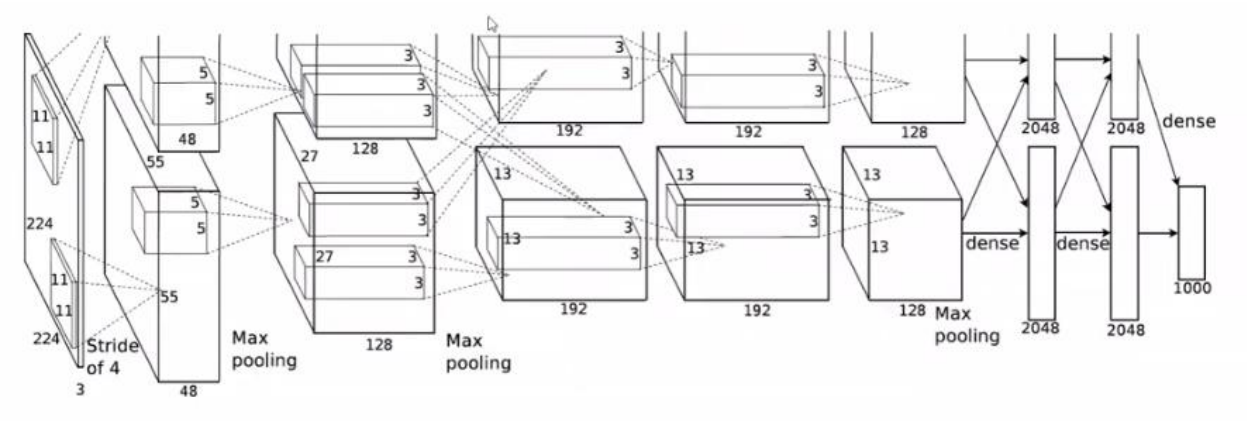
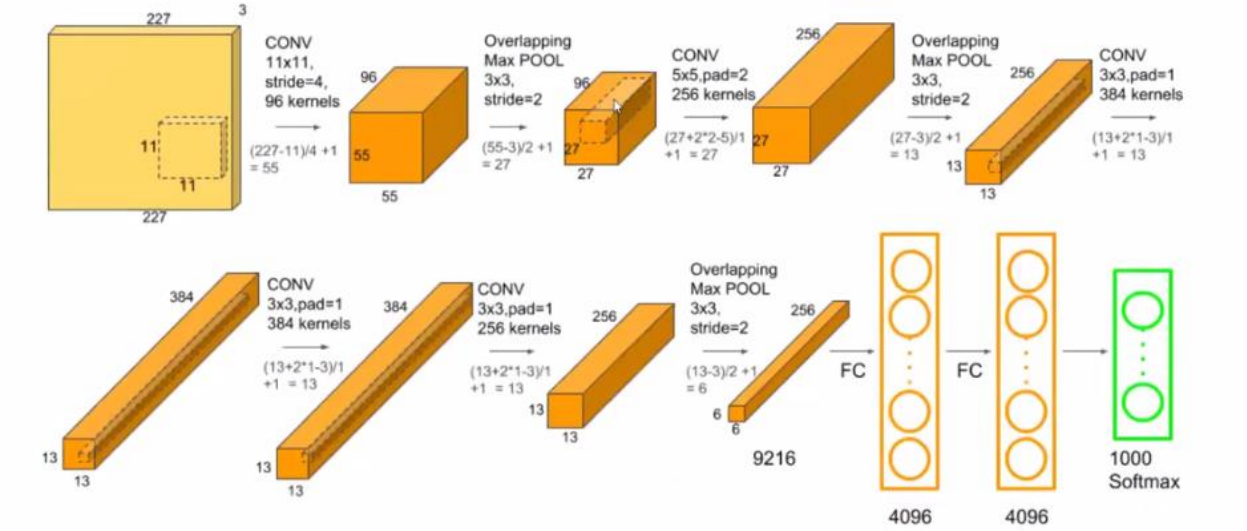
AlexNet:

Alex Krizhevsky - 2012

In this architecture they have made so many changes.

Architecture:





In this model we will be able to get 80.2%

Initially they have tested with CFA10 dataset and they have almost 6000 images was there for 10 classes.

