Deep Portfolio Strategy with CNN

Dawei Zhang
UNI: dz2363

December 24, 2018

Abstract

In this paper, a novel way to implement portfolio selection is proposed. Unlike the classical Markowitz's mean variance tradeoff, we utilize deep learning to classify financial time series of each stock price into 3 classes, representing their level of future returns. Then, stocks with greatest probabilities of having gains in the future are long. To leverage the power of deep learning, especially with advancements in convolutional neural networks (CNNs), we propose to covert the multidimensional time series into multichannel images, which is suitable for CNN. CNNs, in specific AlexNet and GoogLeNet type networks are trained, validated and tested. Additionally, CNN equipped with MCMC scheme is examined as well. Finally, to facilitate a long-only portfolio strategy, we implement a cost-sensitive loss function so as to optimize the odds of succesfully predicting a long signal to a short one. Finally, the portfolio stratgy is backtested and compared against SP500 return. It demonstrates superior predictive power and substantially higher return as compared to a market portfolio.

1 Data

All the data are from Wharton's Research Database (section CRSP). The data are all stock price/volume data which are most complete throughout 1996 to 2017. After forward filling missing values, a dataset consisiting 1,350 stocks, with 5,539 trading days, is obtained. Selected features includes close, high, low and volume of each stock every day (4 features in total). So we are faced with a 1,350 4-dimensional financial with length 5,539 each. In a chronological order, data are partitioned into three parts: training set starts from 01/02/1996, valiation set starts from 01/01/2007, and test set starts from 10/08/2010 and ends at 12/29/2017. This partition results in a rough ratio of $3:1:2$ between training, validation and test sets. For both the concern of computational power, limitation of time and and storage, all stocks across time are pooled together to do the training and prediction, i.e. it is implicitly assumed that panel data is 'nicely' distributed to give predictions for the future. The next section is devoted to illustrating how we could convert multidimensional time-series into multichannel images for CNN learning.

2 Converting the Financial Time Series into Images

There are many ways to convert financial time series into images. Of all newly proposed methods, one can straightforwardly take the correlation matrix of a time series and make it into a image, since it is a matrix, and all the elements are within the range $[-1, 1]$, which makes it a natural form of an image. However, there are two obvious problems with correlation matrix. First of all, it is completely symmetrical, which makes it unable to provide richer information for CNNs to extract. Secondly, to capture the entire time-dependent feature of a time series with size n, one would inevitably have to calculate with a correlation matrix of size $n \times n$, which could be very computationally expensive.

As a remedy, one could resort to other methods. Two newly proposed methods are studied in Wang and Oates (2015).

1. **Graiman Angular Fields (GAF)**
   Under this method, the min-max normalized time series of length t are mapped to the polar coordinate

with the following operation:

$$\begin{cases} \phi = \arccos(x_i), & x_i \in \{x_1, x_2, ..., x_t\} \\ r = \frac{i}{t}, & i \in \{1, 2, ..., t\} \end{cases}$$

After this operation, Gramian Angular Summation Fields (GASF) and Gramian Angular Difference Fields (GADF) are defined respectively as:

$$GASF = [\cos(\phi_i + \phi_j)]_{t \times t}, \quad GADF = [\sin(\phi_i - \phi_j)]_{t \times t}$$

One can show that by defining inner product by $< x, y >= x \cdot y - \sqrt{1 - x^2}\sqrt{1 - y^2}$ and $< x, y >= \sqrt{1 - x^2}y - x\sqrt{1 - y^2}$, the above is very similar to Graham matrix, only in that the inner product does not possess linearity.

