Project

Leon Menzies - 300543278

Introduction:

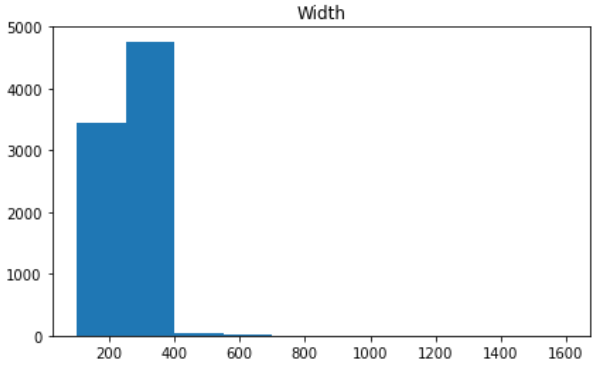
We are given 4,500 images to train a model that can make predictions on the hidden test set (1,500 images). I will build a convolutional neural network (CNN) to achieve accurate predictions. I will approach this in a few stages starting with a simple model (I chose MLP), using the knowledge gained from my first model to build a CNN followed with as much tuning necessary to achieve maximum results with a second CNN likely constructed of many more layers.

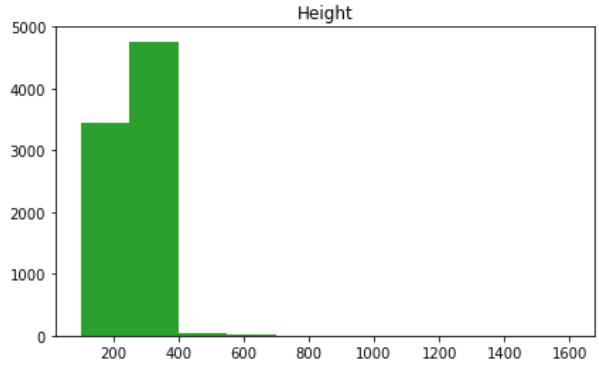
Problem Investigation:

My EDA involved a few different steps with a slightly different approach from previous assignments. Because we are now working with images I found I had to dig a lot deeper into the data to find patterns and useful information.

Data size: The training data set is of size 4500 which is made up evenly of 1500 images of each class (strawberry, cherry, and tomato) This is not a very large dataset especially for 3 classes that are quite similar. I will take this into account when trying to improve my model.

Resize: Although the images are supposed to be all sized at 300 * 300 there is always a possibility for error. Thus I chose to extract all the image sizes from the Dataloader and graph them.

