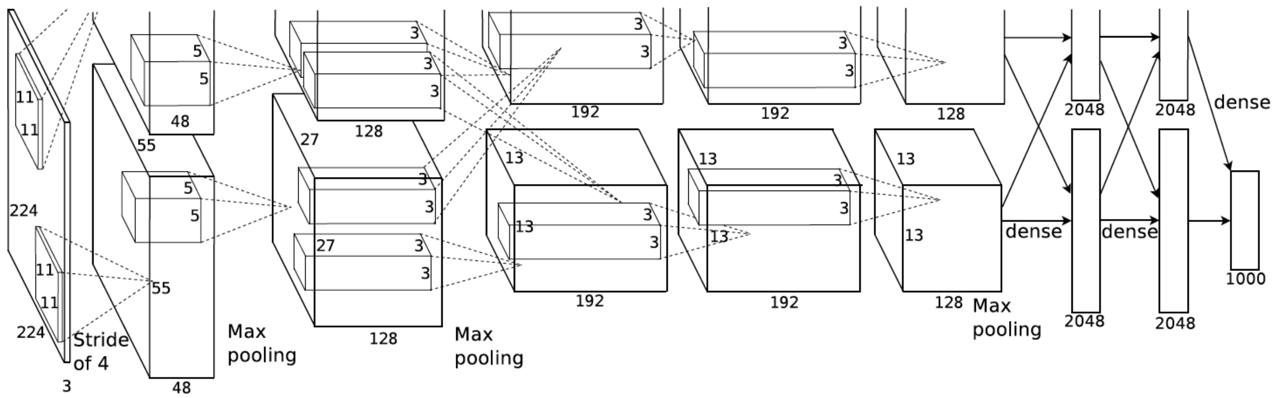
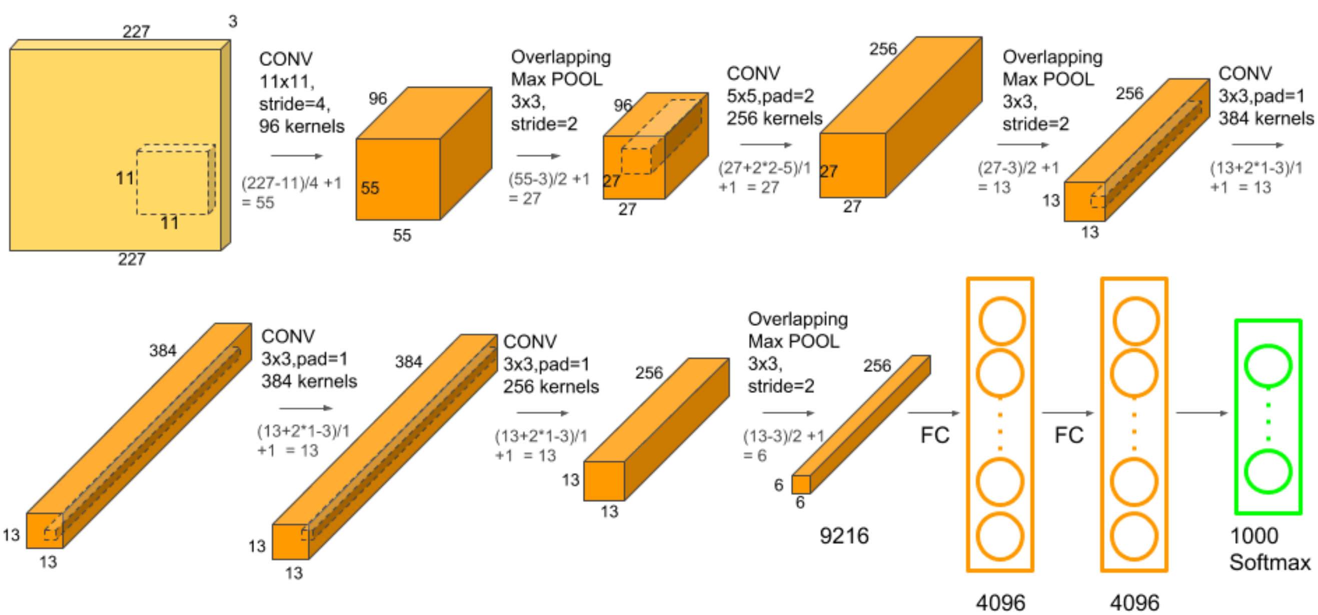
Introduction

AlexNet was designed by Hinton, winner of the 2012 ImageNet competition, and his student Alex Krizhevsky. It was also after that year that more and deeper neural networks were proposed, such as the excellent vgg, GoogleLeNet. Its official data model has an accuracy rate of 57.1% and top 1-5 reaches 80.2%. This is already quite outstanding for traditional machine learning classification algorithms.