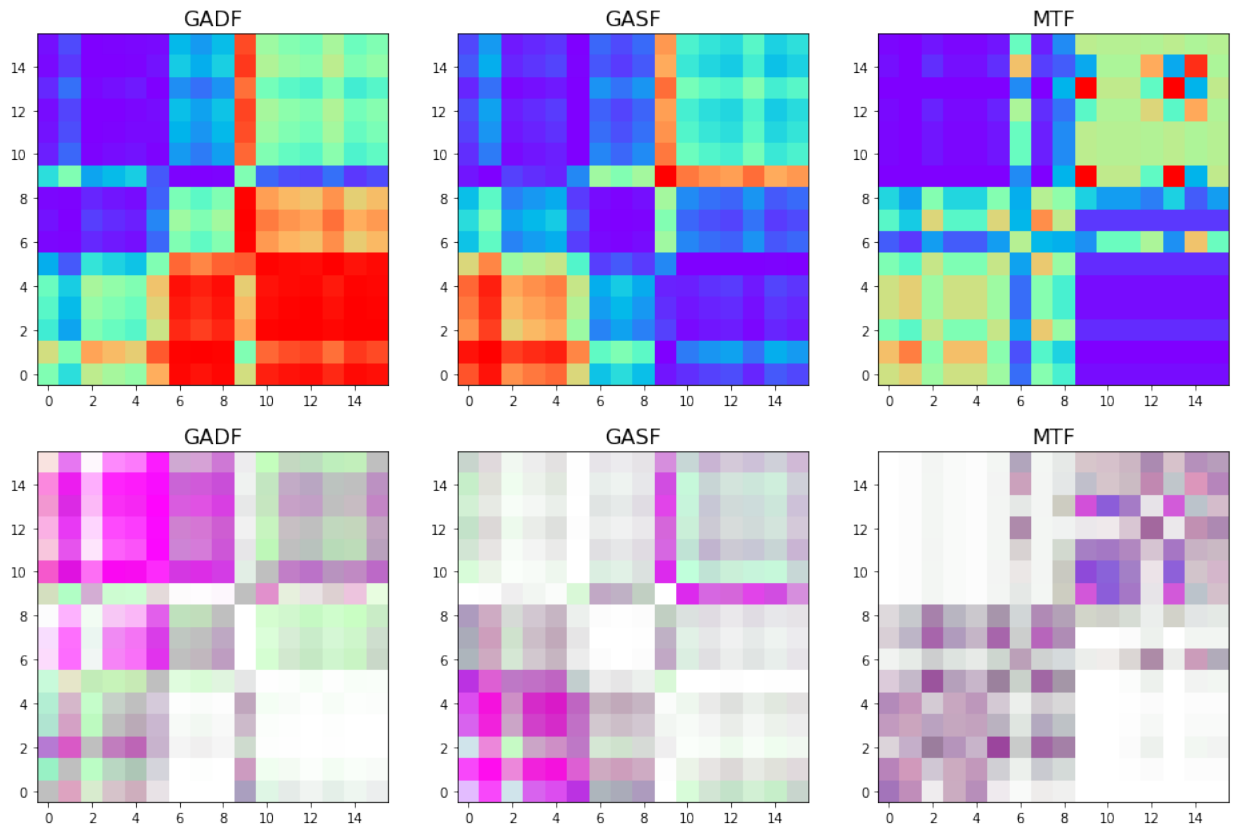
2. **Markov Transition Fields (MTF)**
In essence, MTF is an extended version of Markov transition matrix. Originally, Markov transition matrix (MTM) is defined to be the empirical probability transition matrix of the time series, assuming that we divide the time series into $Q-$quantile bins. Then, for a time series with length t, MTF is defined to be the $t \times t$ matrix with $(i, j)-$th entry to be the entry of the Markov transition matrix corresponding to the value of x_i transitioning to x_j. The following describes the difference of MTM and MTF, and how to go from MTM to MTF (the former is the $(i, j)-$th entry of MTM and the latter is the (k, l)-th entry of MTF):

$MTM =$ [freq from quantile i to quantile j]$_{Q \times Q} \Rightarrow MTF =$ [(i,j)-th entry of MTM given $x_k \in Q_i, x_l \in Q_j$]$_{t \times t}$

As described above, the images constructed from GAF or MTF are also of dimension $t \times t$. However, we often utilize the so-called piecewise aggregation approximation (PAA) technique described in Keogh and Pazzani (2000) to compress the dimension of the matrix. PAA could turn the time series into arbitrary length lower than the original one, while preserving the trend of the time series.

Empirically, in finance, one might argue that it is preferred to use samples created by GADF, as first of all GADF provides higher variability within one image, by noticing that its counterpart GASF is entirely symmetric. It is conceivable that, by having different signs across the diagonal elements, GADF is able to characterize the sequence by which each data point occurs, which is impossible for GASF. For MTF, one needs to do the quantile bins division manually, adding extra subjectivity into the whole process. Meanwhile, the nonstationarity of financial time series will give very sparse representation in MTF, namely a lot of entries of an MTF from financial time series could be 0, providing little information regarding the pattern of the data. As a result, only samples created from GADF are used throughout this study.

