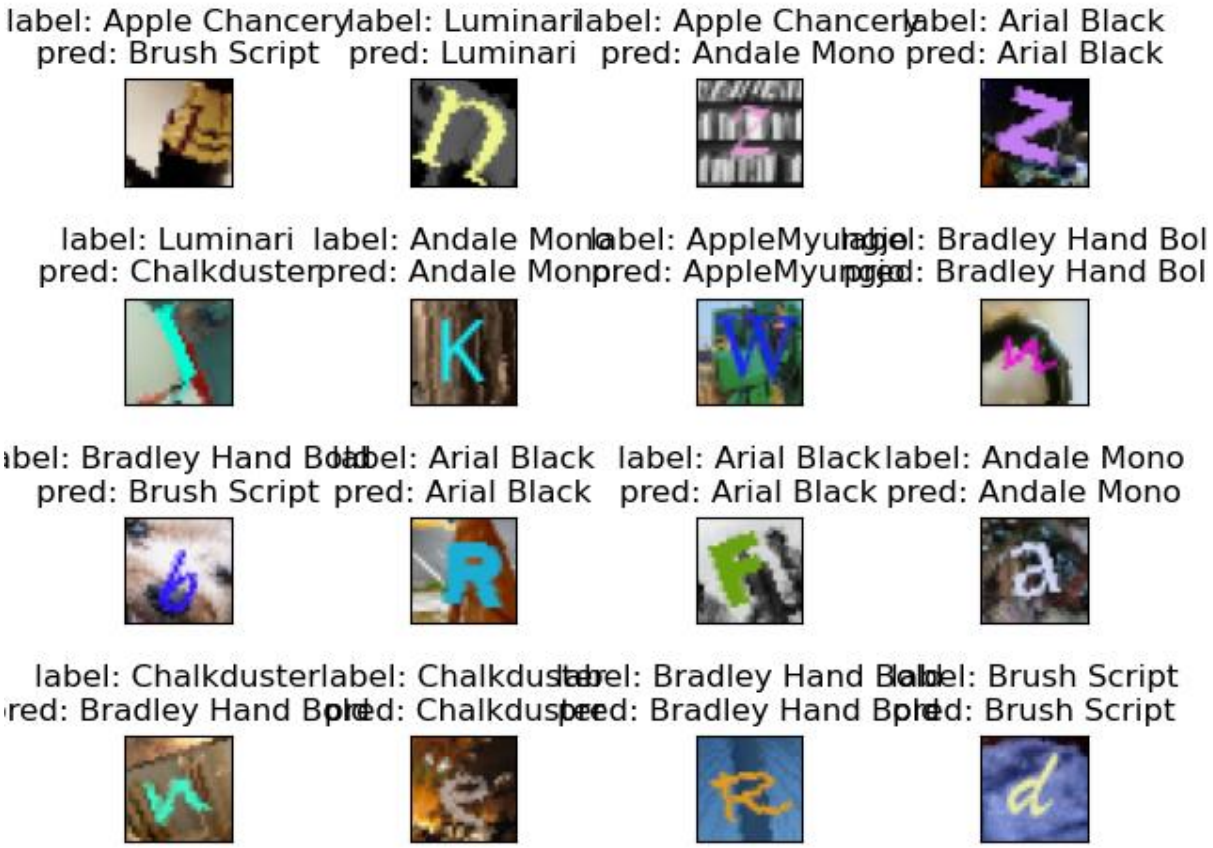
COSC 420 Assignment 1



A convolutional neural network was used for all tasks. This architecture was chosen to retain the relationships between neighbouring pixels in forming a letter, rather than simply flattening the input into 1D for a multi-layer perceptron.

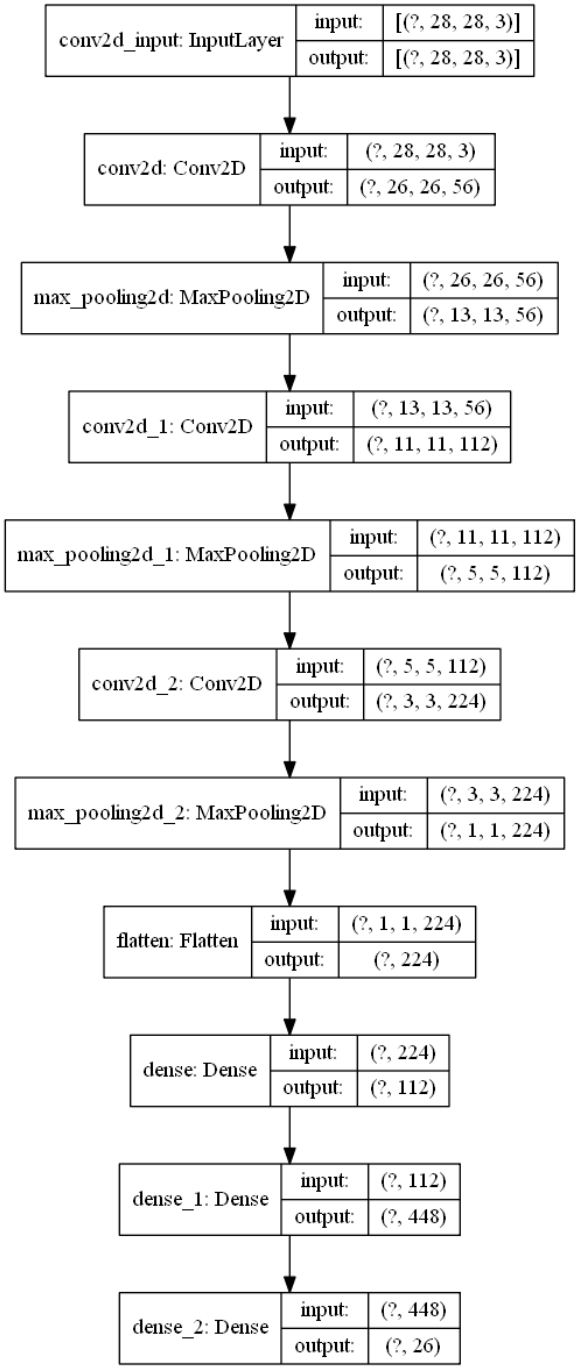
Other neural networks were not explored due to lack of computational power at hand, like stoichastic networks (due to the long settling time required) or the depth of multi-layer-perceptrons required to approach convolutional neural networks in accuracy for multi-dimensional input.

The number of epochs was kept small (10 <= n <= 20) in order to keep training times manageable on the available hardware and test networks with many different strategies and parameters.

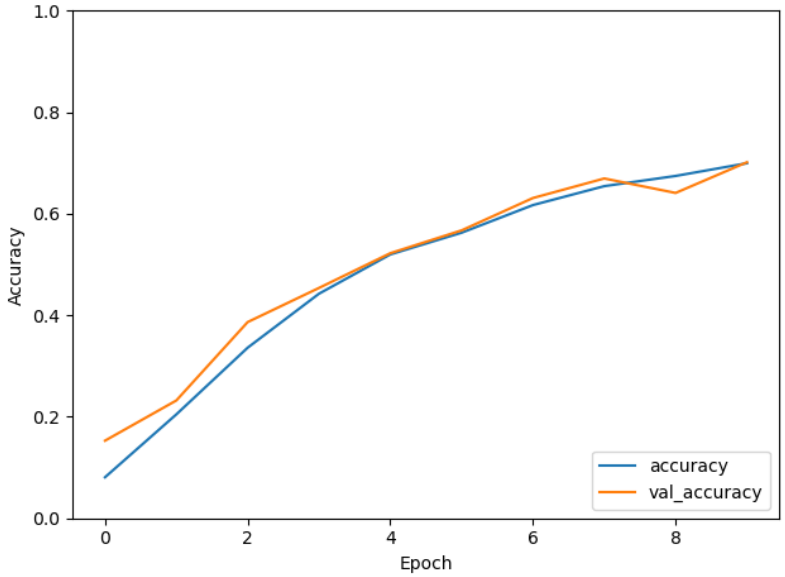
The loss function was kept as Sparse Categorical Cross-Entropy in all trials. Looking at implementation and uses of other loss functions such as Mean Squared Error (relevant to continuous numerical predictions), Binary Cross-Entropy (used for binary classification) and standard Categorical Cross-Entropy (requiring additional one-hot encoding of output labels, and with minimal improvements over the former), the initial loss function was most suited to the context.[1]

[1] https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718

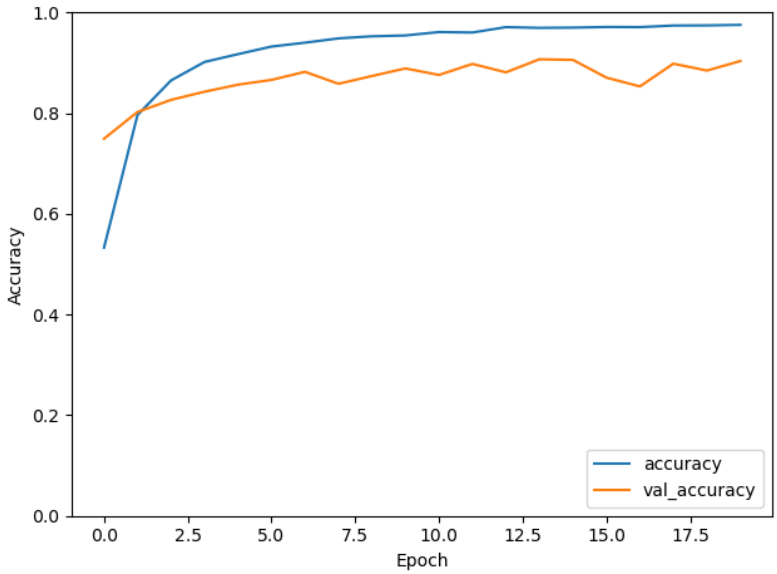
Simple classification, clean data



A basic CNN[2] on the clean data over 10 epochs gave 0.75 training accuracy and 0.70 testing accuracy.

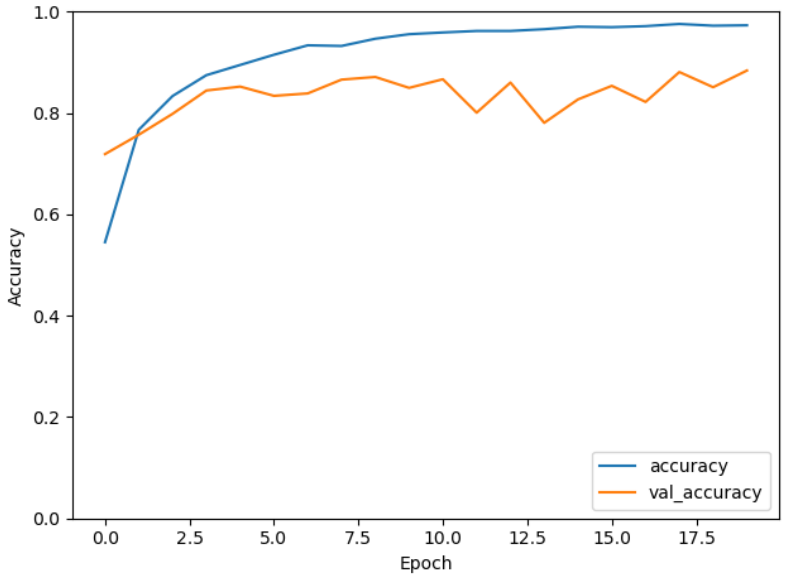


Upping the epochs to 20 with batch normalization to improve training time, the training accuracy was 0.99 and the test accuracy 0.90.

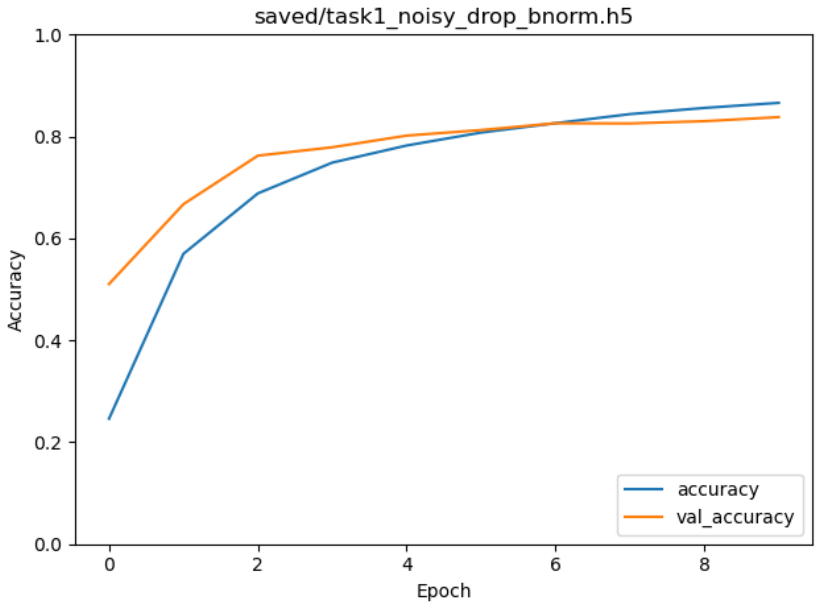


Using weight decay actually increased the degree of overfitting, possibly due to the graph not being deep enough for large weights indicative of overfitting to be present.

[2] Code taken from example3.py, in Lecture 3



Simple classification, noisy data



Running the CNN with dropout and batch normalization on the noisy data for 10 epochs, there was a training accuracy of 0.93 and test accuracy of 0.84.

Looking at the data, the letters were rotated up to 45 degrees from the norm, fully within the frame but not always centered. Thus, data augmentation was tested with the following techniques[3]

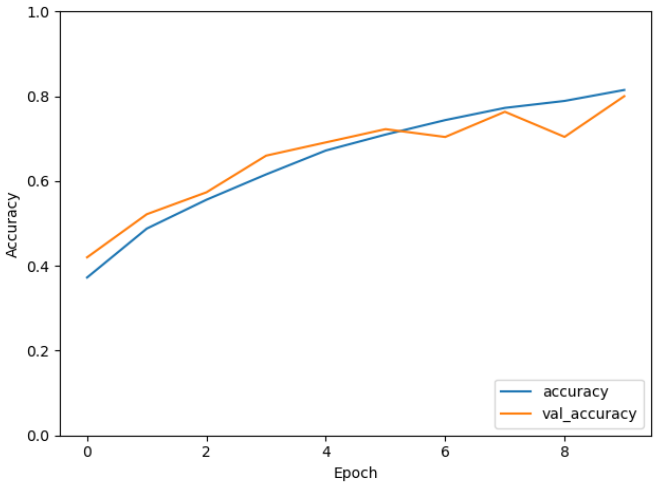
- Image shifts: Yes, as letters should be recognisable anywhere in the frame (1/10 of frame)
- Flips: No, as letters are not normally handwritten backwards
- Rotations: Yes, as handwritten letters can be tilted (limited to 45 degrees to preserve M-W distinction)
- Brightness: Yes, as the ZCA transformation emphasizes letter features to the network
- Zoom: No, as I found it unclear how to preserve the letter wholly in the frame when enlarging or retain minimum line size to be distinguishable when shrinking

```
task1 (1)
600/600 [==============================] - 115s 192ms/step - loss: 0.8176 - accuracy: 0.7463 - val_loss: 0.5910 - val_accuracy: 0.8158
Saved model to saved/task1_noisy_aug_drop_bnorm.h5...
2020-04-14 12:56:14.545839: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 141120000 exceeds 10% of system memory.
Train accuracy (tf): 0.83
Test accuracy  (tf): 0.82
```

Unfortunately the data augmentation ran extremely slowly at 20 epochs with the above error, and gave a reduced accuracy. This prevented further testing of the implementation and accuracy gains of data augmentation.

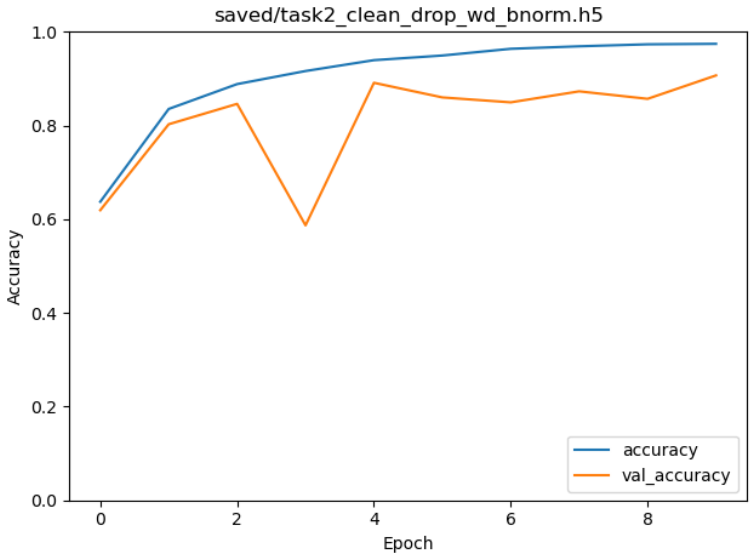
Fine-grained classification, clean data

The initial CNN with no options implemented gave a training accuracy of 0.85 and test accuracy of 0.80 on 10 epochs. Interestingly, this is significantly more accurate than the simple classification 10-epoch result, despite being a “harder problem” – perhaps a reflection of the 1/8 probability of a random font guess being correct as opposed to 1/26 letters.



