Capstone 3: Final Report

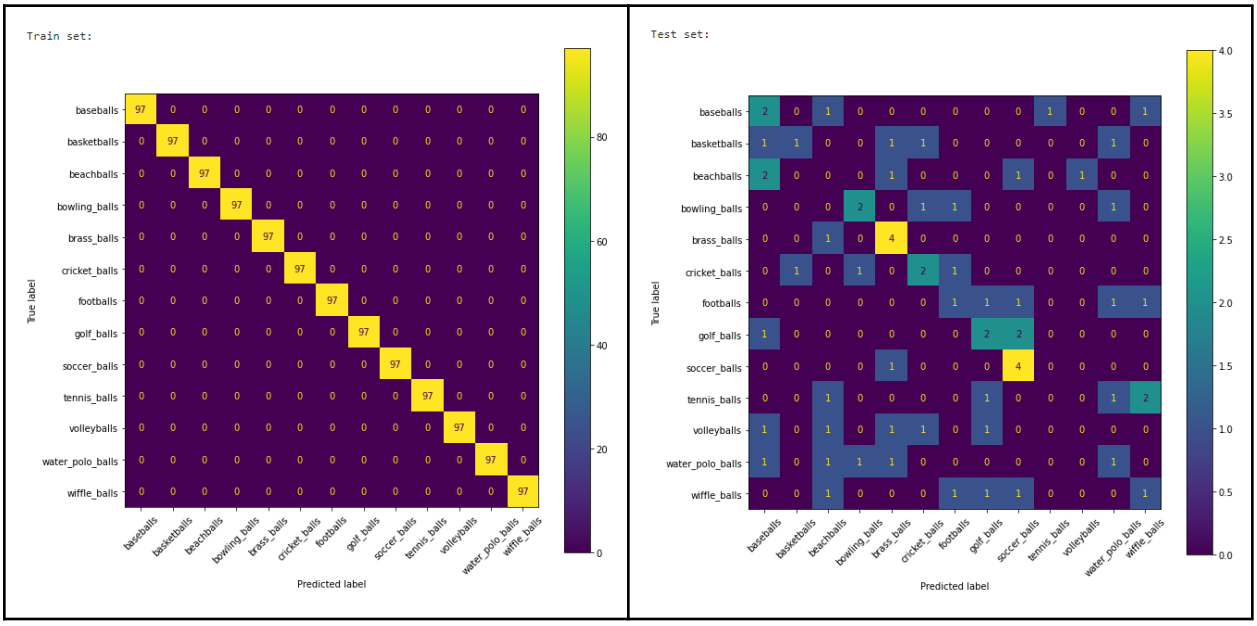
Marcial Medina

Table of Contents:

Decision Tree I:

The first model I tried out was scikit-learn's decision tree classifier. I used a random state instance so that the results can be replicable. I also used scikit-learn's GridSearchCV to try several values of a few parameters. The number of cross-validation folds I used is 5 and the -1 value for the n_jobs parameter allows me to use all available processors to run jobs in parallel. The first parameter I changed around was the criterion function which is used to measure the quality of a split in the decision tree. Every other parameter used its default value.

GridSearchCV determined that the best performing criterion function was the entropy function. The default would have been the gini function. This decision tree model performed fantastically on the training set of images, classifying each image perfectly. But that is to be expected since it was trained on this set. For the testing set, the model didn't perform as bad as I expected. It classified 30.8% of the images correctly which equates to 20 out of 65 images.



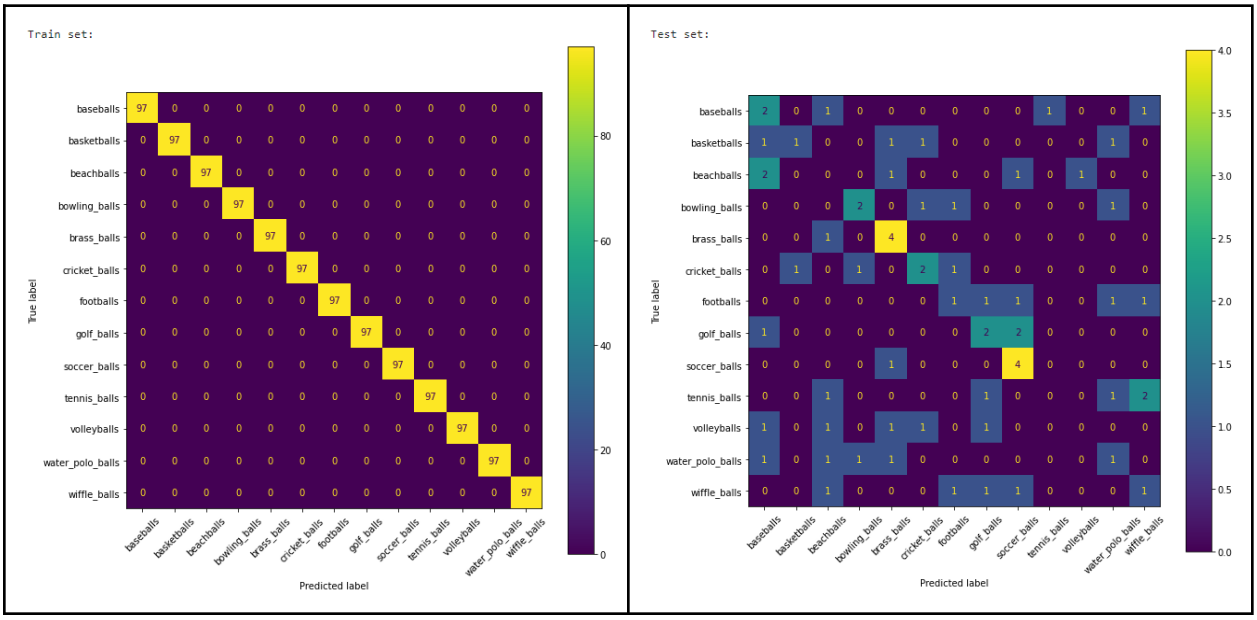
The above images show the confusion matrices for the training set and the testing set. They were generated using scikit-learn's confusion_matrix and ConfusionMatrixDisplay to compute and visualize the charts. The left chart shows that the decision tree model classified each image in the training set correctly. The right chart shows that it performed well with the brass ball and soccer ball images, classifying 4 out of the 5 images correctly. Interestingly enough, one of the true soccer ball images was classified as a brass ball image. The other brass ball image that wasn't classified correctly was placed in the beach ball image class.