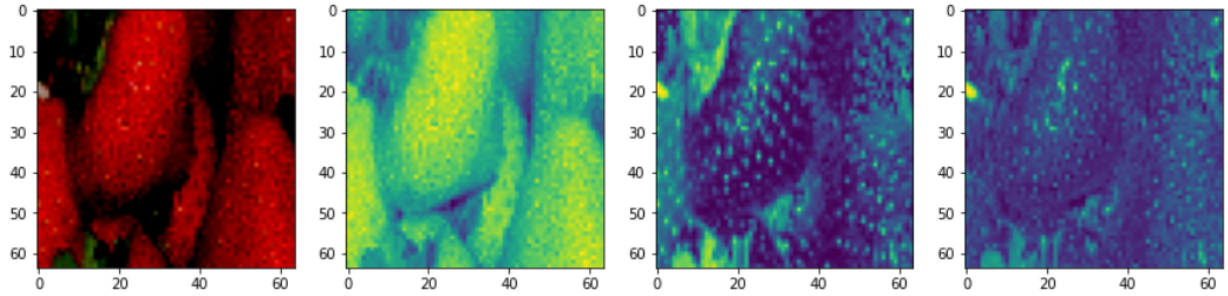




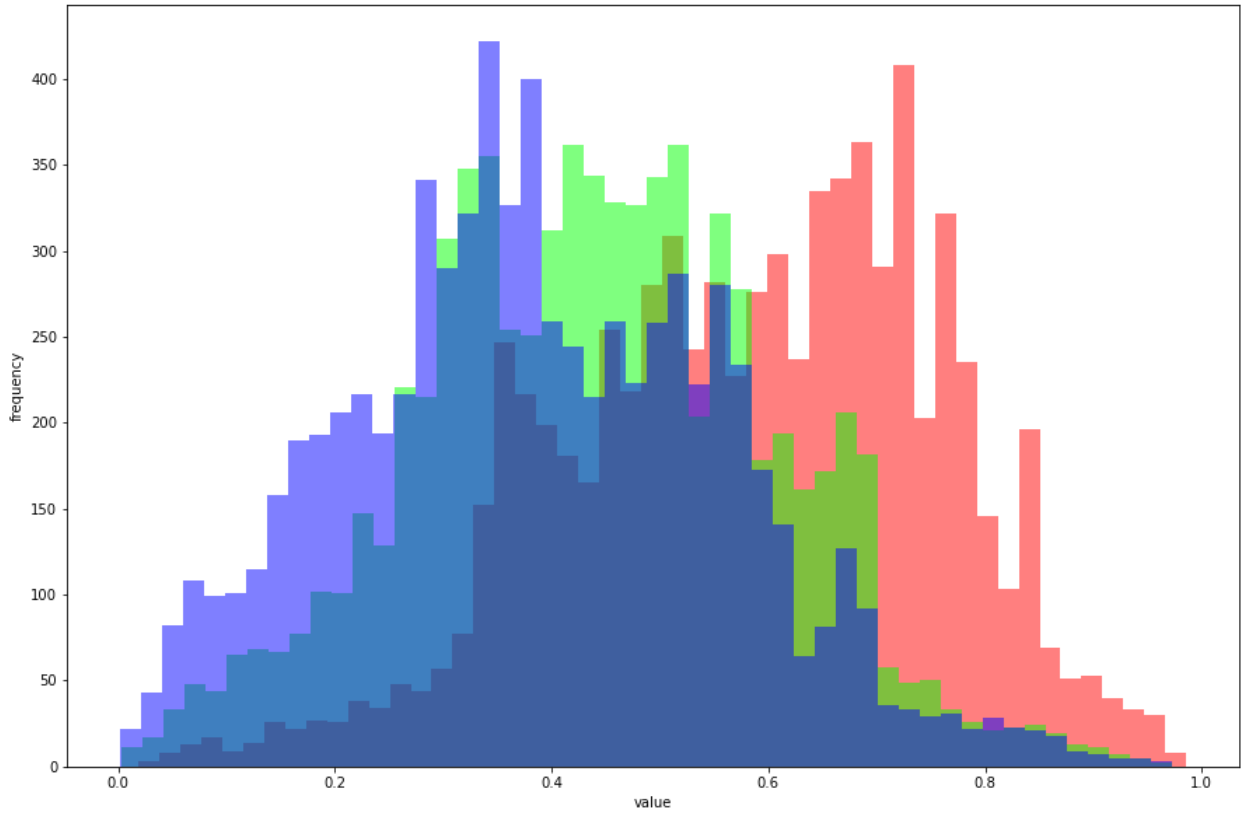
As we can see there are quite a few images with different widths and heights. Most of these images are from the new ones I added to improve my CNN model but a very small few are from the given dataset. This will be a problem when it comes to using convolution layers in our models later on so this must be resized. I started by choosing a size of 64 to increase the speed that the models can be built and trained.

Splitting the channels: I extracted the numerical values for each of the three color channels (RGB) This allowed me to draw the image showing each channel and how much it represents a single image as shown below. We can see in this image of a strawberry the red channel is the

most prominent. We can also see the green channel plays an important role in distinguishing the seeds in the strawberry which is one of the biggest factors that divides it from a tomato or cherry.



