is MaxPooled and dimensions are reduced to 6x6x256. Further the layer is Flatten out and 2 Fully Connected Layers with 4096 units each are made which is further connected to 1000 units softmax layer.
- The network is used for classifying much large number of classes as per our requirement. However in our case, we will make the output softmax layer with 6 units as we ahve to classify into 6 classes. The softmax layer gives us the probablities for each class to which an Input Image might belong.

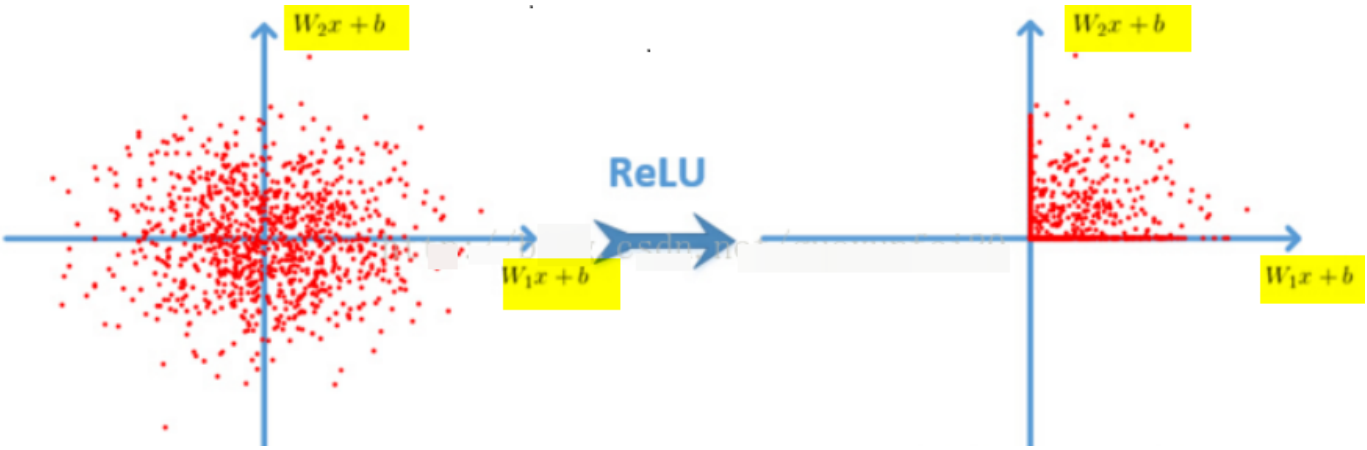
| Size / Operation | Filter | Depth | Stride | Padding | Number of Parameters | Forward Computation |
|---|---|---|---|---|---|---|
| 3 *227* 227 | | | | | | |
| Conv1 + Relu | 11 *11* | 96 | 4 | | (1111*3 + 1)* 96=34944 | (11*11*3 + 1) *96* 55 *55=105705600* |
| 96 55 *55* | | | | | | |
| Max Pooling | 3 3 | | 2 | | | |
| 96 *27* 27 | | | | | | |
| Norm | | | | | | |
| Conv2 + Relu | 5 *5* | 256 | 1 | 2 | (5 5 *96 + 1)* 256=614656 | (5 *5* 96 + 1) *256* 27 *27=448084224* |
| 256 27 *27* | | | | | | |
| Max Pooling | 3 3 | | 2 | | | |
| 256 *13* 13 | | | | | | |
| Norm | | | | | | |
| Conv3 + Relu | 3 *3* | 384 | 1 | 1 | (3 3 *256 + 1)* 384=885120 | (3 *3* 256 + 1) *384* 13 *13=149585280* |
| 384 13 *13* | | | | | | |
| Conv4 + Relu | 3 3 | 384 | 1 | 1 | (3 *3* 384 + 1) *384=1327488* | (3 3 *384 + 1)* 384 *13* 13=224345472 |
| 384 *13* 13 | | | | | | |
| Conv5 + Relu | 3 *3* | 256 | 1 | 1 | (3 3 *384 + 1)* 256=884992 | (3 *3* 384 + 1) *256* 13 *13=149563648* |
| 256 13 *13* | | | | | | |

| Size / Operation | Filter | Depth | Stride | Padding | Number of Parameters | Forward Computation |
|---|---|---|---|---|---|---|
| Max Pooling | 3 3 | | 2 | | | |
| 256 *6* 6 | | | | | | |
| Dropout (rate 0.5) | | | | | | |
| FC6 + Relu | | | | | 256 *6* 6 *4096=37748736* | 256 6 *6* 4096=37748736 |
| 4096 | | | | | | |
| Dropout (rate 0.5) | | | | | | |
| FC7 + Relu | | | | | 4096 *4096=16777216* | 4096 4096=16777216 |
| 4096 | | | | | | |
| FC8 + Relu | | | | | 4096 *1000=4096000* | 4096 1000=4096000 |
| 1000 classes | | | | | | |
| Overall | | | | | 62369152=62.3 million | 1135906176=1.1 billion |
| Conv VS FC | | | | | Conv:3.7million (6%) , FC: 58.6 million (94% ) | Conv: 1.08 billion (95%) , FC: 58.6 million (5%) |