The following table below explains the network structure of AlexNet:

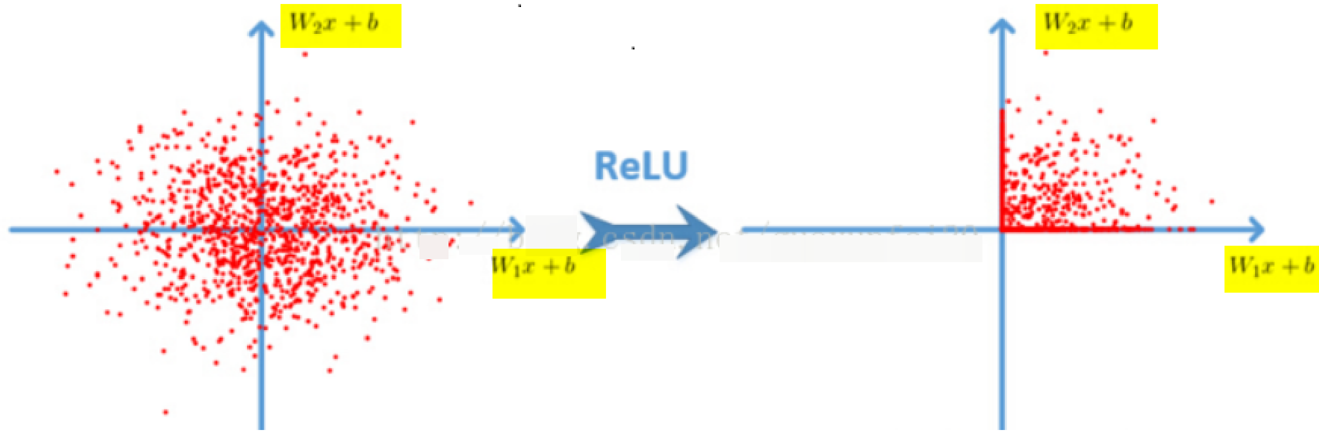
Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
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						
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%)

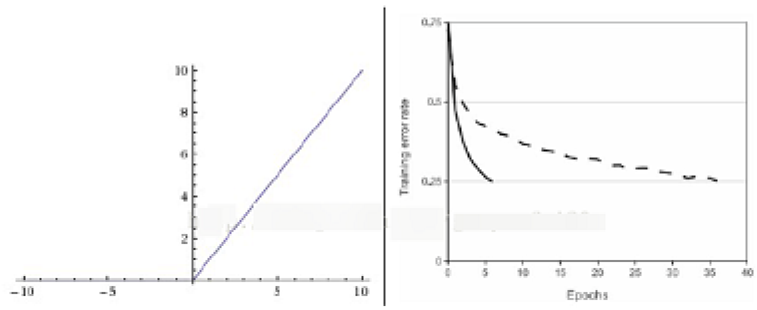
Why does AlexNet achieve better results?

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.

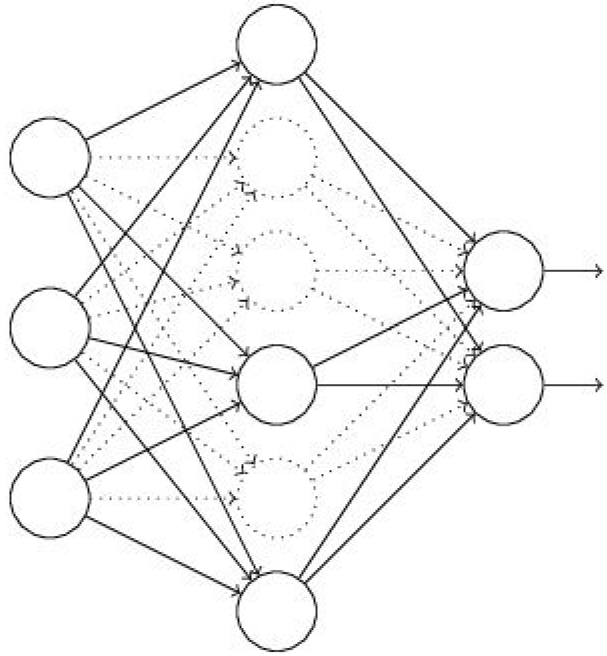
2. **Standardization (Local Response Normalization)**

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.

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

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.



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.

Code Implementation

In [1]:
```
!pip install tflearn
```

```
Collecting tflearn
  Downloading tflearn-0.5.0.tar.gz (107 kB)
     -------------------------------------- 107.3/107.3 kB 1.6 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: numpy in d:\anaconda setup\lib\site-packages (from tflea
rn) (1.23.5)
Requirement already satisfied: six in d:\anaconda setup\lib\site-packages (from tflear
n) (1.16.0)
Requirement already satisfied: Pillow in d:\anaconda setup\lib\site-packages (from tfle
arn) (9.4.0)
Building wheels for collected packages: tflearn
  Building wheel for tflearn (setup.py): started
  Building wheel for tflearn (setup.py): finished with status 'done'
  Created wheel for tflearn: filename=tflearn-0.5.0-py3-none-any.whl size=127290 sha256
=35d3e450583535c181c918100373f68882a1b7fc62f4c54e0149a8b043bb17db
  Stored in directory: c:\users\shehryar gondal\appdata\local\pip\cache\wheels\5d\83\f7
\63e33ac9c0560f1dddb2ecff627b8ab6cb076d4b1996416be1
Successfully built tflearn
Installing collected packages: tflearn
Successfully installed tflearn-0.5.0
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
```