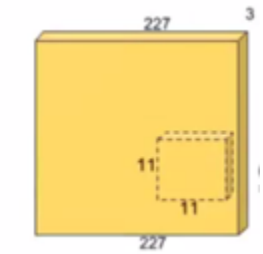
This is the first time were people trained their model in GPU (GTX 580 3GB).

In our previous model we have taken the LeNet and we used the kernel size is 5X5

And the model depth was not that much.

This model will give a breakthrough in computer vision.

Here they have taken of image size 227X227 of channel 3 (R, G, B)



Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						

Step 1:

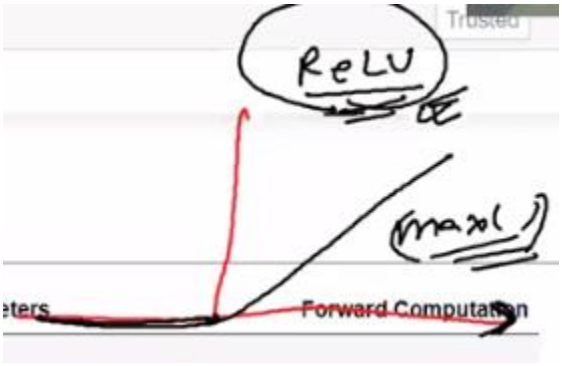
This is the first model people have used relu for the data normalization.

Relu – For all the positive input it always give you the maximum of input and all the negative values or inputs it is going to give you the zero value.

In our previous model we have used tanh function and here we are using relu. Relu is a linear function whereas tanh is an exponential function. So it is little bit more complex and it consumes lot of processing power and the training time will be very high than Relu in terms of computation.

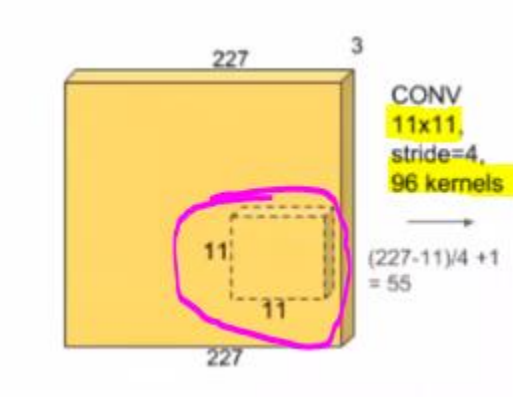
Tanh always fluctuate between -1 to 1 and they normalize the dataset between -1 to 1.

Using Relu we are trying to simplify the calculations and it is better option compare to tanh.



In a first layer, they have applied convolution of kernel size of 11X11 96 filters and they have applied relu function with stride 4 and no padding.

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600



So what is the number of the label you will be able to get in the next layer?

Dimension of the original images (N)

Filters (F)

Strides (S)

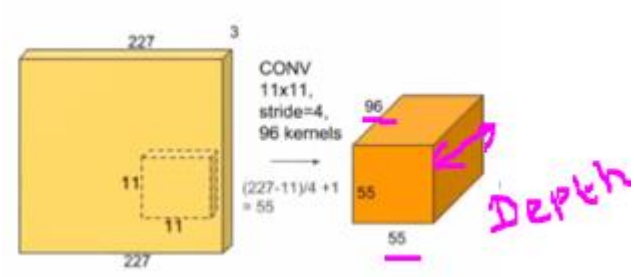
Below is the formula for if stride is more than one:

