Image Classification Project Report

Image classification is a problem which has plagued the machine learning community for years. Before the advent of tools such as Keras and Tensorflow and techniques such as deep neural networks and convolutional neural networks, classification took teams large amounts of time to get to a level where they would be more hit than miss. For the final project of COMP309, Keras has been used as a major tool alongside convolutional neural networks to classify images of strawberries, tomatoes and cherries with a high accuracy. This report will detail how CNNs were applied to this program to get a test accuracy of above 80%, and how adjusting the network using strategies such as optimisation and regularisation improved accuracy on the given data.

Exploratory Data Analysis and Pre-Processing of Data

The first steps undertaken for this project involved pre-processing of the data and exploratory data analysis (EDA). EDA was the first technique applied to the data, getting an understanding of the project from that perspective. This showed that the data was balanced evenly across the three classes, with 1500 provided images of each of the three classes. The data itself had already been cleaned and tidied into images of 300x300 size, and had been set up in directories to be read in with the classes automatically applied. There were some noisy images in the dataset that were found by looking through the images, such as images where the target fruit was a small image surrounded by other details, or where a different fruit had been incorrectly classified as one of the target three. For the point of this project, these noisy images were not removed. This was in order to keep the balance between the three classes, to ensure that the model would be more general than had these images been removed, and also to ensure that if the model was applied to unseen data which itself had not been pre-processed, then these images would be able to be classified without an over fitted model throwing up incorrect classes in as many cases as possible.

Feature selection / design was not applied extensively to this data set. The feature number was the resolution of the image which, due to technical limitations, was limited to a 100x100 size. As the original pictures were given in a 300x300 pixel size shrinking the images to this smaller size did not result in a major loss in data, so 100x100 was deemed to be an appropriate image size. The features were not processed beyond this, a technique that could have applied would be finding the edges of the target fruit, due to the overall quality of the dataset and the accuracy which was applied from later was high enough that pre-processing could have clarified the images more but it did not have a major impact on expectations of accuracy. Had the accuracy been lower than anticipated, these techniques would have been applied.

One way in which the data was pre-processed was through the use of random transformations to the data. Using the openCV image editing techniques images were enhanced in as many as three different ways, two types of rotation and one which blurred the images to decrease the quality. This had the benefit of artificially enhancing the size of the data set as each image could spawn up to three similar but not identical images, and also added different perspectives which the classifier would need to understand. The images were changed randomly through the use of numpy random number generators, which added unpredictability to the model and could have caused issues as the model was changed and retrained. In order to avoid this uncertainty in results, the images were generated at random for the training set once, then saved using openCV image write so they could be stored in with the training set and loaded in the model to ensure reproducibility in the results.

This did cause a slight decrease in the training accuracy but the overall result on the test set was improved by 1%-2% for close to 150 images. The balance was kept in the dataset by ensuring that there were the same number of each class randomly generated.

Another way in which data could have been enhanced was through the use of Keras and the image data generator and flow which applied changes to the stored dataset. Those techniques were not used in this project due to two reasons. The first was that using the image data generator caused an extreme slowdown in the running of code, as it would default to the CPU instead of the GPU and cause the time taken for a single epoch to be approximately 30 times that of the GPU, and the second was that the accuracy was not improved to a measure where the generator seemed to make an extreme change to the model. The techniques applied by openCV were more restricted and fewer in number, but provided enough of a boost that they helped with the accuracy more than using the baseline supplied dataset.

The data was supplied in single folders which stated the class of the images inside. In order to apply a training and test dataset that would ensure reproducible and comparable results, code was used which would randomly split the data into 80% training and 20% test set. Once the data had been split once it was saved with this division, ensuring that all models would run on the same set as test and training. When data was generated by openCV there was a boolean passed which dictated whether the image belonged to a test or training dataset, and the images were stored in the according folders. During early rounds of training the model there was also a defined validation set, but this was changed later on as is explained later in this report.

