Time Series Augmentation through Deep Learning

Samuel Johnson
Utah State University
Logan, Utah, United States
sam.johnson1357@gmail.com

Abstract

Time series data tends to have far fewer observations than traditional tabular data. This makes tasks like time series classification more difficult. Additionally, some datasets may be extremely imbalanced, with many observations of one case and few of another. Due to these difficulties, it makes sense to generate "artificial" observations in the hope that they will increase accuracy on the testing dataset. In this paper we examine various deep learning methods that can be used to augment datasets. We examine how similar augmented data is to real data and how it can affect model accuracy. We perform the analysis using an "Earthquake" dataset [1]. The models were built and evaluated using the PyTorch [2] and Scikit-learn [3] Python libraries.

Keywords: Deep learning, time series, autoencoder, transformer, generative adversarial network, augmentation

ACM Reference Format:
Samuel Johnson. 2022. Time Series Augmentation through Deep Learning. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The classification problem associated with the Earthquake dataset was predicting whether or not a major earthquake event was going to occur based on recent readings in the surrounding area. Each observation had 512 data points, each of which was an average reading for one hour. Thus, data for the previous 512 hours was given for each label. A major event for classification purposes was defined to be a reading of over five on the Rictor scale while a minor event was a reading of less than four. There are 368 negative cases and 93 positive cases in the dataset, making it an imbalanced dataset.

For purposes of generating augmented data, we will generate data for both positive and negative classes. We will then compare them through PCA to see how similar they are.

For purposes of improving classification accuracy, we will create 275 additional *positive* examples so that there are 368 instances of each class. We will then see if classification accuracy has improved.

We will perform the above steps using a few different deep learning models: autoencoders, transformers, and GANs.

2 Autoencoder

2.1 How Autoencoders Work

The first model we used for data augmentation was the autoencoder. Autoencoders are formed by creating two neural networks: an encoder and a decoder. The encoder takes in input and through several fully-connected layers transforms that input into a low-dimensional space.

Next, the decoder takes in the encoder's output as its input. Then, through several fully connected layers, it attempts to transforms the input back into its original state. The original input to the encoder and the output from the decoder are then compared and the loss is computed. Backpropogation is then performed, adjusting the weights in the model, forcing the model to learn.

2.2 Structure

The autoencoder consisted of an input layer of 512 nodes, four hidden layers of 128, 64, 36, and 18 nodes, and an output layer of 3 nodes. It took in an array of length 512 and output a condensed array of length 3. The ReLU activation function was applied between the layers.

The decoder was built in a reverse fashion. It had an input layer of 3 nodes, hidden layers of 18, 36, 64, and 128 nodes, and an output layer of 512 nodes. It took in an array of length 3 and output an array of length 512. The ReLU activation function was once again applied between the layers.

2.3 Loss Function

The SoftDTW loss function, which builds on DTW, was used to train the model. Traditional dynamic time warping uses dynamic programming to optimally align two time series

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and compute their distance. It is known to compute similarity better than traditional metrics like mean squared error on time series data. SoftDTW is a smoothed, differentiable approximation of DTW, such that it can be used by the gradient descent algorithm for parameter optimization [4].

2.4 Augmented Data Generation

Augmented data was generated by first selecting a random observation from the real dataset. Then, the data was repeatedly passed through the autoencoder until error was reduced below a certain threshold. The threshold was a randomly selected value between 0.2 and 0.5 for each augmentation. When the threshold was set to 0, the model created a perfect reconstruction of the original dataset, which wasn't very useful. Thus, a random threshold was introduced to created data that differed from its source.

Last of all, values in the real dataset never went below -0.8 while the autoencoder occasionally produced values below this level. We opted to force all values lower than -0.8 in the augmented dataset to -0.8 in order to make the dataset more realistic.

In figure 1 we show a plot of a randomly selected sample from class one and the resulting augmented data. We can see that the two time series look fairly similar with small variations in the heights of the various bars. They are also different in that the bottom of the first chart is flat while the bottom of the second chart is jagged.

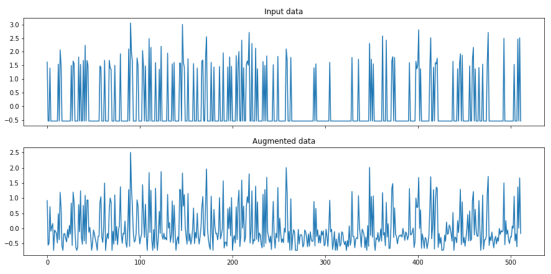


Figure 1. Class 1 Augmented Data

Figure 2 shows a randomly selected sample from class zero and the resulting augmented data. We can once again see that the two time series are similar with small variations in the heights of their bars. Once again, the bottom of the first plot is much smoother than the bottom of the second plot.

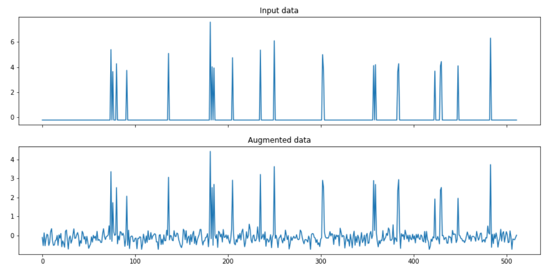


Figure 2. Class 0 Augmented Data

2.5 PCA Analysis

Next we performed principal components analysis to compare the original data with the augmented data. After performing PCA, we selected just the first two principal components. Note that the first two principal components explain just 4% of the variation in class zero and 12% of the variation of class one. This is likely due to the fact that it is difficult to compress a 512-dimensional dataset.

The PCA plot for class zero is given below in figure 3. We can see the original data in purple and the augmented data in green. Both sets of data spread across the graph, although the augmented data is far more clustered in the center of the graph. This would suggest that the augmented data represents the original well but not perfectly. There is room for improvement.

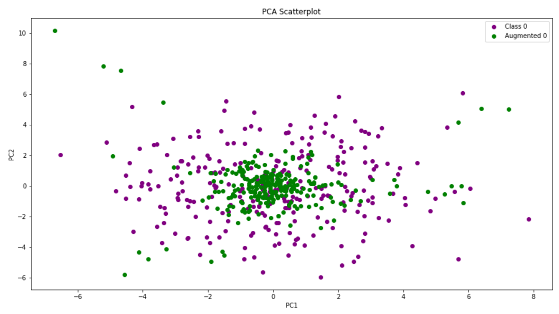


Figure 3. Class 0 PCA

The PCA plot for class one is given below in figure 4. The original data is given in blue while the augmented data is given in red. We can see that both datasets have a good spread but once again the augmented data is more clustered and has less variation than the real data. This suggests that the augmented data quality is lacking.

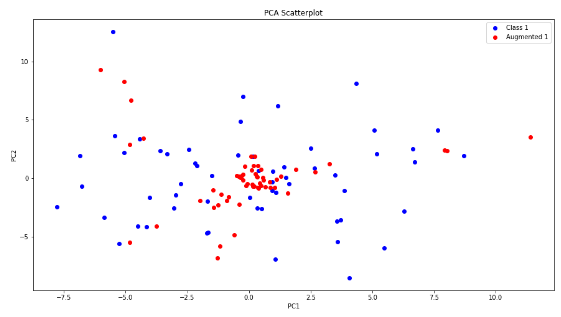


Figure 4. Class 1 PCA

2.6 Time Series Classification

We perform time series classification on the Earthquake dataset using three different machine learning models: Logistic Regression, K-Nearest-Neighbor, and Random Forest Classifier. The goal was to classify each observation as either a minor or major event.

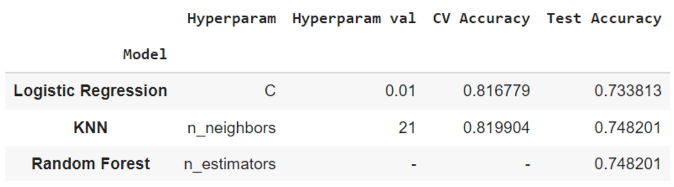
Some of these machine learning methods had important hyperparameters to tune. We ran five-fold cross validation across a range of values to find the optimal hyperparameters. Afterwards, we used the chosen hyperparameters and ran them on a separate test set to evaluate model accuracy on unseen data.

The relevant hyperparameter for logistic regression was C, which indicates the level of regularization. Low values of C lead to high regularization while high values of C lead to low regularization. We tested ten different values of C ranging from 0.01 to 2 in equally sized increments.

The relevant hyperparameter for the K-Nearest-Neighbor classifier was the number of neighbors. We trained the model on ten different values ranging from 1 to 100 in increments of equal size.

One possible hyperparameter for the random forest classifier is the number of estimators. In practice, however, the random forest doesn't overfit as we increase the number of estimators. Due to this fact, we simply choose 300 and estimators and fit the model once.

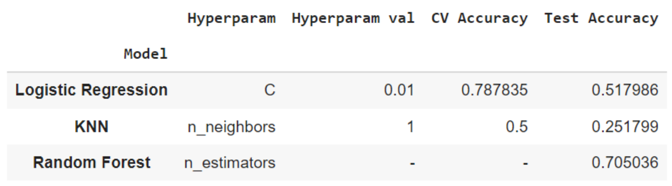
After finding the optimal hyperparameters, we trained a model and then calculated the classification accuracy on the unseen test set. The results of our analysis for the unaugmented data are given below in figure 5. We can see that the accuracy for all three methods hovers around 74%.



	Hyperparam	Hyperparam val	CV Accuracy	Test Accuracy
Model				
Logistic Regression	C	0.01	0.816779	0.733813
KNN	n_neighbors	21	0.819904	0.748201
Random Forest	n_estimators	-	-	0.748201

Figure 5. Classification Accuracy Before Augmentation

We then augment the data using an autoencoder. We create 275 additional *positive* examples so that there are 368 instances of each class. The results of the classification are shown in figure 6. We can see that the accuracy has worsened substantially for the logistic regression and KNN models. The random forest model's accuracy has decreased slightly to 70.5%.



	Hyperparam	Hyperparam val	CV Accuracy	Test Accuracy
Model				
Logistic Regression	C	0.01	0.787835	0.517986
KNN	n_neighbors	1	0.5	0.251799
Random Forest	n_estimators	-	-	0.705036

Figure 6. Classification Accuracy After Autoencoder Augmentation

We conclude that the autoencoder might not be the best time series augmentation method.

3 Transformer

3.1 How Transformers Work

The next model we used for time series data augmentation was the transformer. Transformers are Seq2Seq models that require the input of a source sequence and a target sequence. A common use case is language translation, where we might enter a sentence in English as the source and the translated version in German as the target and teach the model to translate from English to German.

Transformers are built using a series of encoders and decoders. Each encoder and decoder contains a fully connected layer and an attention layer. The attention layer is a recent innovation that is designed to help the model focus on the most important words to improve translation.

3.2 Structure

We can generalize the language translation example to time series augmentation using transformers. We can do so by randomly choosing both a source and target sequence from the dataset. Then, we can train the model to learn the target sequence from the source sequence. After training, we can output the current estimate and treat it as augmented data.

The following model parameters were used:

- Number of heads: 8
- Number of encoder layers: 6
- Number of decoder layers: 6
- Dimension of fully connected layer: 2048
- Dropout: 0.1
- Activation function: ReLU

3.3 Augmented Data Generation

We used the SoftDTW loss function, a learning rate of 0.01, and 10 epochs of training. We took the final output as generated data.

In figure 7 we display an example of augmented data for class zero. The first two graphs are randomly selected source and target data. The last graph is the augmented data. We can see that the augmented data closely resembles the target data with slight variations. Unlike the autoencoder, the bottom is pretty smooth, closely resembling real data.

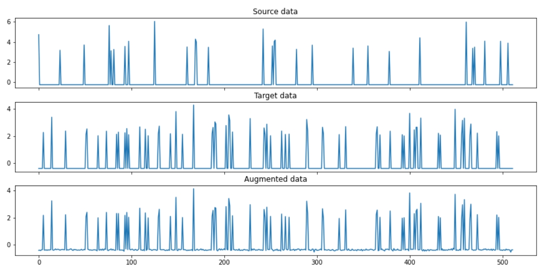


Figure 7. Transformer Augmentation Class 0

In figure 8 we display an example of augmented data for class one. Once again, we can see that the generated data closely resembles the target data with slight variations. The bottom of the chart is smooth, resembling real data samples.

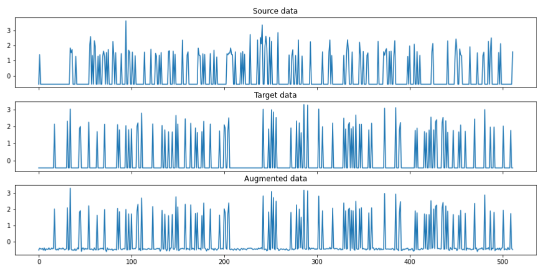


Figure 8. Transformer Augmentation Class 1

3.4 PCA Analysis

Next we performed principal components analysis in order to compare the original data with the augmented data. Due to high dimensionality, the first two principal components explained just 5% of the variation in class zero and 20% of the variation in class one.

The PCA plot for class zero is given below in figure 9. We can see the original data in purple and the augmented data in green. We can see that the purple and green dots cover similar spaces, although the green dots are slightly more clustered together. In general, it looks like the augmented data is very realistic.

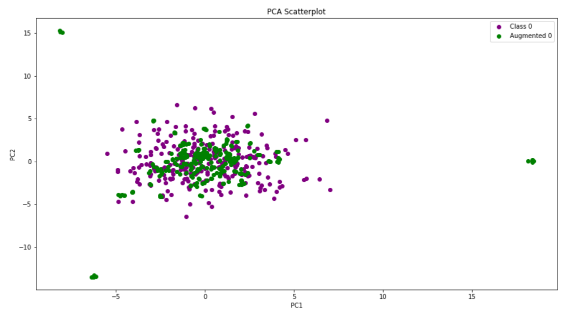


Figure 9. Class 0 PCA

The PCA plot for class one is given below in figure 10. The original data is given in blue while the augmented data is given in red. Both datasets have a good spread although the blue (real) dataset has a much bigger spread. Perhaps the transformer generated data for class zero is better than that of class one.

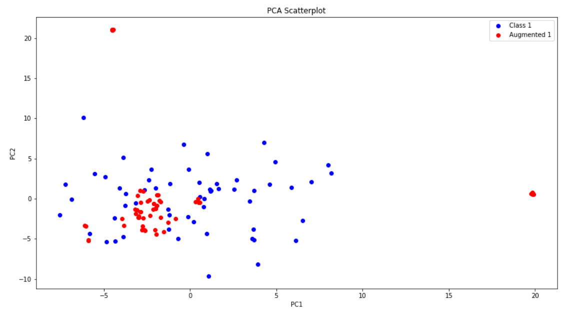
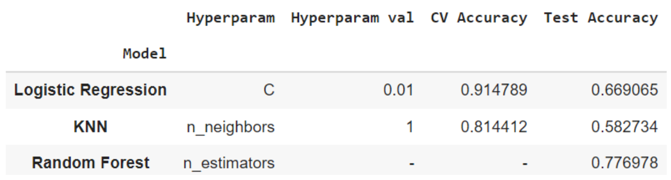


Figure 10. Class 1 PCA

3.5 Time Series Classification

We once again create 275 additional *positive* examples so that there are 368 instances of each class. We then perform hyperparameter tuning through cross validation the same way as before. We fit models with the optimal hyperparameters and then evaluate the model's accuracy on the unseen test set. The results can be seen in figure 11. The Logistic Regression and KNN models are achieving worse accuracy than before. The Random Forest Classifier, on the other hand, achieved accuracy of 77.7%, which is about 3% better than its performance when trained on unaugmented data.



Model	Hyperparam	Hyperparam val	CV Accuracy	Test Accuracy
Logistic Regression	C	0.01	0.914789	0.669065
KNN	n_neighbors	1	0.814412	0.582734
Random Forest	n_estimators	-	-	0.776978

Figure 11. Classification Accuracy After Transformer Augmentation

Although the Logistic Regression and KNN models suffered poor performance, the Random Forest Classifier saw improved accuracy. Ultimately, it is peak accuracy that matters. Therefore, we conclude that transformer augmentation methods led to increased accuracy on this dataset.

4 Generative Adversarial Networks

4.1 How GANs Work

The next model we tried was the generative adversarial network. GANs are models that attempt to generate realistic data, a popular example being human faces. They are built with two deep neural structures: a generator and a discriminator. In this model, these two structures are competing with one another. The generator is creating data and the discriminator is attempting to *discriminate* between the real and generated data. The end goal is that the generator is able to generate realistic enough data that the discriminator is fooled about 50% of the time.

The generator is fed random data to its initial layer. It then contains several hidden layers, after which it outputs fake data.

The discriminator takes in both real and augmented data and attempts to classify them. It consists of several hidden layers before reducing to a dimensionality of one and applying the sigmoid activation function. In this manner, it outputs the probability that a given sample is real.

4.2 Structure & Training

The generator took in an initial 512-dimensional vector. There were then two hidden layers of 750 and 1000 neurons and finally an output array of length 512.

The discriminator took in an initial 512-dimensional vector. It then passed the data through two hidden layers of 300 and 150 neurons before outputting a probability between 0 and 1.

Early GAN models saw the discriminator dominating the generator, which prevented it from learning. As a result, we implemented dropout layers of 0.5 between the hidden discriminator layers.

We also randomly chose 20% of the real input data and told the discriminator that it wasn't real. This slowed down the discriminator enough to allow the generator to learn.

4.3 Augmented Data Generation

The following hyperparameters were used by both the discriminator and the generator. Note that SoftDTW was attempted but didn't end up working very well.

- Loss function: Binary Cross Entropy
- Epochs: 600
- Batch size: 20
- Learning rate: 0.01

In figure 12 we display generated data for class zero against randomly selected class zero data. We can see that the two samples are similar, suggesting that the GAN is doing a good job.

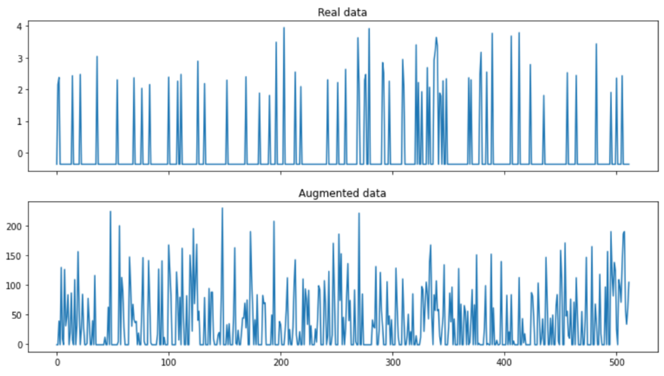


Figure 12. GAN Augmented Class 0 Data

In figure 13 we display generated data for class one against randomly selected class one data. Without labels, it would be difficult to determine which one was the real data.

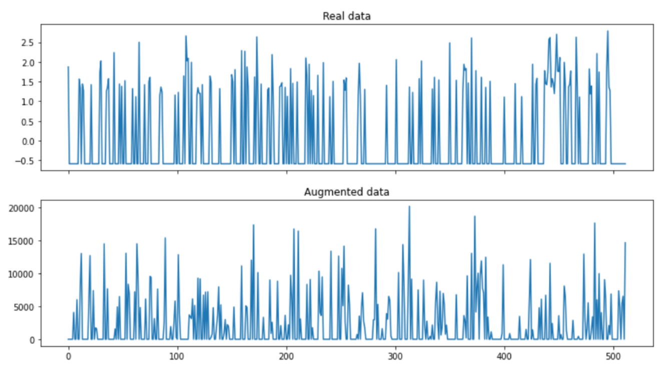


Figure 13. GAN Augmented Class 1 Data

4.4 PCA Analysis

Next we performed principal components analysis on the real vs augmented data. Note that the first two principal components explain 99% of the variation in class zero and 1% of the variation of class one.

Figure 14 displays the principal components of class zero. Surprisingly, the augmented data doesn't look anything like the real data.

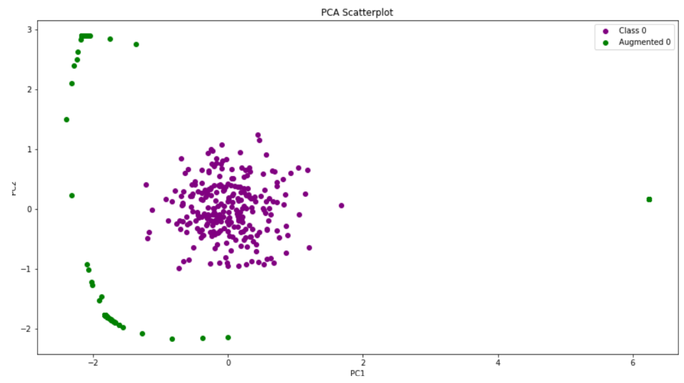


Figure 14. Class 0 PCA

Figure 13 displays the principal components for class one. Interestingly, all the augmented (red) dots are clustered in one area. Upon further examination, the generator is creating the same example every time and has thus fallen into mode collapse.

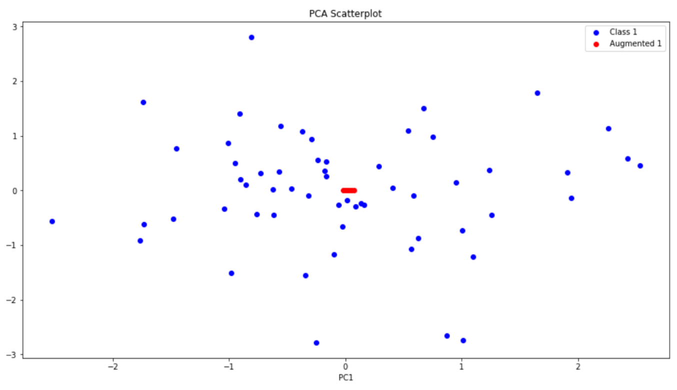
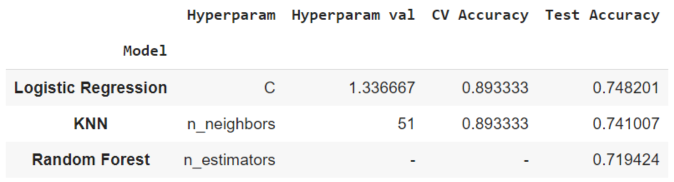


Figure 15. Class 1 PCA

4.5 Time Series Classification

We once again create 275 additional *positive* examples so that there are 368 instances of each class. We perform hyperparameter tuning and then evaluate various models on the unseen test set. The resulting accuracy is given in figure 16.



Model	Hyperparam	Hyperparam val	CV Accuracy	Test Accuracy
Logistic Regression	C	1.336667	0.893333	0.748201
KNN	n_neighbors	51	0.893333	0.741007
Random Forest	n_estimators	-	-	0.719424

Figure 16. Accuracy after GAN augmentation

We can see that accuracy has stayed roughly the same as that with no augmentation, although the random forest accuracy has slipped slightly. We conclude that this particular GAN model has room for improvement.

5 Wasserstein GAN

In the previous section we noted that the GAN resulted in mode collapse, wherein all the samples generated by the GAN were essentially equivalent. Wasserstein GANs are designed to prevent mode collapse. They do this by having a critic instead of a discriminator. Rather than classifying data, the critic tries to make the output for real instances bigger than that of fake instances [5]. We employ the WGAN in this section to see if we can improve our augmentation.

5.1 Structure

The structure of the WGAN model was the same as the regular GAN except the leakyReLU activation function was used.

The following hyperparameters were decided on after multiple training attempts:

- Epochs: 500
- Critic generator train ratio: 5

- Gradient clip range: [-1, 1]
- Batch Size: 64
- Critic learning rate: 0.001
- Generator learning rate: 0.001

During training, each time the generator loss reached a new low, that model was saved. After 500 epochs, the saved model was used to generate augmented data.

5.2 Data Augmentation

Figures 17 and 18 show the real and generated data for both classes. Upon visual inspection, the augmenter appears to be doing a good job.

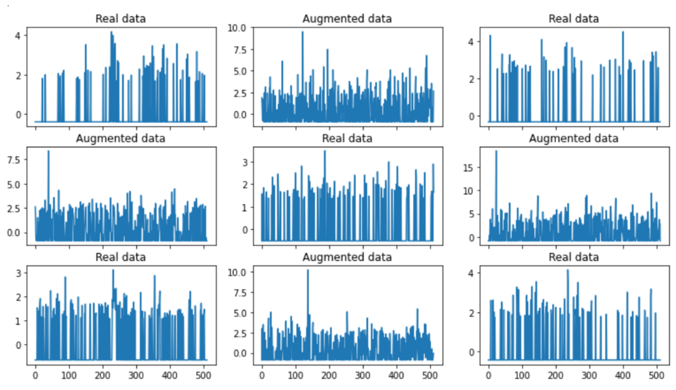


Figure 17. Wasserstein GAN Class 0 Augmented Data

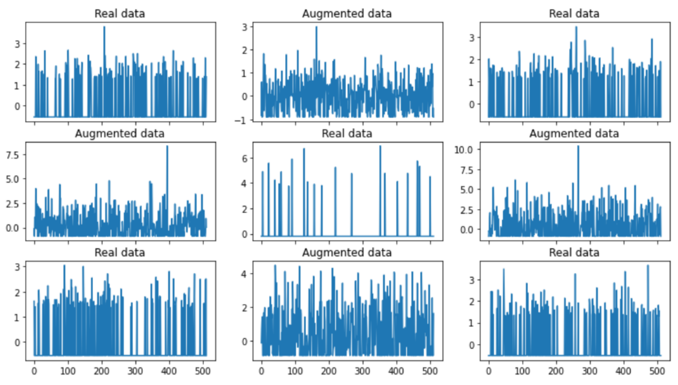


Figure 18. Wasserstein GAN Class 1 Augmented Data

5.3 PCA Analysis

Figures 19 and 20 display the PCA plots for class zero and class one respectively. The augmented data doesn't seem to represent the real data very well.

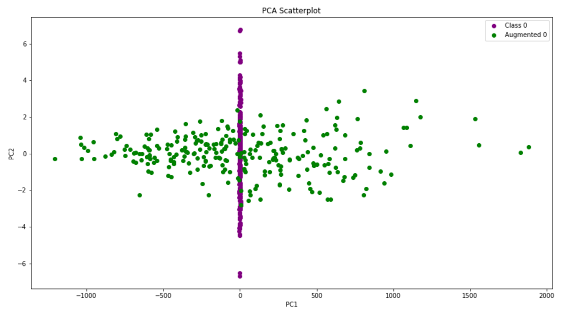


Figure 19. Class 0 PCA

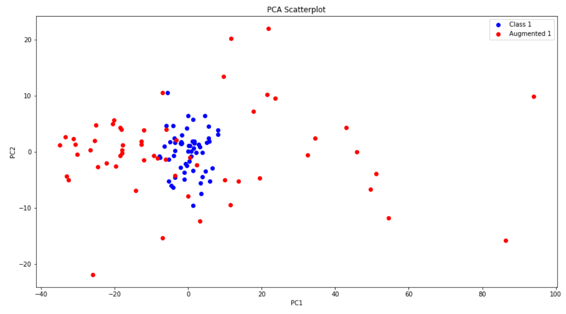
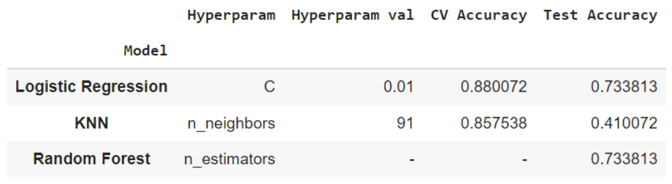


Figure 20. Class 1 PCA

5.4 Time Series Classification

In figure 21 we show the classification accuracy after WGAN augmentation in the usual manner. We can see that the accuracy of Logistic Regression and Random Forest have more or less remained constant while the accuracy of KNN has dropped significantly.



Model	Hyperparam	Hyperparam val	CV Accuracy	Test Accuracy
Logistic Regression	C	0.01	0.880072	0.733813
KNN	n_neighbors	91	0.857538	0.410072
Random Forest	n_estimators	-	-	0.733813

Figure 21. Classification Accuracy After WGAN Augmentation

6 Conclusion & Future Research

In this paper we examined various deep learning methods for time series augmentation on an "Earthquake" dataset. We tried out autoencoder models, transformers, GANs, and WGANs. We found that autoencoder and GANs underperformed while transformers led to an increase in classification accuracy.

Future research could include the following:

- Try out different hyperparameters for deep learning models.
- Implement an Unrolled GAN [5], which might help prevent mode collapse.
- Try out other non-deep-learning time series augmentation methods.
- Try out a variational autoencoder[6].

References

[1] A. Bagnall. Dataset: Earthquakes.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series, 2017.

[5] Common problems; generative adversarial networks; google developers.

[6] T. Fuertes. Variational autoencoder as a method of data augmentation, Jun 2020.