Taking all the separate channels and plotting the means on a histogram can give us a better idea of the distribution of each color value. We can see from this graph that red is more prominent as expected with three fruits that are normally red. Green is also a big factor due to most of the images containing green stems of green plants in the background.



We can also take a look at the distribution of the pixel values which clearly shows two very large peaks at -1 and 1 which is expected when it comes to colored photos.

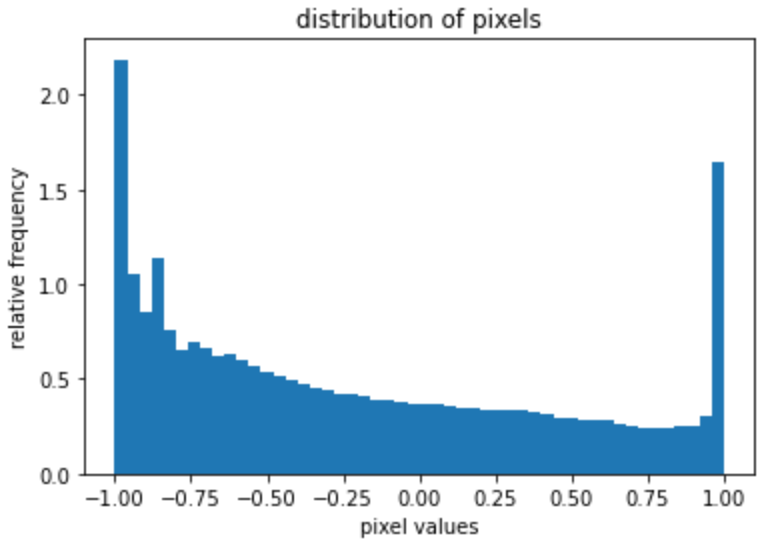


Photo outliers: I took the time to browse through each of the three classes and check for any outlying photos. As a part of the preprocessing of the data after finding such outliers, I removed photos from all three categories. Most of them either didn't represent the given class at all, for example in cherry there was a picture of a cherry tomato or a book cover with the word cherry on them. Even worse, some photos contained multiple fruits including a photo of a pile of cherries on top of some strawberries. This would negatively affect our model as it would take into account both fruits for the cherry class. The training data now had the following sizes.

Cherry: 1471
Strawberry: 1456
Tomato: 1461

Although it is not a huge difference from the original it is significant enough to justify this process as the removed images would have decreased the accuracy of the model.

Pre-Processing:
From the EDA I had identified some images were of different sizes. This would be a problem for the model so I resized all the images to 64x64. This was significantly smaller than the original 300X300 and will increase training and fitting times a lot. It is also necessary to resize all the images as I may add more later to help increase my CNN model and it will make it much easier than manually resizing each photo.

I also manually preprocessed the data by removing photos that did not represent the given class. For example the below photo on the left was in the cherry folder but also contains strawberries which would confuse the model. As for the photo on the right found in the tomato folder, it does not represent a physical tomato in any sense thus would not be good for the