Methodology

For the initial building of the CNN, the philosophy of add layers until there is a decrease in accuracy was applied. In the earliest stages CNNs online were looked at to see the common structure of their layers, as was the CNN which was created during the tutorial in week 10. These followed a similar structure that seemed to be applicable towards many smaller or more trivial problems, while datasets which had millions of images with high definition tended to be far more complex and complicated than was necessary for this problem. The model settled on had three convolution networks with different numbers of filters (32, 64, 64) which provided the best performance on the data set. Each convolution layer was followed by a max pooling layer in order to decrease the number of neurons in the network, then finally there was one fully connected dense layer and the final output dense layer which had three neurons, one for each class probability. This provided a CNN which achieved an above 80% accuracy on many of the aspects which are listed below, while not overcomplicating the model to the point that it was too large or unnecessarily complex for the task it was applied to.

Default / Base settings

For all the different methods applied below, there were base settings used in the model in order to ensure that results could be compared accurately. The base settings of the CNN are as follows, with each being swapped out when it was the aspect being tested:

Optimisation: Adam, validation: none, regularisation: dropout layers as detailed in regularisation description, loss function: categorical crossentropy, activation: relu, ensemble model: no, extra images: no, 100 epochs, batch size of 32

Validation

For validation use in the CNN, three options were considered. The first was to use no validation in the training of the model, instead allowing the model to run without considering any data other than the training set. As would be expected this meant that there was a high accuracy on the training set, approximately 98%, with a very low in comparison test set accuracy of 64%, showing the bias and overfitting which occurred with this model. Keeping a validation set in mind the second choice which the model was run on was by using the test set as a validation set. This is not a preferred option as a model will give a better idea of the accuracy when the test set is completely unseen, but it did give a strong accuracy of 80.33 on the test set and 84.83 on the train.

K-fold cross validation was considered, but instead the choice was made to run the model with a validation split of 0.1 and 0.2. This meant that each epoch would have a random 10% of training data used as the validation data. It caused fluctuation of the validation accuracy in each epoch, but also ensured that the model would be more generalised than if a dedicated validation set was chosen. This did bring up the risk of a lower validation accuracy with wider range, as well as some images not making it into the validation set at all, but the benefit of the randomness was that it kept the data from overfitting. This is reflected in the accuracies, with training accuracy of 10% validation at 85.28 and test accuracy at 80.49, while the accuracies on 20% validation were 75.86 and 72.73 respectively. Due to this, the choice for validation for the model was 10% random validation split, due to the strong performance and benefit in terms of reducing overfitting.

Loss Function

For the loss function, two measures performed better than the rest. Mean absolute error (MAE) performed consistently bad, giving the accuracy of 33.33% which is akin to randomly guessing the classes. Because of this, despite a small amount of time spent in adjustment, MAE was discounted. MSE had a strong performance, with 80.22 on the test set and 96.17 on the training set, but using categorical cross entropy had the highest accuracy with 82.11 on the test set and 97.31 on the training set. This is most likely due to the categorical nature of the data being optimised with the loss function set to categorical cross entropy, and MSE having a stronger performance when it came to regression based data. Due to these outcomes of accuracy (and also a lower loss which was not recorded during the time of the experimentation), categorical cross entropy was chosen as the strongest loss function for this model based on the default settings chosen for the model as well.

Optimisation method

For this model, three optimisation methods were chosen. The first, Adadelta, was picked as it was a notable choice during online research for building the CNN. The accuracy given when using Adadelta with the default settings was 78.56 on the test set and 94.94 on the training set, giving it a slight bias towards the training data and causing it to be dropped due to concerns that it pushed the model towards overfitting on training data. The second option, Adam, performed very well with the learning rate set to 0.001. This stopped the problem noted at learning rate 0.01, where accuracies were set around 33.33. When the learning rate was reduced, Adam performed well on the dataset with fewer epochs than the other optimisers, most likely due to how it works as a combination of AdaGrad and RMSProp, training the learning rate for each network weight and adapting each separately during the course of learning. Adam trained the fastest, getting accuracies of 82.00 on the test set and 98.19 on the training, a closer comparison than Adadelta.