The model could not correctly classify any beach balls, tennis balls or volleyballs. These images really gave the model a hard time. It classified 6 non-beach ball images as beach balls and it only classified 1 image as tennis balls and 1 image as volleyballs. The 1 image it classified as tennis balls was actually an image of baseballs which makes sense. They are both relatively similar size balls. And the 1 image it classified as volleyballs was actually an image of beach balls which can have similar designs to volleyballs.

Decision Tree II:

The second model I tried was still scikit-learn's decision tree model with GridSearchCV but with two variable parameters, the criterion function and the maximum depth of the tree. The criterion parameter had the same two choices I used in the first model and the maximum depth had 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and None.

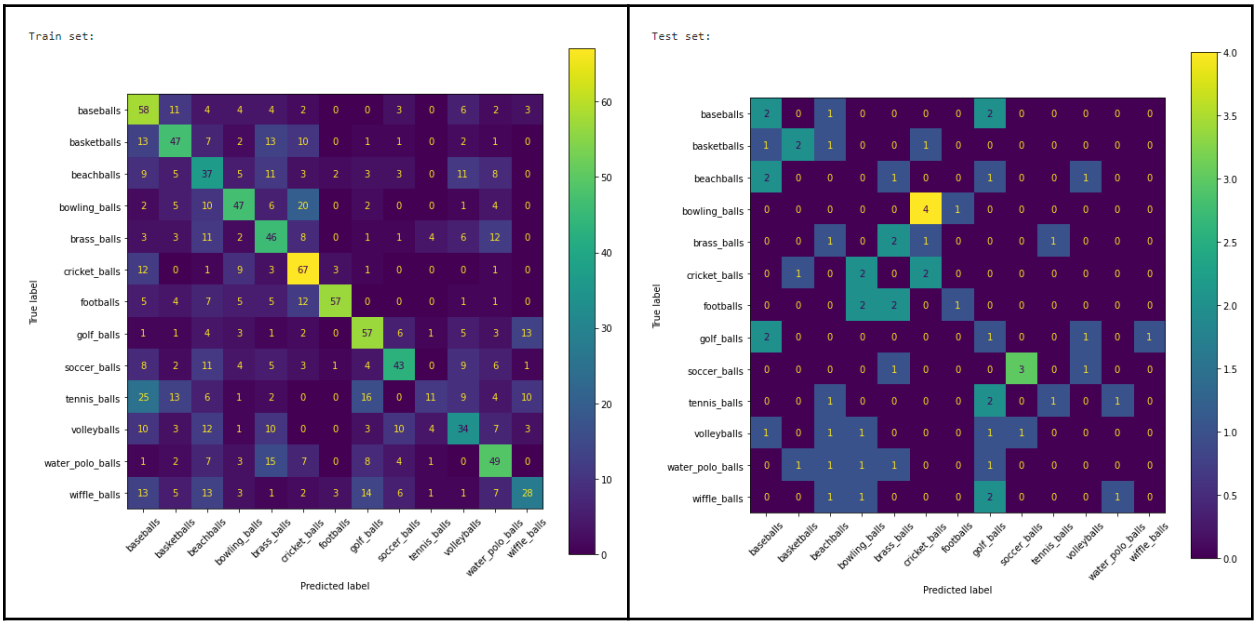
GridSearchCV determined that the best performing combination was the entropy criterion function and a maximum depth of None. None means that the nodes of the tree are expanded until all leaves contain only 1 sample. None is also the default value for the maximum depth parameter. Which means that this model and the first decision tree model have the exact same parameters including the same random instance parameter which means both models produced the same results. It classified all the train set images correctly but only 30.8% of the test set images.



Decision Tree III:

The third model I tried was yet another scikit-learn decision tree model. Again using GridSearchCV, I increased the amount of variable parameters to 3. I chose the same two as before, criterion function and maximum depth, and also min_sample_leaf which dictates the minimum number of samples required to be at a leaf node. Its options were: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 (similar values to maximum depth paramater).

GridSearchCV determined that the best performing combination was the entropy criterion function and a maximum depth of None and a min_samples_leaf of 20. The default for min_samples_leaf is 1 so this means this model is different from the first two decision tree models (despite having the exact same parameter values for the criterion function and the maximum depth). On the training set, this model performed much worse than the first two, getting only 46.1% correct. For the testing set, it also performed worse with a score of 21.5%.



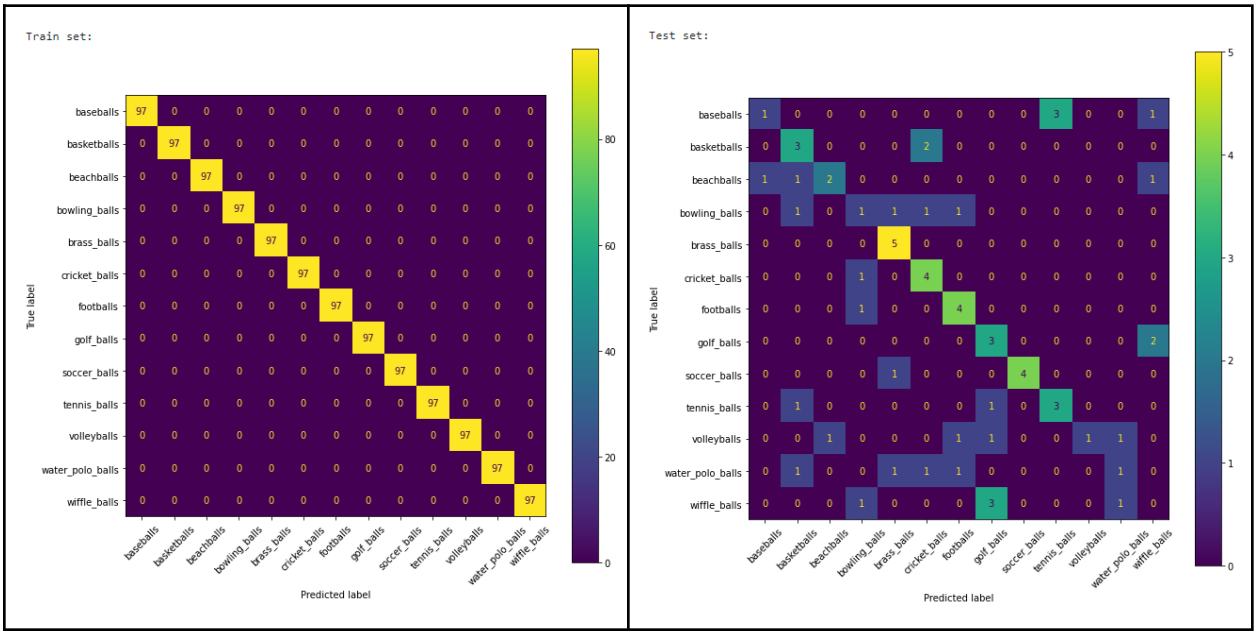
On the left chart (above) is the confusion matrix for the classifications for this model. It performed the best cricket ball images. This makes sense; cricket balls do have a distinct size, shape and color. The model seemed to be the most confused about images of wiffle balls. On the bottom row of the chart you can see that it only classified them correctly 28 times (out of 97, about 28.7%). The rest of the wiffle ball images were classified as every other possible option. The model only classified 11 tennis ball images correctly (11.3%). Instead, it classified most of them as baseballs (25 images), basketballs (13 images), and golf balls (16 images). Also, it classified 20 bowling ball images as cricket balls

The right chart shows the results when the model attempted to classify the test set images. The first thing that stands out is the fact that it classified 4 of the bowling ball images as cricket balls images, something we saw in the training set classifications. It performed very well with soccer ball images, better than it did with the training set soccer balls images. Also, the models continue to perform poorly with wiffle ball images. It only classified 1 image out of the whole test set as a wiffle ball image and it was actually a golf ball image. The wiffle ball images were all classified as either basketballs, beachballs, golf balls or water polo balls.

Random Forest I:

The fourth model I tried out was scikit-learn's random forest classifier. So instead of a single decision tree I will use a whole forest's worth of decision trees. I used a random state instance so that the results can be replicable. I also used scikit-learn's GridSearchCV to try several values of a few parameters. The number of cross-validation folds I used is 5 and the -1 value for the n_jobs parameter allows me to use all available processors to run jobs in parallel. The first parameter I changed around was the criterion function which is the first parameter I changed around for the decision tree classifier as well. As a reminder, the criterion function is used to measure the quality of a split in the decision trees. Every other parameter used its default value.

GridSearchCV determined that the best performing criterion function was the entropy function, just like the previous decision tree models. It classified every image in the train set perfectly. Additionally, it got just under half of the images in the test set correctly which is a huge improvement over the previous models. The best performing decision tree I tried had a score of 30.8% while this basic random forest model had a score of 49.2%, which equates to about 32 out of the 65 test images. Let's look at the confusion matrices for the training set and the testing set.



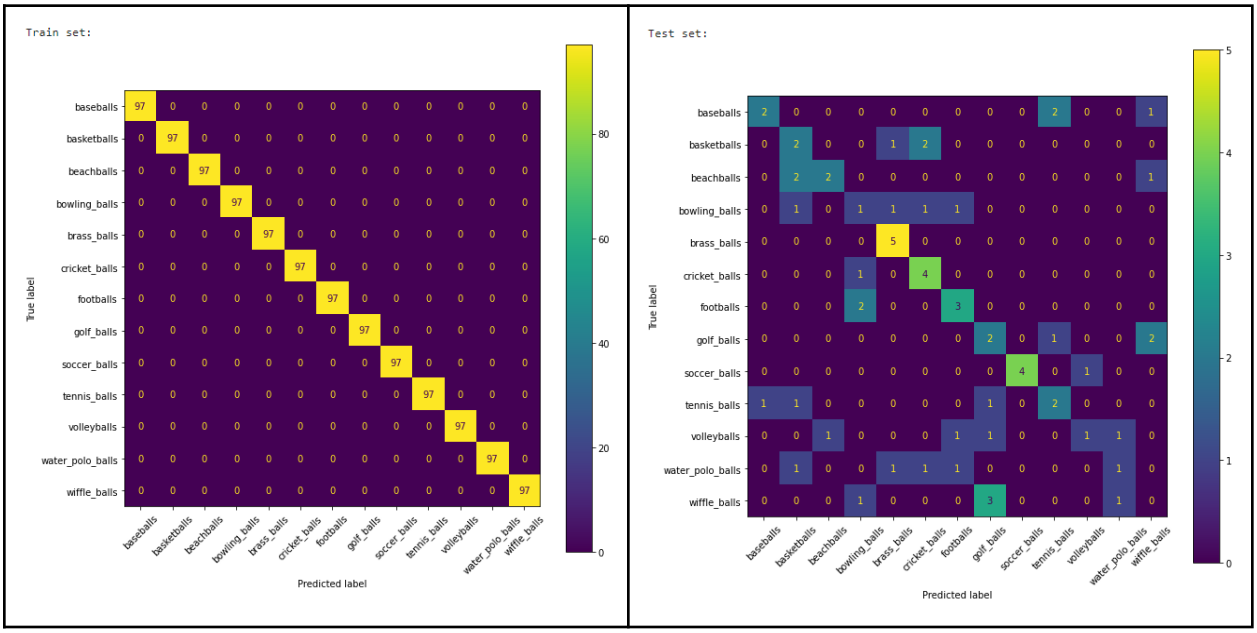
The fact that this random forest model also struggles with wiffle ball images is the first result that stands out. It did not classify any wiffle ball image correctly. It did however classify a couple of golf ball images, a baseball image and even a beach ball image as wiffle ball images. And it also classified 3 wiffle ball images as golf balls. The confusion between these types of ball images makes a lot of sense since they are both relatively small balls with circles all around the sphere. I'm noticing a trend where all the models so far have struggled with a few categories. Namely, the last three categories are volleyball, water polo, and of course wiffle ball images. This group can also include baseball, beachball, and bowling ball images.

The model did a great job with brass balls, which are arguably the most uniform ball in terms of color. They are just spheres with a brass-like color (because they are made of brass). It also performed very well with cricket ball, football, and soccer ball images. These groups are also distinct in their designs.

Random Forest II:

The fifth model I tried was another random forest model but with GridSearchCV searching through a second parameter. This time I chose the criterion function again but paired with the parameter n_estimators. This parameter dictates the number of decision trees in the forest. Its options were: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 (the default is 100).

GridSearchCV determined that the best combination was the entropy criterion function (again) and the number of trees set to 80. This model performed as well on the training set as the previous random forest model did, classifying each image correctly. However, it performed slightly worse with the test set. It scored 44.6%, so 29 out of 65 test images.

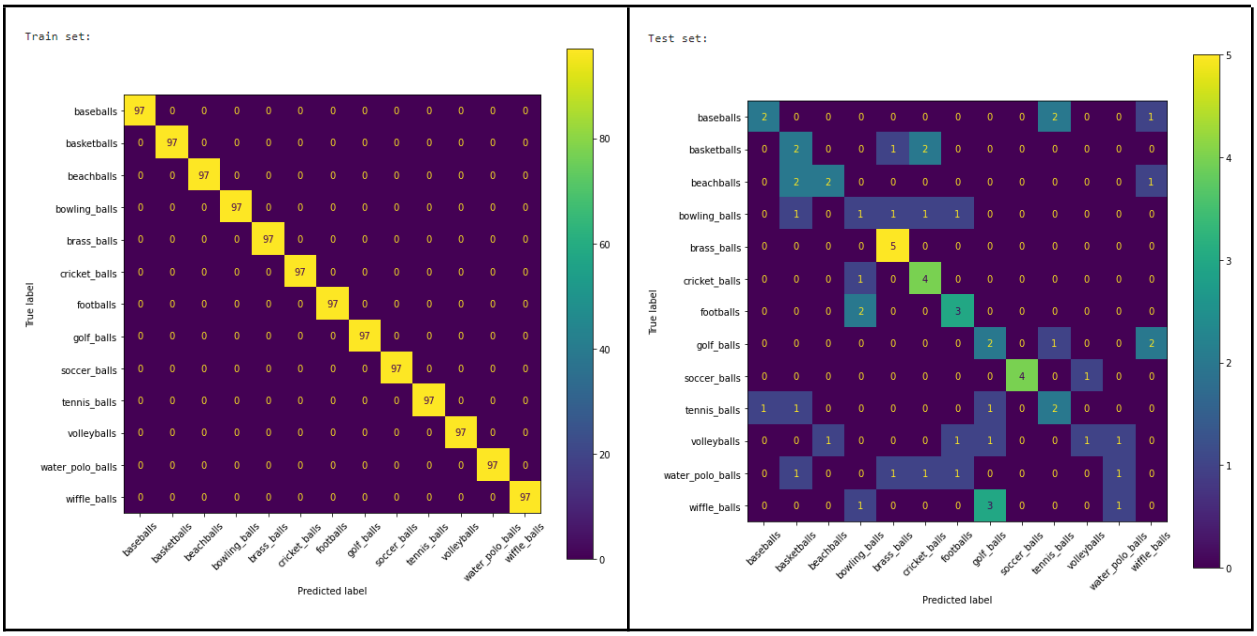


The confusion matrix on the left is identical to the one for the first random forest model. The one on the right only looks very similar. It classified one more baseball image correctly that the previous model, but classified 1 less correctly for basketballs, footballs, tennis balls and golf balls.

Random Forest III:

The sixth model was yet another scikit-learn random forest classifier with GridSearchCV. This time I used three variable parameters: criterion function, number of trees, and the maximum depth of the trees. The first two parameters had the same options as before. The maximum tree depth had options 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and None.

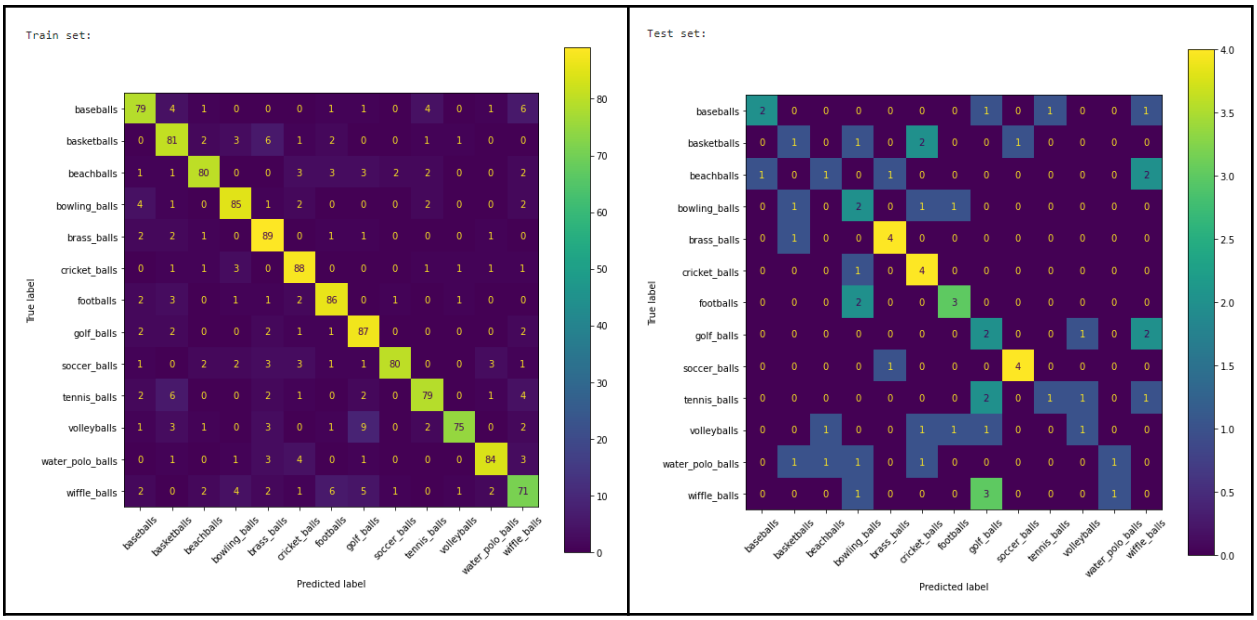
GridSearchCV determined that the best combination was the entropy function, 80 trees, and a maximum depth of None. However, since None is the default value for this parameter and the same random instance was used, this means that this random forest model, random forest III, is almost identical to random forest II. The “almost” comes from the number of trees this model chose. It chose 80 which is different from the default number of 100. Despite this difference, the two models produced the same results. The train set images were all classified correctly and the test set images were classified correctly 44.6% of the time. Not only is the percentage identical, but the confusion matrix is a mirror image.



Random Forest IV:

The seventh model I tried was another random forest model but with GridSearchCV looking through 4 parameters. They were the same ones as before, criterion function, number of trees, maximum depth of the trees, and with a new one called min_samples_leaf. This is essentially the same parameter I used in the decision tree III classifier. It dictates the minimum number of samples required to be at a leaf node for each tree. Its options were: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100.

GridSearchCV determined that the best combination is the gini criterion function, 60 trees, no maximum depth and a min_sample_leaf value of 10. This is the first time that the gini function was chosen as the best option. And although the maximum depth value didn’t change, the number of trees did. This, along with 10 being chosen instead of the default value 1, means this model should have been sufficiently different enough to produce different results. Which it did: it got 84.4% of the train set images correct and 40.0% of the test set images correct.



All in all this model did not perform that well. Once again, these models struggle with wiffle balls. It correctly classified 71 out 97 wiffle ball train images which is not great., a poor 73.2% score. It got over 90% of the brass ball and cricket ball images classified correctly, and even golf balls it performed well on. However it did classify 9 volleyball images as golf ball images.

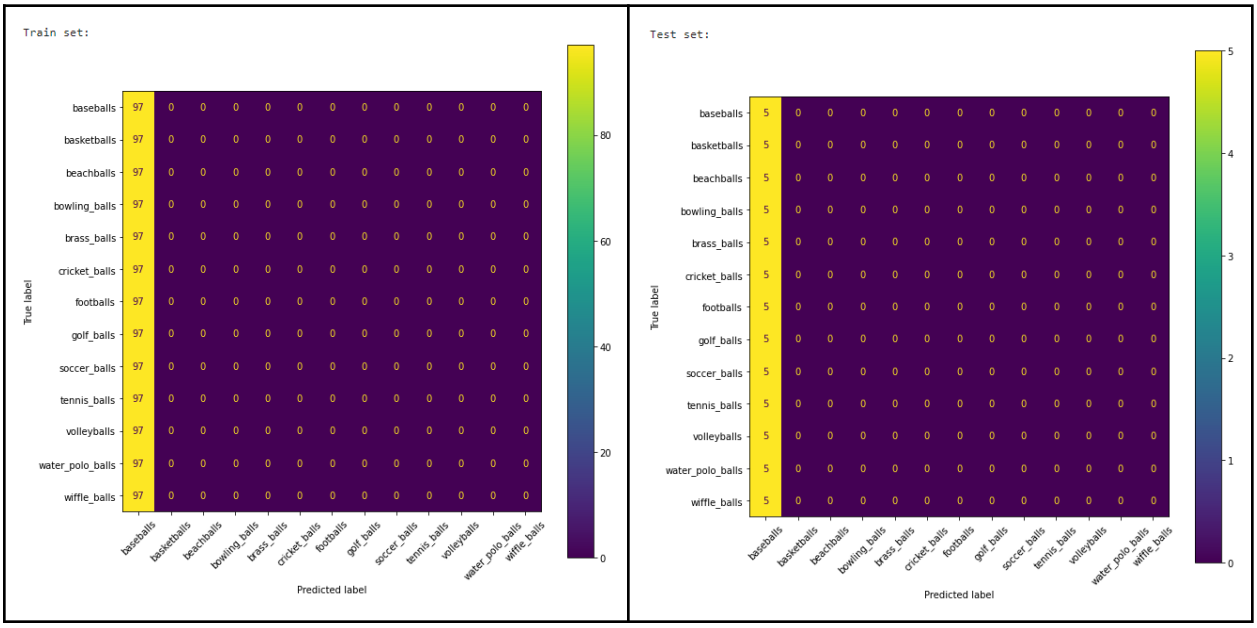
As for the test set images, things do not look good. Wiffle ball images continue to completely confuse the model. It classified 2 beach ball images as wiffle balls. These two always seem to confuse the models. Other than brass balls, cricket balls, and soccer balls, it performed terribly.

PyTorch CNN I:

For the eighth model, I tried something different. I used a PyTorch convolutional neural network. It consists of just two linear layers and the ReLU (Rectified Linear Unit) activation function. There was no special reason I chose 4500 as the out_features parameter of the first linear layer other than it was just under half of the value in the in_features parameter (9408).

The DataLoader for the train images are shuffled in order to train the model more effectively. I used the PyTorch Adam optimizer with the learning rate set at 0.001 and the weight decay set at 0.0001 and also used the cross-entropy loss function.The neural network looped through the entire shuffled training dataset 10 times.

This model performed horribly. For the training set images, it correctly classified 7.7% of them. That is 97 images out of the 1,261 images that are in that set. For the testing set, it got a score of 7.7% as well. Which is equivalent to 5 out of the 65 that are in that set.

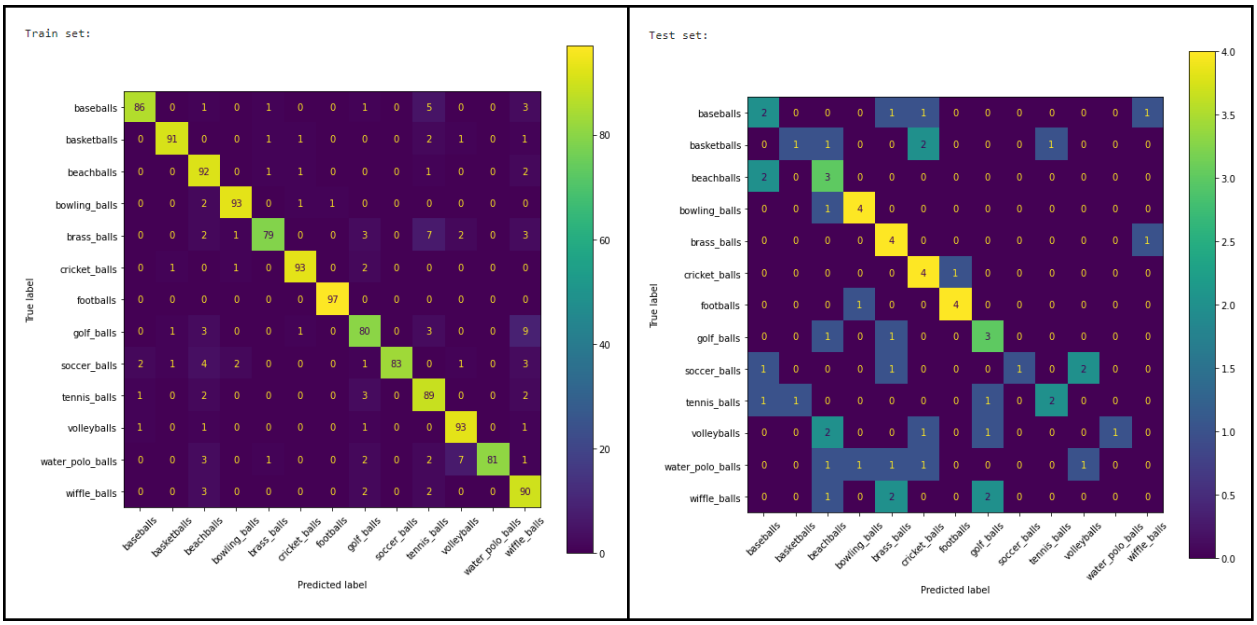


The problem is very obvious: the model is classifying every image as a baseball image. It's like the saying goes, “even a broken clock is right twice a day”.

PyTorch CNN II:

The ninth model was also a PyTorch CNN but this one has more layers.I removed one of the linear layers and introduced 3 2D convolutional layers. Beyond the layers themselves and the parameters of those layers, this model is the same as the first PyTorch CNN. So it uses the same activation function, loss function and the same Adam optimizer.

This model performed much better with an accuracy score of 91.0% for the training set and 43.1% for the testing set.

