Convolutional Neural Networks for Trading Strategies in Stock

Elior Illouz (ei2239), Zehui Jin (zj2248)

Sept 2019

Abstract

Convolutional neural network rarely show up in applications in finance, while other deep learning methods, such as Long short-term memory (LSTM) and Deep Q-learning are widely attempted in the financial market for optimal trading time, optimal frontier and event-driven predictions.

In this paper, we aim to implement Convolutional Neural Networks for portfolio selection. In order to fully utilize the advantage of CNN on feature caption and image classification, we converted time series stock data into 2D images first by several different algorithms. Then we used GoogLeNet, AlexNet and ResNet for training, validation and testing. Finally we examined the performance of CNN-based strategy against S&P 500, and the result of GoogLeNet showcases a strong predictive power and feasible future extensions. We also talked about future improvements that could be done on this project.

1 Data Selection and Image Encoding

1.1 Data Selection

The stock data is accessed through The Center for Research in Security Prices (CRSP) at Wharton Research Data Service, where the most comprehensive collection of security price, return, and volume data for the NYSE, AMEX and NASDAQ stock markets is maintained.

Regarding to the timelines and prediction power of historical stock data on future stock data, data from 2010/01/01 to 2018/12/31 was used for the project. Only stocks which have complete recordings throughout this period are kept to avoid null values. The final dataset consists of 2,264 trading days and 3,730

stocks. The ratio of train, validation, and test set is roughtly 0.4*2/3, 0.6*2/3, 1/3. We also added a filter on average daily volume over the whole period to keep stocks where liquidity is not an issue. Indeed as described later we did not do any assumptions on transaction costs, therefore removing illiquid stocks enables us to be more confident with this assumption.

The raw dataset includes 12 features: 'DATE', 'TICKER', 'COMNAM' (company name), 'BIDLO' (lowest bid of the day), 'ASKHI' (highest ask of the day), 'PRC' (close mid), 'VOL' (daily volume), 'RET' (holdings returns, i.e. reinvesting dividends), 'SHROUT' (nb of shares outstanding), 'SPRTRN' (SP500 returns). While 'PERMNO' served as the identifier of the stock, we also took 'COMNAM' into considerations in case any significant merge or acquisition happened. Small volume stocks were removed by a floor on 'SHROUT' to avoid huge fluctuations in stock prices (exceeding half or double) driven by events.

DATE	Date
PERMNO	CRSP Permanent Security Identifier
TICKER	Ticker Symbol
COMNAM	Company Name
BIDLO	The lowest trading price during the day
ASKHI	The highest trading price during the day
PRC	Closing price or Negative Bid/Ask Average on calendar date
VOL	Shares traded during the day (units of one share)
RET	Return
SHROUT	Publicly held shares on NYSE, NYSE American, NASDAQ Stock Exchange
SPRTRN	S&P 500 Composite Index Return

Table 1: Features

We encountered a challenge with stocks splits, indeed when doing the backtest we noticed huge spikes at some points, we noticed that we did not take stock splits into account when shorting. In our data when looking at prices and there is a merge or a stock split 'PRC' would move by a really huge amount. This is why we decided to use the field 'RET' for returns, in addition to that 'RET' computes returns reinvesting dividends so this is what we want. Finally the number of shares outstanding 'SHROUT' was used for stocks splits and merge because in our data volume is multiplied/divided by the multiplier after a merge/stock split. Thus, we used 'SHROUT' to keep consistency in the volume data.

We transformed BIDLO and ASKHI to a comparable ratio, while it also help indicating volatility: $BIDLO = (PRC - BIDLO)/PRC$, $ASKHI = (ASKHI - PRC)/PRC$.

Five features after adjusting remained when building and dumping images: 'RET', 'ASKHI', 'BIDLO', 'VOL', 'SPRTRN'; 'DATE' and 'PERMNO' are kept as index.

1.2 Image Encoding

This section mainly illustrates how GAF and MTF are adopted to convert stock data into images.