(N – F / S)+1

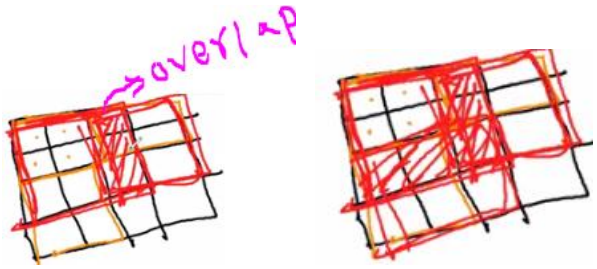
(227 – 11/4)+1

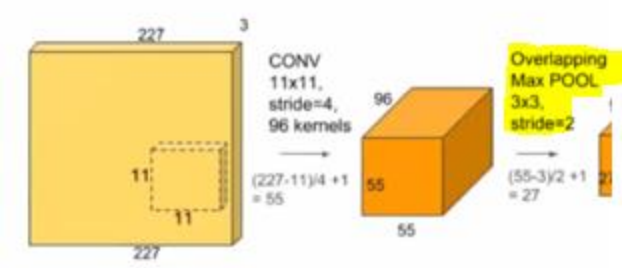
54 + 1 = 55

So totally we will be able to get 55X55 of 96 features or properties



Then they have used **overlapping max pool.**





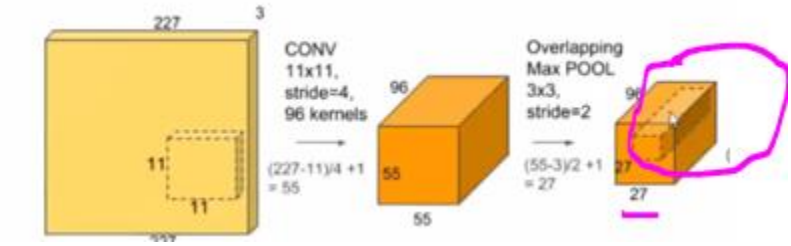
Here we have taken 3X3 of stride 2 and it will be some overlap over like above. This is the new concept which have introduced in the AlexNet architecture.

Why this overlapping max pool?

In our max pool we take the maximum or dominant property and we lose some data. So that why they have introduced the concept called Overlapping maximum pool. So it goes to the next layer by doing overlapping max pool

Calculation:

55-3/2 + 1 = 26+1 = 27



Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27 →	O/P					

This portion indicates they have applied overlapping Max pool in all the layers or in short in-depth. They have this kind of pooling on 96 layers. So depth wise there is no changes.

After this they have implemented one concept is called as **LRN (Local Response Normalization)**

To control the relu activation function they have used LRN.

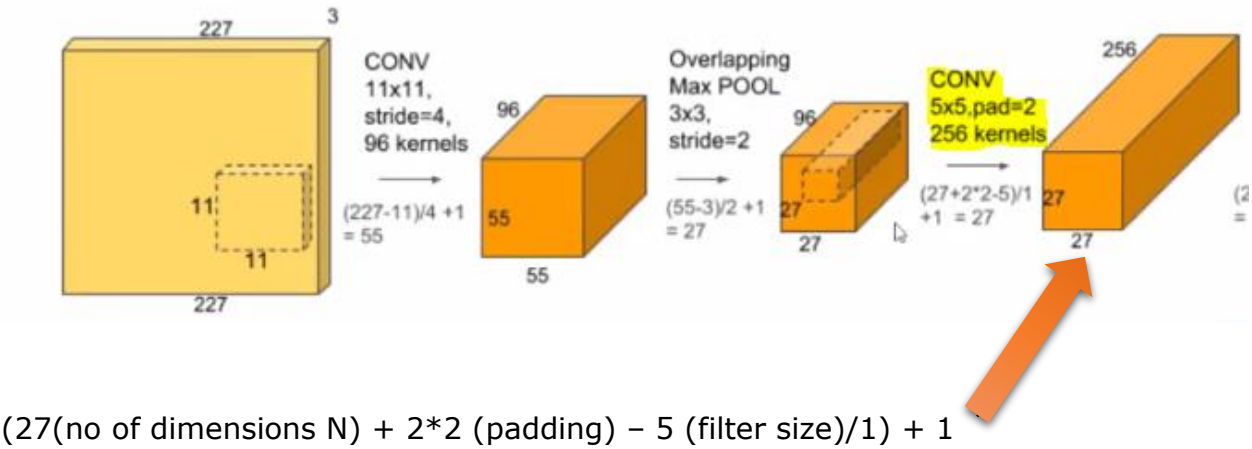
This is also the first time which they have been introduced in the AlexNet architecture.

In **LRN,** Suppose I have 1X1 array and in-depth you will be able to find out multiple arrays behind one array. So, instead of taking all the arrays they have normalized this array that is called as LRN.

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						

After Normalization again they have implemented a Convolution with filter size 5X5 with padding 2 and stride 1.