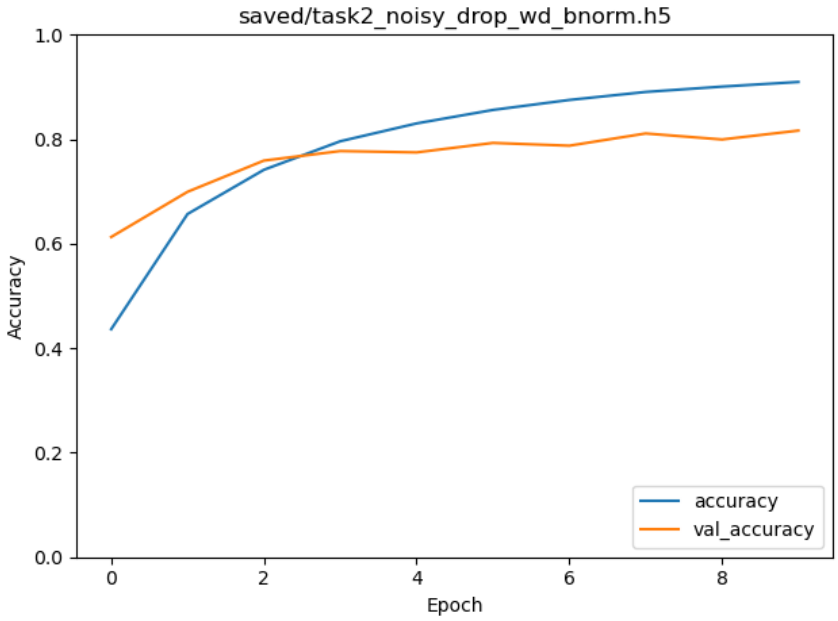
[3] https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Adding dropout, weight decay and batch normalization to address the overfitting gave training accuracy of 0.97 and testing accuracy of 0.91.

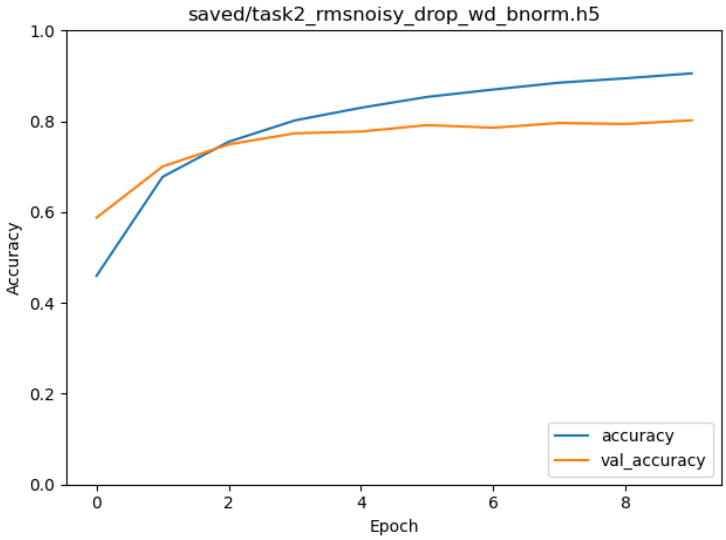


Fine-grained classification, noisy data

Running the above network on the noisy data gave a training accuracy of 0.95 and a testing accuracy of 0.87.



Experimenting with the optimizer in this task did not increase the accuracy, with the closest being a training accuracy of 0.93 and testing accuracy of 0.80 when switching from Adam to RMSProp.



Conclusion

The most accurate neural networks on the 4 tasks attempted were:

Simple clean: task1_bnorm
Simple noisy: task1_noisy_drop_bnorm
Fine-grained clean: task2_clean_drop_wd_bnorm
Fine-grained noisy: task2_noisy_drop_wd_bnorm

Given more resources and better circumstances, I would have like to set up TensorFlow operations in parallel to address the performance issues on my personal computer. I would aim to increase epochs and layers with more computing power, and to implement the extra examples from data augmentation fully.