# COMP9517

**Lab 5, S1 2018**

**Convolutional Neural Networks**

In this tutorial we will look at an implementation of a simple convolutional neural network (CNNs) using the Keras library in Python.

**Keras** is a high-level level neural network library. Keras can be used with several backend libraries such as Tensorflow, Theano, or CNTK for computing numerical dataflow graphs which is how most neural networks are implemented. These backends allow neural networks to be trained across a wide range of devices include CPUs, GPUs, and custom ASICs such as Google's TPUs. Some libraries like Tensorflow are very large and complex (some people say it is overengineered), and since you already have had to deal with OpenCV there is no reason to punish you with another big library.

As discussed in the lectures, the main building blocks for CNNs are typically repeated series of layers. Typical layers include:

- convolutional layers
- max-pooling layers
- dense layers

These components are then typically assembled like LEGO into architectures. They can be very complicated, although we will not discuss this here.

There are two APIs in Keras for implementing neural networks: the functional API and the sequential API. In this tutorial we shall use the sequential API. To get the layers to build your model using the sequential API, import them as follows:

```
from keras.layers import Dense
```

A very simple two layer neural network may be implemented as follows:

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

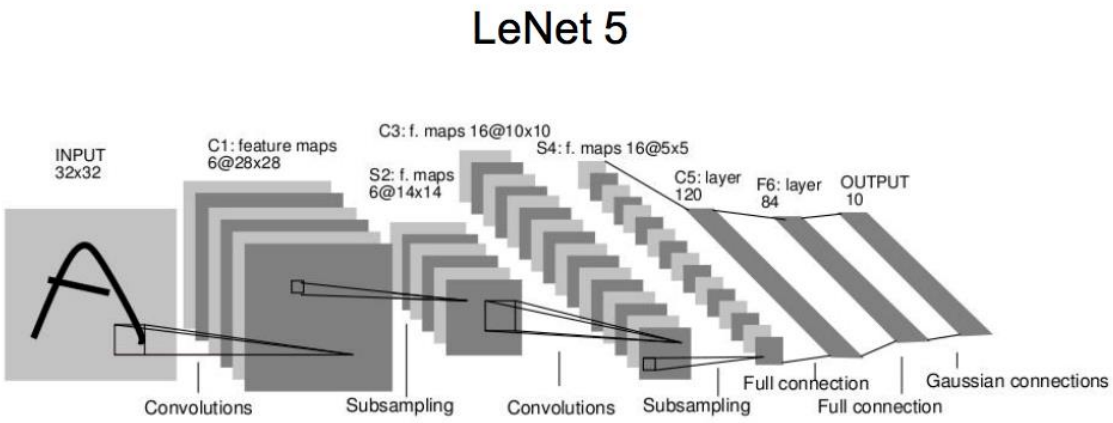
Dense layers are layers where every neuron is connected to every other in the next layer. It is usually computationally prohibitive to train deep fully connected neural networks, this is why convolutional layers are widely used.

In this tutorial, we shall examine an image classification problem with a simple dataset, the Asirra dataset. There are 2 types of images in this dataset, images of cats and images of dogs, so this is a binary classification problem.

You can download the dataset from here:
https://drive.google.com/file/d/1eYe4EK1LPaKv6Soi-Twv-dYIfXIUL2uv/view?usp=sharing

It will take too long to train even this relatively small dataset on our whimpy vlab machines, so we shall downsample the training images and only select a subset of the data to work with when training. You can find a script to do this in ***tut5_process_data.py***

Next, we shall form and train our model with Keras. Early CNNs did not have many layers, but nevertheless they were able to achieve state of the art results for tasks such as optical character recognition. Below is a diagram of the architecture developed by Yann LeCun for the MNIST dataset.




The final Gaussian layer is what we would now call a fully connected sigmoid layer. Note there are 10 nodes in his output layer due to the 10 classes. We only need one, but this is already in the code. The notation 6@28x28 mean 6 sets of 28x28 filters.

**QUESTION FOR ASSESSMENT:**
Complete the code in the skeleton (***tutorial5_incomplete_code.py***). All you need to do is input the missing dimensions for the layers. Show the tutor when you have completed 10 epochs of training on your machine.

**FURTHER TASKS:**
Try experimenting with the architecture improve the accuracy:

- add more layers of convolutions and pooling
- use a different optimiser
- add some regularisation such as a dropout layer or L2 regularisation

**FOR FURTHER READING:**
Andrey Karpathy has an excellent set of tutorials on CNNs for computer vision applications which can be found here:
http://cs231n.github.io/convolutional-networks/

The Tensorflow playground can be instructive to understand the filters which the convolutional layers are learning:
https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.82627&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false