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*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						



(27(no of dimensions N) + 2*2 (padding) – 5 (filter size)/1) + 1

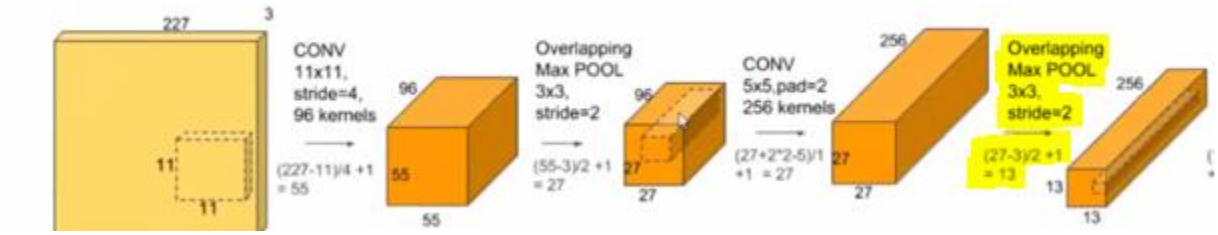
(27+2*2-5)/1 + 1 = 27

Here from 96 features they have extracted again 256 features.

So here the kernel is changing from 11x11 to 3X3 to 5X5

After this I'm doing Overlapping maximum pool of filter size 3X3 with stride 2

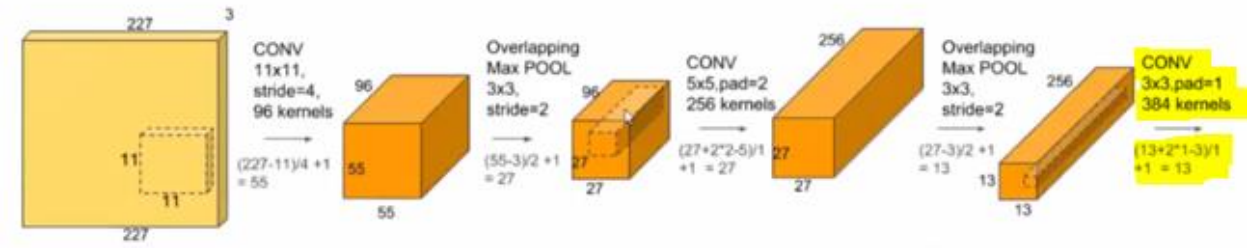
Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656	(5 * 5 * 96 + 1) * 256 * 27 * 27=448084224
256 * 27 * 27						
Max Pooling	3 * 3		2			
256 * 13 * 13						

