Image Classification Comparison with a Deep, Convolutional, Siamese Neural Network

Patrick Knott

Sunday 27th October 2019

Table of Contents

List of Figures

Executive Summary

This project implemented a Deep, Convolutional, Siamese Neural Network classifier to distinguish between images of the same class and images of different classes. It did not attempt to identify particular classes, only whether two images were from the same or different classes. The resulting model was tested with the image classes used to train it, as well as with non-trained image classes. In the optimum model, the non-trained image pairs were correctly identified as being of the same or different classes with an accuracy of 82%. There were models with higher accuracy on the image classes used for training, but this came at the expense of accuracy for the non-trained image classes.

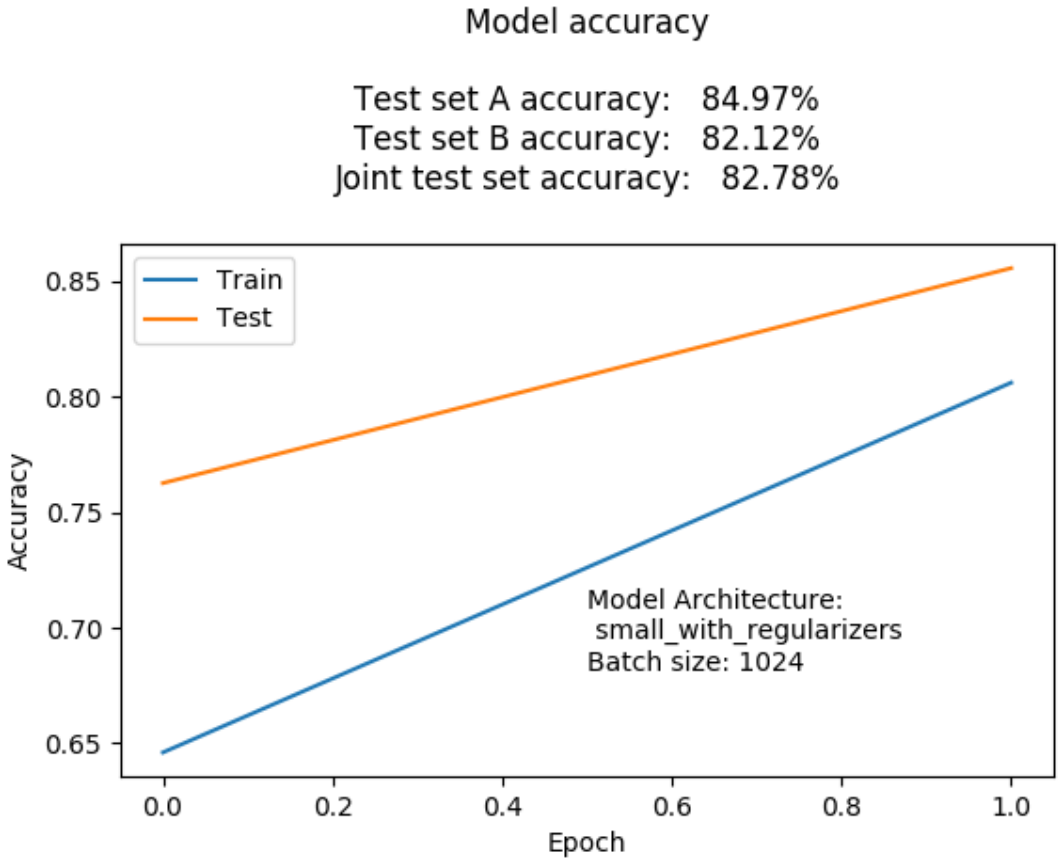


Figure 1. Optimum model, small architecture, regularized kernel, 1024 batch size, 2 epochs. Test set B contains images from the non-trained classes.

Statement of completion

I worked on this project alone. However, two functions (*euclidean_distance()*, *eucl_dist_output_shape()*) were copied and one function (*contrastive_loss()*) was adapted from the Keras documentation website.