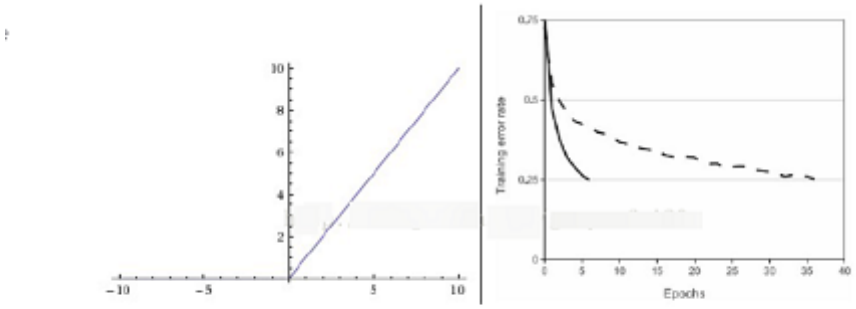
**Why does AlexNet achieve better results?**

1. **Relu activation function is used.**

Relu function: f (x) = max (0, x)




ReLU-based deep convolutional networks are trained several times faster than tanh and sigmoid- based networks. The following figure shows the number of iterations for a four-layer convolutional network based on CIFAR-10 that reached 25% training error in tanh and ReLU:




**Left:** Rectified Linear Unit (ReLU) activation function, which is zero when x < 0 and then linear with slope 1 when x > 0. **Right:** A plot from Krizhevsky et al. (pdf) paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.

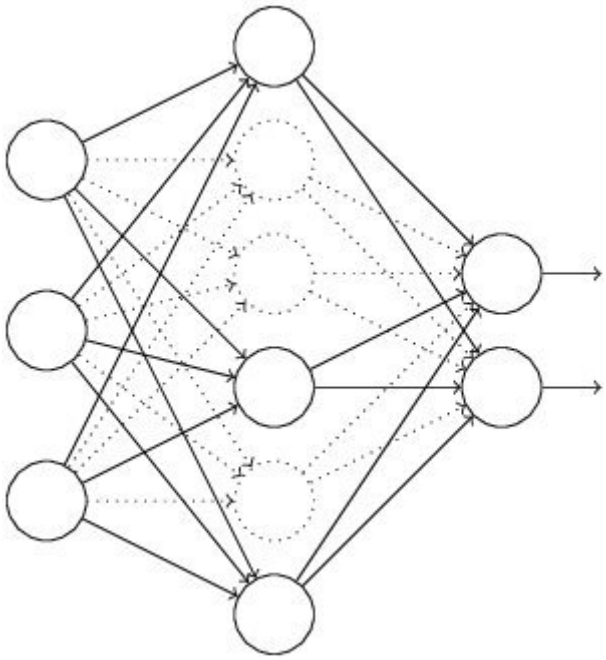
1. **Standardization ( Local Response Normalization )**

After using ReLU f (x) = max (0, x), you will find that the value after the activation function has no range like the tanh and sigmoid functions, so a normalization will usually be done after ReLU, and the LRU is a steady proposal (Not sure here, it should be proposed?) One method in neuroscience is called "Lateral inhibition", which talks about the effect of active neurons on its surrounding neurons.

$$b^i_{x,y} = a^i_{x,y} / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

1. **Dropout**

Dropout is also a concept often said, which can effectively prevent overfitting of neural networks. Compared to the general linear model, a regular method is used to prevent the model from overfitting. In the neural network, Dropout is implemented by modifying the structure of the neural network itself. For a certain layer of neurons, randomly delete some neurons with a defined probability, while keeping the individuals of the input layer and output layer neurons unchanged, and then update the parameters according to the learning method of the neural network. In the next iteration, rerandom Remove some neurons until the end of training.



1. **Enhanced Data ( Data Augmentation )**

**In deep learning, when the amount of data is not large enough, there are generally 4 solutions:**

Data augmentation- artificially increase the size of the training set-create a batch of "new" data from existing data by means of translation, flipping, noise

Regularization——The relatively small amount of data will cause the model to overfit, making the training error small and the test error particularly large. By adding a regular term after the Loss Function , the overfitting can be suppressed. The disadvantage is that a need is introduced Manually adjusted hyper-parameter.

Dropout- also a regularization method. But different from the above, it is achieved by randomly setting the output of some neurons to zero

Unsupervised Pre-training- use Auto-Encoder or RBM's convolution form to do unsupervised pre-training layer by layer, and finally add a classification layer to do supervised Fine-Tuning

In [1]:

```
#Importing library
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
import numpy as np

np.random.seed(1000)
```

In [6]:

```
model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(filters = 96, input_shape = (224, 224, 3), kernel_size = (11, 11), strides
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(10))
```

```
model.add(Activation('softmax'))
```

In [7]:

```
#Model Summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
activation (Activation)      (None, 54, 54, 96)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952
_________________________________________________________________
activation_1 (Activation)    (None, 17, 17, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
```

In [ ]: