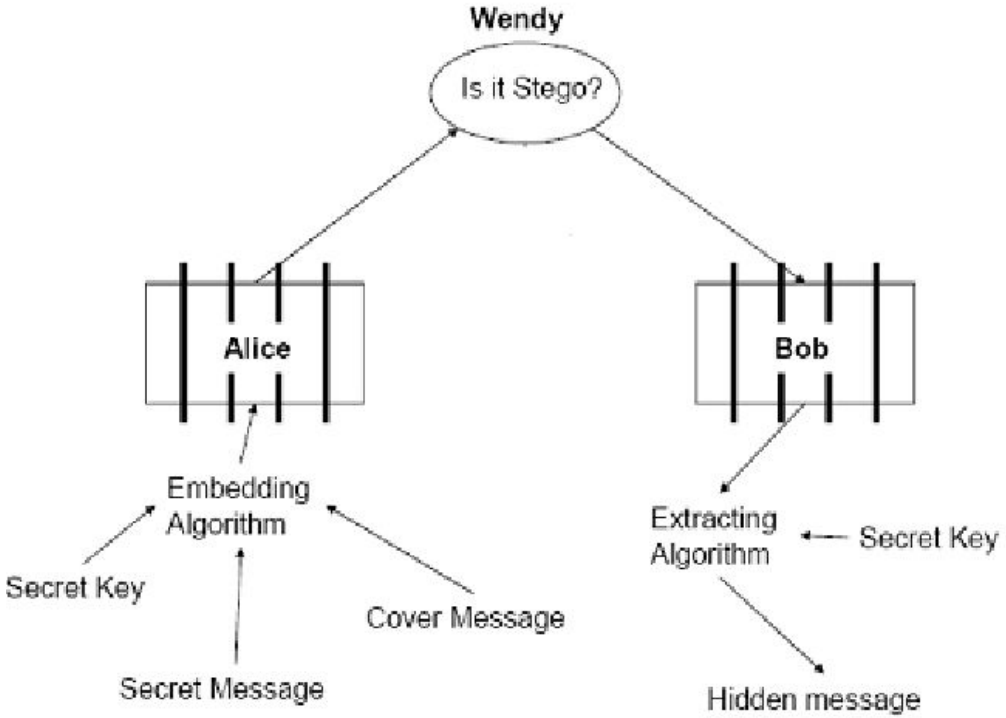
Steganography detection with ML

Illia Andrieiev

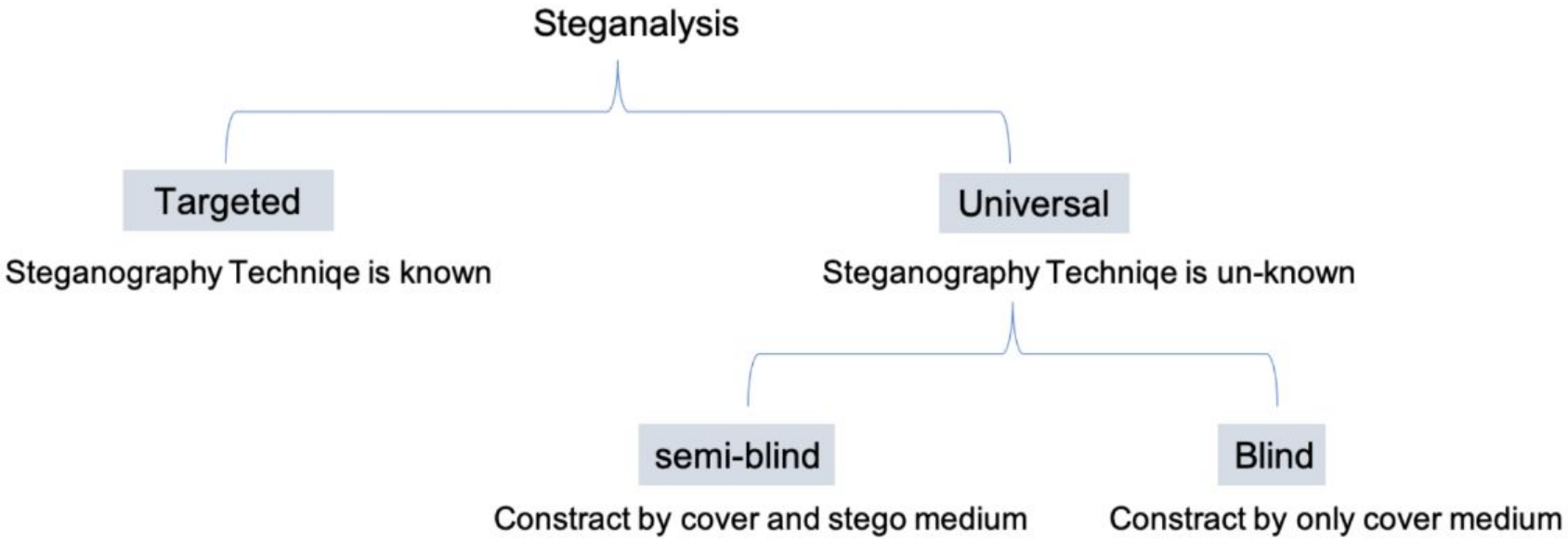
Steganography

Steganography is the practice of hiding a secret message inside of something that is not secret. The purpose of steganography is to conceal and deceive. Where *cryptography* is a science that largely enables privacy, *steganography* is a practice that enables secrecy – and deceit.

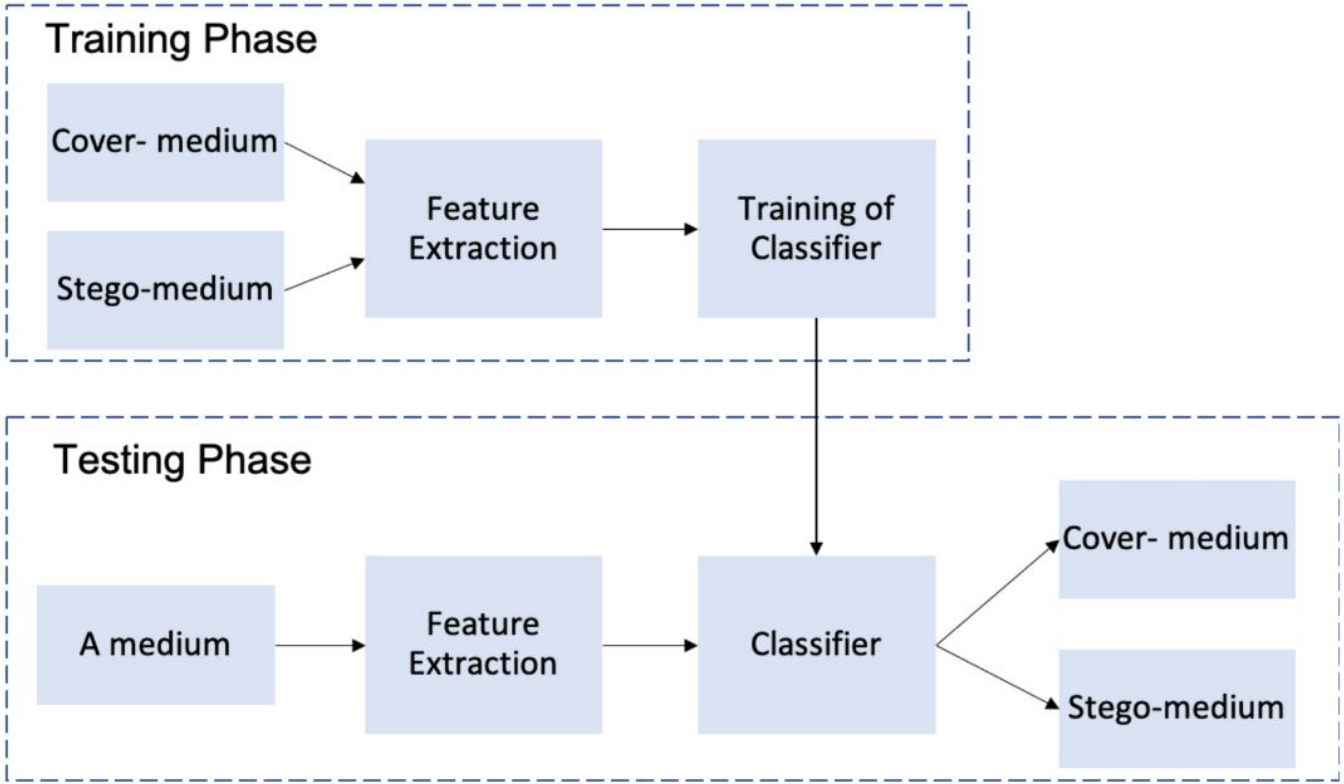
Problem statement



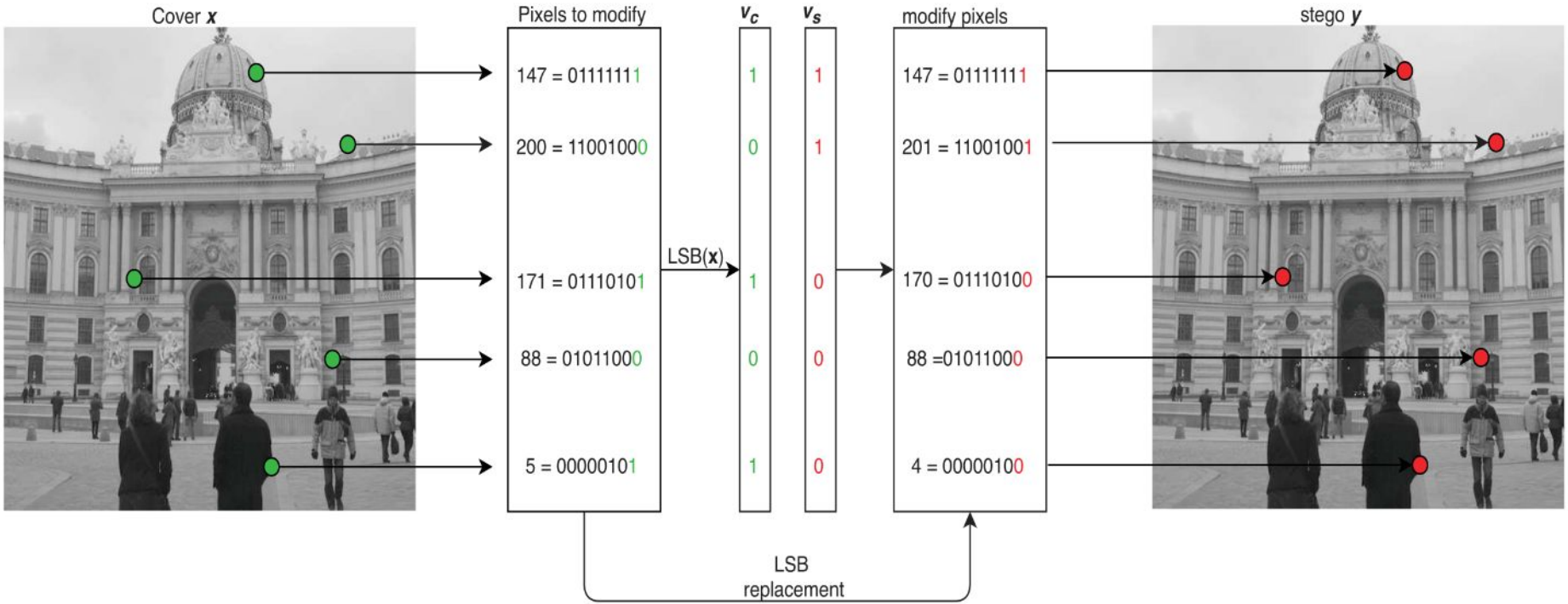
Problem statement



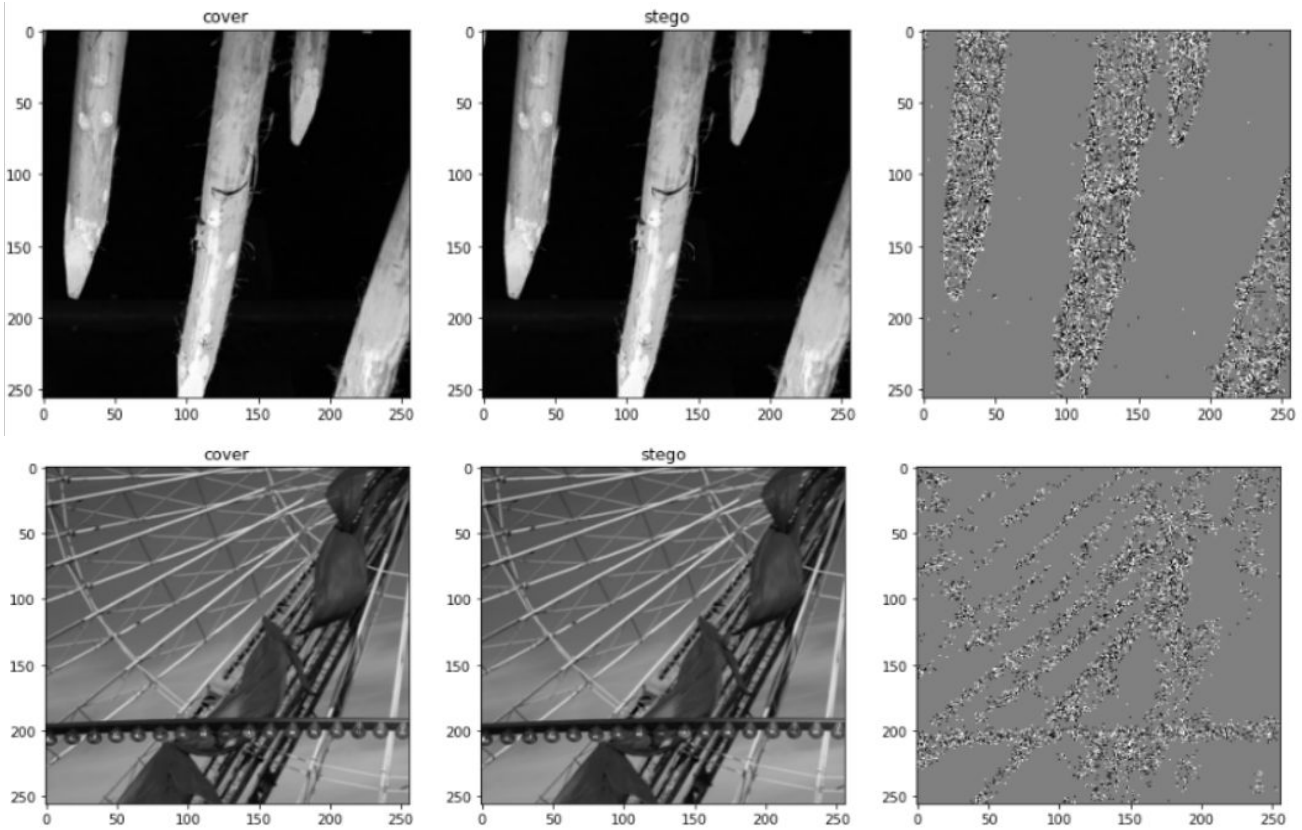
Project scheme



Example algorithm



Adaptive Steganography



Adaptive steganography

1. determine which positions are best for the embedding,
2. encode the message,
3. embed the message.

Adaptive steganography

$$D : \mathcal{C} \times \mathcal{S} \rightarrow [0, \infty[$$

$$D(\boldsymbol{x}, \boldsymbol{y}) = ||f(\boldsymbol{x}) - f(\boldsymbol{y})||$$

$$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} \rho_i \, |x_i - y_i|$$

$$\boldsymbol{\rho} = \{\rho_i \in [0, \infty[\}_i^n$$

$$\rho_i = D(\boldsymbol{X}, \boldsymbol{X}_{\sim}\boldsymbol{X}_i)$$

Adaptive algorithms (Spatial domain)

