Capstone Three Documentation

Objective

The purpose of this project was to attempt to use transfer learning from FastAi's pre-trained convolutional neural nets to classify chart patterns as being predictive of price moves in financial assets.

Data Required

The best things about using financial price data are that they are 1) very easy to source and 2) generally very clean. I was able to get price data from a variety of sources including IG Index, Tradermade and Yahoo finance. For those wishing to explore further I recommend rapidapi.com. I mostly used daily price data over a period of about a year, for roughly 500 equities and about 50 different price indices, commodities and currencies. To mix things up as a way of making the model more robust, I also added a few datasets that were on minute timeframes rather than daily.

Data Preparation

The raw price data was simply the daily (or minute) open, high, low and close prices, coupled with - in some instances - volume data. I algorithmically transformed the data into candlestick representations, so that it would suitable for feeding into a visual CNN.

Once this was complete, I scanned the datasets for instances where the price had a substantial price move. For the daily price data I calculated this as a move greater than 3.7% within 5 days, and for the minute timeframe data I used 1% and 2% changes over the course of 5 minutes.

Once the algorithm found instances of these prices moves, it would save a chart of the 20 candles that preceded the price move, and save them into classification folders of Up or Down.

I generated about 30,000 features in this manner.

Data Preprocessing

One of the (many) remarkable aspects of FastAi is that it automates the pre-processing. With one line of code I was able to automate transformation and resizing of all the features, so that the images are slightly different for each epoch that the model trains through. This ensures robust models, by preventing overfitting.

I used two different sizes of image, 224x224 and 352x352.

Infrastructure

I tried to use GPU enabled notebooks such as Colab and Paperspace for this project to speed up the model training, since the feature sets were so large, but unfortunately I could not get them to work properly, despite many, many hours of trying! As a result I wound up using Jupyter on my local Mac, which took about 4x as long. It was, however, at least effective.

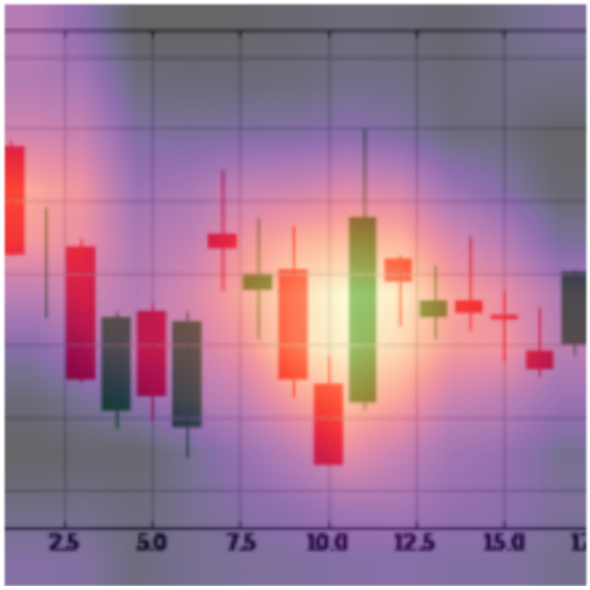
Results

The second model, which was trained on the larger images, eventually achieved an accuracy of about 68.5%, which for trading purposes is phenomenal. Generally, in trading, an edge of anything over 50% is considered sufficient to deploy in liquid markets.

epoch	train_loss	valid_loss	error_rate	time
0	0.546471	0.607613	0.325480	4:49:54
1	0.544967	0.601937	0.320244	4:45:24
2	0.531639	0.602111	0.320419	4:40:23
3	0.521480	0.599843	0.316405	4:40:24
4	0.532581	0.603534	0.318848	4:41:33

Feature Importances

FastAi's callback hooks enable visualisation of the feature importances. I had to switch to Colab to generate these as they require CUDA functionality in pytorch to be enabled, which is only available on Nvidia chips.



Model Summary

```
Sequential
==========================================================================
Layer (type)         Output Shape         Param #    Trainable
==========================================================================
Conv2d               [64, 176, 176]       9,408      False
_________________________________________________________________________
BatchNorm2d          [64, 176, 176]       128        True
_________________________________________________________________________
ReLU                 [64, 176, 176]       0          False
_________________________________________________________________________
MaxPool2d            [64, 88, 88]         0          False
_________________________________________________________________________
Conv2d               [64, 88, 88]         36,864     False
_________________________________________________________________________
BatchNorm2d          [64, 88, 88]         128        True
_________________________________________________________________________
ReLU                 [64, 88, 88]         0          False
_________________________________________________________________________
```

Conv2d	[64, 88, 88]	36,864	False
BatchNorm2d	[64, 88, 88]	128	True
Conv2d	[64, 88, 88]	36,864	False
BatchNorm2d	[64, 88, 88]	128	True
ReLU	[64, 88, 88]	0	False
Conv2d	[64, 88, 88]	36,864	False
BatchNorm2d	[64, 88, 88]	128	True
Conv2d	[64, 88, 88]	36,864	False
BatchNorm2d	[64, 88, 88]	128	True
ReLU	[64, 88, 88]	0	False
Conv2d	[64, 88, 88]	36,864	False
BatchNorm2d	[64, 88, 88]	128	True
Conv2d	[128, 44, 44]	73,728	False
BatchNorm2d	[128, 44, 44]	256	True
ReLU	[128, 44, 44]	0	False
Conv2d	[128, 44, 44]	147,456	False

BatchNorm2d	[128, 44, 44]	256	True
Conv2d	[128, 44, 44]	8,192	False
BatchNorm2d	[128, 44, 44]	256	True
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True
ReLU	[128, 44, 44]	0	False
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True
ReLU	[128, 44, 44]	0	False
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True

ReLU	[128, 44, 44]	0	False
Conv2d	[128, 44, 44]	147,456	False
BatchNorm2d	[128, 44, 44]	256	True
Conv2d	[256, 22, 22]	294,912	False
BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	32,768	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	589,824	False

BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[256, 22, 22]	589,824	False

BatchNorm2d	[256, 22, 22]	512	True
ReLU	[256, 22, 22]	0	False
Conv2d	[256, 22, 22]	589,824	False
BatchNorm2d	[256, 22, 22]	512	True
Conv2d	[512, 11, 11]	1,179,648	False
BatchNorm2d	[512, 11, 11]	1,024	True
ReLU	[512, 11, 11]	0	False
Conv2d	[512, 11, 11]	2,359,296	False
BatchNorm2d	[512, 11, 11]	1,024	True
Conv2d	[512, 11, 11]	131,072	False
BatchNorm2d	[512, 11, 11]	1,024	True
Conv2d	[512, 11, 11]	2,359,296	False
BatchNorm2d	[512, 11, 11]	1,024	True
ReLU	[512, 11, 11]	0	False
Conv2d	[512, 11, 11]	2,359,296	False
BatchNorm2d	[512, 11, 11]	1,024	True

Conv2d	[512, 11, 11]	2,359,296	False
BatchNorm2d	[512, 11, 11]	1,024	True
ReLU	[512, 11, 11]	0	False
Conv2d	[512, 11, 11]	2,359,296	False
BatchNorm2d	[512, 11, 11]	1,024	True
AdaptiveAvgPool2d	[512, 1, 1]	0	False
AdaptiveMaxPool2d	[512, 1, 1]	0	False
Flatten	[1024]	0	False
BatchNorm1d	[1024]	2,048	True
Dropout	[1024]	0	False
Linear	[512]	524,800	True
ReLU	[512]	0	False
BatchNorm1d	[512]	1,024	True
Dropout	[512]	0	False
Linear	[2]	1,026	True

```
Total params: 21,813,570

Total trainable params: 545,922

Total non-trainable params: 21,267,648

Optimized with 'torch.optim.adam.Adam', betas=(0.9, 0.99)

Using true weight decay as discussed in https://www.fast.ai/2018/07/02/
adam-weight-decay/

Loss function : FlattenedLoss

======================================================================

Callbacks functions applied
```

Next Steps

For this project I only attempted using a single pre-trained model. There are, however, many others that exist that I would like to try, such as VGG-16, Inception, Resnet 50 and EfficientNet.

After I have found the most efficient model, I will look to deploy it and begin trading with it.