Introduction

Image classification, verification and recognition algorithms have found much success in recent years. However, they tend to suffer when the point-of-view of the image shifts. Some applications of these programs have inherently inconsistent 'poses' for their image target, such as wildlife monitoring. This project sought to investigate the ability of Siamese convolutional networks to overcome this problem. This involves embedding images into a vector space, with images from an equivalence class near one another but distant from the other equivalence classes.

Approach

A model framework was built in the Python language using the Keras and TensorFlow packages. Then, different combinations of hyper-parameters were implemented. The images are from the Modified National Institute of Standards and Technology Fashion dataset. It contains seven-thousand greyscale, twenty-eight by twenty-eight-pixel images, evenly distributed between ten classes.

Six of the classes (image set A) were used to train, validate and test the model. 83988 positive and negative pairs were created (6*2*(7000-1)). Twenty percent (16797) were set aside to be used only for testing the completed model. Of the other 67191 pairs, 60000 were used to build the model. The remaining 7191 were used to validate the models at the end of each epoch of training.

Four of the classes (image set B) were completely set aside during the model building. They were used only to evaluate the completed model. After generating the positive and negative pairs of images, there were 55992 pairs in this set (4*2*(7000 -1)).

The completed models were also evaluated by the twenty percent of image set A which was set aside, as well as the combination of these pairs with those of image set B (labelled Joint test set on plots).

The tested hyper-parameters consisted of the size of the batch of examples used in each step of model training, whether the model used regularization of the kernels, and the number of epochs. There were also two sets of model sizes used. Both consisted of the same structure, but with values for 'X' of 16 and 32:

1. input vector (784, 1)
2. convolutional layer, (3,3) kernel, ReLU activation, X filters
3. max pooling, (2,2)
4. convolutional layer, (3,3) kernel, ReLU activation, 2*X filters
5. max pooling (2,2)
6. dense layer, some with kernel regularizer, ReLU activation, 4*X nodes
7. 25% dropout
8. dense layer, no regularizer, softmax activation, 2*X nodes
9. Euclidean distance metric between the two outputs
10. contrastive loss cost function

The code contains detail on the functions and variables implemented.

Code Verification

As a part of the implementation, verification was performed on the correctness of the contrastive loss function, the class labels of the original data, the partition into the two class set subsets, and the pair generation.

To test the contrastive loss, function *test_contrastive_loss()* was built. It replicates the contrastive loss formula using NumPy. Then, it creates vectors (for NumPy) and tensors (for Keras) with the same values. These are passed to the two contrastive loss functions. It then prints out the calculated cost for each and the difference between them.

```
In[35]: x = [0.01,  0.05, -0.07, -0.023, 2.6,  0.012, 0.548,  0.79, -.2]
   ...: y = [0.01, -0.05, -0.047, 0.023, 2.4, -0.02,  0.0548, 0.7,  -.2]
   ...: test_contrastive_loss(1, x, y) # positive pair
   ...:
This tests a positive pairing:
The contrastive loss cost calculated by the euclidean_distance and contrastive_loss functions is:  0.15251
The contrastive loss cost calculated with numpy is:  0.15251
The difference between the outputs is:  0.0
In[36]: test_contrastive_loss(0, x, y) # negative pair
This tests a negative pairing:
The contrastive loss cost calculated by the euclidean_distance and contrastive_loss functions is:  0.10023
The contrastive loss cost calculated with numpy is:  0.10023
The difference between the outputs is:  0.0
```

Figure 2, console output from *test_contrastive_loss()*

Function *confirm_data_partition()* is used to check that image set A and B contain only the six and four classes as required by the instructions.

Another function, *visual_test_random_pairs()* is used to verify both the original class labels of the images, and the accuracy of the pair generation. During the creation of positive and negative pairs, a second lookup table of class labels is generated. During verification, a random pair of images are selected and displayed side by side. Also displayed is the class labels from the second lookup table, as well as the pairs label of 'same' or 'different' from

the pairs generation function. This assesses both the original class labels (from the second lookup table) and the pairs generation (from the 'same' or 'different' label).

