**Weekly report**

Split our work into studying theory (Sebastian) and implementing our data into the MNIST tutorial (Elliot) so that we wouldn’t be both working on the same program.

We reduced the one-hot vector saved file size by changing values from floats to ints, but the file size is still far bigger than the file size for actual images.

The current plan is to create one-hot vectors on the fly, as needed for the program instead of creating them all beforehand. Hopefully this will reduce memory taken.

Fixed some issues with the zipped list method to make it faster and more lightweight.

*Theory overview*

Looked at a discussion of kNN (k Nearest Neighbours, unsupervised learning) vs SVM (support vector machine, supervised learning). kNN has been used by some as a preliminary step to supervised learning, but is not going to be useful for our project as it struggles with a large number of classifiers, and 3755 is probably far beyond its scope.

SVM is an optimal classifier (in that it solves an optimization problem and finds a maxima/minima) however it is a binary classifier. As such we would have to use as many SVMs as classes that we have. This would be far too expensive to run on our 3755 classes.

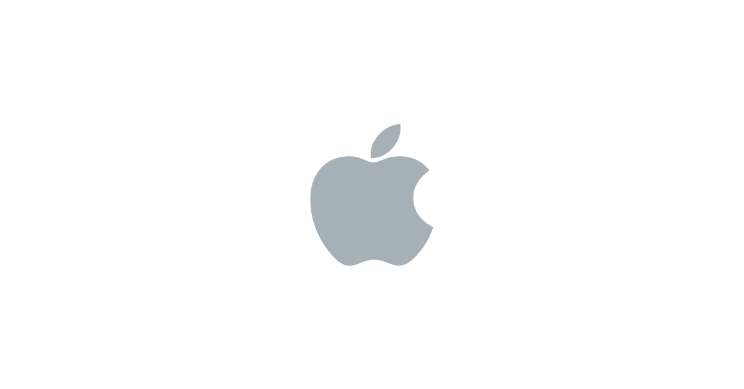


Figure : Key players in the OCR game

Dropbox uses a somewhat exotic pipeline for their optical character recognition, combining CNNs with BDLSTM (bi-directional long short term memory, a speech recognition tool) alongside connectionist temporal classification.

Google Translate switched to neural networks from a statistical translation machine in November of 2016. They also use long short term memory, but as part of a recurrent neural network (RNN). They say that using an RNN lets use the context in a paragraph to identify each word, rather than just trying to identify the word itself. This is clearly a far more complex implementation: RNNs work over time, and use time as a variable unlike CNNs.

Apple is also in the OCR game but they use a refreshingly simple approach. They employ a CNN architecture consisting of just 2 convolutional and pooling layers and 1 fully connected layer.

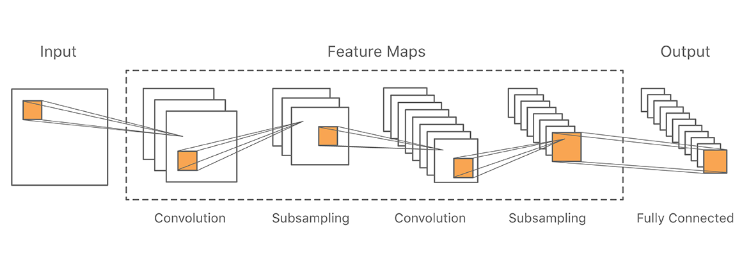


Figure : The architecture of Apple's deep learning CNN

They model their approach on two papers published in 2011, using a committee of CNNs and then averaging the results of independent classifiers. This decreases training time and increases accuracy however it requires parallelisation in the training (on GPUs) if it is to be implemented fast.

The fact that Apple uses CNNs is very promising for our CNN approach. Apple highlights that they want a lightweight and fast detection method over a very accurate one. They expressed concerns that CNN might be too complex, but after implementing CNNs for over 30,000 characters (standard Chinese characters, nouns and names, region-specific characters) they found the CNN implementation was still fast after training.

The committee approach seems promising, and is worth trying. However we would like to try to transfer learning approach first as detailed in previous reports, as we think there is more scope for originality there, and it does not favour implementation on GPUs so we could keep working with CPUs for now.

Two interesting things highlighted in the papers regarding the committee is firstly how the authors distorted their data set (through skewing, scaling, and rotating the MNIST data) at every **epoch** rather than just as a method of generating more data to work with at the start. This is definitely something we want to implement to prevent our model from overfitting and to keep it seeing ‘new’ data. Secondly they mentioned **annealing**, that is, gradually decreasing their learning rate over time by some multiplicative factor.

What if we could separate the Chinese characters not by those that are similar, but by those that are most dissimilar? Training separate CNNs on ‘dissimilar groups’ would make each CNN far more accurate on characters within that group. These CNNs could then be run separately for OCR or formed into a committee.

**Action points for the next week**

1. Read in all the files we have, created a zipped list for all these files so we have a list to label every unique character in our data set.

2. Read in every file we have and create saved files of each image, character, and so on to work with.

3. Keep working on a ML implementation for our data >> is there a better method than one-hot vectors?