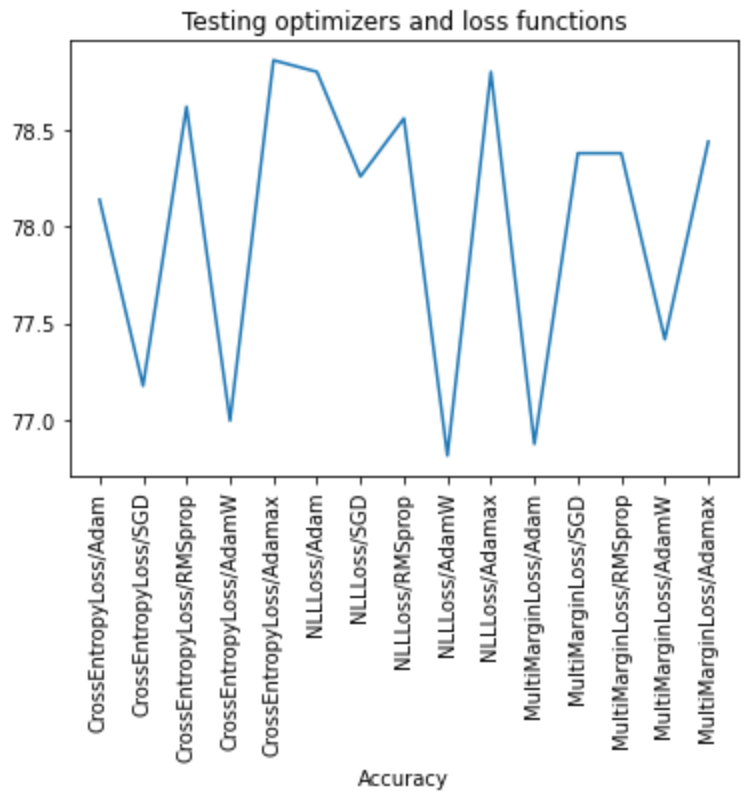
model. After removing all the bad photos I could find I noticed my accuracy slightly increased which justified this decision.

Using the torch vision transformation library I applied some extra preprocessing steps for loading in the data.
I used a random rotation of 30 degrees to randomly rotate batches of images to increase the information the training step could gain from the images. I also used a random resized crop which resizes and crops images to make them smaller or crop out certain parts of the image, A random horizontal flip that flips the images horizontally, and most importantly a normalization transformation to normalise the data and decrease any noise that could interfere with the results.

Methodology:

Cross validation:
Using cross validation to train my model was a really good way to prevent overfitting and allowed me to change the model with the confidence that I was getting an accurate representation back of how the accuracy was being affected. I ran 10 folds with 20 epochs per fold. This trained the model with every possibility and created an accurate model that did not overfit the data. Using the info I gained from the cross validation I did some hyper parameter tuning. I started by adding in another convolutional layer, this slightly increased the accuracy. Next I looked into batch normalisation and using dropout layers. After using batch normalisation after my first to max pools and a dropout layer before flattening the data for linear training I was able to bump the accuracy up even further whilst knowing it was fitting the data well from the cross validation score.

Loss functions:



To test the loss function and optimizers I looped over a few different options to test all possibilities. For loss functions, the options I found for my testing were CrossEntropyLoss, NLLLoss, and MultiMarginLoss. As seen above the results are not very consistent but after running my entire test a few times it became apparent that CrossEntropyLoss built a more accurate model most consistently compared to the other 2 options thus I chose this as the Loss function for my final model.

Optimisation techniques:
When it came to optimisers there were a few more options compared to the loss functions. I ended up choosing Adam, SGD, RMSprop, AdamW, and Adamax. Again the results were not very consistent but Adamax seemed to do a slightly better job at optimizing the parameters vs the other options. I believe this is because unlike adam which uses the L2 norm for optimisation the Adamax uses an infinity norm which has proven slightly better among many use cases. Thus this was my choice of optimiser for the final model.