- HUGO
- WOW
- S-UNIWARD

Dataset generation

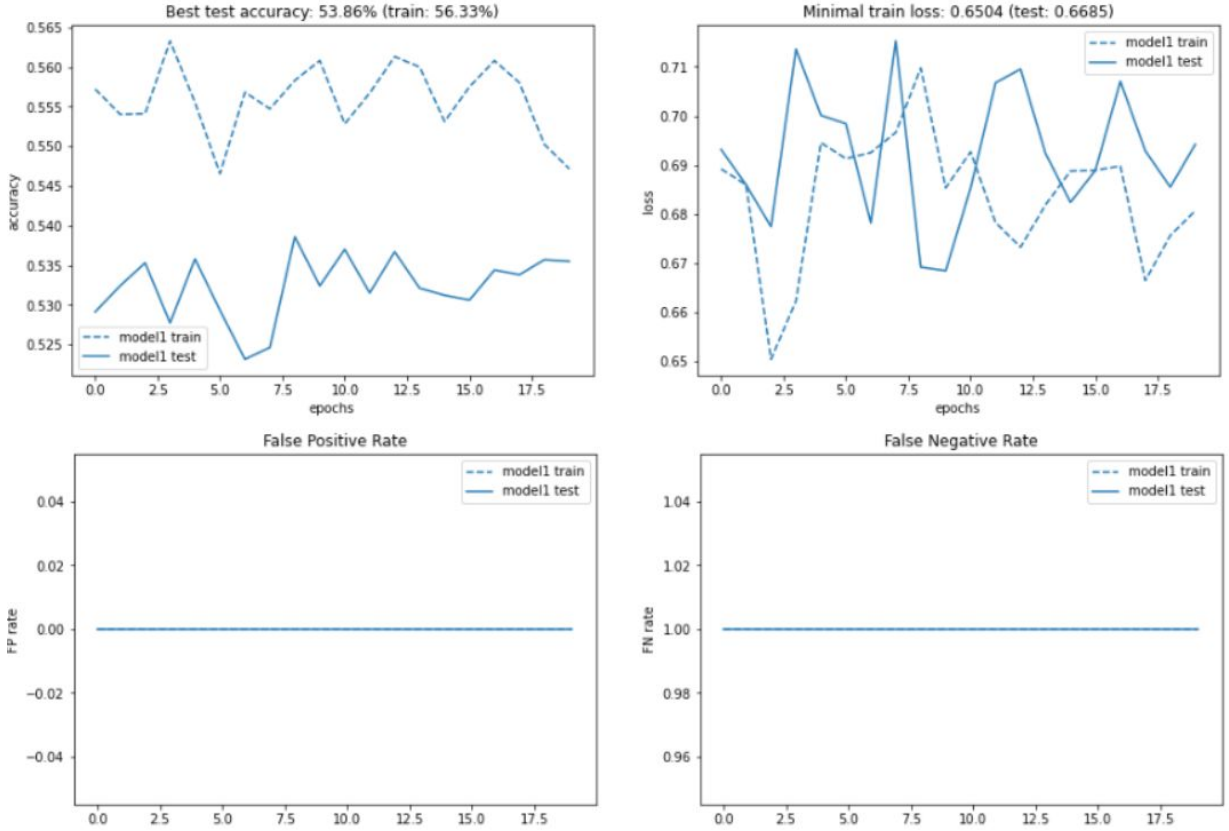
Bossbase 1.01 - 10 000 grayscale pictures 512x512

1. Split each picture into 4 equal parts
2. train/test split with proportions 60%/40%
3. Create corresponding stego pair with random adaptive algorithm (uniform distribution).

Baseline CNN

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 256, 256]	416
Tanh-2	[-1, 16, 256, 256]	0
AvgPool2d-3	[-1, 16, 128, 128]	0
Conv2d-4	[-1, 30, 128, 128]	12,030
Tanh-5	[-1, 30, 128, 128]	0
AvgPool2d-6	[-1, 30, 64, 64]	0
Conv2d-7	[-1, 32, 64, 64]	24,032
ReLU-8	[-1, 32, 64, 64]	0
AvgPool2d-9	[-1, 32, 32, 32]	0
Conv2d-10	[-1, 64, 32, 32]	18,496
ReLU-11	[-1, 64, 32, 32]	0
AvgPool2d-12	[-1, 64, 16, 16]	0
Conv2d-13	[-1, 128, 16, 16]	73,856
ReLU-14	[-1, 128, 16, 16]	0
AvgPool2d-15	[-1, 128, 1, 1]	0
Squeeze-16	[-1, 128]	0
Linear-17	[-1, 256]	33,024
ReLU-18	[-1, 256]	0
Linear-19	[-1, 2]	514