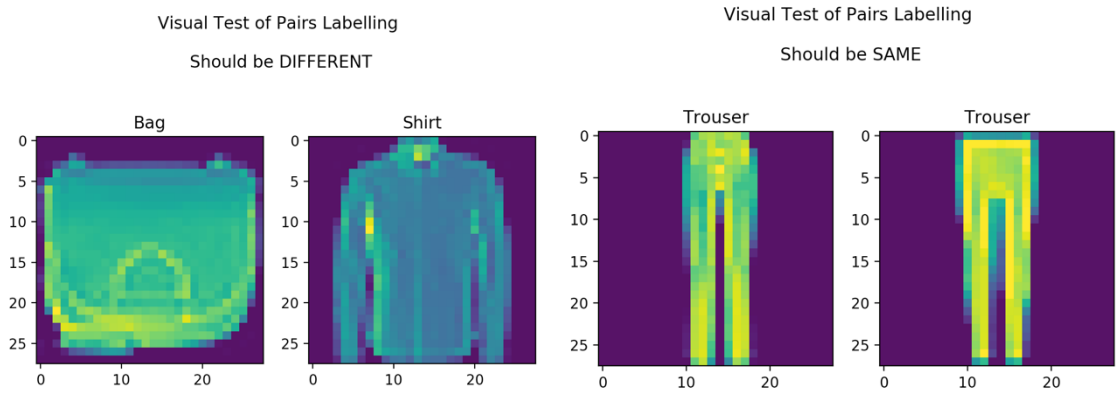


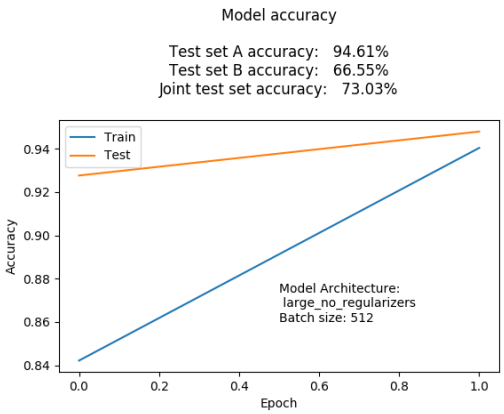
Figure 3 and 4. Output from *visual_test_random_pairs()*

Finally, *numeric_test_random_pairs()* was created to confirm that the pairs of images are from the same or different classes. A pair is randomly selected. Then, the 15th row of each image is extracted. This row was chosen as it is near the middle of the picture and therefore certain to contain non-zero elements. Then, the original, full dataset is flattened into a single vector. The small vectors from the pair of images are then searched for in the original, flattened data set. When found, their exact position in the flattened large vector is used to calculate their index from the original dataset. This index is then used to find the image class from the original label vector. Now, the classes of the image pair are compared to each other as well as to the pair's labelling as being the same class or different. This function may be more understandable to read from the code (line 227) than from this explanation.

Key Findings

There is a major vulnerability with overtraining the model not only to the training examples images but also to the six training classes as a whole. An ability to classify well with non-trained classes required a substantial sacrifice in the accuracy of the six classes used in training. The optimal model ended up with very similar accuracies for both sets of images, suggesting a near to ideal trade-off between over-fitting and under-fitting.

Generally, the middle-range batch sizes (1024, 2048) worked better with the non-trained classes. This makes intuitive sense as it would prevent the model from calibrating too sharply or too broadly, leaving it more applicable to non-trained classes. Preventing the model from becoming too specific corresponds with the findings that regularizers were generally helpful, as were fewer epochs and the smaller model architecture size.