The third optimisation method chosen was SGD, Stochastic Gradient Descent. At first the learning rate was set to 0.01, and the momentum varied between 0.2, 0.5 and 0.9. The best setting for this learning rate was a momentum of 0.5, which gave training data accuracy of 94.38 and test of 80.11. When the learning rate was lowered to 0.001 and the epochs increased beyond 100 to compensate,

this gave a test accuracy of 80.22 and a training accuracy of 87.86 on the test set. Due to the similar test accuracies of the two learning rates, it can be seen that SGD performs better on learning rate of 0.01 and momentum of 0.5 due to the higher test performance. However, both were lower than the results given by Adam, so while this was considered as a possibility for making an ensemble model the chosen optimisation for a single model was Adam. This is in line with a number of papers found online, which believe that Adam is a strong contender in optimisation methods, due to finding the convergence of the optimisation function and the quicker results than can be found in methods such as SGD.

Regularisation Strategy

For regularisation of the CNN model, the main two options were considered. These are dropout layers and the L1/L2 regularisation. In the most naïve run of the model, without any kind of regularisation, the train accuracy reached 99.28%, while the test accuracy was merely 68.22%. This shows that there was overfitting involved with the model, as would be expected when no validation was applied, and that some form of regularisation would be needed to increase test accuracy. The first method chosen was dropout layers, something that is commonly used due to the easy implementation and strong performance across most models. In this case, using two dropout layers proved to be the optimal number, losing at first a 50% dropout after the first two convolution and pooling layers, then 20% of the nodes after the dense layer and before the classification layer. The performance on test and training set was 82.00 and 97.61, giving a strong training performance but also ensuring that the model did not over fit to the detriment of new data and giving one of the highest test accuracies in the models.

L1, L2 and L1/L2 regularisation methods were all applied to the model to see what their performance would be. L1 performed the best, with test and training at 72.56 and 99.44 % respectively, while L2 had the worst performance with a lower accuracy than using no regularisation at all, test accuracy at 65.56% and training at 87.83%. The mixture of L1/L2 had a better performance, with test 71.22 and training 98.36%. This may be due to the implementation of the regularisation strategies, as replacing them where the dropout layers were may have affected their influence on the model. The dropout layer performance was strong enough that these were not looked into more than a basic discovery of their accuracy, and dropout was chosen due to the strong performance and also popularity within CNNs at the moment.

Activation function

For the baseline idea of a activation function, linear regularisation was chosen at first. By picking this more basic, less sophisticated function the accuracies on test and training were both lower than all other activations, as would be expected. Test accuracy sat at 67.67 while test accuracy only achieved 81.0%. Due to the high number of input features for each image, a non-linear function was then explored to see how it would work better on the model. The three explored were sigmoid, both as internal activations and the final layer, tanh and relu. Softmax was also used on the final layer, compared with the softmax output, but for the inner layers (3 conv, 1 dense) for the sake of time and sanity, the activation functions were kept the same.

Sigmoid performed the worst of the three non-linear activation functions, though this could be due to not having enough time to fully perform and get a higher accuracy. It managed only 54% test accuracy with 75% training, performing worse than the basic linear but this could be highly influenced by the smaller number of epochs of training which suited other functions and gave a base to compare between. Tanh was the next tried, as it was one of two most common functions seen in research, and it performed better than linear or sigmoid, achieving a test accuracy of 70% and a training accuracy of 96.83. The high difference between the two numbers seemed to suggest

overfitting towards the training model when this activation was used, making it a less desirable function to use within the model.

The last activation function chosen and tested was the relu function, which was seen in the tutorial example given and also in a number of online examples. As expected this performed the best out of the activation functions, achieving an accuracy of 79.22% on test and 96.5% on training data when used with the baseline model described above, though higher accuracies when it was run with the optimal settings from each of the other aspects of the CNN. This was likely due to the quicker speed with which relu activations can train, along with the capping of negative elements at 0, useful for multi-class classification such as this project. Due to these results and logic, this was the optimal choice for activation function in the method.

Extra images obtained

Extra images were obtained during the training of the CNN in order to enrich the training data and test data provided within the assignment handout. Images were downloaded with the use of the google images downloader python package, with command line arguments to search for pictures with the correct dimension and matching keywords, as seen below:

googleimagesdownload --keywords "cherries, tomatoes" --limit 150 --format "jpg" --exact_size 300,300

This allowed for the addition of three hundred training images to the set before random processing occurred, meaning that there could be a number more images than expected within the training set. Using the images without the potential for preprocessing led to an increase in accuracy on the test set with the optimal settings chosen for the CNN, with the test accuracy jumping up to 83.76% while the train accuracy remained at 98.59%. This showed that getting extra images, even as few as 100 per class, helped to enrich the training data set and give the classifier more images to train on and thus a wider ability to predict the images fed through based on the images previously classified.

Ensemble model

An ensemble model was built though did not function exactly as desired within this project. An ensemble model was created based on the two best convolution neural networks created, similar models which differed in their losses and in their optimisation functions. The two models were run and trained based on the same training data, then had predict called on the same test data. However the problem was in translating this into a comparable format with the assigned labels for the data, meaning that there was not an output of accuracy that could be achieved during the running of this program. Code for the ensemble model was left within the code for train.py, showing the attempted made. There was also an issue which appeared where the ensemble model was required to use predict instead of evaluate, meaning that the test data was not as accessible or succinct as that provided by evaluate. However, due to the results of ensemble models giving a better accuracy and smaller loss than single models, this is something that would be useful if more time and understanding was applied to the problem, such as being able to convert the predictions into the same [0,1,0] format to get the accuracy and show an overall comparison.

CNN vs MLP Results

For the MLP, a small, dense layer only version of the CNN was trained. This MLP had a hidden layer in that there was a dense layer which took as an input the size of the image and had 128 nodes, then a flatten layer to get the values into an output format before having a dense layer of 3 nodes for the output. The time taken for the MLP to run 30 epochs with a batch size of 32 was 1101.41 seconds (18.37 minutes), and the output accuracy for test data was 48.88% with train accuracy at 54.35%. In

comparison for the optimal CNN to run 30 epochs the time taken was 271 seconds (4.5 minute) and the accuracy was at 82.3 for test and 94.59 for the training data set. Both were run with only the base dataset and identical optimisers/loss functions etc.

The MLP took far, far longer than the CNN, most likely due to the use of filters and pooling layers in the CNN, along with dropout. As the CNN runs the number of nodes in the CNN becomes less and less, meaning that there are not as many gradients which need to be calculated, weights which need to be adjusted and inputs into each node. In comparison the MLP runs one fully connected dense layer, only flattening in order to get the output correct, then has another fully connected dense output layer with three nodes. This slows the progress down greatly as nothing is removed and every image is processed through the MLP with the full number of image pixels, which is 100x100. Due to the MLP not reducing data there is not the same emphasis within the MLP on reducing data to the relevant parts of each image, which leads to a great decrease in accuracy. The MLP accuracy is higher than simply guessing at the class which gives an accuracy of only 33.33, but it is almost half as accurate on test data as the CNN, while taking far longer.

Conclusion and Future Work

Through the use of CNNs and tools such as Keras and Tensorflow it is possible to build a image classification system for the three fruits within this project which achieves an accuracy of over 80%. It is important for all aspects of the CNN to be considered, optimising parts such as loss functions and regularisation, in order to get the best performance from a CNN, and the comparison between an MLP and CNN shows how useful the latter is for image classification. Further work that could be done on this CNN would include a deeper look into regularisation techniques L1 and L2 and optimising how they are placed in the CNN, along with combining the best variety and trained CNNs into an ensemble model which would show off the benefits of the techniques involved in each individual CNN.

Acknowledgements of borrowed code:

Images:

https://github.com/hardikvasa/google-images-download

Example of how the images were downloaded from the command line:

googleimagesdownload --keywords "cherries, tomatoes" --limit 100 --format "jpg" --exact_size 300,300

Splitting the data into training and test (Also saved as split notebook)

https://stackoverflow.com/questions/46717742/split-data-directory-into-training-and-test-directory-with-sub-directory-structu

Building an ensemble model:

https://medium.com/randomai/ensemble-and-store-models-in-keras-2-x-b881a6d7693f