Gramian Angular Field (GAF)

Rescale time series data $X = \{x_1, x_2, \ldots, x_n\}$ to make sure they fall in interval $[-1, 1]$ corresponding to the cosine function in $[0, \pi]$ by $\tilde{x}^i_{-1} = \frac{(x_i - \max(X) + (x_i - \min(X))}{\max(X) - \min(X)}$, or in interval $[0, 1]$ corresponding to the cosine function in $[0, \frac{\pi}{2}]$ by $\tilde{x}^i_0 = \frac{x_i - \min(X)}{\max(X) - \min(X)}$.

Then represent $\tilde{x}$ in polar coordinate by taking $\tilde{x}$ as angular cosine and the time stamp as the radius.

$$\begin{cases} \phi = \arccos\left(\tilde{x}_i\right), -1 \leq \tilde{x}_i \leq 1, \tilde{x}_i \in \tilde{X} \\ r = \frac{t_i}{N}, t_i \in N \end{cases}$$

Gramian Angular Difference Field (GADF) requires $\tilde{x}^i_0$ rescaling while Gramian Summation Angular Field (GASF) requires $\tilde{x}^i_{-1}$ rescaling. They provide different information granularity in the Gramian Angular Field for classification tasks.

$$GASF = \left[\cos\left(\phi_i + \phi_j\right)\right] = \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2} \cdot \sqrt{I - \tilde{X}^2}$$
$$GADF = \left[\sin\left(\phi_i - \phi_j\right)\right] = \sqrt{I - \tilde{X}^2} \cdot \tilde{X} - \tilde{X}' \cdot \sqrt{I - \tilde{X}^2}$$

Markov Transition Field (MTF)

Markov Transition Field (MTF) is an extension of Markov Transition Matrix (MTM). For MTM, data is divided into Q quantile bins, and each x_i is assigned to the corresponding bins q_j ($j \in [1, Q]$), then a MTM is constructed by normalizing the $Q * Q$ weighted adjacency matrix W built by counting transitions among quantile bins in the manner of a first-order Markov chain along the time axis.

$$M = \begin{bmatrix} w_{ij|x_1 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_1 \in q_i, x_n \in q_j} \\ w_{ij|x_2 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_2 \in q_i, x_n \in q_j} \\ \vdots & \ddots & \vdots \\ w_{ij|x_n \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_n \in q_i, x_n \in q_j} \end{bmatrix}$$

For MTF, Each pixel M_{ij} represents the probability from the quantile bin at time step i to go to the quantile bin at time step j. The MTF M encodes the multi-span transition probabilities of the time series. Generally:

$$MTM = [freqfromquantileitoquantilej]Q \times Q$$

$$MTF = [(\mathrm{i},\mathrm{j}) - th entry of MTM given x_k \in Q_i, x_l \in Q_j]_{t \times t}$$

GADF is taken as our default and main image encoding method for its advantage in accurate inverse map and temporal dependency preserving.

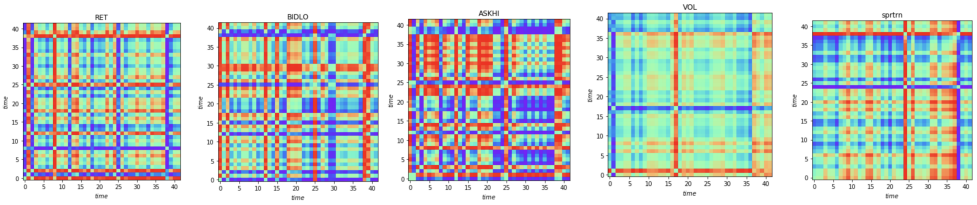
Hyperparameters for Image

The image size indicates the length of data dumped in one image. A large size length may make the image incorporate too much out-of-date information which could have weak predictive power; a small side length may lead to under-fitting. We tried 64, which is roughly a quarter that follows financial statements period, so that one image would be comprised of 64 days of stock information. We also tried 42, which is roughly two months.

There is a trade-off in the image size because we will use this data as an input into the first layer of our CNN. Taking a size too big will force us to convert our 64x64 pixels into a smaller one (averages before using GADF) because this size of images make the training last days or weeks. Obviously there is another reason that we should not use a period that would be too long: regime shifts, we assume behaviours change fast (even faster in Finance) and thus taking more than a few months would not be fine. Therefore, we recommend using at most 42 (2 months) as an input for one sample.

We also set the frequency to rebalance our portfolio as 5 days, which follows a continuous weekdays rotation. In other words, the images will be rebuilt every week.

Figure 1 is an example of five features in GADF Encoding, images are in 42*42 size. We represented them with color to see differences better but a more realistic representation would be black and white since we just have values between $[-1, 1]$. This five images form a tensor of size 42x42x, we have 5 input channels here.



(a) RET (b) BIDLO (c) ASKHI (d) VOL (e) SPRTRN

Figure 1: GADF Encoding

1.3 Labels

Our target variable consists of three positions: long, hold, short. These three actions are determined by the return of the stock. At the beginning we just defined two hard-coded threshold as cutoffs for 'long' and 'short'. However, we thought that there was a problem in doing so because volatility is not the same across our universe. Indeed for a low volatile stock if the predicting result is 'hold' all the time then we are more likely to be correct and it is the same for either 'long' and 'short' for volatile stocks.

Thus, if the return is greater than $\alpha\sigma$ then the stock will be longed; if the return is less than $-\alpha\sigma$ then the stock will be shorted; if the return falls in the interval $[-\alpha\sigma, \alpha\sigma]$ then we will neither long nor short. In practise in the code we consider $\alpha = 4 \times \beta$, where β is a parameter that is adjustable as a manual input from the user when building labels. We chose 4 as a multiplier so that $\beta = 1$ for a volatility of 25%. We chose thresholds of about 1.25% because it makes approximately 33% of samples fall in each category. Also it corresponds to an annual volatility of 20% which makes sense for our study.

2 Neural Networks

We applied ResNet, AlexNet and GoogLeNet in this project. Their architectures from Tensorboard are listed respectively in the Appendix (Figure 6, Figure 7, Figure 8). We recommend looking at them to understand how these structures differ. For all these networks we always considered condensed versions of them (i.e less layers) since the training time was too long for the original versions.

AlexNet won the 2012 ILSVRC competition , there are three major characteristics in the AlexNet: firstly, it uses Relu instead of Tanh to add non-linearity; secondly, it uses dropout instead of regulation to avoid overfitting; lastly, it uses overlapping pooling to reduce the network size. This architecture can be summarized by a series of convolution with pooling and then a series of dense layers (simple feed-forward network).

GoogLeNet won the 2014 ILSVRC competition. It uses batch normalization, image distortions and RMSprop as optimizer for better results. The inception layers are designed to cover a bigger area, but also keep a fine resolution for small information on the images, which reduce the number of parameters. This architectures is a series of "Inception" blocks that consist of a "weighted average" of convolutions with a different filter size.

ResNet won the ILSVRC classification competition in 2015. The most striking part of ResNet is the residual connections that solve the problem of vanishing gradients and optimization difficulty. In this architecture we have blocks

of blocks where each subblock outputs the sum of: the output of a series of convolution, the input of this subblock. Thus, if the output of the series of convolution is zero (which could be reached by zero weights) the result of the whole subblock is the input (and not zero). This is why it does not have the problem of vanishing gradient anymore.

Besides, we also applied batch normalization and cost-sensitive loss function in our models. Batch normalization draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. For cost-sensitive loss, we impose different penalties for different categories. Table 2 is a cost matrix we've used in the project. We put a higher penalty for misclassifying 'long' and 'short' (1.4) and a relatively lower penalty for misclassifying 'hold' and 'long'/'short' (1.1).

	Long	Hold	Short
Long	0	1.1	1.4
Hold	1.1	0	1.1
Short	1.4	1.1	0

Table 2: Cost Matrix for Misclassification

Training Evolution For most of our attempts to build a meaningful network we got overfitting. What we observed was an increasing validation loss while the training loss was decreasing w.r.t the number of epochs. We reached the best results with the GoogLeNet style network so we decided to focus on this one for further fine-tuning. We plotted the evolution of both losses for our best result for GoogLeNet. Surprisingly, we obtained this result without a cost sensitive loss. We normalized both losses to 1 for the first epoch to be able to compare only the evolution of the losses.

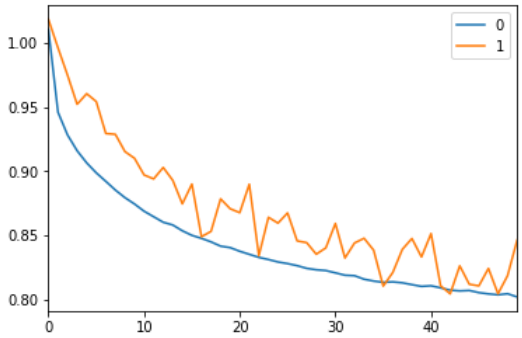


Figure 2: Normalized training/validation losses (blue/orange)

3 Backtesting

In addition to the metrics from validation, namely accuracy, we wanted to test our model based on the PnL that we would make if we follow our strategy. We considered the same rebalancing scheme as in training: 5 days and backtested the strategy over a following period of time. We considered reinvesting any benefit made.

The output of the model every week is a matrix of size Nx3 where N is the number of stocks. Indeed, every week we are given a probability of going 'long', 'short' and 'hold' for each stock in our universe.

At first for simplicity we only consider 'long' strategies where we only purchase a selected list of stocks and get returns. We also did the same for short and for a mixed backtest but results are quite similar anyway.

If our model was perfect we would just go long the stock with the highest probability of going 'long'. Actually, we want stability and as we have seen with accuracy the model is not perfect at all. At first we considered taking the 10, 20 "best" ranked stocks in term of probability with an equal weight at each period. Then we allowed customization on the weights and number of considered stocks. Now you can customize your strategy based on the output vector of probabilities from the model. The code is within the object *Backtester* and strategies are highly flexible.

The *Backtester* objects works with 2 inputs: the samples data (i.e. encoded images standing for time series data) and a tensorflow checkpoint file that is the trained model that we will use. Thus you can choose exactly which model to backtest and on which data.

GoogLeNet shows the best result compared to ResNet and AlexNet. Therefore, all of the following results are based on GoogLeNet. Besides, results are without cost-sensitive loss because it showed little improvement.

Our best result for 3 strategies are listed in Figure 3, the label 'Cash' stands for S&P500 that we consider as our benchmark. The Sharpe ratio of '20max' is 0.91, '10max' is 0.62, '2max' is 0.26. As expected the larger the number of stocks considered the smoother is the rolling returns curve. What is quite surprising is that with only 10 or 20 stocks variations are not really high, even similar to SP500 that boasts many more stocks.

A way of checking whether the backtester is producing good results is to consider "bins" (Figure 4). Instead of considering the N-best stocks at each re-balancing date, we split our stock universe into M subsets of stocks ranked by their probability of going long. Then we expect the first subset to perform really well and the last one to perform bad. Here are the results for GoogLeNet over the

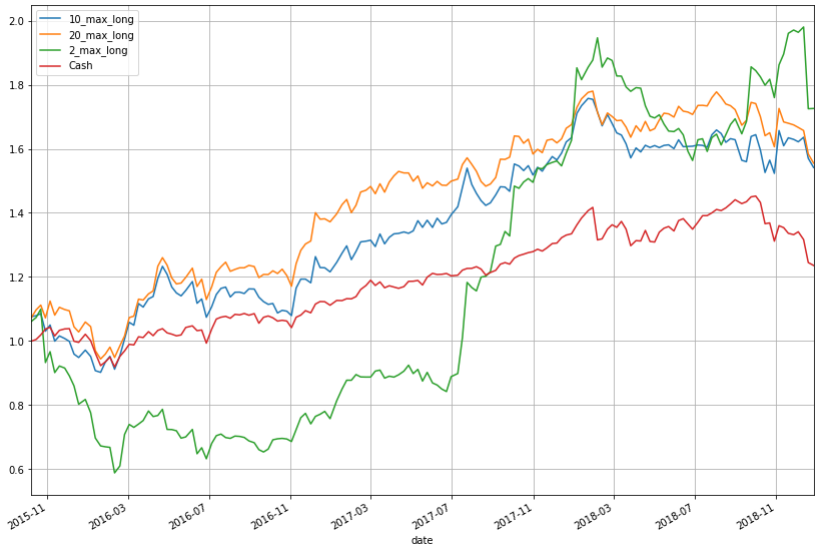


Figure 3: Long only strategies based on GoogLeNet

period 2015-2018. It is similar to what we expected, small ranked bins perform better. We can see that there is a huge spike in one of the bins. We checked in detail and this is not an error but the stock DRYS took 2000% in one week then returned to normal which causes the spike when going long that stock in one of our strategies.

Another thing to notice is that different bins may seem to have similar trends as S&P500, this is mainly due to two reasons: all strategies here are long only; a long time scale in a limited plot (If we use a wider plot then possibly we'll get a different visual effect). If we look into details, we can find many differences which finally lead different bins into different levels of PLs.

To analyze this plot with more than the eyes we plotted the average ranking of each of these bins over the time period (Figure 5). What we expect is a positive relationship between the best ranking and the best performing stocks. For example, if the n^{th} bin (decile) has a average rank of n then the strategy delivered by the model is stable and thus more confidence could be secured by its performance in the real market.

It is not a perfect straight line but we are quite satisfied with the result. The distinction of each bin's performance and the average ranking tell us that the model is working as expected.

