**Building AlexNet with Keras**

As the legend goes, the deep learning networks created by Alex Krizhevsky, Geoffrey Hinton and Ilya Sutskever (now largely know as AlexNet) blew everyone out of the water and won Image Classification Challenge (ILSVRC) in 2012. This heralded the new era of deep learning. [AlexNet](https://en.wikipedia.org/wiki/AlexNet) is the most influential modern deep learning networks in machine vision that use multiple convolutional and dense layers and distributed computing with GPU.

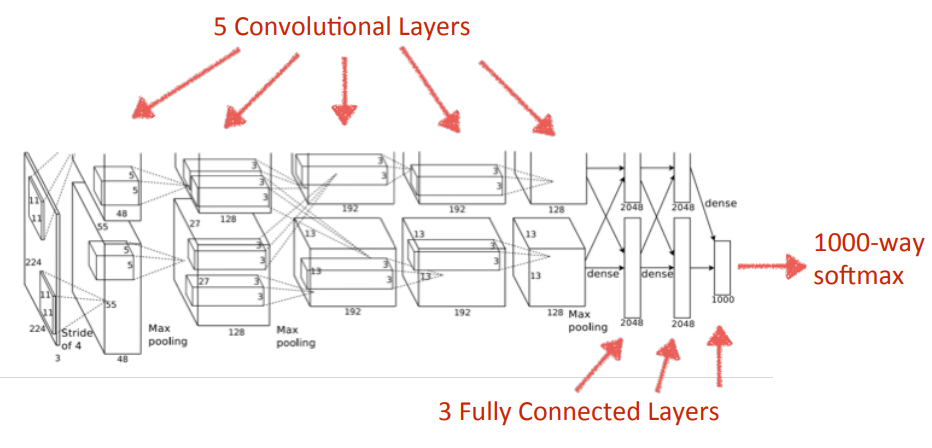
Along with [LeNet-5](http://yann.lecun.com/exdb/lenet/), AlexNet is one of the most important & influential neural network architectures that demonstrate the power of [convolutional layers](https://en.wikipedia.org/wiki/Convolutional_neural_network) in machine vision. So, let’s build AlexNet with [Keras](https://keras.io/) first, them move onto building it in [TensorFlow](https://www.tensorflow.org/).

**Dataset**

We are using [OxfordFlower17](http://www.robots.ox.ac.uk/~vgg/data/flowers/17/) in the tflearn package. The dataset consists of 17 categories of flowers with 80 images for each class. It is a three dimensional data with RGB colour values per each pixel along with the width and height pixels.

**AlexNet Architecture**

AlexNet consist of 5 convolutional layers and 3 dense layers. The data gets split into to 2 GPU cores. The image below is from the first reference the AlexNet Wikipedia page [here](http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf).



**AlexNet with Keras**

I made a few changes in order to simplify a few things and further optimise the training outcome. First of all, I am using the sequential model and eliminating the parallelism for simplification. For example, the first convolutional layer has 2 layers with 48 neurons each. Instead, I am combining it to 98 neurons.

The original architecture did not have batch normalisation after every layer (although it had normalisation between a few layers) and dropouts. I am putting the batch normalisation before the input after every layer and dropouts between the fully-connected layers to reduce overfitting.

When to use batch normalisation is difficult. Everyone seems to have opinions or evidence that supports their opinions. Without going into too much details, I decided to normalise before the input as it seems to make sense statistically.

**Code**

Here is the code example. The input data is 3-dimensional and then you need to flatten the data before passing it into the dense layer. Using cross-entropy for the loss function, adam for optimiser and accuracy for performance metrics.

As the network is complex, it takes a long time to run. It took about 10 hours to run 250 epochs on my cheap laptop with CPU. The test dataset accuracy is not great. This is probably because we do not have enough datasets. I don’t think 80 images each is enough for convolutional neural networks. But, it still runs.

It’s pretty amazing that what was the-state-of-the-art in 2012 can be done with a very little programming and run on your $700 laptops!

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

**Next Steps**

Let’s write AlexNet with TensorFlow.

In the next post, we will build AlexNet with TensorFlow and run it with [AWS SageMaker](https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html) (see [**Building AlexNet with TensorFlow and Running it with AWS SageMaker**](http://www.mydatahack.com)).