WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

# CHART PATTERN CATEGORIZATION WITH CNNS

27. JANUAR 2019

Christoph Ostertag
Milan Krivokuća

# Contents

Goal ..... 2
Methods ..... 2
Synthetic Chart Patterns ..... 2
Processing Input Charts ..... 5
Keras CNN ..... 5
Results ..... 7
Noise 0 ..... 7
Noise 10 ..... 8
Noise 30 ..... 9
Noise 50 ..... 10
Noise 100 ..... 11
Conclusion ..... 12

Christoph Ostertag
Milan Krivokuća

# Goal

The goal of this paper is to find out to which extend CNNs can categorize patterns in charts and give a visual intuition about it.

The goal of this paper is not to find out, just if CNNs can categorize chart patterns, nor is it to classify real chart patterns. Rather we are working with synthetic charts that show different predefined patterns that will be altered in a way to make it less similar to the original pattern. By visually comparing the charts to the percentage of right classification we aim to give the reader a visual intuition about how accurate this classification can be.

This paper does not aim to fulfil the typical standards for a scientific paper, but should rather only show my personal experience with chart pattern classification and give a brief overview and visual understanding of the topic for the reader.

# Methods

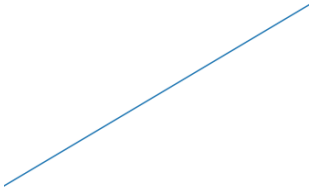
## Synthetic Chart Patterns

To create those synthetic or fake charts, we first must choose what pattern we want to mimic. We chose five different ones, namely:

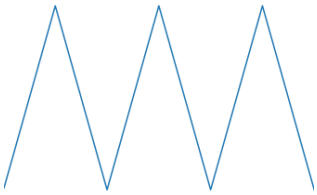
- None



- Linear

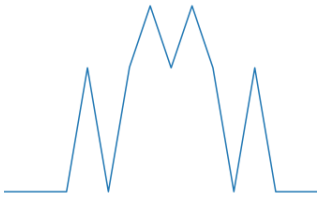


- Triple Top



- Triangle

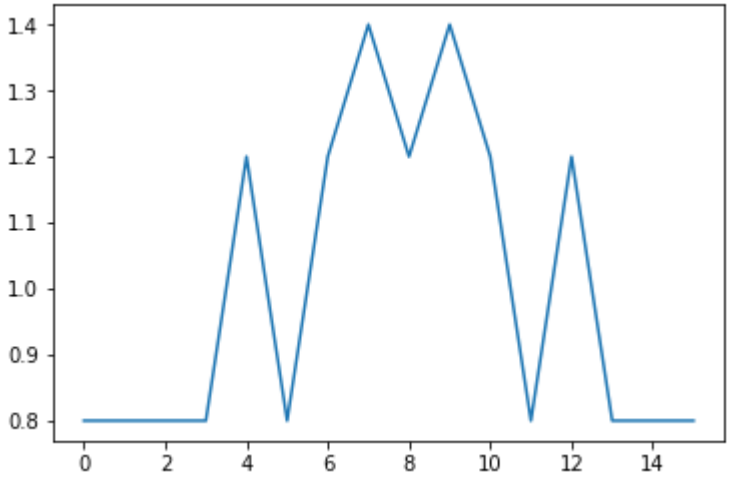
- Head and Shoulders



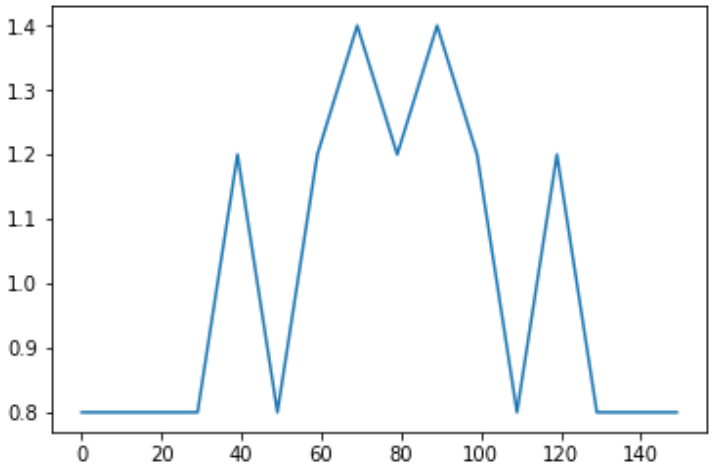
After having figured out and drawn our patterns, we want to modify them with some noise, so they don't look exactly as the originals.

But first we need to be using extrapolation to first extend our charts with more datapoints we can later fill with noise.

<u>Around 15 datapoints:</u>




<u>Around 150 datapoints (10x more datapoints):</u>




Now we have more datapoints, but still no noise, as you see, the chart, except for the axis looks exactly the same.

To give noise to our chart, we initialize a variable that holds an over the x-axis (time) accumulating difference to our original y-value (price). Here is an example:

```
differ = 0
for _ in range(n):
    # returns a variable differ that is changing every iteration, by
    # a random draw from the standard normal distribution
    differ = differ + np.random.randn()
```

To price this difference into our original prices, we loop over them and add every iteration changing accumulated differences to the original prices.

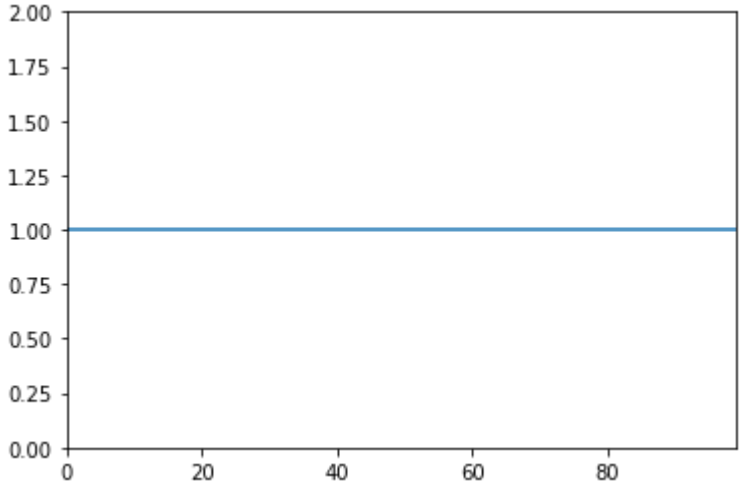
```
syntheticPrices = []
differ = 0
for price in originalPrices:
    differ = differ + np.random.randn()
    syntheticPrice = price + differ
    syntheticPrices.append(syntheticPrice)
```

To change the magnitude of how much noise we want to have in our chart we define a variable sensitivity that we multiply with the standard deviation “np.random.randn()”.

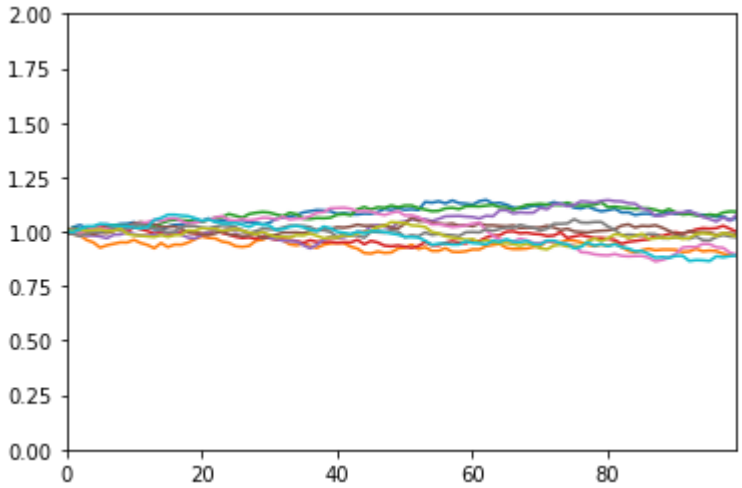
```
syntheticPrices = []
differ = 0
sensitvity = 30*10e-3
for price in originalPrices:
    differ = differ + np.random.randn()*sensitvity
    syntheticPrice = price + differ
    syntheticPrices.append(syntheticPrice)
```

Now look at some results:

No noise:




Noise (10 different charts on top of each other):




## Processing Input Charts

First, I got rid of the axis. These should not be used by the model to learn patterns. Then I reduced the input charts dimension by using only their blue colour channel, as the others do not carry any useful information. They carry only the value 255. Also I changed the data type to a np.float32 to reduce memory use and scaled the input to 0 to 1. Unfortunately float 16 or float 8 are not available in Keras. That is especially painful as 1 bit would be enough to store the information for every pixel. The only contain the value 0 (black) or 1 (white).

## Keras CNN

After playing around with some pretrained nets, like the InceptionV3, I found out that the wouldn't produce superior results. It seemed, because they were not trained for charts, there pretrained CNN layers were not very useful. Therefore, I trained some different NN with multiple CNNs, MAxPooling2D, fully connected Dense, Dropout and a softmax output layer. The architecture was inspired by the VGG16 net and works very well. That is especially pleasant considering we only have about 260.000 parameters.

For the hidden layer activation functions, I used ReLu, for the optimizer "Adam" (Adaptive Momentum) and as loss function categorical crossentropy.

Christoph Ostertag
Milan Krivokuća

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_5 (Conv2D) | (None, 215, 332, 4) | 40 |
| conv2d_6 (Conv2D) | (None, 213, 330, 4) | 148 |
| max_pooling2d_3 (MaxPooling2 | (None, 106, 165, 4) | 0 |
| conv2d_7 (Conv2D) | (None, 104, 163, 4) | 148 |
| conv2d_8 (Conv2D) | (None, 102, 161, 4) | 148 |
| max_pooling2d_4 (MaxPooling2 | (None, 51, 80, 4) | 0 |
| flatten_2 (Flatten) | (None, 16320) | 0 |
| dense_3 (Dense) | (None, 16) | 261136 |
| dropout_3 (Dropout) | (None, 16) | 0 |
| dense_4 (Dense) | (None, 16) | 272 |
| dropout_4 (Dropout) | (None, 16) | 0 |
| dense_5 (Dense) | (None, 5) | 85 |

Total params: 261,977
Trainable params: 261,977
Non-trainable params: 0

Christoph Ostertag
Milan Krivokuća

# Results

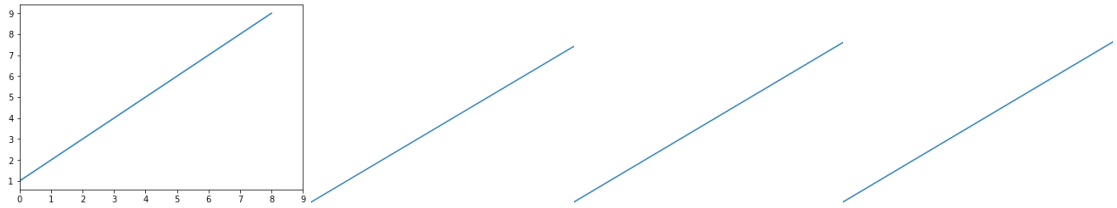
First picture is the original. Last three pictures are with noise.

## Noise 0

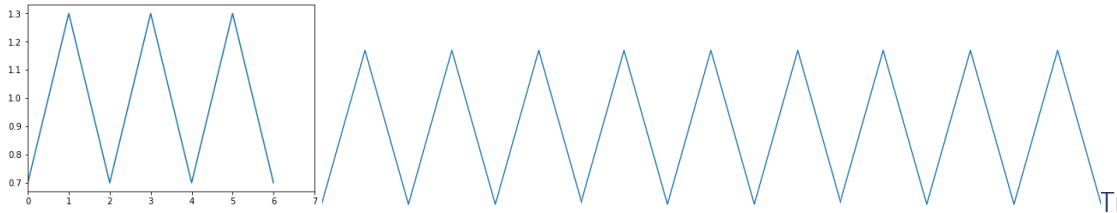
Accuracy: 100%
Test sample size: 10

### None

### Linear



### Triple Top



### Triangle

### Head and Shoulders

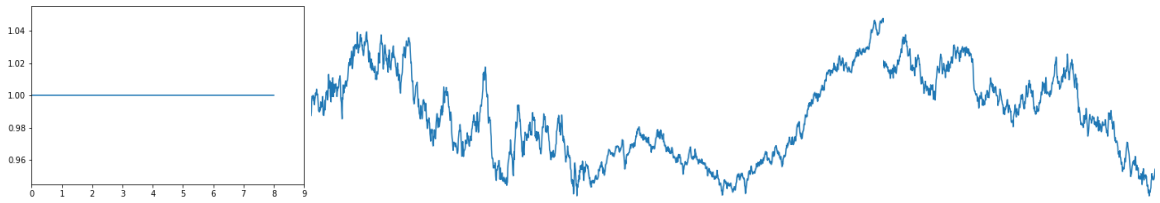
Christoph Ostertag
Milan Krivokuća

## Noise 10

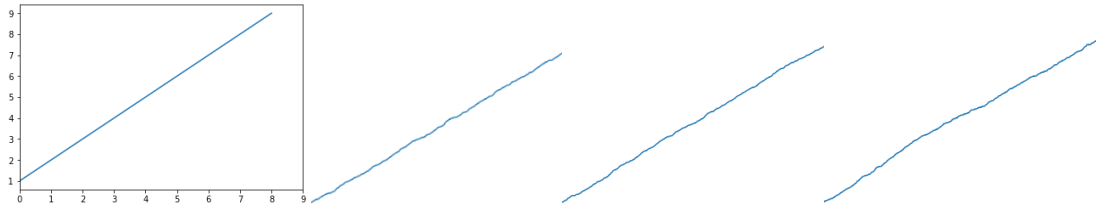
Accuracy: 100%
Test sample size: 1000

### None

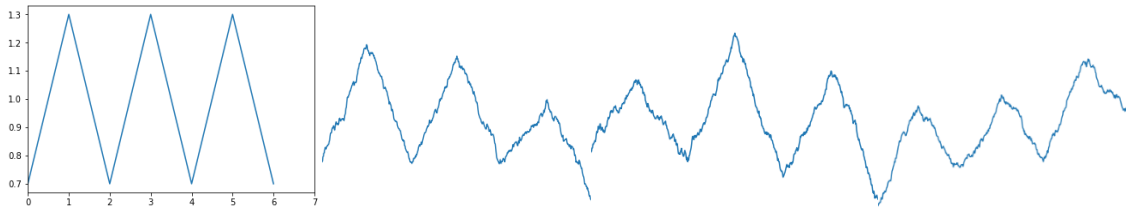
Remark: The line is pulled up and down because the plotting library auto-scales the y-axis. Therefore this can be considered zoomed in.



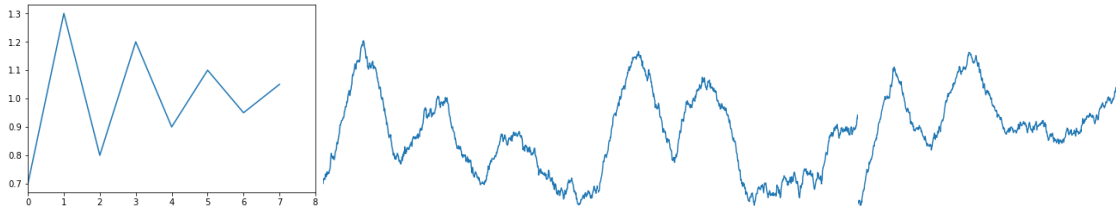
### Linear



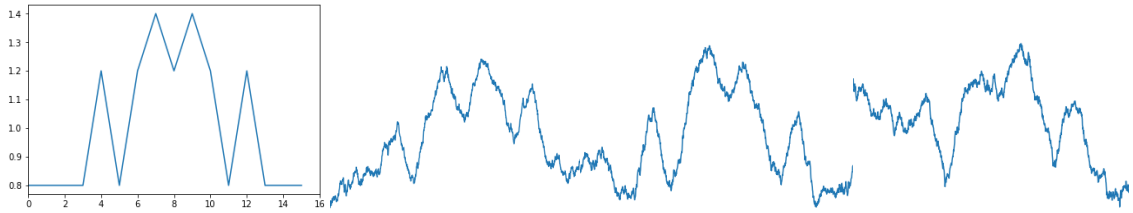
### Triple Top



### Triangle



### Head and Shoulders

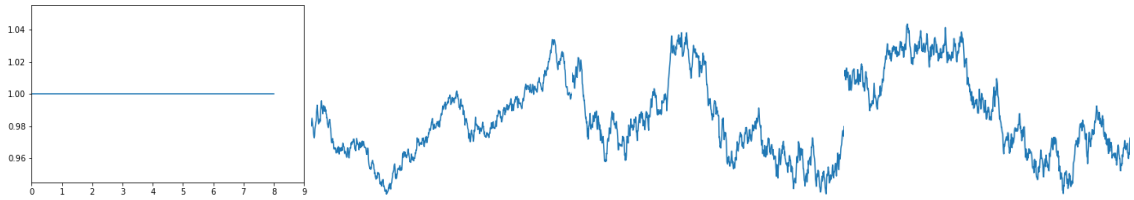


Christoph Ostertag
Milan Krivokuća

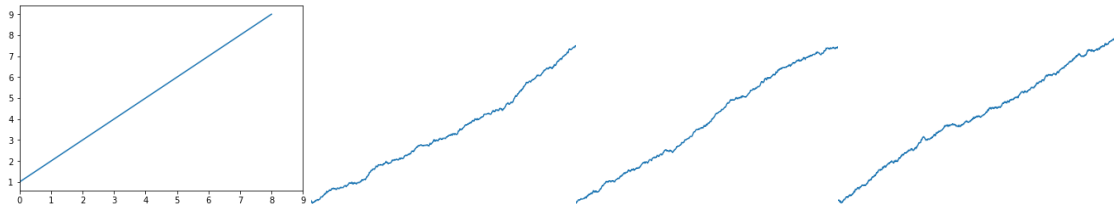
## Noise 30

Accuracy: 96%
Test sample size: 1000

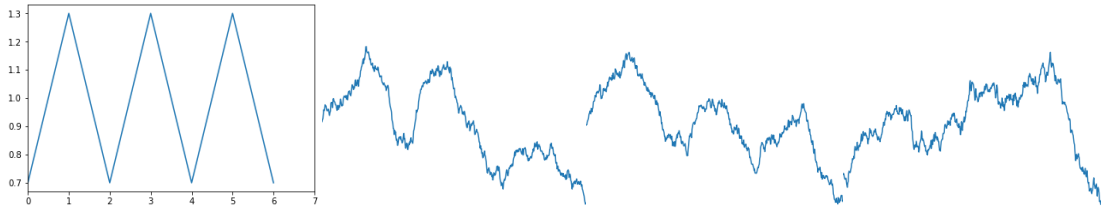
### None



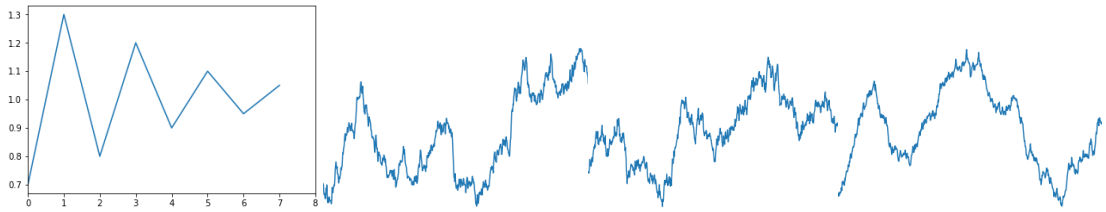
### Linear



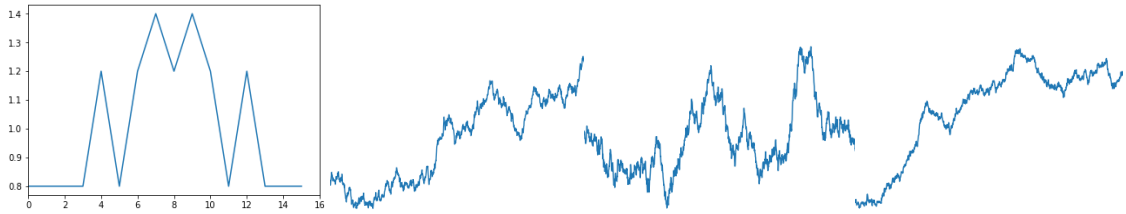
### Triple Top



### Triangle



### Head and Shoulders

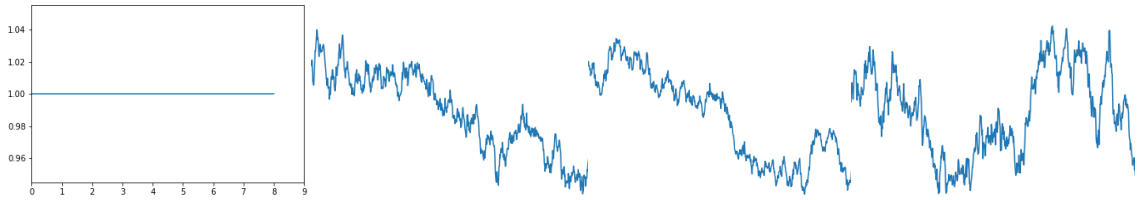


Christoph Ostertag
Milan Krivokuća

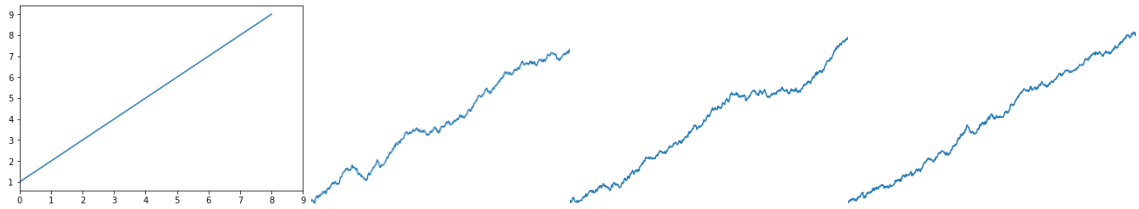
# Noise 50

Accuracy: 74%
Test sample size: 1000

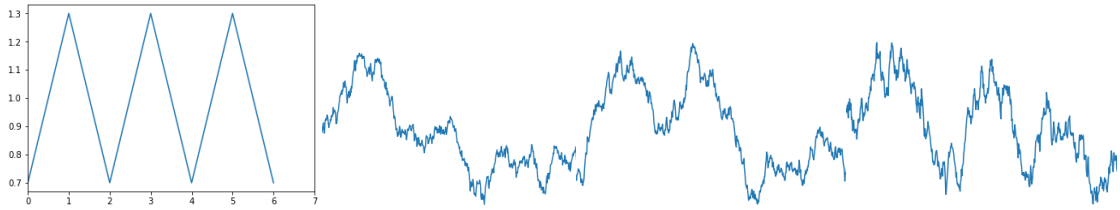
## None



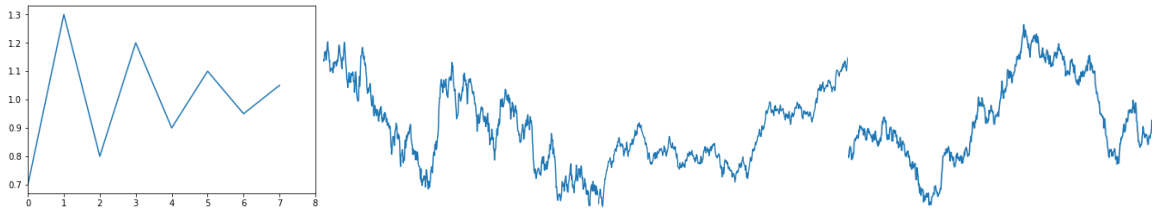
## Linear



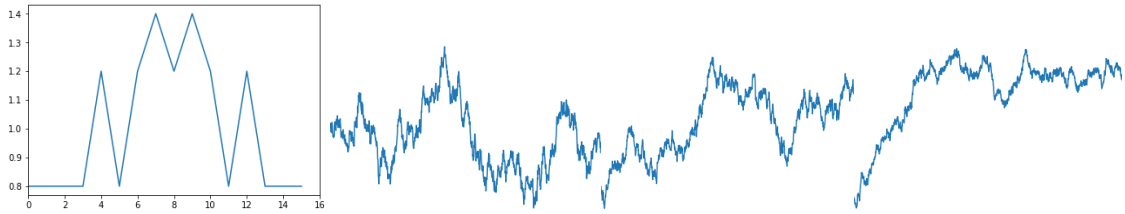
## Triple Top



## Triangle



## Head and Shoulders

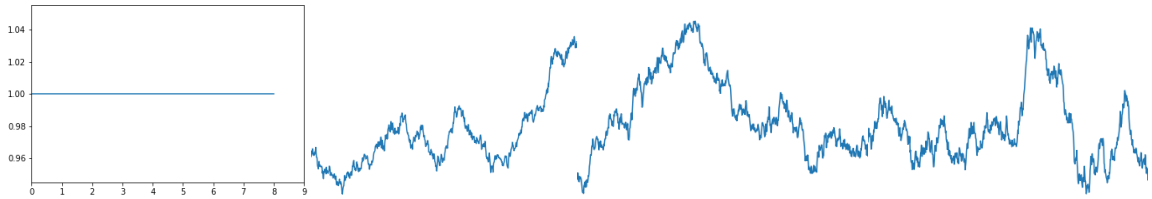


Christoph Ostertag
Milan Krivokuća

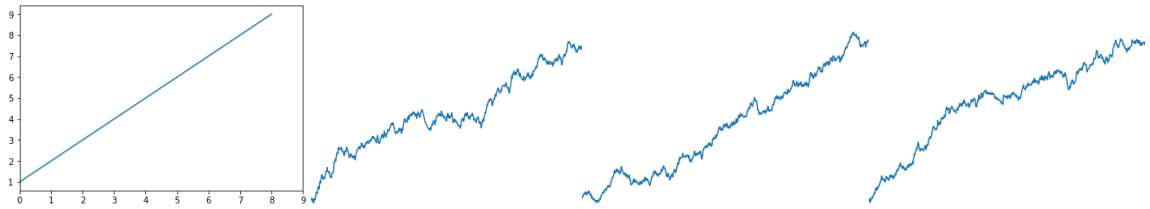
# Noise 100

Accuracy: 58%
Test sample size: 1000

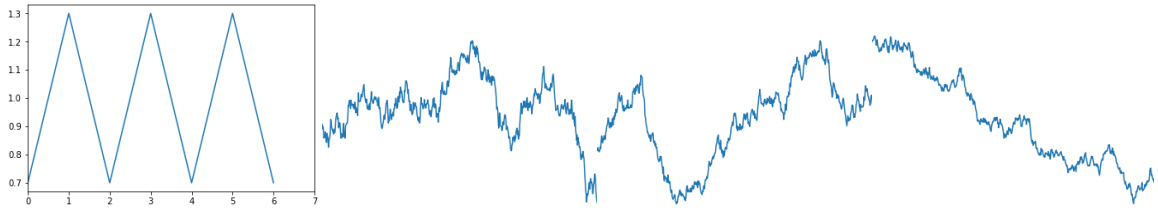
## None



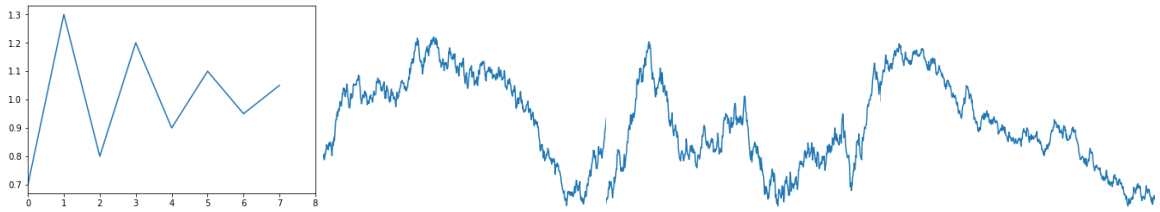
## Linear



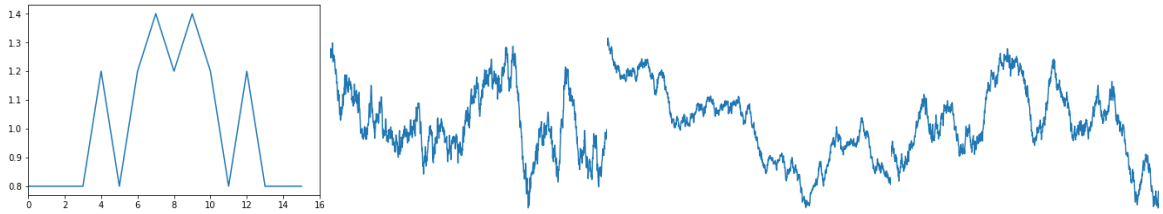
## Triple Top



## Triangle



## Head and Shoulders



### Interpretation

Would our algorithm just be guessing it would be correct about 20% of the time, as every chart appears exactly the same time in the training and test data. As we see beginning with a noise of 30 classification can become quite tricky for the human eye. Our CNN still classifies 96% of the test data correctly. At a noise of 100, where our model classifies 58% correctly, the charts all look more or less the same for the human eye, with the exception of the linear chart. A benchmark with the human capability of detecting chart patterns would be interesting and may prove or disprove the usefulness of technical analysts.

## Conclusion

These results show that the technical analyses of trends may not only be possible by human chart analysts but could also be done by a machine. Furthermore, a CNN that would be used to binary categorize up or down trends, would not be limited to patterns we teach this model, but it could learn its own patterns and decide for itself which are relevant, and which are not.