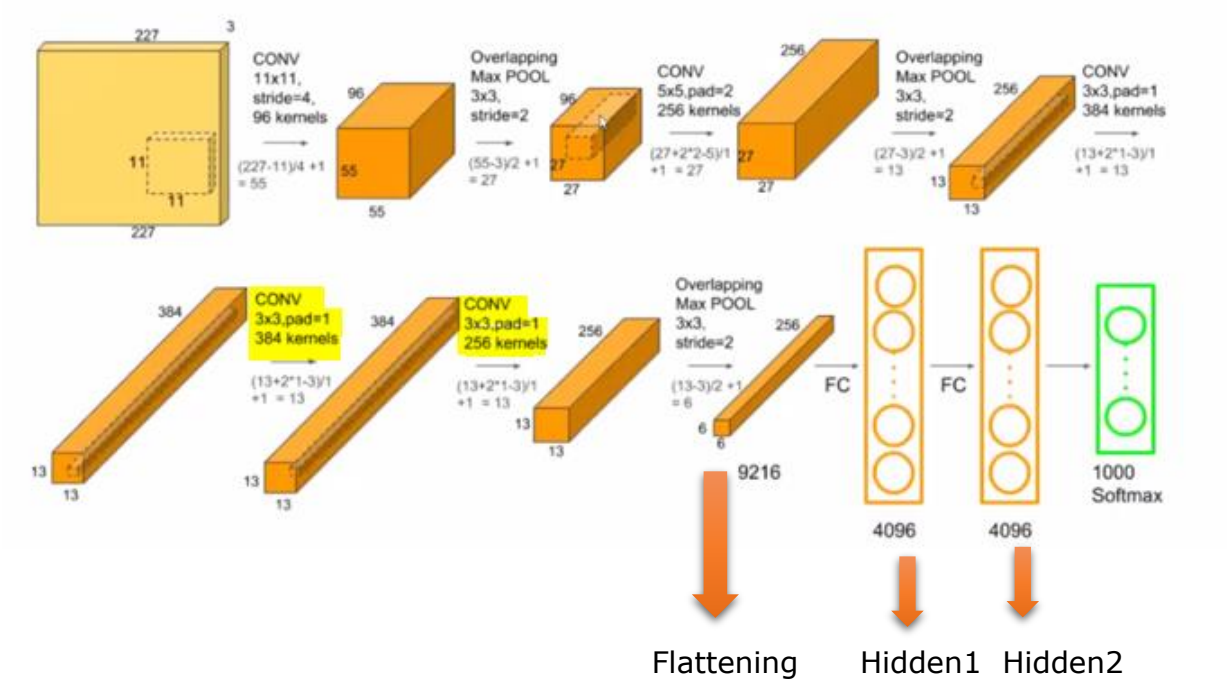


Then we will be able to get 256 features of size 13X13.

And they doing again LRN.

After that they have used convolution and they have taken 384 kernels of size 3X3 along with Relu.

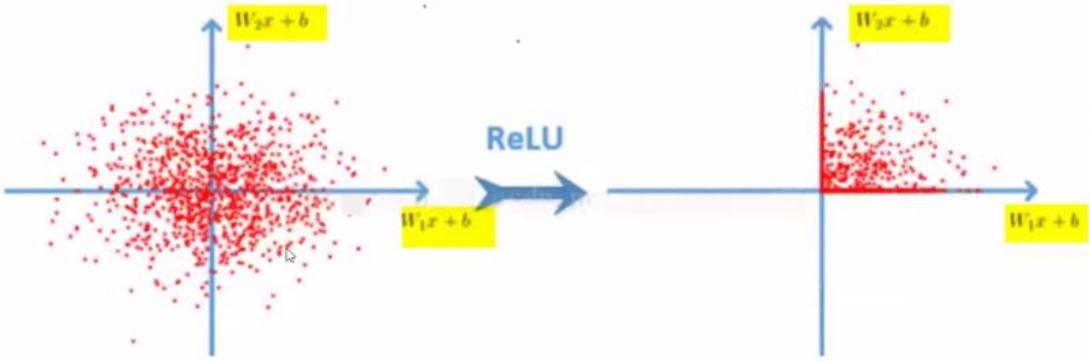




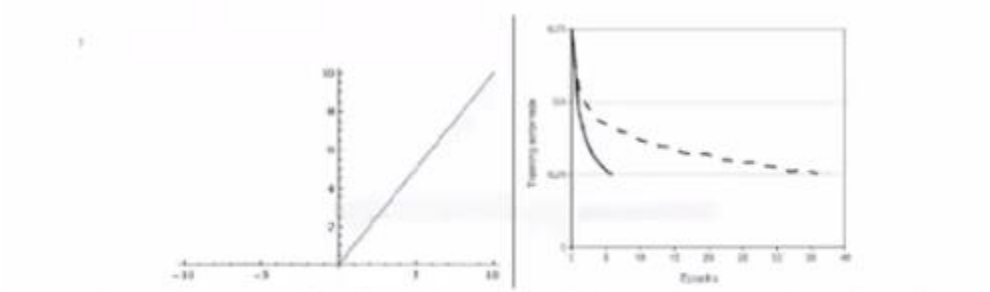
Why does AlexNet achieve better results?

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.

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.

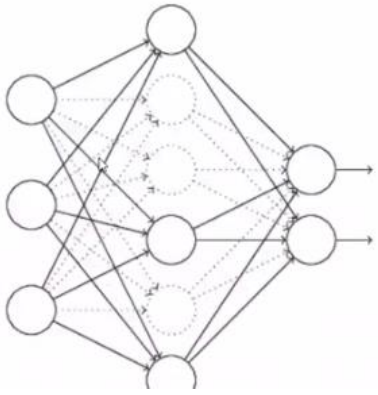
2. **Standardization (Local Response Normalization)**

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.