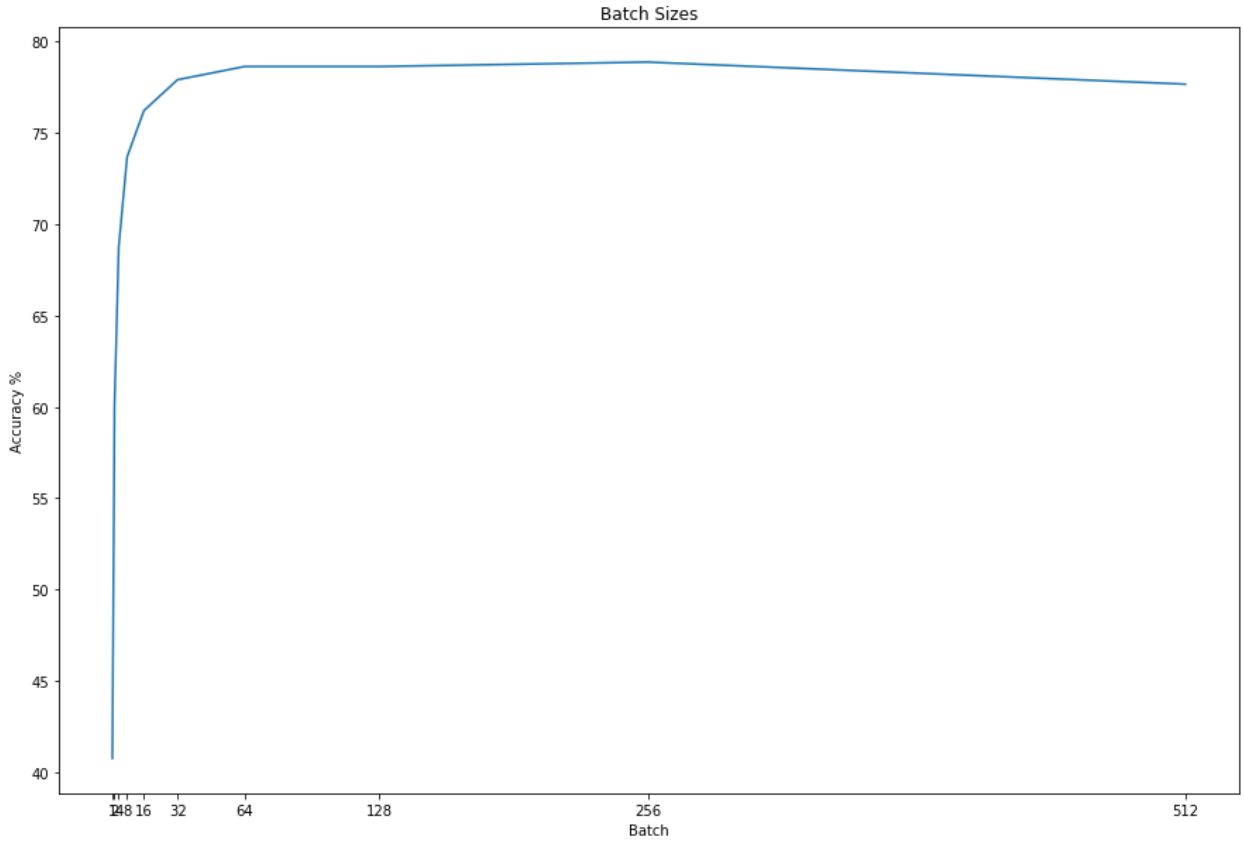
Get more data:
I manually obtained 300 more images (100 per class) to help enrich my training data set. I did this using google images. This allowed me to pick out images that represented each class well and avoided the possibility of adding images that could potentially negatively affect my model. I also found some large fruit datasets on kaggle which I downloaded and extracted the cherry, strawberry, and tomato sets. The first data set was 100-200 close up images of each fruit from all angles with a plain background. It also included multiple variants of each fruit such as

different coloured cherries, different shaped strawberries, and unripe tomatoes to name a few. The only problem I came across was the fact that there was more data for tomatoes and cherries than strawberries which would create an uneven training set and could affect the model negatively. I solved this by finding another data set and adding more strawberry data. This proved to be an extremely valuable optimisation technique as it bumped my base model CNN up over 10%.