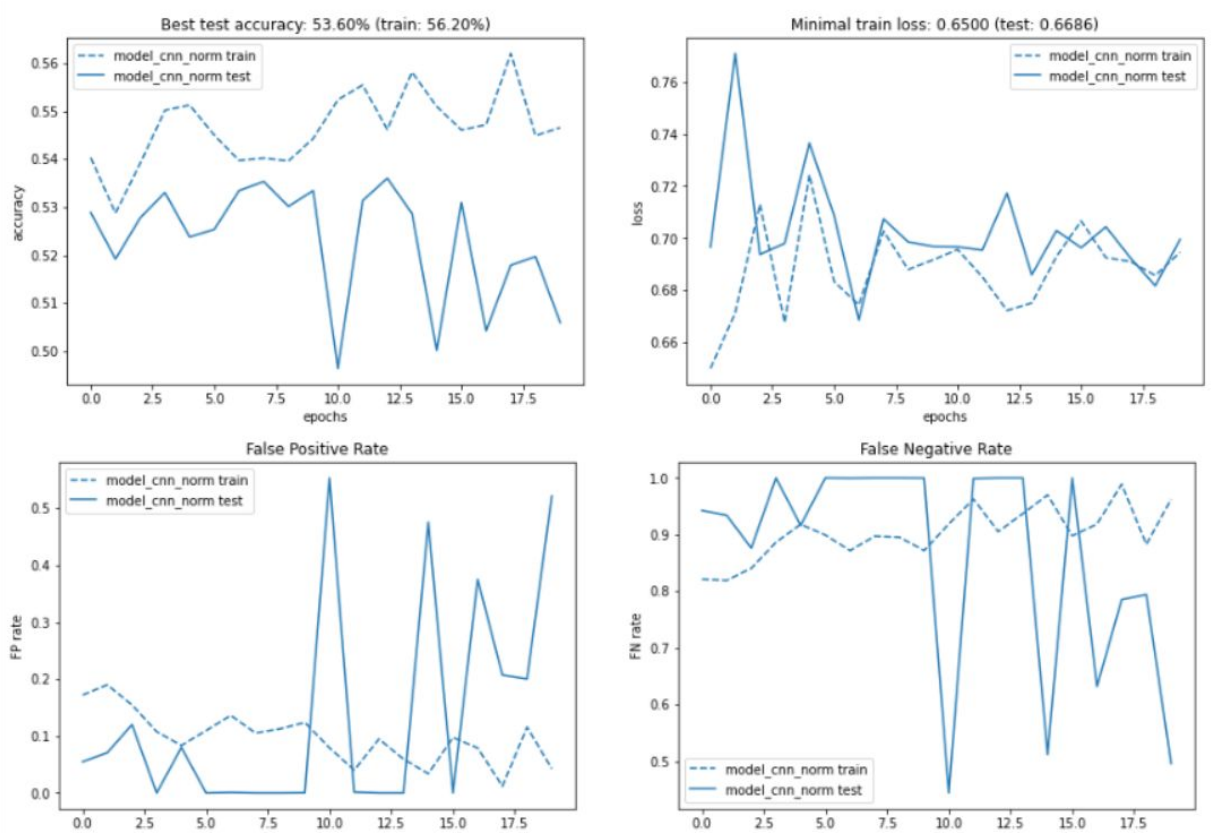
Baseline CNN



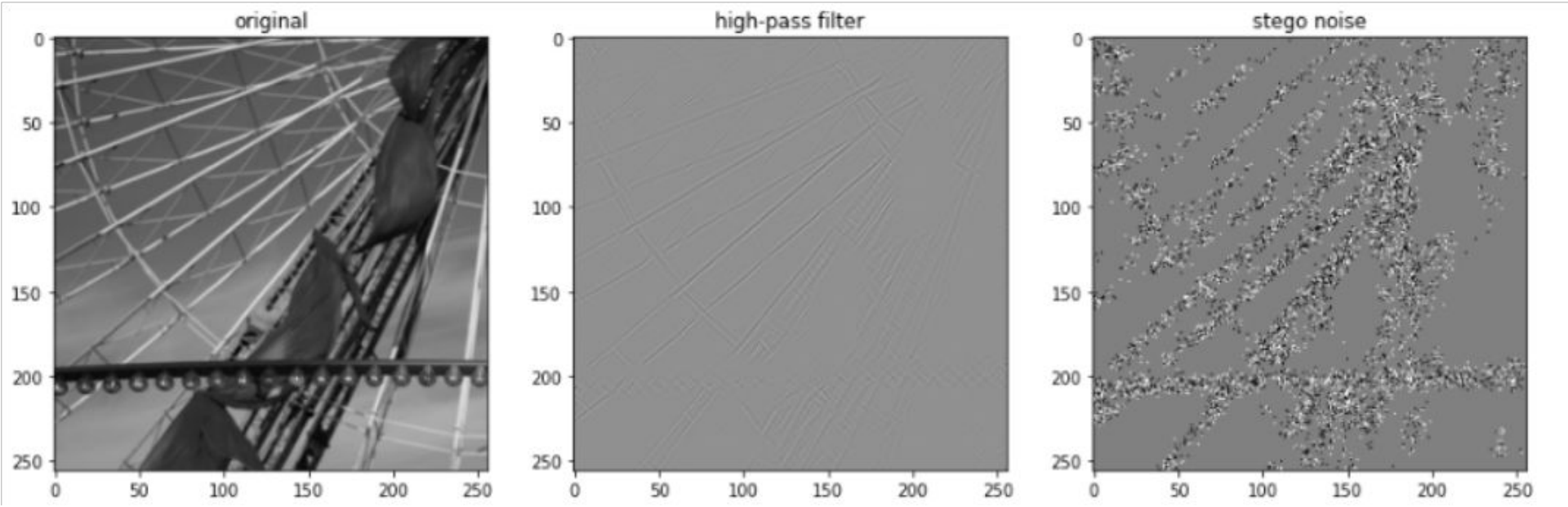
Baseline CNN + Batch Normalization

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 256, 256]	208
BatchNorm2d-2	[-1, 8, 256, 256]	16
Tanh-3	[-1, 8, 256, 256]	0
AvgPool2d-4	[-1, 8, 128, 128]	0
Conv2d-5	[-1, 16, 128, 128]	3,216
BatchNorm2d-6	[-1, 16, 128, 128]	32
Tanh-7	[-1, 16, 128, 128]	0
AvgPool2d-8	[-1, 16, 64, 64]	0
Conv2d-9	[-1, 32, 64, 64]	12,832
BatchNorm2d-10	[-1, 32, 64, 64]	64
ReLU-11	[-1, 32, 64, 64]	0