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

3. **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.



Dropout – Suppose if I have a multiple layers, based on the probability if I activate some of the weights or deactivate some of the weights in a learnable parameter that is called as dropout.

In every epoch or in every step size there will be a different kinds of networks that we are going to activate or going to deactivate.

4. **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

Data Augmentation:

Suppose I have images, whenever you work with the real time environment you will face this kind of challenge for sure.

Whenever I'm going to train my images for sure you have taken or captured the images from certain distance or light condition or some rotations. Now in a real time if you are going to get an images which is little bit distorted or very far or with very low light conditions. So in this case whenever you are trying to test your model and our model is not going to perform better. Even though the loss was very less but still my model is not performing well.

To prevent this kind of scenario we always try to perform Data augmentation. In each and every project we need to apply data augmentation technique. We written own functions as well by combining different kinds of functions. So that, by calling one function you will be able to do data augmentation and it is applicable for any kind of a datasets.

In short augmented dataset's means. We try to flip the dataset or rotate the dataset horizontally or vertically, we try to change the color of the dataset's far or near. So this is called as data augmentation.

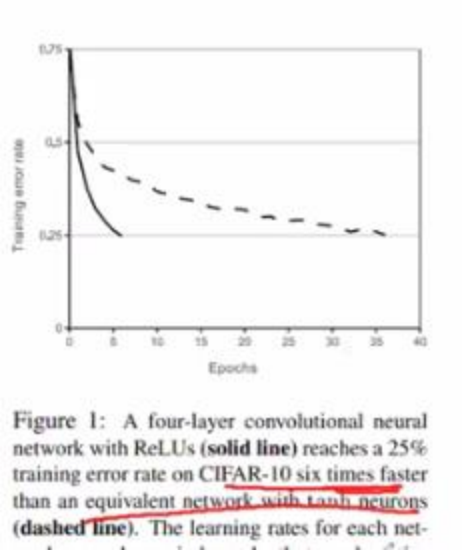


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each net-

It is six times faster than network with tanh functions.

In case of tanh it reaches 5 epochs and in case of a relu it reaches 35 epochs.

5 Details of learning

We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model's training error. The update rule for weight w was

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

Momentum – Whenever you are trying to calculate a new weight or whenever you are trying to fall a local minima or minima basically how much dependencies or effect is going to consider from a past.

Adam optimizer, Ada delta and Ada grade Here we are talking about the momentum where we trying to consider this much effect of previous changes and then I will try to do a next changes.

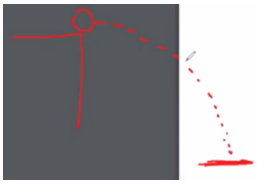
Here you will be able to observe that momentum of 0.9 and weight decay of 0.0005

So what they have done is they have controlled the weight decay like we are not allow not to drop down of this weight under 0.0005

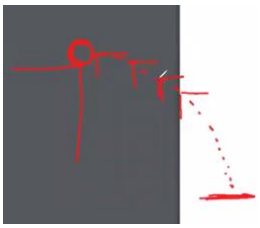
Second this they have applied the momentum

When we talk about the momentum. Consider there is a ball which kept at some height.

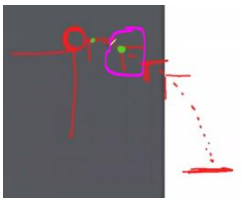
If we try to throw the ball or kick the ball, for sure the ball will try to touch the particular surface.



To touch this surface it's going to follow some path or direction or trajectory. So every point of a time there will be a multiple forces which will be applied like centripetal and centrifugal force.



So every time, whenever it will try to move from first point to the next point in a air. So basically in the next point what will be the velocity

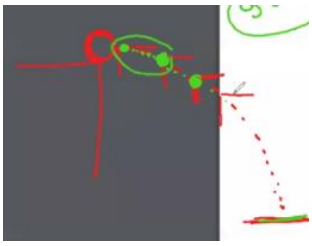


Because if it is trying to fall can I say that it will be able to accelerate itself with respect to $9.8 m/s^2$. So this is the acceleration based on Gravitational force.

Acceleration - Velocity is not same actually it changes the velocity every point of a time.