Data added

- Cherry: 1240
- Strawberry: 1311
- Tomato: 1330

Batch sizes:

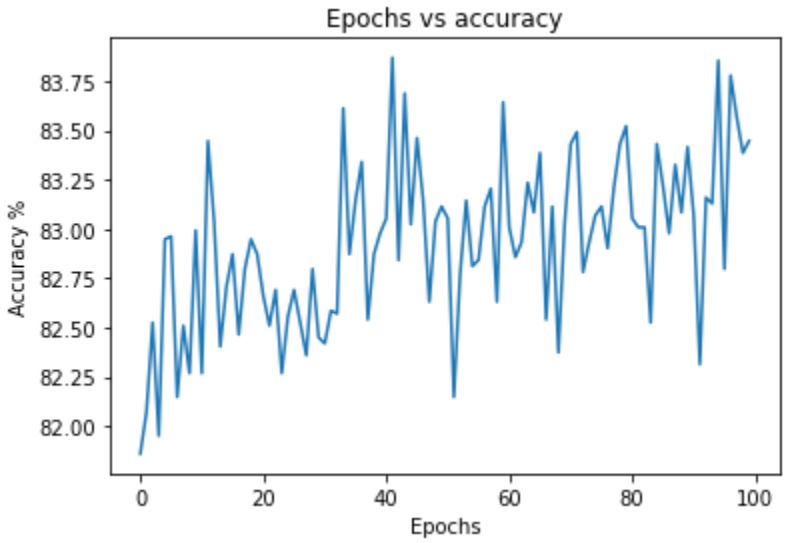


I created a method for testing a few different batch sizes. As seen from the graph above the accuracy jumps significantly at the start and then tapers out and even decreases slightly. After running this method a few times it became apparent that a batch size of 64 was the most consistent and returned the best results on the test data.

Others:

I tried a few different ranges of epochs throughout this project. During the early stages, I used as little as 5 epochs. Although much faster which was good for testing the original accuracy

and the final accuracy were not largely different. I increased the epochs to around 10 which bumped up the final accuracy by roughly 10%. I used 10 epochs for building my first CNN model. Although roughly four minutes slower than my MLP it created a much more accurate model. I then bumped it up again by double to 10 epochs for the cross validation. I had to leave my computer running this process overnight so doubling the epochs didn't affect my wait time and helped increase the model accuracy again by 5-10%. For the final build of the tuned CNN model, I tried running it for 1000 epochs but after five hours I noticed the accuracy was not increasing enough to justify the wait so I backed it off to 100 which was enough to create a good model but only took 1 hour. You can see from the graph below that it helped increase the model slightly.



Result discussion:

MLP:

The MLP model took 2 minutes and 44 seconds to train running 5 epochs. It produced a 48% classification accuracy. Although this does not seem very high a random guessing algorithm would score roughly 33% which means the model was 15% more likely to pick the right class for each fruit. The model only contained two linear layers and one Rectified linear layer which struggled to make accurate predictions on the test data.

CNN:

Using the cnn from the week 11 tutorial with minor adjustments increased the accuracy a lot. My baseline CNN scored 74% accuracy which was 26% better than the MLP model. The model itself however did take 6 minutes and 48 seconds to train which is over 4 minutes longer.

Reference for baseline cnn: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Tuned CNN:

Finally, the tuned CNN which took advantage of cross validation as much more training cycles and a different optimizer. I also added in another convolutional layer, some batch normalisation and a dropout layer. The now more complex model was able to score 83% on the test set. The final model took 67 minutes and 7 seconds to train 100 epochs which was over 10 times longer. It is also notable that the accuracy struggles to get any higher than 83-85% on all data which I believe is because of how similar the classes are.

Conclusion:

Pros:

The pros of my approach to creating a tuned CNN model were definitely adding more data that was a clearer representation of each class (fruit). This helped the performance of the model immensely. Another pro was the use of graphs and visuals to represent how each cycle affected the models accuracy. This made it easier to see what was a good improvement and what was not.

Cons:

The scale of the data and amount of training steps the model took was too much for my laptop which makes it very hard to run a lot of tests and look for areas of improvement. Another con for the approach I took was running the cross validation before testing the optimisers, loss functions, and batch size. This is because the model learns all the data which could cause overfitting.

Future work:

If I was to do this again or continue working on this model I would look into ways of simplifying or saving my models to do transfer learning to prevent the need to continuously re-run the large stages of the build process.