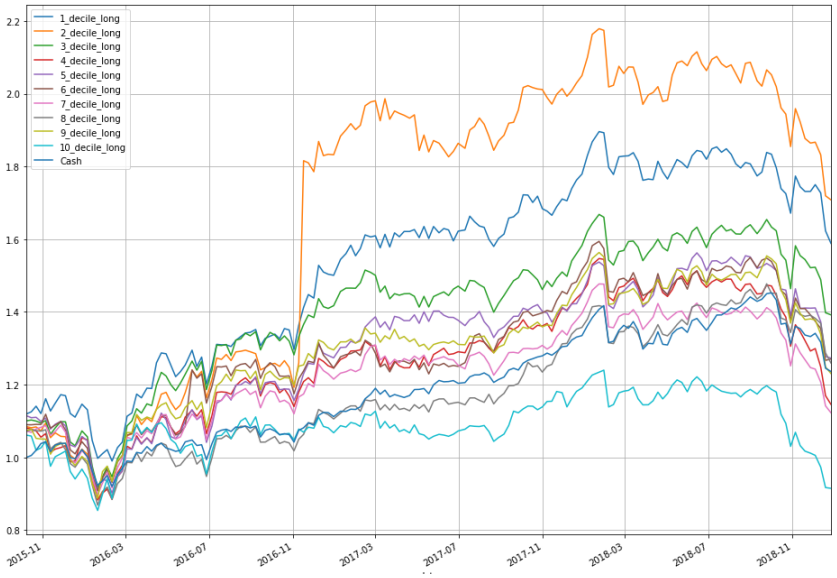


Figure 4: Stock bins based on GoogLeNet ranked by probabilities of going long

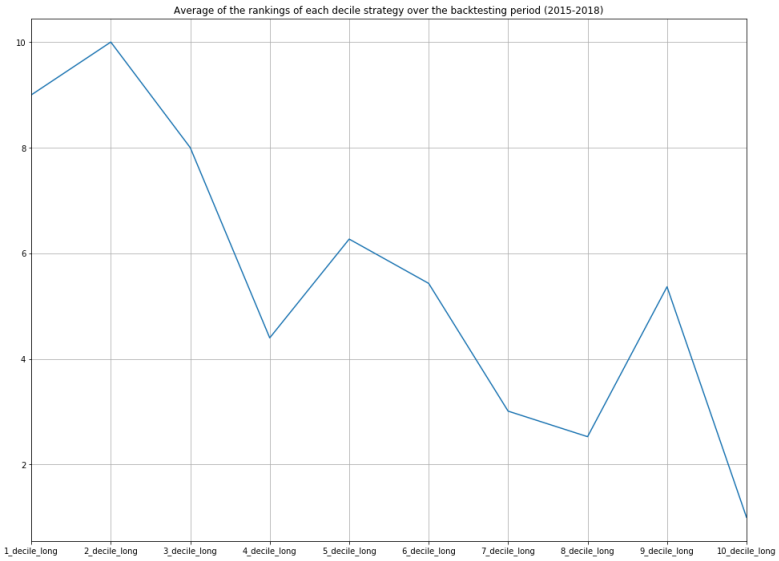


Figure 5: Average ranking for each stock bins

4 Conclusion

In this project, ResNet, AlexNet and GoogLeNet are applied in the stock selection process. GoogLeNet showcased a better predictive power than the other two models. With the analysis of stock bins performances and average ranking of stock bins, the prediction of GoogLeNet is proved to be somewhat stable rather than lucky guess.

Future improvements could be made by training data with a rolling window, so the model could be exposed to data more timely and capture the regime better. This may require expensive computation resources but could deliver a better result. Another improvement we came up with is that more meaning features could be used in this project. So far we only used very simple and easy features, which may also limit the information the model could learn from.

References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012.

[2] Christian Szegedy and Wei Liu and Yangqing Jia and Pierre Sermanet and Scott Reed and Dragomir Anguelov and Dumitru Erhan and Vincent Vanhoucke and Andrew Rabinovich. *Going Deeper with Convolutions*. Computer Vision and Pattern Recognition (CVPR), 2015.

[3] Ioffe, S. and Szegedy, C. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv preprint arXiv:1502.03167. 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[5] Khan, S., M. Hayat, M. Bennamoun, F. Sohel, and R. Togneri. *Cost-sensitive learning of deep feature representations from imbalanced data*. IEEE Transactions on Neural Networks and Learning Systems, 29 (8), 3573-3587, 2015.

[6] Krizhevsky, A., I. Sutskever, and G. Hinton. *ImageNet Classification with Deep Convo- lutional Neural Networks*. Advances in neural information processing systems, 1097-1105, 2012.

[7] Kukar M., and I. Kononenko. *Cost-Sensitive Learning with Neural Networks*. European Conference on Artificial Intelligence, 445-449, 1998.

[8] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovic. *Going Deeper with Convolutions*. Proceedings of the IEEE conference on com- puter vision and pattern recognition, 2015.

[9] Wang, Z., T. Oates. *Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks*. Trajectory-Based Behavior Analytics: Papers from the 2015 AAAI Workshop, 2015.

[10] Wang, Z., T. Oates. *Imaging Time-Series to Improve Classification and Imputation*. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015