In [8]:
```
# Get Data
import tflearn.datasets.oxflower17 as oxflower17
from keras.utils import to_categorical

x, y = oxflower17.load_data()

x_train = x.astype('float32') / 255.0
y_train = to_categorical(y, num_classes=17)
```

In [10]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(1360, 224, 224, 3)
(1360, 17)
```

In [5]:

```
# 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)
model.add(Activation('relu'))

# Pooling
model.add(MaxPooling2D(pool_size=(3,3), 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='same'))
model.add(Activation('relu'))

# Pooling
model.add(MaxPooling2D(pool_size=(3,3), 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=(3,3), 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
```

```python
model.add(Dense(4096))
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()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/keras/layers/normalizat
ion/batch_normalization.py:581: _colocate_with (from tensorflow.python.framework.ops) i
s deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
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, 26, 26, 96)       0         
 )                                                               
                                                                 
 batch_normalization (BatchN  (None, 26, 26, 96)       384       
 ormalization)                                                   
                                                                 
 conv2d_1 (Conv2D)           (None, 26, 26, 256)       614656    
                                                                 
 activation_1 (Activation)   (None, 26, 26, 256)       0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 12, 12, 256)      0         
 2D)                                                             
                                                                 
 batch_normalization_1 (Batc  (None, 12, 12, 256)      1024      
 hNormalization)                                                 
                                                                 
 conv2d_2 (Conv2D)           (None, 10, 10, 384)       885120    
                                                                 
 activation_2 (Activation)   (None, 10, 10, 384)       0         
                                                                 
 batch_normalization_2 (Batc  (None, 10, 10, 384)      1536      
 hNormalization)                                                 
                                                                 
 conv2d_3 (Conv2D)           (None, 8, 8, 384)         1327488   
                                                                 
 activation_3 (Activation)   (None, 8, 8, 384)         0         
                                                                 
 batch_normalization_3 (Batc  (None, 8, 8, 384)        1536      
 hNormalization)                                                 
                                                                 
 conv2d_4 (Conv2D)           (None, 6, 6, 256)         884992    
                                                                 
 activation_4 (Activation)   (None, 6, 6, 256)         0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)        0         
 2D)                                                             
                                                                 
 batch_normalization_4 (Batc  (None, 2, 2, 256)        1024      
 hNormalization)                                                 
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 4096)              4198400   
                                                                 
 activation_5 (Activation)   (None, 4096)              0         
                                                                 
 dropout (Dropout)           (None, 4096)              0         
                                                                 
 batch_normalization_5 (Batc  (None, 4096)             16384     
 hNormalization)                                                 
```

```
 dense_1 (Dense)             (None, 4096)              16781312

 activation_6 (Activation)   (None, 4096)              0

 dropout_1 (Dropout)         (None, 4096)              0

 batch_normalization_6 (Batc  (None, 4096)             16384
 hNormalization)

 dense_2 (Dense)             (None, 17)                69649

 activation_7 (Activation)   (None, 17)                0

=================================================================
Total params: 24,834,833
Trainable params: 24,815,697
Non-trainable params: 19,136
_________________________________________________________________
```

In [11]:

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [12]:

```python
# Train
model.fit(x_train, y_train, batch_size=64, epochs=5, verbose=1,validation_split=0.2, shu
```

```
Train on 1088 samples, validate on 272 samples
Epoch 1/5
1088/1088 [==============================] - ETA: 0s - loss: 3.6493 - acc: 0.2858

/usr/local/lib/python3.10/dist-packages/keras/engine/training_v1.py:2335: UserWarning:
`Model.state_updates` will be removed in a future version. This property should not be
used in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates

1088/1088 [==============================] - 11s 10ms/sample - loss: 3.6493 - acc: 0.28
58 - val_loss: 3.1504 - val_acc: 0.0699
Epoch 2/5
1088/1088 [==============================] - 2s 2ms/sample - loss: 2.0367 - acc: 0.4357
- val_loss: 5.0040 - val_acc: 0.0699
Epoch 3/5
1088/1088 [==============================] - 2s 2ms/sample - loss: 1.7029 - acc: 0.5055
- val_loss: 7.8723 - val_acc: 0.0551
Epoch 4/5
1088/1088 [==============================] - 2s 2ms/sample - loss: 1.5586 - acc: 0.5441
- val_loss: 5.6248 - val_acc: 0.0699
Epoch 5/5
1088/1088 [==============================] - 2s 2ms/sample - loss: 1.3488 - acc: 0.6094
- val_loss: 7.2296 - val_acc: 0.0699
```

Out[12]:

```
<keras.callbacks.History at 0x7fe7577e01f0>
```

In [ ]: