Assignment 4:

Graphically Encoded Time-Series & Convolutional Neural Networks for Classification

Jay D. Vaidya

School of Professional Studies, Northwestern University

MSDS 458: Deep Learning and Artificial Intelligence

Dr. Syamala Srinivasan

December 6, 2021

Abstract

The application of Convolutional Neural Networks (CNNs) to the classification of graphically encoded time-series images is a novel approach attempting to overcome some of the most well-known issues in time-series analysis, with respect to analyses undertaken using deep-learning (DL) architectures. The use of more standard time-series model architectures, such as Recurrent Neural Networks and 1-Dimensional Convolutional Neural Networks, can frequently suffer from data leakage and the diminishing ability to transport critical information regarding long-ranging dependencies to current time-steps for the model to analyze. By encoding images using Gramian Angular Fields, a method by which temporal dependencies are preserved via a polar coordinate system, and using CNNs as the model architecture, long-range dependencies can be more accurately preserved and accounted for and patterns can be detected that are both translation and time-invariant, providing for more robust classification opportunities.

Keywords: Time-Series Analysis, Deep Learning, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Gramian Angular Field (GAF)

Assignment 4: Classification of Graphically Encoded Time-Series Using CNNs

Introduction

The forthcoming report has been developed at the request of XYZ Asset Management Corporation, who has engaged my services for the purpose of determining if deep-learning models developed to scrutinize historical time-series data on liquid equity securities can be used to forecast whether the next day's return – i.e., percentage change of tomorrow's closing price as compared to today's closing price – on a given security will be positive or negative. In the case of a forecast of a positive return, XYZ Asset Management Corporation would direct portfolio managers to go 'LONG' and purchase a stock at the market open the next day, selling at the close of the day's trading; in the case of a forecast of a negative return, XYZ Asset Management Corporation would direct portfolio managers to sell 'SHORT' (i.e. borrow a security, sell it in the open market and then settle the claim by later purchasing the security – hopefully for a lower price – and returning it to the original lender, thereby pocketing the difference) and close a trade by the end of the day. Reliable trading histories and participant returns are notoriously difficult to ascertain, but XYZ Asset Management estimates that previously published SEC estimates indicating that up to 70 percent of traders lose money on a quarterly basis are reflective of its' trader's performance as well (Securities and Exchange Commission, 2011).

XYZ Asset Management Corporation has attempted, unsuccessfully, to use Deep Neural Networks (DNNs), Convolutional 1-Dimensional (Conv-1D) and Recurrent Neural Networks (RNNs) for this classification task in prior work, but to no avail. My services have been procured to apply graphical encoding to the time-series data and then build Convolutional Neural Networks to analyze the three-color-channel images generated to determine if accurate classification is possible, and to what extent. My work, as will be detailed in the following report, indicates that accurate classification using ensembled CNN models is possible, but individual model performance is unreliable, at best. Rather, ensembles of trained models, trained using randomly generated starting weights, provide a more balanced decision space that can be used for applications by XYZ Asset Management Corporation.

Literature Review

Beginning with the inception of centralized stock exchanges – the first, the Amsterdam Stock Exchange, created in 1602 by the Dutch East India Company – securities traders have sought to find discernible patterns in securities trading that could be exploited at a profit (Lodewijk, 2014). The first systematic iteration of this desire took the form of technical, or chart, analysis, wherein traders plotted historical price and volume statistics, at chosen time intervals, in hopes of finding repeating patterns; once patterns were believed to have been identified, chartists – as they became known – would attempt to identify the first stages of a newly unfolding pre-specified pattern and trade accordingly (i.e. go long, or purchase a security, if it was set to rise, or go short, or sell a borrowed security, if it was set to fall). Though the practice has persisted, and no doubt evolved, the basics of charting remain unchanged and, largely, remain human-based.

The advent of machine learning algorithms capable of parsing vast sums of time-series data, however, has generated a step-change in the sophistication and potential applications of technical, or chart-analysis, with respect to securities trading. One of the foremost authorities on financial machine learning, Marcos Lopez De Prado, said the following on the topic, “The algorithmization of finance is unstoppable… Because the next wave of automation does not involve following rules, but making judgement calls. As emotional beings, subject to fears, hopes, and agendas, humans are not particularly goo at making fact-based decisions, particularly when those decisions involve conflicts of interest” (Lopez De Prado, 2018).

One such application of algorithmic trading, or machine learning (ML), to the realm of finance and technical analysis is the utilization of convolutional neural networks to scrutinize time-series data converted to Gramian Angular Fields (GAFs), first put forward by Zhiguang Wang and Tim Oates of Maryland University (2015). According to the authors, in a GAF time-series, data is organized into a polar coordinate system, as opposed to a Cartesian coordinate system, ensuring that the temporal structure of the data is kept intact. Specifically, GAFs are created by first scaling all of the time-series data being scrutinized between negative one and positive one, and then encoding each observation as an angular

cosine with the timestamp of a given observation acting as the radius in the calculation. According to the authors, “As time increases, corresponding values warp among the different angular points on the spanning circles, like water rippling.” After generating the polar coordinate system, the GAF method exploits the angular nature of the system by using the trigonometric sum between each point in the matrix to identify temporal correlations at different time intervals, as dictated by the input data (See Appendix A for Details). The end result is an *n x n* matrix (where *n* is the number of observation in the series) that has preserved temporal relations between time-series data and can now be used for graphical encoding and thus the application of CNNs (Wang & Oates, 2015).

The work of Wang and Oates was pushed further in 2020 by Barra, Carta, Corriga, Podda and Recupero in a piece titled *Deep Learning and Time Series-to-Image Encoding for Financial Forecasting*. The researchers posited that by generating graphically encoded RGB color-mapped images of GAFs (i.e. *n x n* matrices) at varied time-scales (i.e. minutes, hours, days) and creating one composite image containing multiple time-scales, for each chosen ‘step’ through the time-series, could enable better deep-learning pattern recognition via CNNs, as patterns that manifest at varied time scales may be better discerned (2020). The author’s found that their approach demonstrated clear superiority over Conv-1D networks and was able to clear the threshold of 50 accuracy that is typical of many marginal market-predictive algorithms, due to the high variability of securities prices that make difficult “BUY” and “SELL” forecasting. The remainder of this report builds on the work of Barra et al. and examines the feasibility of their approach using real-word intraday tick data on the S&P 500 value index.

Importantly, Johann Faouzi and Hicham Janati published in 2020, in the Journal of Machine Learning Research, an open-source Python package known as ‘pyts’ (or, Python Time Series), a package specifically designed to assist in time-series related research (2020). One of classes of data manipulation provided by ‘pyts’ is the production of Gramian Angular Fields; the ‘pyts’ package, and GAF class, are used in this project to explore securities time-series data.

Methodology: Data Preparation, Exploration, Visualization & Analysis

The methodological approach undertaken in this project was that of using the intraday tick data for the Standard and Poor's 500 Value Index (ticker: IVE), spanning from beginning of pre-trading (9:00am EST) on September 28, 2009 to November 8, 2021 at the close of the trading day (4:00pm EST), to produce Gramian Angular Field (GAF) matrices that were then encoded into RGB images and fed into a series of Artificial Neural Networks to determine the feasibility of predicting, and classifying, the next trading day's return – positive or negative. The ultimate goal of this project is to identify an approach and ANN architecture to recommend to XYZ Asset Management Corporation for use in predicting the class – i.e. positive or negative – of the next trading day's security returns on the S&P 500 Value Index.

The IVE intraday tick datatset is comprised of 'ticks', each of which represent a trade executed on a registered U.S. stock exchange; additionally, each observation, or 'tick', in the dataset contains six pieces of information: date, price, time, bid, ask and volume. For the purposes of this study, only date, price, and time were required. Prior to cleaning, the intraday tick dataset was comprised of 9,314,767 'ticks' over 4,425 total days; only 3,161 of the 4,425 total days, however, were trading days (i.e. weekdays Monday through Friday), for an average of just under 2,964 'ticks' per each trading day. The daily volume of shares traded, which correlates highly with number of ticks per day, shows significant variation at the tick, daily, weekly, monthly and even yearly levels (as can be seen below).

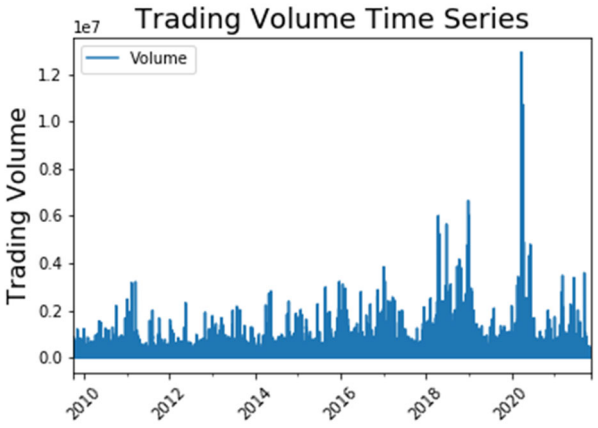

Fig 1. Volume Plot - Number of Shares Traded Each Trading Day

To ensure the highest level of integrity of the results of the foregoing analyses, the cleaning of the dataset, prior to data processing required to generate the GAFs, first required that all trading data that did not fall within normal trading hours and normal public trading days was discarded from the dataset. This was accomplished by eliminating all days outside of Monday thru Friday from the dataset; only trades occurring between 9:00am EST (which includes 30 minutes of pre-opening trading prior to 9:30am EST market openings) and 4:00pm EST were retained for usage; all public holidays, which coincide with bank and therefore market holidays, were also discarded; finally, the 'tick' data was grouped into one-hour intervals to permit for incremental, time-based analyses. What remained were publicly available trading information with reasonable daily volumes and therefore a lower likelihood of detecting anomalous feature correlations that would provide spurious or misleading results. This cleaned dataset contained information on 2,512 valid trading days, each of which contained 8 hourly (9am to 4pm, inclusive) observations, for a total of 20,096 remaining data points.

The next step in preparing the data for use in this project required that I select the time intervals at which the data would be scrutinized – i.e., the intervals at which the GAFs would be generated. Four intervals were chosen – 1hr, 2hr, 4hr & 1day – in order to permit for the production of balanced, square matrices comprised of 4 composite RGB images of GAFs, one produced at each distinct time interval, for a given trading day. For each chosen interval, a loop was created to calculate the 20 previous time-steps in of the same interval length (i.e. 1hr, 2hr, 4hr & 1day), which were then used to produce a GAF; the beginning period for each time-step departure is the ending of a given trading day, ensuring the each GAF has the same point of departure, or in other words that no data leakage from data points in the future was introduced into the analysis. The resulting 4 GAFs produced for each trading day in the dataset were then stitched into a single image and saved in either the 'LONG' or 'SHORT' local filepaths for later applications using the ANNs. Examples of the individual and stitched GAFs can be seen below.

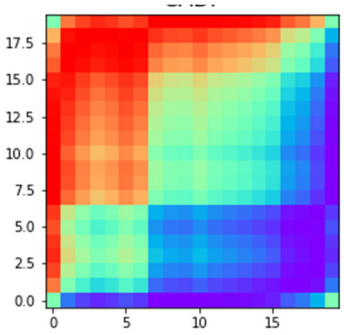

Fig 2. Sample GAF image in a 20x20 RGB image Matrix at the 1hr interval

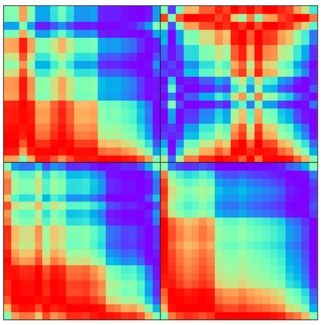

Fig 3. Sample stitched GAF image, each image in a 20x20 image Matrix at the 1hr, 2hr, 4hr and 1 day intervals

After producing the stitched GAF images and saving them to the respective 'LONG' and 'SHORT' filepaths, as determined by the next day's return (i.e., 'LONG' if the next day return was positive, and 'SHORT' if the next day return was negative), I produced a dataframe that contained the filepath for each image along with it's respective class, 'LONG' or 'SHORT'. The dataframe was used to create the train, validation and test datasets for this project. Approximately 10 percent of the data, or 240 images, was retained for testing, while the remaining 90 percent was split into training and validation sets; this was accomplished by withholding 15 percent of this dataset (which equates to 13.5 percent of the original total dataset) for validation purposes. The number of observations in each set are summarized below. Of note, the dataset is slightly unbalanced – for the full datatset, 51.6 percent of the data belongs to the 'LONG' class:

```
Full Dataset Value Counts:
 LONG     1297
SHORT    1215
Name: Labels, dtype: int64

Training Value Counts:
 LONG     1168
SHORT    1104
Name: Labels, dtype: int64

Testing Value Counts:
 LONG     129
SHORT    111
Name: Labels, dtype: int64
```

Fig 4. Value counts for the full, training and test datasets for each Class of data

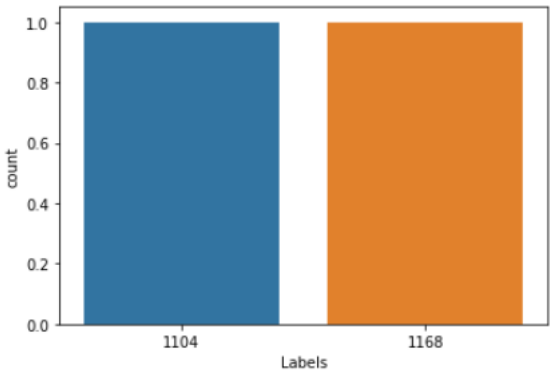

Fig 5. Value plot of training data by class; the dataset is slightly unbalanced. 'LONG' represents 51.4% of observations.

The final step in the data preparation process was to dictate a size of the images, rescale the pixel values to fall between 0 and 1, and to bifurcate the data into a 3-channel dataset that could be applied to CNNs and a flattened version of this same dataset that could be fed into DNNs. Given that each Gram matrix produced was a 20x20 matrix, and four were stitched into each image, the natural dimensions of 40x40x3 were retained for the image sizes, to avoid unnecessary distortions. The flattened data to be used in the DNNs resulted in an array of 4,800 pixel values (i.e., 40 pixels by 40 pixels by 3 channels = 4,800).

Research Design & Modeling Methods

The research design for this project entailed the production of multiple iterations of both DNN and CNN models in order to develop an understanding of the benefits and limitations of each method, as well as to determine the best architecture from which to achieve XYZ Asset Management Corporation's stated goal, with respect to the classification of the next day's return for a given security to inform portfolio managers' trading decisions. The classification task at hand is that of binary classification, as the next day's trading action results in either a positive or negative return on a given security. Both DNN and CNN model architectures were first tested without regularization methods, before regularization was applied to subsequent model architectures, so as to determine the appropriateness and impact of various regularization techniques, such as using dropout between NN layers. Where kernel initialization is not being tuned as a hyperparameter, the 'he_normal' initializer is used to permit for better comparison across model architectures.

Given the slightly unbalanced nature of the dataset, wherein the 'LONG' class represents approximately 51.6 percent of the data points, each model tested in this project included a "class_weight" object inserted into the model 'fit' procedure. By generating a dict with the class weights included, and passing it into the Keras model 'fit' procedure, I effectively weighted the loss calculation for each epoch of the models produced, ensuring that loss contributions from the less-prevalent class (i.e., 'SHORT') received a greater weight than those of the more prevalent class. In other words, the model rewarded the identification of the 'SHORT' class more richly than that of the 'LONG' class.

For each of the primary model architecture types (DNN & CNN), I engaged in hyperparameter tuning over a broad feature space in order to determine the efficacy of a variety of model architectures. For the DNN models, the basic structure I tested for hyperparameter tuning was that of a two-layer DNN with regularization applied. The model input shape was that of the 4,800 pixel array detailed in the previous section; dropout was used after both the first and second dense layers; the second dense layer in the DNN had half the number of nodes as the first; the kernel initializer, optimizer and learning rate were varied, as will be detailed below; and the final dense layer was fed into a single-node sigmoid output layer for binary classification at the end.

For the DNN hyperparameter tuning, I tested the following feature space: number of nodes were tested at the levels of 10, 16, 32 and 64, with each second dense layer in the models having half these amounts – higher levels were not tested as earlier tests demonstrated that model capacity was sufficient to produce overfitting at these levels; dropout was tested at the 0.4 and 0.5 levels; kernel initializers 'he_normal' and 'he_uniform' were both tested; learning rate was tested at four levels from 1e-4 to 1e-7 – earlier tests showed high learning rates led to erratic results; two optimizers were tested, 'SGD' and 'Adam'. The total number of DNN models tested in this tuning analysis was 128.

As with the DNN models, before engaging in hyperparameter tuning on the CNN models, I first produced unregularized models to determine a baseline of performance. The CNN models used two convolutional layers, each followed by a max-pooling layer. Stride sizes of 1x1 were used along with a kernel size of 3x3; the input images fed into the model were of the size of a 40x40 pixels and 3 color channels (40x40x3). The results from the CNN were then flattened and fed into a dense layer before being passed to a single node sigmoid output layer for classification.

For the two-convolutional layer CNN architecture hyperparameter tuning, I tested the following feature space: the dense output layer before the classification layer contained either 256 or 512 nodes; the first convolutional layer contained 16, 32, or 64 filters and the second convolutional layer doubled that of the first; six different kernel initializers were tested, the normal and uniform versions of the LeCun,

Glorot and He initializers; learning rates were tested at the 1e-2 to 1e-4 levels. In total, 432 CNN architectures were tested in this hyperparameter tuning step.

As will be detailed in the results section of this paper, one of the key findings from the hyperparameter tuning of DNN and CNN models was the sensitivity of model accuracy on unseen test-data to the weight initializations of a given model. For this reason, the final methodological undertaking for this project was the production of an ensemble model that would produce a series of CNN's whose results could be pooled to make ensembled predictions. In this manner, an ensemble could help overcome the limitations inherent in the sensitivity of a given model's outputs to its weight initializations.

Results & Interpretation

The results of the time-series DNN and CNN classification experiments undertaken in this project demonstrate three important findings: first, model capacity achieved with two hidden layers is sufficient to map the complexity of the input feature space, and overfitting can, and does, occur quickly; second, model accuracy and usefulness is highly impacted by the initialization weights for a given model iteration; and third, CNN models demonstrate superiority with respect to avoiding immediate overfitting, and ensembles of CNN models are far more useful for classification than any individual model with which I experimented. The experimental results indicate that the capacity of DNN and CNN models to fit training data can be achieved with relatively little computational expenditure; unregularized models do begin overfitting the training data and learning extraneous, idiosyncratic patterns, but that process can be partly mitigated by early-stopping, which is undertaken as training set validation-loss (the measure by which models are optimized) plateaus, as well as by ensembling multiple model iterations to help dampen the impact of extraneous feature learning.

Among the most important items I learned quickly, through model iteration, is that both CNN and DNN models will rapidly converge on the local optima of predicting all classes as 'LONG' if regularization and early stopping are not used. Furthermore, even when regularization and early stopping are used, non-optimal local optima can still be settled upon. In these instances, model accuracy after fitting is reported as 54.12 percent accuracy on the validation accuracy and 53.75 percent accuracy on the

test accuracy: these two figures reflect the proportion of 'LONG' classifications for each respective dataset, indicating that the model converged on the *lazy* answer of assuming the most likely class – this occurs despite my use of class weights to balance the loss calculations by over-emphasizing the loss attributed to incorrectly identifying 'SHORT' predictions. This was confirmed by plotting the confusion matrix for the one such result (see below).

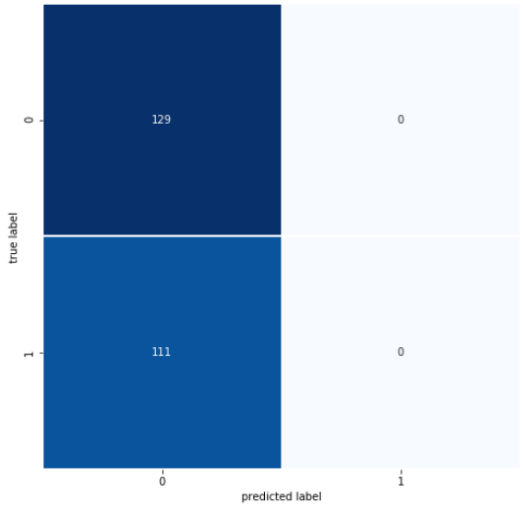

Figure 6. Confusion Matrix Result – Non-optimal Local Optima Reached Wherein All Classes Predicted As '0' or 'LONG'

The single best individual model result, providing a test set accuracy of 58.75%, was achieved using a two-layer CNN with max-pooling after each layer, followed by a dropout layer of 0.4. For each convolutional layer the kernel size was 3x3 with 1x1 strides; max-pooling was 2x2 with strides of 2. The first convolutional layer contained 32 filters and the second was doubled to 64 filters. Following the second max-pooling layer, the outputs were flattened and fed directly into a one-node binary classification layer using the sigmoid activation function. Important to note, however, despite the high level of accuracy (as compared to a 50 percent baseline, or approximate 54 percent that can be achieved by predicting all observations as 'LONG'), the model performance peaked after the first epoch, and deteriorated thereafter, indicating that weight initialization can *significantly* impact overall performance. Additionally, in reviewing the confusion matrix of the results, it can be noted that the distribution between predicted 'LONG' and 'SHORT' was imbalanced in the opposite direction of the training dataset

– i.e., 49.2 percent of test dataset predictions were ‘LONG’ and the remainder were ‘SHORT’, a somewhat intuitively unsatisfactory result.

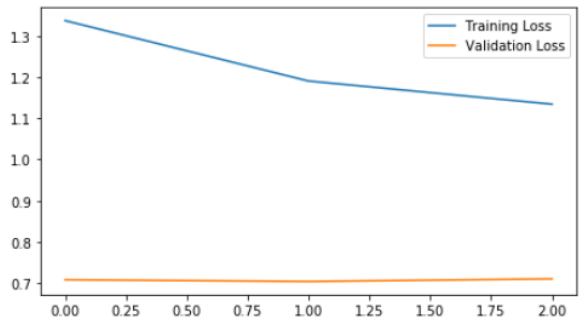

Figure 7. ‘Best’ Model Training & Validation Loss

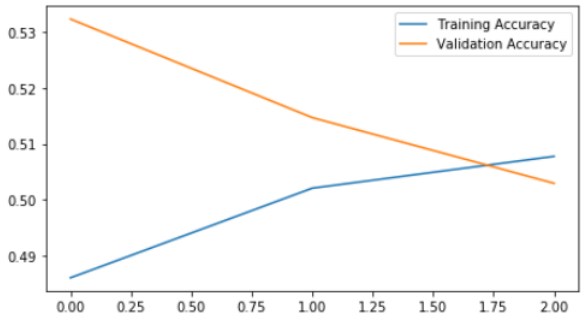

Figure 8. ‘Best’ Model Training & Validation Accuracy

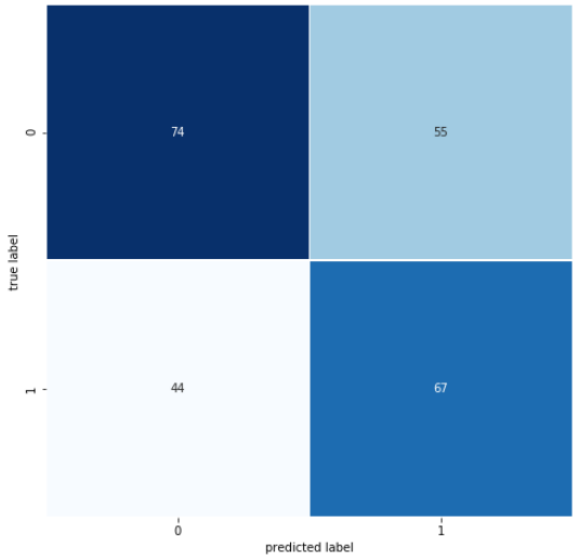

Figure 9. ‘Best’ Model Confusion Matrix

Of the range of model architectures I tested – two and three-layer, stacked convolutional layers, with and without data augmentation, etc. – the only consistent patterns that emerged were twofold: first, the primary obstacle to overcome was overfitting leading to the prediction of only the most prevalent class; second, model efficacy rested most strongly with the random weight initializations, as opposed to a specific architecture choice, indicating that the degrees of freedom available to even simple two-layer model architectures were sufficient to describe the complexity of the feature space. The results of my individual architecture experiments can be seen below.

Dense & Convolutional Neural Networks - Summary of Best Results by Architecture (Not Including Hyperparameter Tuning)										
Model Type	*Num Units*	*Dropout*	*Optimizer*	*Kernel Initializers*	*Filters*	*Learning Rate*	*Test Accuracy*	*Time (sec.) Per Epoch*	*Num. Epochs*	*Total Time*
Basline DNN - 2 Layer (No Reg.)	10	N/A	Adam	he_uniform		0.0001	53.75%[1]	0.2	32	6.4
Baseline CNN - 2 Layer (No Reg.)	N/A	N/A	Adam	he_uniform		0.0001	56.25%	8.7	3	26.1
CNN - 2 Layer	N/A	0.4	Adam	he_uniform		0.000001	58.75%	8.7	3	26.1
CNN - 2 Layer w/ LR Reduction	N/A	0.4	Adam	he_uniform		0.0001	51.67%	8.71	8	69.68
CNN - 3 Layer Stacked	1024	0.4	Adam	he_uniform		0.000001	54.17%	9.71	10	97.1
CNN - 3 Layer Stacked w/ Aug.	1024	0.25	Adam	he_uniform		0.0001	52.49%	9.76	8	78.08
CNN - 3 Layer Stacked Ensemble	1024	0.3	Adam	RANDOM[2]		0.00001	52.50%	8.7	75	652.5

1) Baseline DNN without regularization quickly reached a point of local-optima wherein all predictions were '0' or 'LONG'

2) The ensemble model is generated in such a way that every iteration of CNN randomly selects from one of 6 kernel initializers - He, Glorot and LeCun, both 'normal' and 'uniform' versions

Table 1. Model Architecture Results – Benchmarked Using Two-Layer DNN and CNN & No Regularization

The assertion that model initializations have a significant impact on the test dataset accuracy of the final models is borne out by the results of the hyperparameter tuning on both regularized DNN and CNN models. For the DNN and CNN models, respectively, 19 of 128 and 21 of 432 models outperformed the local optima (i.e., 53.75% test set accuracy), wherein the only discernible thread connecting these model outperformances is the random weight initializations, which help escape the local optima.

Dense Neural Networks: Two-Layer DNN with Regularization									
Num Units	*Dropout*	*Optimizer*	*Kernel Initializers*	*Filters*	*Learning Rate*	*Test Accuracy*	*Time (sec.) Per Epoch*	*Num. Epochs*	*Total Time*
10	0.5	SGD	he_uniform	N/A	0.00001	56.67%	0.2	3	0.6
32	0.5	Adam	he_normal	N/A	0.000001	56.25%	0.2	4	0.8
16	0.5	SGD	he_normal	N/A	0.0001	55.83%	0.2	4	0.8
32	0.4	SGD	he_uniform	N/A	0.0001	55.83%	0.3	6	1.8
64	0.4	SGD	he_uniform	N/A	0.0000001	55.42%	0.3	3	0.9
32	0.5	SGD	he_normal	N/A	0.0000001	55.42%	0.2	3	0.6
10	0.5	SGD	he_uniform	N/A	0.0001	55.42%	0.2	12	2.4
16	0.5	SGD	he_normal	N/A	0.000001	55.00%	0.2	4	0.8
64	0.5	Adam	he_uniform	N/A	0.000001	55.00%	0.2	3	0.6
10	0.4	SGD	he_uniform	N/A	0.0001	55.00%	0.3	3	0.9

Table 2. DNN Hyperparameter Tuning Results – Best 10 of 128 total

Convolutional Neural Networks: Two-Layer CNN with Regularization									
Num Units	*Dropout*	*Optimizer*	*Kernel Initializers*	*Filters*	*Learning Rate*	*Test Accuracy*	*Time (sec.) Per Epoch*	*Num. Epochs*	*Total Time*
256	0.4	Adam	he_normal	16	0.001	57.08%	9.72	6	58.32
512	0.3	SGD	he_uniform	32	0.001	56.67%	9.7	4	38.8
256	0.3	Adam	lecun_uniform	32	0.0001	56.25%	9.73	5	48.65
256	0.3	Adam	glorot_normal	32	0.001	56.25%	8.7	5	43.5
256	0.3	Adam	he_normal	64	0.001	55.83%	9.7	5	48.5
512	0.3	Adam	lecun_uniform	32	0.0001	55.83%	8.7	3	26.1
512	0.3	SGD	lecun_uniform	32	0.0001	55.83%	9.72	3	29.16
512	0.4	SGD	he_normal	32	0.0001	55.42%	9.71	5	48.55
256	0.4	SGD	lecun_normal	32	0.001	55.00%	9.71	6	58.26
256	0.3	SGD	he_uniform	32	0.001	55.00%	9.72	4	38.88

Table 3. CNN Hyperparameter Tuning Results – Best 10 of 432 total

In order to better understand what features the CNN models were learning, I next produced plots of the feature maps generated by the first three layers – two convolutional and one max-pooling – of a stacked-CNN model that had achieved a test dataset accuracy of just over 52 percent. The images below show these feature activation plots:

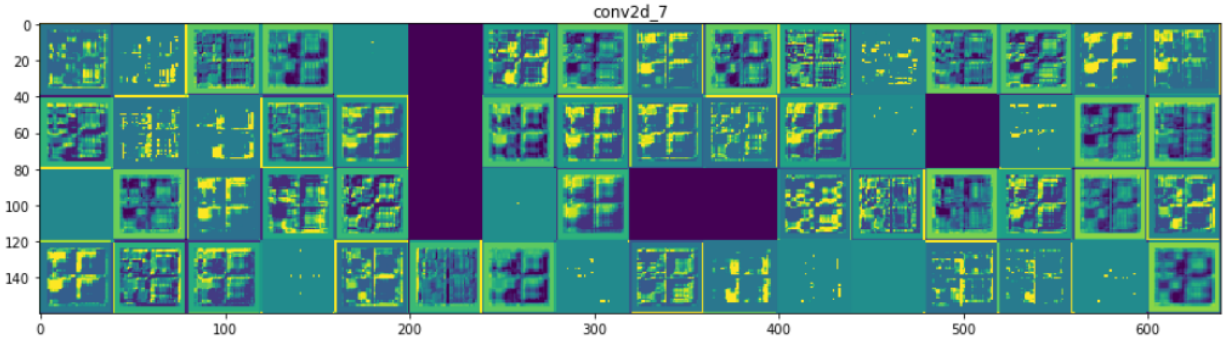

Figure 11. First Convolutional Layer Feature Activation Plot

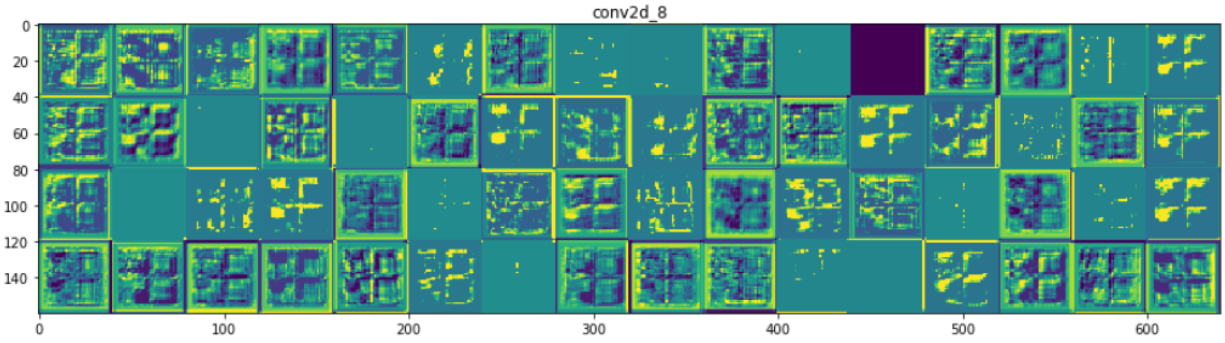

Figure 12. Second Convolutional Layer Feature Activation Plot

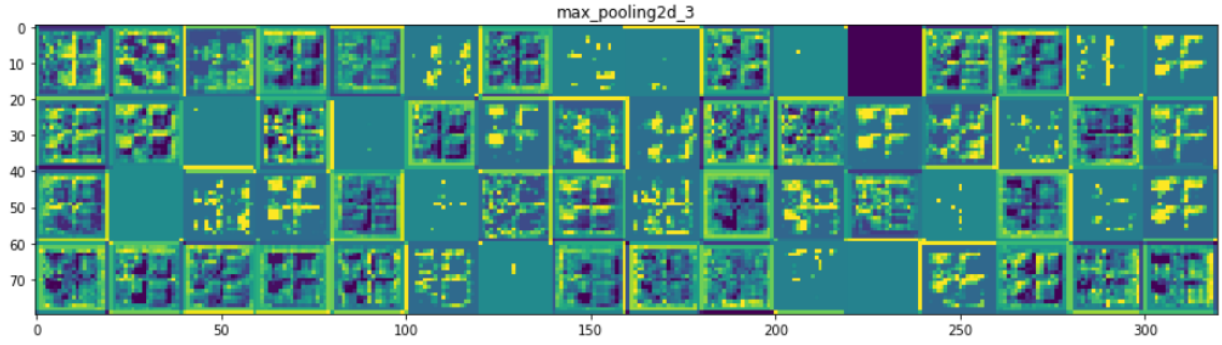

Figure 13. First Max-Pooling Layer Feature Activation Plot

Interesting to note in the above, each panel in the activation map tends to show similar regions in each of the four quadrants present in each image being dissected as activated; in other words, the model is making special-temporal correlations at varied time steps (each quadrant represents a GAF at a different time-interval) for patterns detected in the data. This may indicate that patterns, which are fractal in nature across time-intervals (i.e., self-similar), are being detected, which would confirm both theoretical assertions and empirical observations in investment finance made by Benoit Mandelbrot (Mandelbrot & Hudson, 2004). Unfortunately, however, the CNN model still has difficulty separating and unfurling the manifold upon which the dataset rests, as can be seen in the T-Stochastic Neighbor Embedding (T-SNE) plot of the test dataset, which possesses no discernible separability between classes of data.

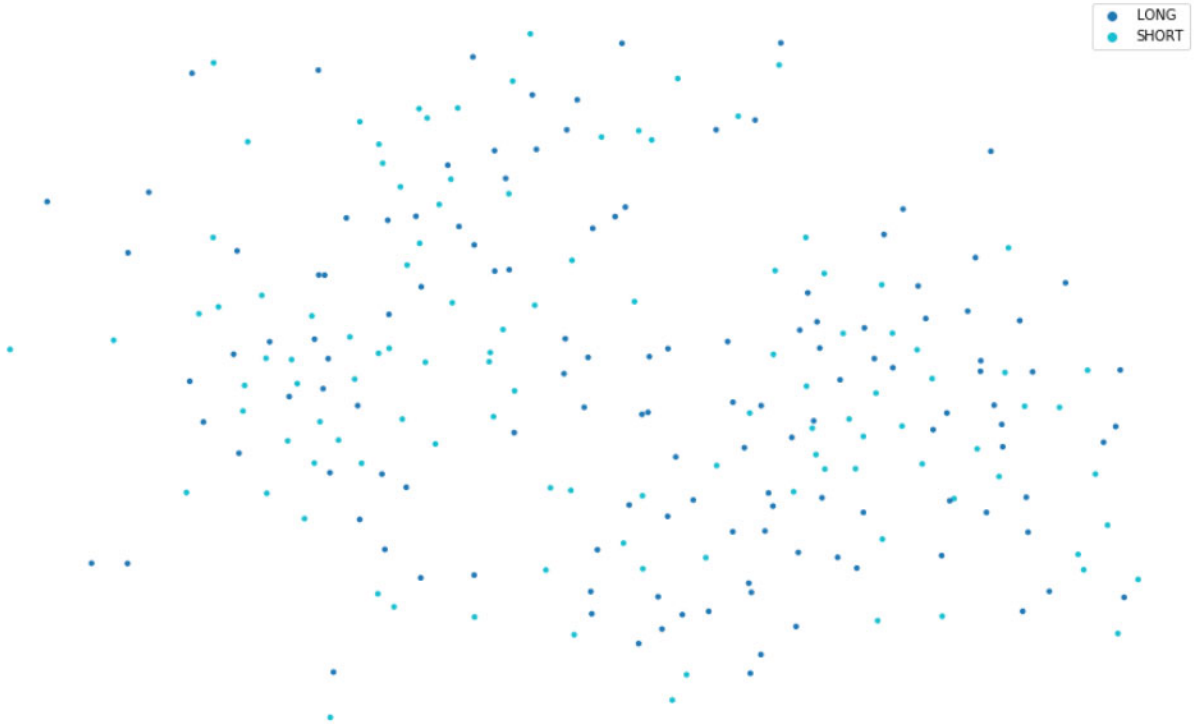

Figure 14: T-Stochastic Neighbor Embedding (T-SNE) Plot of Stacked CNN Model Architecture Prediction Results

The final step in my experiments, which demonstrates the greatest potential for use in predicting the class of the next day's security return, was that of model ensembling. During these experiments, I produced a loop that would generate a pre-specified number three-layer CNN's with the following

architecture: one convolutional layer with 32 filters followed by a max-pooling and dropout layer with dropout set to 0.3; two stacked-convolutional layers, with convolutional layers containing 64 filters, stacked one after another and after each layer stack a max-pooling and then dropout layer with dropout set to 0.3; finally, the outputs of the second stack-convolutional layer were flattened and fed into a fully-connected dense layer containing 1024 nodes, which itself was fed into a single node classification layer using a sigmoid activation function (See Appendix B). Additionally, I generated the model loop such that each model would randomly choose a kernel initializer in order to maximize the distribution of initialized starting weights to – hopefully – produce a more diverse set of models from which to make ensemble predictions.

By producing, fitting and then saving the resulting models built to a list object, I was able to use the models, in concert, to generate ensembled predictions. The manner in which I chose to make the ensemble predictions was through a consensus threshold; in other words, of the 10 models produced for a given ensemble, I could toggle the number of required predictions in agreement for 'LONG' predictions and only accept the prediction if a certain number of models (or greater) agreed on the prediction. By toggling through these options (0 through 10), I was able to determine that a threshold of 4 was most appropriate for this project. This threshold level produced a test set accuracy of 52.5 percent; moreover, the distribution of predictions between the 'LONG' and 'SHORT' classes most closely reflects the actual occurrence percentages in the training data (when compared to other produced models) – 56.25 percent 'LONG' predictions in the ensemble model test set versus 53.75 percent in the training set and 43.75 percent 'SHORT' predictions in the ensemble model test set versus 46.25 percent in the training set.

Lastly, I also tested producing ensemble predictions wherein a third category of class could be selected, titled 'HOLD', which was meant to indicate situations in which the model prediction threshold was not met in the 'LONG' or 'SHORT' direction, the model would assign a class of '2', or 'HOLD'. In such a scenario, a portfolio manager could choose not to trade the given security on the following day, choosing instead to wait until the model was more certain of the next day's class. The results, however,

were not superior to that of the first ensemble model, but this approach may merit further scrutiny in the future.

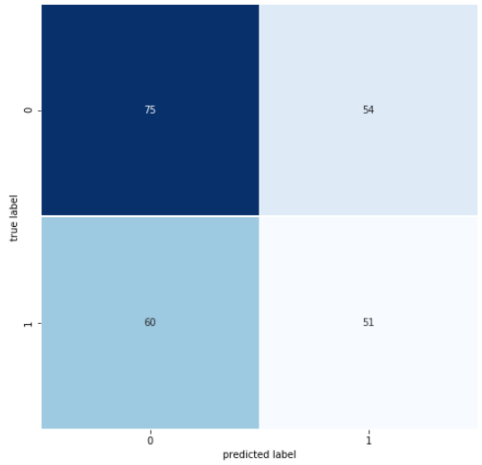

Figure 15. Ensemble CNN Model – Confusion Matrix

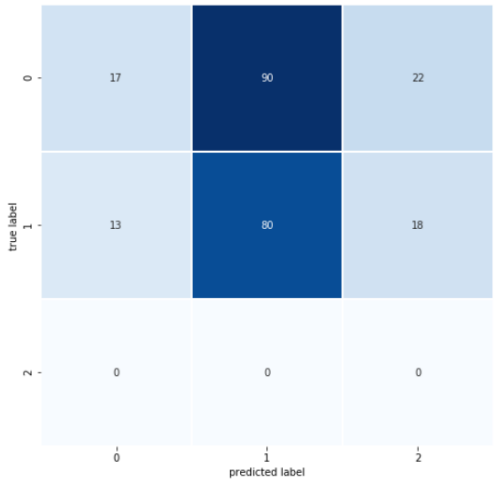

Figure 16. Ensemble CNN Model – Confusion Matrix, With '2', or 'HOLD' Option Added

Recommendations & Next Steps

For the purpose of developing a classification model capable of classifying the next trading day's return as positive – indicating the appropriateness of a 'LONG' position – or negative – indicating the appropriateness of a 'SHORT' position – it is my recommendation that XYZ Asset Management Corporation produce an ensembled CNN model based on a simple stacked-model architecture, and containing 10 CNN models in the ensemble. The stacked architecture should be structured as follows: input images will be 40x40x3, representing 4 quadrants of GAF images at the 1hr, 2hr, 4hr and 1day intervals; the first convolutional layer, containing 32 filters, will be followed by a max-pooling layer and then a dropout layer with dropout set to 0.3; this will be followed by two stacked convolutional layers, the first stack containing 64 filters for each convolutional layer and the second containing 128, which each flow into a max-pooling and then dropout layer with dropout set to 0.3; following the stacked convolutional layers the model output will be flattened and fed into a fully-connected dense layer

containing 1024 nodes; the dense layer will flow into the output layer that contains a single node and uses the sigmoid activation function to make a binary classification determination.

Due to the high impact of model initialization weights, kernel initializers should be randomly selected from a pool of no less than six initialization techniques (He, Glorot & LeCun, Uniform and Normal versions, respectively). Furthermore, XYZ Asset Management Corporation should produce five such ensembles, each stored and able to be called upon separately. These five ensembled models should be used over no less than 90 trading days to 'paper trade' the model predictions. In other words, each model should be used on real-time data, following the close of trading each day for 90 trading days, to predict the class of the next trading day's return. The accuracy of each ensemble model should be recorded for each day; at the end of the paper trading period, XYZ Asset Management Corporation should assess each model in turn and determine if the results merit live application to real trading decisions moving forward. If implemented, models should be recalibrated and re-tested on a no-less than quarterly basis to ensure new trading patterns – which may occur as market conditions change – can be captured by the models.

Given the sparsity of data available in this project (i.e., roughly 2,500 total stitched GAF images available), it is also recommended that XYZ Asset Management Corporation use their superior resources to test the same model architecture at narrower time intervals – i.e., intraday. By setting the smallest time increment to one minute and the largest to 1hr, it is possible to develop a larger dataset upon which to train the model, which may provide a more robust feature space from which to fit the model and avoid local optima.

Finally, it is highly recommended that XYZ Asset Management Corporation explore, during the paper-trading phase of their assessment, ensemble model agreement thresholds below which portfolio managers should opt not to engage in either 'LONG' or 'SHORT' positions. By seeking thresholds upon which more clear signals are being detected by the ensemble, it is possible to avoid unnecessary errors and improve long-term investment performance.

References

Barra, S., Carta, S.M., Corriga, A., Podda, A.S., & Recupero, D.R. (2020). Deep learning and time series-to-image encoding for financial forecasting. IEEE/CAA Journal of Automatica Sinica, 7(3), 683–692. https://doi.org/10.1109/JAS.2020.1003132

Faouzi, J. & Janati, H. (2020). Pyts: A Pythong package for time series classification. Journal of Machine Learning Research, 21(46), 1–6.

Mandelbrot, B. & Hudson, R. L. (2004). The (mis)behavior of markets : a fractal view of risk, ruin, and reward . Basic Books.

Lodewijk, P. (2014). The world's first stock exchange. Columbia Business School Publishing.

Lopez De Prado, M. (2018). Advances in financial machine learning. John Wiley & Sons, Inc. Hoboken, New Jersey.

Securities and Exchange Commission. (2011). Release No. 34-64874, File Number: S7-30-11. https://www.sec.gov/comments/s7-30-11/s73011-10.pdf

Wang, Z. & Oates, T. (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*.

Appendix A: Gramian Angular Field Encoding Steps

Given a time series $X = \{x_1, x_2, ..., x_n\}$ of n real-valued observations, we rescale X so that all values fall in the interval $[-1, 1]$:

$$\tilde{x_i} = \frac{(x_i - max(X) + (x_i - min(X))}{max(X) - min(X)} \tag{1}$$

Thus we can represent the rescaled time series $\tilde{X}$ in polar coordinates by encoding the value as the angular cosine and time stamp as the radius with the equation below:

$$\begin{cases} \phi = \arccos(\tilde{x_i}), -1 \leq \tilde{x_i} \leq 1, \tilde{x_i} \in \tilde{X} \\ r = \frac{t_i}{N}, t_i \in \mathbb{N} \end{cases} \tag{2}$$

After transforming the rescaled time series into the polar coordinate system, we can easily exploit the angular perspective by considering the trigonometric sum between each point to identify the temporal correlation within different time intervals. The GAF is defined as follows:

$$G = \begin{bmatrix} \cos(\phi_1 + \phi_1) & \cdots & \cos(\phi_1 + \phi_n) \\ \cos(\phi_2 + \phi_1) & \cdots & \cos(\phi_2 + \phi_n) \\ \vdots & \ddots & \vdots \\ \cos(\phi_n + \phi_1) & \cdots & \cos(\phi_n + \phi_n) \end{bmatrix} \tag{3}$$

$$= \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}' \cdot \sqrt{I - \tilde{X}^2} \tag{4}$$

I is the unit row vector $[1, 1, ..., 1]$. After transforming to the polar coordinate system, we take time series at each time step as a 1-D metric space. By defining the inner product $< x, y >= x \cdot y - \sqrt{1 - x^2} \cdot \sqrt{1 - y^2}$, G is a Gramian matrix:

$$\begin{bmatrix} < \tilde{x_1}, \tilde{x_1} > & \cdots & < \tilde{x_1}, \tilde{x_n} > \\ < \tilde{x_2}, \tilde{x_1} > & \cdots & < \tilde{x_2}, \tilde{x_n} > \\ \vdots & \ddots & \vdots \\ < \tilde{x_n}, \tilde{x_1} > & \cdots & < \tilde{x_n}, \tilde{x_n} > \end{bmatrix} \tag{5}$$

Appendix B: Recommended Ensemble Model Architecture

```
# Generate 10 different models and append themto a list; then compile that list of models

cnn_networks = 10
model = []
for j in range(cnn_networks):

    model.append(
        tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu, input_shape=(40, 40, 3),
                                   kernel_initializer=rand_kernel),
            tf.keras.layers.MaxPool2D((2, 2),strides=2),
#            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu,
                                   kernel_initializer=rand_kernel),
            tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu,
                                   kernel_initializer=rand_kernel),
            tf.keras.layers.MaxPool2D((2, 2),strides=2),
#            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu,
                                   kernel_initializer=rand_kernel),
            tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1), activation=tf.nn.relu,
                                   kernel_initializer=rand_kernel),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(units = 1024, activation=tf.nn.relu,
                                  kernel_initializer=rand_kernel),
            tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)
        ]))
    # Compile each model
    model[j].compile(
        optimizer=tf.optimizers.Adam(learning_rate=1e-5),
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics=['accuracy'],
    )
```

```
Net : 1
rand int =  365
Found 1932 validated image filenames belonging to 2 classes.
Found 340 validated image filenames belonging to 2 classes.
Found 240 validated image filenames belonging to 2 classes.
Epoch 1/20
120/120 [==============================] - 9s 73ms/step - loss: 0.7580 - accuracy: 0.4885 - val_loss: 0.7188 - val_accuracy: 0.4735
Epoch 2/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7568 - accuracy: 0.5167 - val_loss: 0.7163 - val_accuracy: 0.4824
Epoch 3/20
120/120 [==============================] - 8s 71ms/step - loss: 0.7386 - accuracy: 0.5188 - val_loss: 0.7145 - val_accuracy: 0.4882
Epoch 4/20
120/120 [==============================] - 8s 71ms/step - loss: 0.7365 - accuracy: 0.5120 - val_loss: 0.7133 - val_accuracy: 0.4882
Epoch 5/20
120/120 [==============================] - 8s 71ms/step - loss: 0.7456 - accuracy: 0.4984 - val_loss: 0.7126 - val_accuracy: 0.4824
Epoch 6/20
120/120 [==============================] - 9s 71ms/step - loss: 0.7588 - accuracy: 0.4922 - val_loss: 0.7116 - val_accuracy: 0.4824
Epoch 7/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7619 - accuracy: 0.4854 - val_loss: 0.7111 - val_accuracy: 0.4912
Epoch 8/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7495 - accuracy: 0.5094 - val_loss: 0.7106 - val_accuracy: 0.4912
Epoch 9/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7277 - accuracy: 0.5292 - val_loss: 0.7105 - val_accuracy: 0.4882
Epoch 10/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7484 - accuracy: 0.5037 - val_loss: 0.7102 - val_accuracy: 0.4853
Epoch 11/20
120/120 [==============================] - 8s 70ms/step - loss: 0.7461 - accuracy: 0.5146 - val_loss: 0.7098 - val_accuracy: 0.4912
CNN Model 1: Kernel Initializer=he_normal Epochs=20, Training Accuracy=0.52923, Validation Accuracy=0.49118
5/5 [==============================] - 1s 119ms/step - loss: 0.7035 - accuracy: 0.5125
accuracys: 51.25%
```

NOTE: This is an example of 1 of the 10 CNN's that generated the ensemble network predictions.