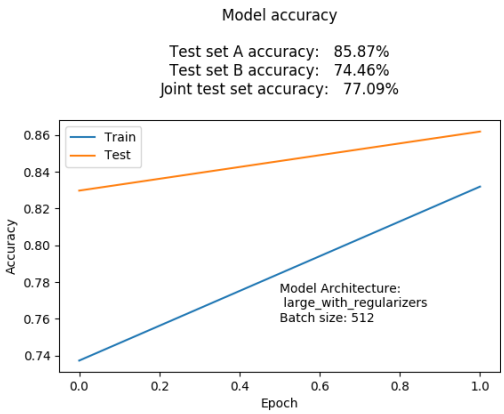
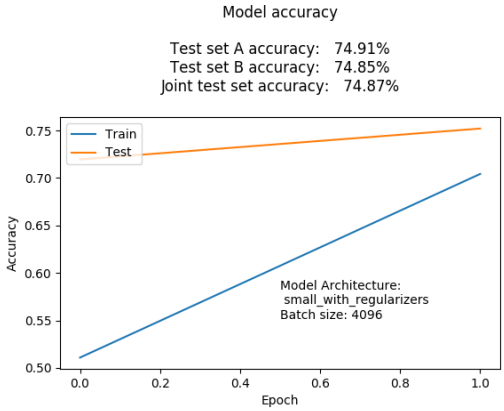


Figure 5 and 6, same model hyper-parameters except that only the model on the right used kernel regularization, leading to an 8% improvement in the non-trained classes accuracy at the cost of a 9% decrease in the trained classes accuracy.



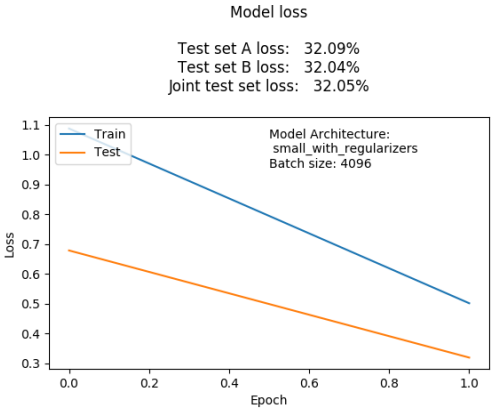
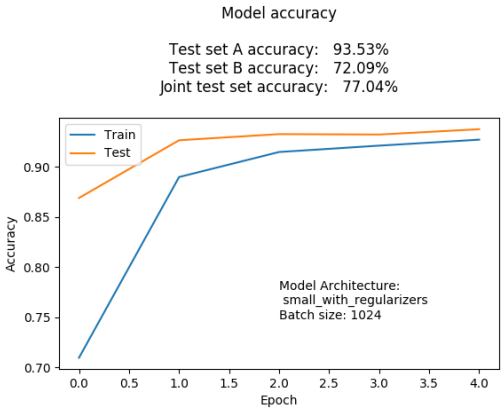


Figure 7 and 8, this model (both plots are the same model) has the same hyper-parameters as the optimum model, except that in this one the batch size is four-times larger. The accuracy on the trained and non-trained classes have converged to a similar value, but that value is about 8% lower in this model than the optimum model. The loss values for both sets of classes have converged too.



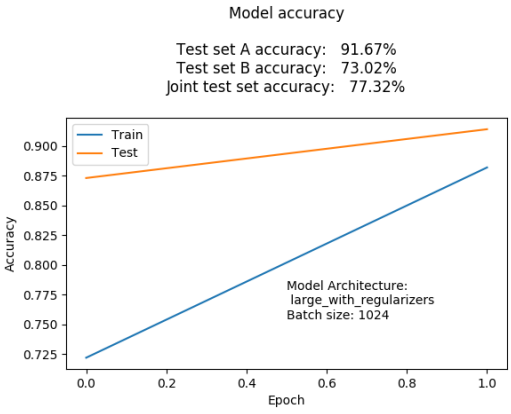


Figure 9 and 10. The left model has the same hyper-parameters as the optimum except that it was trained for five epochs instead of two. The result being that, relative to the optimum

model, the trained classes have an accuracy that is almost 10% higher, but the non-trained classes have an accuracy that is 10% lower. The right model has the same hyper-parameters as the optimum except that it uses the larger architecture (number of kernels and numbers of nodes). This also resulted in overfitting of the training classes.

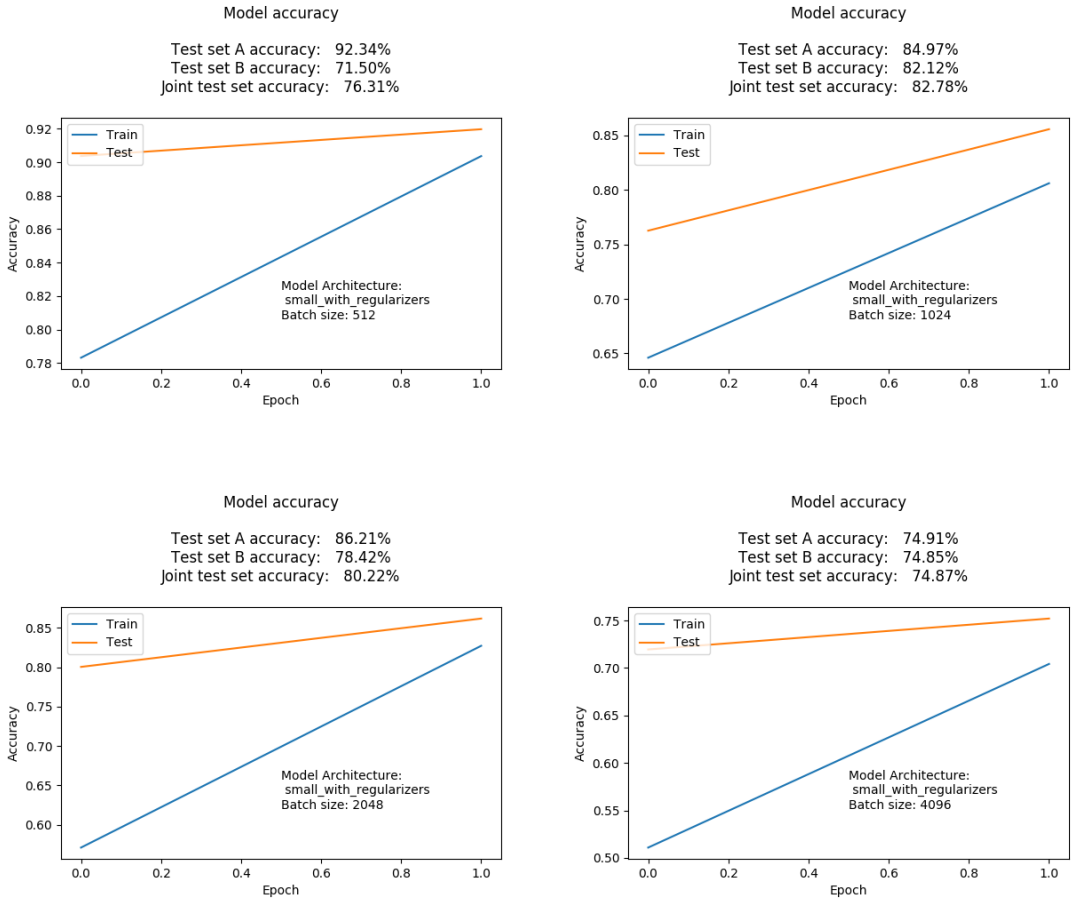


Figure 11, 12, 13 and 14. Same hyper-parameters except for the batch size. The best non-trained classes accuracy is in the middle two batch sizes, the worst were the smallest and largest batch sizes. 512 with 72%, 1024 with 82%, 2048 with 78% and 4096 with 75% (left-to-right, top-to-bottom).

Conclusion

Using a Deep, Convolutional, Siamese Neural Net classifier to distinguish between images of equivalence classes works. Hyper-parameter turning was very impactful. There is a fine line between overfitting to the training classes and underfitting to the non-trained classes, which would need to be thoroughly investigated for any application. This investigation strongly suggests that this kind of algorithm could be applied to overcome the weakness to geometric transformations for object classification, verification and recognition tasks.