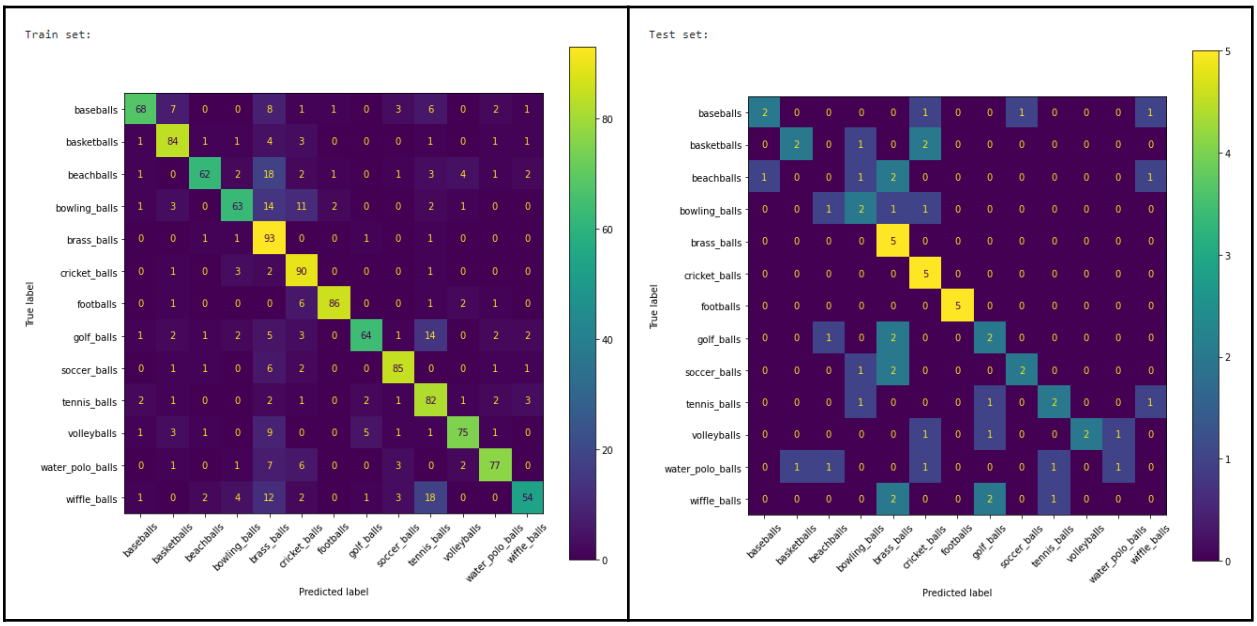


This model performed very well with wiffle ball training images, getting 90 out 97 correct (about 92.8%). It did predict 9 golf ball training images as wiffle ball images however. Golf ball images were amongst the most confusing for the model since. Interestingly enough, its worst class was brass ball images. It correctly classified 79 out of the 97, which is still above 80% but below what I would expect given how well the scikit-learn models performed with these images relative to the other images. The model incorrectly classified 7 brass ball training images as tennis ball images. This could be a consequence of tennis balls and brass balls having roughly the same size and roughly similar colors. The last interesting nugget for the training images results are the 7 water polo ball images that the model misclassified as volleyballs. These two types of balls look very similar, the two main distinctions are the size and texture. The sizes are not that far off and it is difficult to tell from the images. The same goes for texture. However, there is one thing that sets water polo ball images and volleyball images apart: the presence of water. Some, but not all, water polo ball images show water, which could have helped the model distinguish between the two.

As for the testing set, this model's accuracy score was lower than that of the first 3 random forest models. It performed very well with the bowling ball, brass ball, cricket ball and football image classes, but did not get any of them perfect. Also, it did not correctly classify any water polo ball, volleyball or wiffle ball images. Fun fact: there were only 2 water polo ball images that showed water. However, given that it only confused 1 water polo ball image as a volleyball image and vice versa, the lack of water in the images wasn't the only factor. The model also misclassified water polo ball images as ones containing beachballs, bowling balls, brass balls, or cricket balls. For volleyball images, it misclassified some as beach ball images, cricket ball images and even golf ball images. Based on this, it is possible that the model is getting confused with the color schemes of the water polo balls and the volleyballs. They come in different colors and so do the balls it confused them with.

PyTorch CNN III:

This cnn (the tenth model overall) adds a 2D max pooling operator into the mix. It uses the same layers as the previous model but applies this max pooling operator between the first and second convolutional layers. This model performed worse than PyTorch CNN II with the training images, 78.0% vs. 91.0%, but somewhat better with the testing images, 46.2% vs. 43.1%. That is a 2-image improvement, 28 vs. 30. Still not as good as the first couple of scikit-learn random forest models.



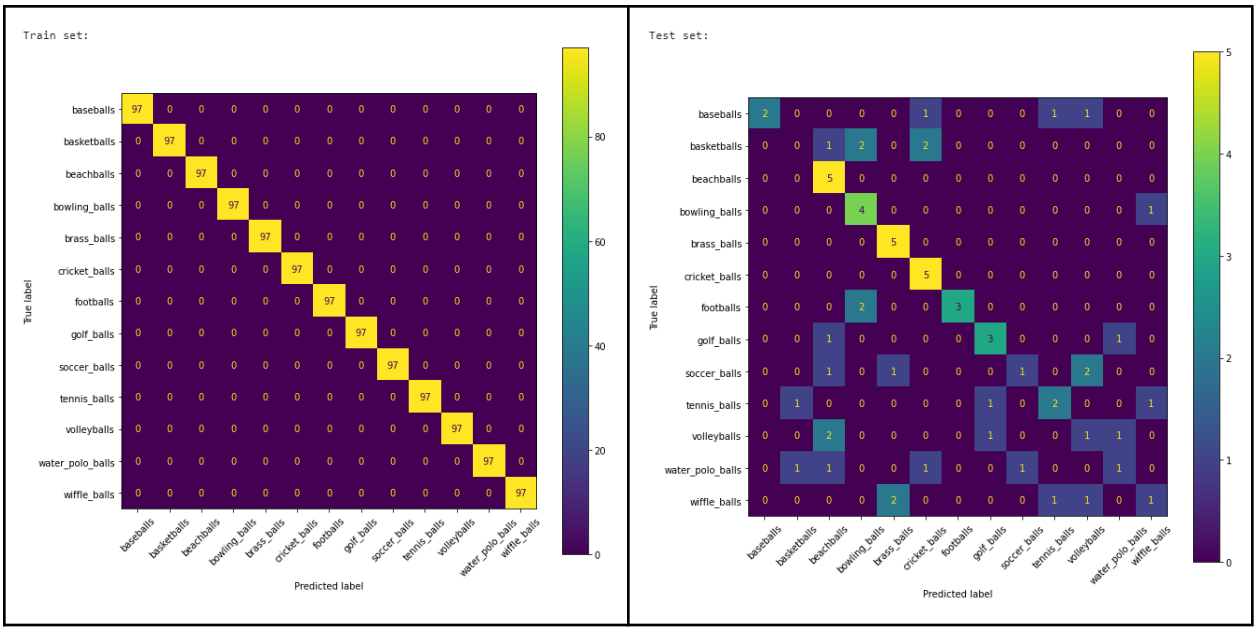
The confusion matrix for the training images looks like a mess. It correctly classified 93 brass ball images correctly, about a 95.9% rate, which is very good. However, the model cheated by classifying 180 images as brass ball images. That is almost double the amount of true brass ball images there are in the set and about 14% of all images in the set. It employed similar strategies for cricket balls, which worked almost as well, and tennis balls, which didn't work as well. 90 out 97 correctly classified cricket ball images is very good, but it also misclassified 11 bowling ball images as cricket ball images. This makes sense, bowling balls and cricket balls, although different sizes, can both have a "shiny" look to them as a result of polishing. The one very important and obvious characteristic that golf balls, tennis balls, and wiffle balls have is that they are relatively small and many of their images contain multiple balls of the same class. This would explain why so many golf ball and wiffle ball images were placed in the tennis ball class. What can't be explained is why this model performs so terribly with wiffle balls (55.7%) when the previous model, which is just missing the max pool operator, had a score of 92.8% with that group.

The big takeaway from the testing images confusion matrix is the perfect classification of brass ball images, cricket ball images and football images. Although, the perfect brass ball score is ruined by the same problem this model had with the brass ball training images. It classified 14 images as brass ball images. Those 14 images make up over 20% of all images in the test set. The perfect accuracy score for the football images are very impressive. Not only did

the model classify all 5 football images correctly, it also avoided classifying any non-football images as football images. Another takeaway, the model continued to struggle with wiffle ball and beach ball images.

PyTorch CNN IV:

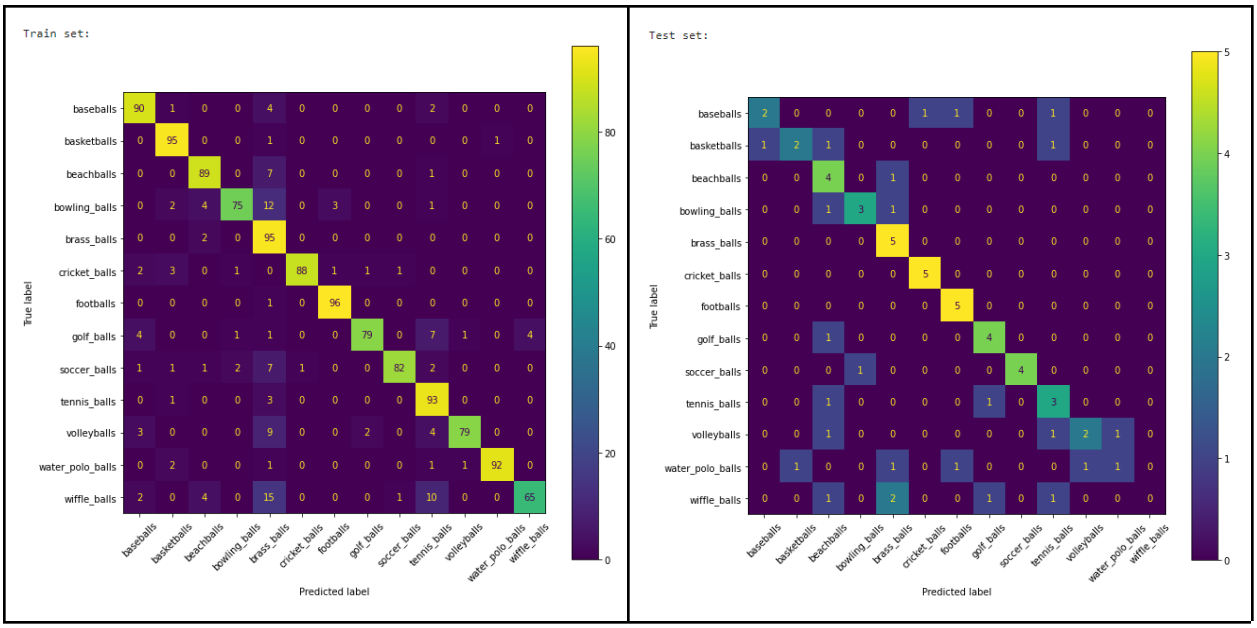
The 11th model, another PyTorch convolutional neural net, builds on the previous model. PyTorch CNN IV has the same layers as III, but adds two batch normalization layers. The first one is placed before the second convolutional layer and the second one is placed before the linear layer (final layer). This model classified each image in the training set correctly and approximately 50.8% of the testing images correctly. It is thus far the best performing model, just barely outclassing the first scikit-learn random forest model by 1 image (100% for training set, 49.2% for testing).



There is nothing to be said about the training set results, so I'll just start with the testing image results. This model did not use the "classify everything as brass balls" strategy as the previous one did. It's possible it did use that strategy with beach ball images, but since this isn't reflected in the training set confusion matrix nothing can be concluded.

PyTorch CNN V:

The 12th and final model added another convolutional later in the cnn. It is nestled between the second batch normalization layer and the linear/final layer. It's accuracy score for the training set was 88.7% and for the testing set it was 61.5%. This improves on the last one by a big margin; it correctly classified 40 out of the 65 test images.



It seems like this model also likes to use the “brass ball” strategy, although not to the same degree as PyTorch CNN III. In particular, it liked to classify bowling ball and wiffle ball images as brass ball images. As explained earlier, this is possibly as a result of brass balls being small, like wiffle balls, and “shiny”, like bowling balls. Another similarity between this model and some of the previous ones, particularly PyTorch CNN IV, is that it struggled with wiffle ball images. This class of images had the lowest score, 67.0%. It was followed by the bowling ball image class which had a score of 77.3%. Wiffle ball images are just difficult to classify when there are also tennis ball and brass ball images that look similar.

This is again confirmed when looking at the testing set confusion matrix (right). It could not classify any wiffle ball images correctly and, what’s more interesting is that it did not classify any non-wiffle ball images as wiffle ball images.

Recommendation:

My recommendation on the type of model that should be used for image classification is a PyTorch convolutional neural network. The best performing model for this project was a PyTorch CNN that correctly classified 88% of the training images and 61% of the testing images. Although this performance isn’t mindblowing, it does prove that this type of model can work very well. It outperformed the scikit-learn decision tree and random forest models that I trained. The best performing non-PyTorch model was the third random forest model, it correctly classified every single training image and 49% of the testing images. However, there are less avenues to improve decision tree and random forest models than PyTorch CNNs. That being said, it is difficult to figure out what exactly can be modified in the neural net in order to make it more accurate. There are two things that come to mind that could improve performance: the number epochs and the number of training images. I limited the cnn’s to 10 epochs for time and because I didn’t want to overfit the models to the data of which an argument could be made that there wasn’t enough. I only used 97 images for each training class. Perhaps using more images, the models could make more meaningful distinctions between the different classes.