Q: In the given notebook they standardise the input values using the mean and standard deviation before training, but I can't really see a reason for doing this mentioned. Does this just enable the CNN to perform better on the data?

> **A:** CNNs learn by continually adding gradient error vectors * learning rate computed from backpropagation to various weight matrices throughout the network as training examples are passed through. If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ proportionally from one another. We might be over-compensating a correction in one weight dimension while under-compensating in another. This is non-ideal as we might find ourselves in an oscillating (unable to centre onto a better maxima in cost(weights) space) state or in a slow moving (traveling too slow to get to a better maxima) state. It is of course possible to have a per-weight learning rate, but it's yet another hyperparameter to introduce - not ideal. So we standardize images before inputting them in CNN.

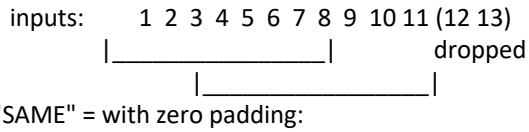
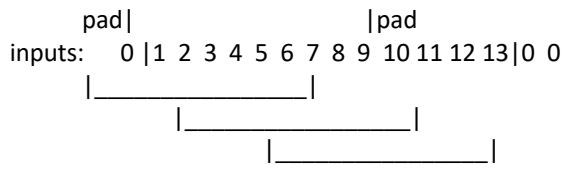
Q: I believe that one of the reasons the CNN performs better than the traditional classifiers is that it accounts for translational invariance. Are there any non-deep-learning classifiers which account for this?

> **A:** First of all, there are other characteristics of CNN that helps its performance, e.g. weight sharing and transfer learning. Translational invariance is unique to deep learning architecture. Math here: https://arxiv.org/pdf/1512.06293.pdf.
>
> Translational invariance in CNNs has pros and cons – it works best if the positional shifts are irrelevant to the prediction, e.g. in the MNIST data. It's not so great if the location of features matters to the prediction.
>
> Pretty good explanation of everything above here: https://medium.com/@divsoni2012/translation-invariance-in-convolutional-neural-networks-61d9b6fa03df.

Q: The notebook says 'In practice cross-validation experiments are quite expensive on CNNs, and instead you may rely on one of the common CNN architectures available in Keras and TensorFlow'. What does this mean?

> **A:** It means CNNs are difficult to build. In deep learning you would normally tempt to avoid k-fold cross-validation because of the computational cost associated with training k different models. Instead, you just use a train/test/validation set split. In Keras you can use validation_split (% to hold out) or validation_data (tuple of (X, y)).

Q: What is the difference between 'SAME' and 'VALID' padding in tf.nn.max_pool of TensorFlow?

> **A:** 'SAME' pads the input such that the output has the same length as the original input. 'VALID' option may discard the border elements of input. See the documentation for formula: https://www.tensorflow.org/api_docs/python/tf/nn/convolution.
>
> "VALID" = without padding:
>
> 
>
>
> "SAME" = with zero padding:
>
> 
>

Source of this ascii art: https://stackoverflow.com/questions/37674306/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-t

Q: Loss on validation set increases by a lot while accuracy remains more or less the same. Why is that?

A: There can be many reasons, off the top of my head.

First is imbalance. The accuracy is going to be how many correct observations out of all observations, vs. the loss function can be more complex and can be more/less robust to class imbalances. It will also depend on how well the classifier is doing for each class. It might be good to look at a confusion matrix to see where the model may be failing.

Second is the threshold. Accuracy can remain flat while the loss gets worse as long as the scores don't cross the threshold where the predicted class changes. Some images with borderline predictions get predicted better and so their output class changes (eg a cat image whose prediction was 0.4 becomes 0.6). This is a common case where loss decreases while accuracy increases.

Third is over-fitting: high validation accuracy + high val loss score vs high training accuracy + low loss score suggest that the model may be over-fitting on the training data.

Q: Why does 1 vs 1 need different number of iterations than 1 vs all?

A: Not sure exactly what your results were, but 1 vs rest approach takes one class as positive and rest all as negative and trains the classifier. So for the data having n-classes it trains n classifiers. One vs One considers each binary pair of classes and trains classifier on subset of data containing those classes. So it trains total n*(n-1)/2 classes.

Q: Using ".predict" threw an error, so I used predict_classes, but it says it's deprecated. What are other options?

A: model.predict_classes is deprecated and removed after 2021. model.predict(test_images) will give you an array of probability scores from the 10 classes, so it's throwing an error because you're expecting the class itself. You can convert this y using class_names[np.argmax(predictions)].

Q: I got an error "depthwise max pooling is currently only implemented for cpu devices" - why?

A: Looked this one up, and it's a known issue: https://github.com/tensorflow/tensorflow/issues/636.

As far as the placer is concerned MaxPool is defined to run on GPU, so it will be placed on a GPU device if available (using hard or soft placement). However, for certain input values, it will raise an error at runtime if the op tries to execute on GPU.

TL;DR: For now it's necessary to annotate any depthwise max-pooling ops with a with tf.device("/cpu:0"): block.