Nonetheless, one critical issue is not addressed in Wang and Oates (2015). They do not discuss how to create image representation for multi-dimensional time series. As a result, all their images created from time series are of single channel, which largely limits the predictive power of their model. Moreover, it is common practice to add extra features than the price series itself in financial studies, so here in this paper, a way to account for multiple features is proposed. The way to do is to stack different single channel images created from different feature time series. Then a multi-channel image is formed. By moving the filters of a CNN along the channel, we could hopefully capture the interactions of different features. An example of images as such, with comparison to former single channel images is given in below.



Top: 3 Types of single channel images converted from 64 days of trading data of one stock
Bottom: 3 Types of multichannel images converted from 64 days of trading data of one stock

With the above methodology, it is able to convert multidimensional time-series easily into multichannel images. Notice that we have selected 4 features for this study, the number of channel will be 4. This could be extended to arbitrary number once we incorporate more features. Typically, some hyperparameters are specified to do the conversion:

1. Length of data to convert one image
   Too long would make an image incorporate irrelevant information from the long past; too short would cause underfitting. In this paper, 64 days of data is used, which is roughly a quarter of data. We argue that this is appropriate, as regime switches across quarters when financial statements are released.

2. Pixel of the image
   An image of size $64 \times 64 \times 4$ is still potentially too large. As discussed, PAA could be applied to compress data information and make learning more effective. In this paper, we choose pixel of 16. This implies that we "average" 4-day of information of a stock to form a pixel.

3. Time interval between which to create an image
   This is equivalent to asking between how many days do we believe that new data becomes relevant to make predictions of future. Here, we set it as 5. Namely, we create a new image from past 64 days of data every 5 days. This is also the frequency we rebalance our portfolio.

Finally, we discuss the target variable. Generally, we would use chaging direction of close price to make target variables. Here, since we are rebalancing every 5 days, it is more appropriate to find a target variable that is better able to capture true price changes over a week. A naive use of close price of a week will be highly noisy. Consequently, we calcualte the volume-weighted average price (VWAP) of a stock each week, and use the VWAP return as our target variable. This is also a "smoothing" technique. To make our sample balanced,

we choose a partition interval $[-\infty, -0.0125], [-0.0125, 0.0125], [0.0125, +\infty]$ for the target variable. Namely,

$$Y = \begin{cases} 1, & weekly\ VWAP\ return\ > 0.0125 \\ 0, & weekly\ VWAP\ return\ \in [-0.0125, 0.0125] \\ -1, & weekly\ VWAP\ return < -0.0125 \end{cases}$$

The following table summarizes the distribution of the samples:

	$Y = 1$	$Y = 0$	$Y = -1$
Training set	277987	206914	254899
Validation set	96703	56356	93991
Test set	179225	158245	153929
Total	553915	421515	502819

3 AlexNet and GoogLeNet Experimental Results

3.1 AlexNet-like Architecture

AlexNet (Krizhevsky, Sutskever and Hinton (2012)) is one of the first proposed deep convolutional type neural networks. Specifically, it leverages the power of stacking multiple convolutional and max pooling layers combination to extract features, after which the images are fed into some fully connected layers. In this paper, original inputs are of dimension $224 \times 224 \times 3$, as opposed to our case with input dimension $16 \times 16 \times 4$. Therefore, instead of having a full version of AlexNet, which have 5 convolutional layers (interpolated with max pooling) and 3 dense layers, the following AlexNet-like architecture is proposed and tuned in this paper:

Layers	**input dimsion**	**output dimension**	**kernel / weight**	**# of parameters**
Conv1	(16, 16, 4)	(16, 16, 4)	(2, 2, 6)	102
MaxPooling1	(16, 16, 6)	(16, 16, 6)	(2, 2)	0
Conv2	(16, 16, 6)	(16, 16, 10)	(2, 2, 10)	250
MaxPooling2	(16, 16,10)	(16, 16, 10)	(2, 2)	0
Conv3	(16, 16, 10)	(16, 16, 8)	(2, 2, 8)	328
Conv4	(16, 16, 8)	(16, 16, 4)	(2, 2, 4)	132
MaxPooling3	(16, 16, 4)	(15, 15, 4)	(2, 2)	0
Flatten + Dropout (15%)	(15, 15, 4)	900	-	0
FullyConnected1	900	150	(150, 900)	135,150
FullyConnected2	150	25	(25, 150)	3,775
Softmax output	25	3	(3, 25)	78
Total	-	-	-	139,815

If otherwise specified, all the activation functions used in this research are $tanh(\cdot)$, which, with validation, has shown best predictive power. Indeed, we have tailored the network to our use in several ways. First, instead of having really large strides and kernel size as in the original AlexNet, our kernel size and strides are small, since we have a smaller-pixel sample, and also more compressed information in each pixel. To prevent further information loss, most of the layers are padded with zeros. We make training and validation on the training and validation sets, and obtain the following experimental results for the "condensed" AlexNet: validation loss for AlexNet is 1.0482 and validation accuracy is 0.4521, while training loss and training accuracy are 1.0327 and 0.4680, respectively.

3.2 GoogLeNet-like Architecture

A more recent CNN architecture, called GoogLeNet (Szegedy et. al. (2015)), which is consisted of a so-called inception module, is considered another landmark of deep learning. The basic unit of GoogLeNet, inception

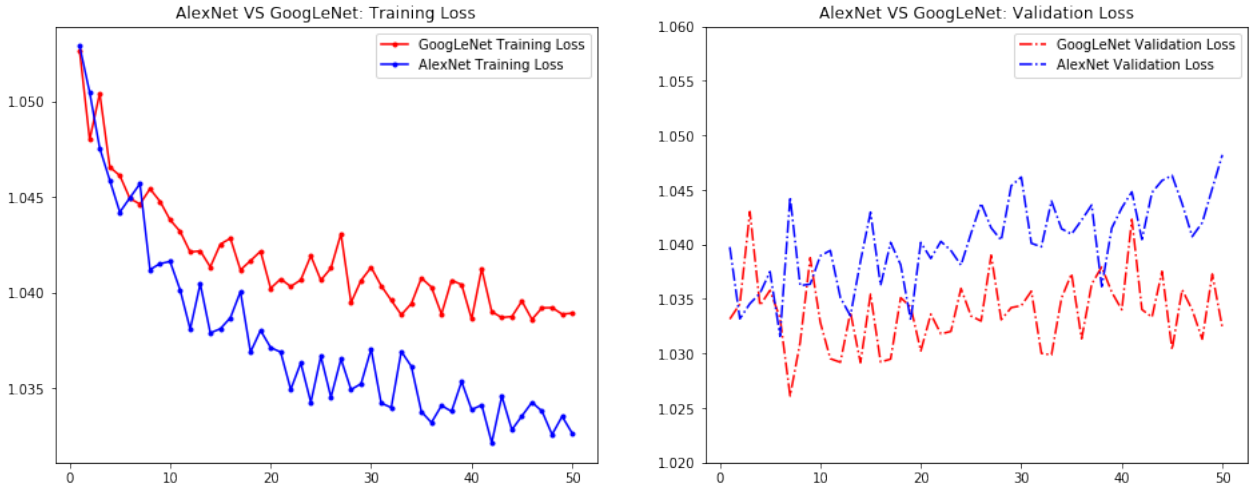
module, makes use of stacking of different convlution (with different size of kernels) of the same input. The intuition behind this module is that different size of kernel provides different level of extraction of information, and a simple concatenation will help extract hidden information on a even deeper level.

As before, since the original GoogLeNet has 22 layers, which is too large for our problem, we have to create a condensed version of it. The following table summarizes the architecture of condensed GoogLeNet used in this study. Refer to the code for more detailed definition of an inception module, and the tuned hyperparameters for each inception module.

Layers	**input dimsion**	**output dimension**	**kernel / weight**	**# of parameters**
Conv1	(16, 16, 4)	(16, 16, 16)	(2, 2, 16)	272
MaxPooling1	(16, 16, 16)	(16, 16, 16)	(2, 2)	0
Conv2	(16, 16, 16)	(16, 16, 8)	(2, 2, 8)	520
MaxPooling2	(16, 16, 8)	(16, 16, 8)	(2, 2)	0
Inception1	(16, 16, 8)	(16, 16, 25)	-	2,029
MaxPooling3	(16, 16, 25)	(16, 16, 25)	(2, 2)	0
Inception2	(16, 16, 25)	(16, 16, 19)	-	1,818
MaxPooling4	(16, 16, 19)	(16, 16, 19)	(2, 2)	0
Inception3	(16, 16, 19)	(16, 16, 14)	-	872
AvePooling1	(16, 16, 14)	(15, 15, 14)	(2, 2)	0
Flatten + Dropout (10%)	(15, 15, 14)	3,150	-	
Softmax output	3,150	3	(3, 3,150)	9,453
Total	-	-	-	14,964

Similar to AlexNet training and validation, we obtained the following experimental results: validation loss and validation accuracy are 1.0324 and 0.4736, while training loss and accuracy are 1.0389 and 0.4657.

To facilitate validation, the following training graphs are presented:



Left: Training Loss Comparison; Right: Validation Loss Comparison

It is easily seen, using the validation result, GoogLeNet architecture outperforms AlexNet in our classification problem, and this is achieved with much less parameters. Henceforth, we consider exclusively the application of GoogLeNet.

3.3 Variational Autoencoder with AlexNet Structure

In this research, a VAE with AlexNet-like archicture on both the encoder and decoder sides are proposed and tested. Then, the hidden units are taken to train a multilayer dense network. However, the training and validation results show very strong overfitting and can hardly be alleviated, the outcomes will not be showed herein. However, codes are incorporated in the GitHub.

4 Bayesian GoogLeNet: An Easy MCMC Approach

Neural network is notoriously known to be volatile in making prediction. This volatility arises from the fact that training procedure tends to overfit in-sample data, and nonconvexity of the optimization problem combined with the random initialization makes the final weights unequal, or even vary greatly, over different training paths. To overcome this issue, one might easily come up with an ensemble schedule, by averaging prediction of N independently trained neural nets of same architecture. Nonetheless, it is often observed that the weights/validation prediction results bumps within certain range during final epochs. This leads to the advancement of Bayesian neural networks proposed and implemented by Neal and Zhang (2006).

To see how a straightforward MCMC can be done, we assume that there is indeed a prior for the weights of a neural network. Then, as batches come in and optimization proceeds, the network updates its weights and reaches stationarity after E large enough number of epochs. As it reaches stationarity, we could sample from these weights and thus acquire a bayesian learning of neural networks. The posterior gives rise to more stable prediction of out-of-sample data. To implement the idea, one simply samples weights from last e epochs of training result, and average the prediction over them. To see that it is truly effective, we could compare the 1-st and 2-nd order moments of the training set prediction results of : 1) last e epochs of training; 2) e independently trained networks. Since training AlexNet in our study is more time-saving, we only illustrate this result with AlexNet. See the following tables for a comparison of the statistics.

Statistics	**Ensemble AlexNet**	**Bayesian AlexNet**
Mean Training Accuracy	1.037	1.033
95% Confidence Interval	[1.035, 1.039]	[1.033, 1.034]

It is seen that the result is pretty closed, and Bayesian network provides an even tighter CI for training accuracy, indicating a better convergence.

5 Cost-Sensitive Cross Entropy Loss

Proposed by Khan et. al. (2015), applying cost sensitive loss function is a way to train the neural networks to understand how to use expected penalty to balance sample size. Simply put, if the training samples have class distribution $\pi(\mathbf{y}) = (\pi_1(1), ..., \pi_m(m))'$, which may not balanced, the neural network will be trained to be biased towards more abundant classes. To modify it, one must penalize misclassifying the more rare classes more. To this end, we first define a cost matrix $C_{m\times m}$ where m is the number of classes. The $(i, j)-$th entry of cost matrix is the factor by which one wants penalize classifying class i into j. Then, an expected penalty factor of misclassifying class i is obtained:

$$\bar{F}(i) = (C\pi(\mathbf{y}))_i/(1 - \pi(i))$$

In this paper, a simpler version of calculating the modified cross entropy $\tilde{E}$ with the above penalty factor is implemented (denote the original cross entropy $E(\cdot, \cdot)$, where the two arguments are true target and predicted target, respectively, in one-hot encoded form):

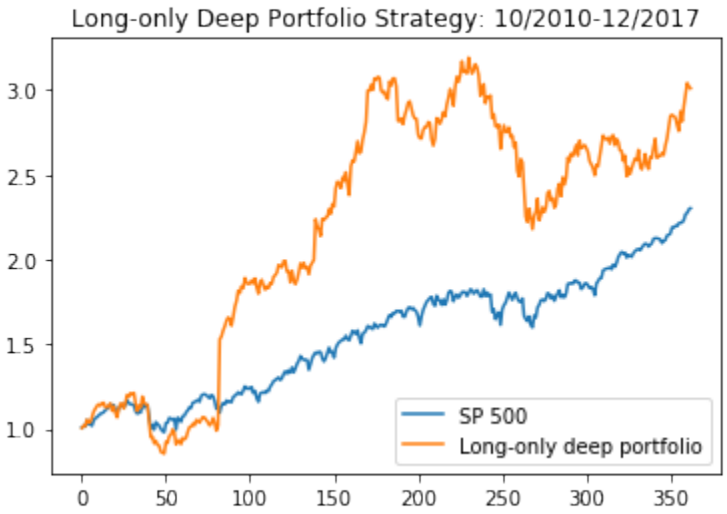
$$\tilde{E} = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{y}_i, \bar{F}(\mathbf{y})'\hat{\mathbf{y}}_i/(\mathbf{1}'\bar{F}(\mathbf{y})))$$

Although originally proposed to penalize misclassification of more rare class, in this study, we could apply it to facilitate long-only strategy. Because in the proposed strategy, class 0 will give an action of holding,

instead of taking any position, this would not result in any changes in PnL. On the other hand, when we are doing long-only trading, the only way that would hurt the PnL is to long a stock that actually drifts downward. Therefore, we reversely use this idea, by making $(1,3)$-th and $(3,1)$-th entry larger to penalize more the misclassification of 1 to 3 or vice versa. In other words, this would make our prediction more conservative, since we would make the network more reluctant to give prediction to class 1 and 3. But this makes sense in finance, since we will not be holding a portfolio so large that it incorporates all stocks that should be long or shorted, only stocks about whose moving directions we are very confident are added to our portfolio. Alternatively, this loss function enables us to achieve optimization of the odds of successfully predicting a long signal to a short one. To illustrate the result, I train both a GoogLeNet with and without the cost sensitive loss function, and then compare their confusion matrix. Then, I compute the ratio between the number of cases that is predicted to be a long signal actually is a long signal, and the number of cases that is predicted to be long but actually is a short, for both networks, and obtain that, for the network trained without cost sensitive loss, this odds ratio is 1.94, while for the one trained with cost sensitive loss, this odds becomes 2.12. The latter is a more optimal result if we want to implement a long-only portfolio strategy.

6 Strategy Backtest

Finally, I apply the Bayesian GoogLeNet model trained with cost sensitive loss function to do long-only portfolio selection on the test data. The strategy is straightforward. At the date of prediction, we acquire the output probability distribution of VWAP return of all stocks, and then we select top-15 long stocks (stocks with greatest probability of achieving return over 1.25%) to form a long-only portfolio. Using the close price next week to close the portfolio and close price of previous week to open the portfolio, we acquire the following PnL result, where S&P returns are accompanied. This shows that the model indeed has superior predictive power for top-15 stocks long-only portfolio.



7 Concluding Remarks

In this paper, I explored the possibility of applying CNNs on multidimensional financial time series classification. GADF is applied to convert multidimensional time series into multichannel images, which is then fed to AlexNet and GoogLeNet. It is found that GoogLeNet demonstrates stronger predictability on our data than AlexNet, with much less parameters to be fitted. Also, leveraging the power of Bayesian learning and cost sensitive loss function, we optimize the odds between a signal predicted to be long actually is long to the signal predicted to be long but actually is a short. Based on the Bayesian GoogLeNet with cost sensitive loss, I formulate a simple long-only deep portfolio strategy, which shows extra profitability, thus

extra predictability, than the market portfolio.

Reference

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

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

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

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

[5] **Wang, Z., T. Oates**. 2015. 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.*

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