So whenever we are trying to calculate velocity in that particular point (second point)

In third point the velocity will be like having the effect of combined affect or previous all of the velocities.



At every point of a time the velocity is calculated is based on all the previous velocities. That was been represented by here

$$\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

Means whatever changes we are trying to make over here, my next weight will depends upon all the previous changes of weights that we have done so that is called as momentum.

Here 0.9 momentum describes, It simply means that changes I have observed in a previous one, all of those changes are considered as 0.9 % of changes not a weight. Which means the changes I have done. Here 0.9 is somewhat a constant. Eta = 0.9

5 Details of learning

We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model's training error. The update rule for weight w was

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.

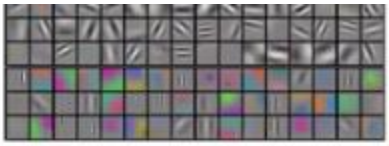


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

Momentum – It is a cumulative summation of the previous changes that you have done.

Coding:

```
!pip install tflearn
```

```
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)

# (2) Get Data
import tflearn.datasets.oxflower17 as oxflower17
x, y = oxflower17.load_data(one_hot=True)

# (3) Create a sequential model
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'))
# Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation before passing it to the next Layer
model.add(BatchNormalization())
```

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

# Passing it to a dense layer
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())
```

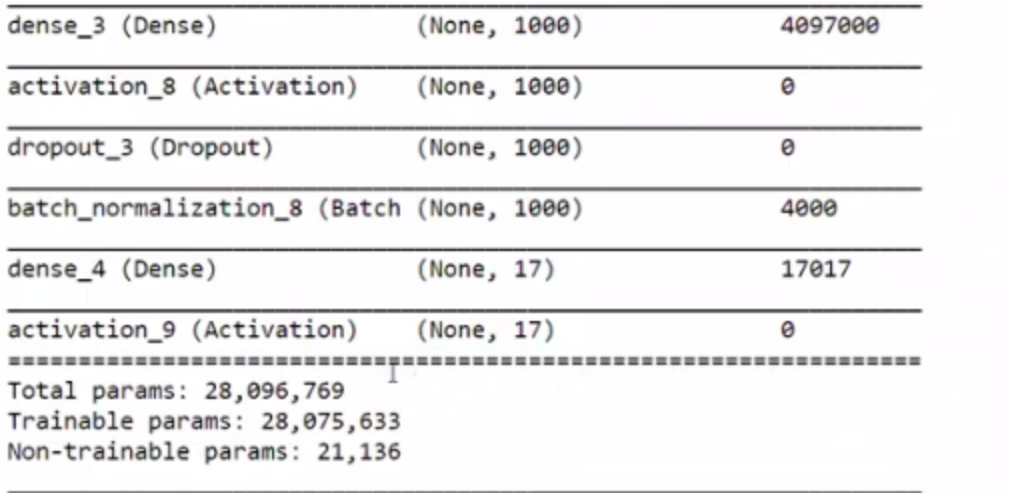
```
# 3rd Dense Layer
model.add(Dense(1000))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# Output Layer
model.add(Dense(17))
model.add(Activation('softmax'))

model.summary()

# (4) Compile
model.compile(loss='categorical_crossentropy', optimizer='adam',\
 metrics=['accuracy'])

# (5) Train
model.fit(x, y, batch_size=64, epochs=1, verbose=1, \
validation_split=0.2, shuffle=True)
```



dense_3 (Dense)	(None, 1000)	4097000
activation_8 (Activation)	(None, 1000)	0
dropout_3 (Dropout)	(None, 1000)	0
batch_normalization_8 (Batch	(None, 1000)	4000
dense_4 (Dense)	(None, 17)	17017
activation_9 (Activation)	(None, 17)	0

Total params: 28,096,769
Trainable params: 28,075,633
Non-trainable params: 21,136

Disadvantages:

Here what people have done is they have taken the kernel size and number of kernels also is different for every layer. Suppose if you start creating more number of layers in that case it will cross even 16 million.

Although we are using Relu, so that our learning complexity will be less but still you will be able to find out the loss that is able to drop. If my model is too dense, so in that case I will be like start losing the accuracy.