Plates

There are just 40 plate images for training which is not much, so I considered the following approach. First I undertook some fine-turning to try and improve my score.

1. The first thing i did was to use a pre-trained model InceptionResNet V2.
2. I froze the model preserving the learnings from the model and unfrozen the model with the following

	froze	unfrozen
freeze	59274	3494

3. The finding was that it was better to freeze the model: `base_model.trainable = False`
4. The next thing was to update hyper parameters. The approach was to change one hyperparameter and begin training again.
5. The first hyper parameter was batch_size and i undertook the following with these results

batch_size	16	56586
batch_size	32	59274
batch_size	64	45564

6. The take away is that there is an optimal batch_size for a model that has an impact on the performance of the models prediction.
7. There is a relationship between the batch_size and learning_rate and its worth adjusting this value if you change a batch_size. I found the following

learning_rate	0.0001	59274
learning_rate	0.005	54435

8. The result showed that learning rate has an impact on the models performance when generalizing on test data. Again this is a case of adjusting one parameter and retraining the model and observing the results.

9. Next up was epochs, the number of times we go through each cycle key for training.

epochs	50	59274
epochs	100	54564

10. A key observation is that this actually becomes irrelevant if you use early stopping. This stops training before the model begins to overfit. In most occasions the model will call early stoping before it completes all epochs.
11. For small datasets epochs from 50-100 are recommended for better learning as the model sees the same data repeatedly. For large data sets it is recommended to use 10 to 30.
12. Without early stopping if you have to many the model can overfit.
13. The next step was to include a validation_generator which is best practice, something that i did not include. Its worth noting without validation you miss out on some metrics such as val_accuracy and val_loss and cannot use early stopping.
14. A key parameter in early stopping is patience. I adjusted this parameter with the following findings

patience	5	52419
patience	10	58870
patience	15	56989

15. So patience is just giving training a chance e.g. if there are 5 epochs and no improvement in say val_loss and say it starts getting worse training is terminated early, and the weights are restored from the best epoch.
16. This stops overfitting. Using verbose=1 tells you when it terminates and why.
17. The last part of hyper tuning was using steps_per_epochs. This is in general the number of batches an epoch will see before completing and moving to the next epoch. Again I found this influenced the final score when using predict with the model.

steps_per_epochs	100	59946
steps_per_epochs	50	42607
steps_per_epochs	150	32215

18. Again with making an adjustment, then retraining, steps per epoch had some influence over the models prediction.
19. If to small the model cannot see enough data per epoch and results in poor generalization.
20. if too large then the model overfits and gives poor predictions.
21. I took it as far as i could. Then i considered the pre-trained model.
22. It turns out that the inceptionResNetv2 model is super complex, great for large images with complex data sets and tends to overfit with small data sets.
23. My dataset is 40 images so obviously changing to a less complex, small data friendly model makes sense.
24. I employed Resnet50 a simple model that works great with small data sets so i used this.

Before

```
base_model = applications.InceptionResNetV2(weights='imagenet',
include_top=False,
                                          #
input_shape=(image_size,image_size, 3))
```

After

```
base_model = applications.ResNet50(weights='imagenet', include_top=False,
input_shape=(image_size,image_size,3))
#%%
```

25. I decide to keep the pre-models weights from the convolutional layers to dense (the top) frozen to preserve the learnings from the new model.
26. There was one more adjust required involving the flattening layer in the top layer of the CNN.
27. I used

```
x = Flatten()(x)
```

28. This is a complex function that flattens the output to pass this on to the dense layer. However this layer is great for complex models with substantial data, a poor fit for a small model with 40 images. This is the axes between the convolutional layers and the connected layer. I found something bettering fitting with a small scale model such as mine and used this.

```
x = GlobalAveragePooling2D()(x)
```

29. This compress the feature maps to small vectors and is optimal for small datasets and reduces overfitting. This passes on to the dense layer (connected layer)
30. The results

ResNet 50 + GlobalAveragePooling	63440
inceptionResnet + Flatten	57930