A Appendix

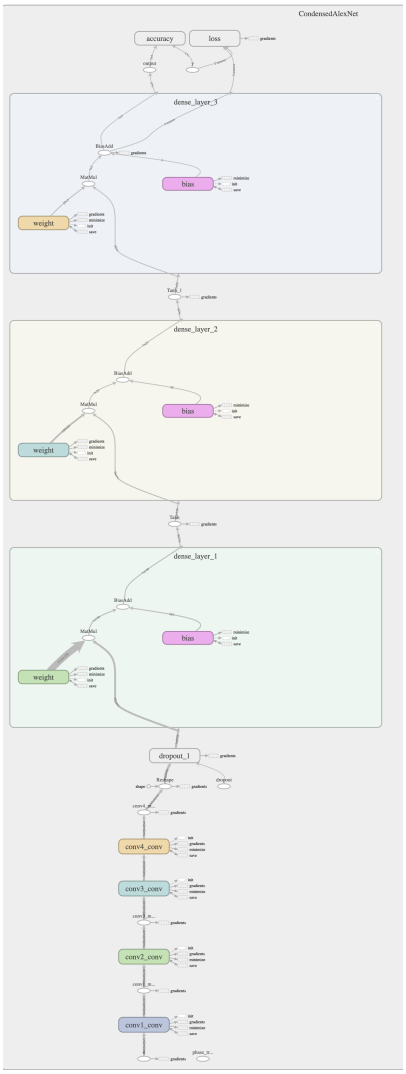


Figure 6: AlexNet architecture used

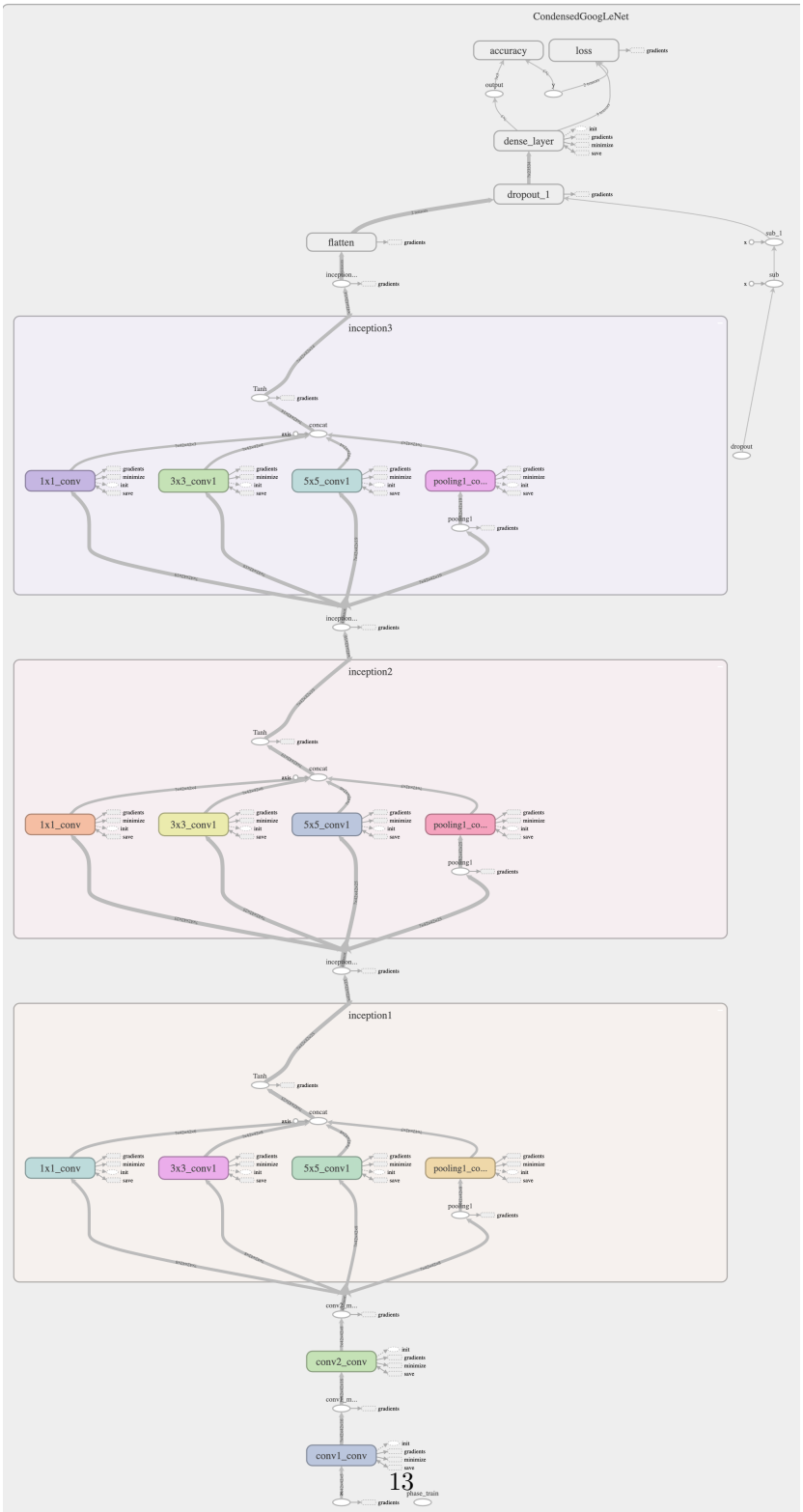


Figure 7: GoogLeNet architecture used

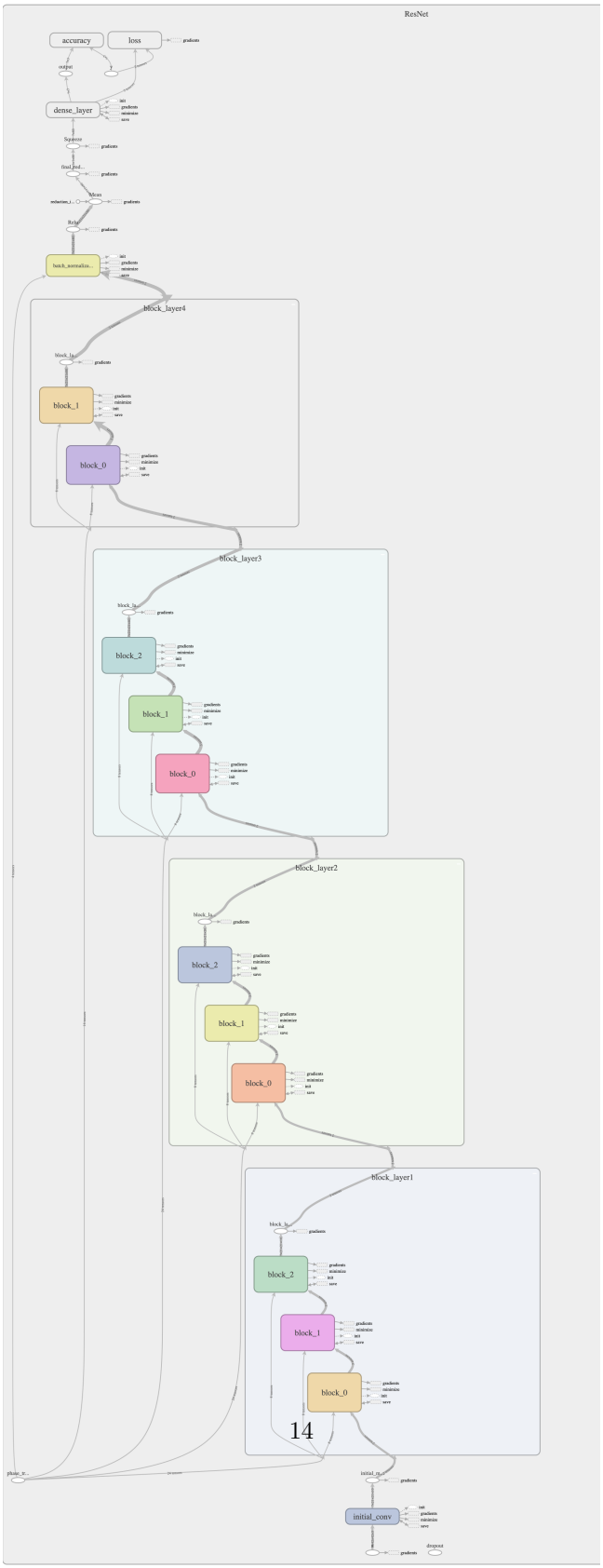


Figure 8: ResNet architecture used