AvgPool2d-12	[-1, 32, 32, 32]	0
Conv2d-13	[-1, 64, 32, 32]	18,496
BatchNorm2d-14	[-1, 64, 32, 32]	128
ReLU-15	[-1, 64, 32, 32]	0
AvgPool2d-16	[-1, 64, 16, 16]	0
Conv2d-17	[-1, 128, 16, 16]	73,856
BatchNorm2d-18	[-1, 128, 16, 16]	256
ReLU-19	[-1, 128, 16, 16]	0
AvgPool2d-20	[-1, 128, 1, 1]	0
Squeeze-21	[-1, 128]	0
Linear-22	[-1, 32]	4,128
ReLU-23	[-1, 32]	0
Linear-24	[-1, 2]	66

Baseline CNN + Batch Normalization



High-Pass filtering



High-Pass Filtering

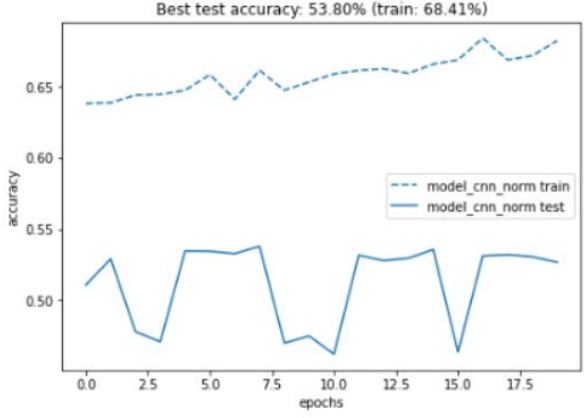
```
kernel = torch.tensor([
        [-1,2,-2,2,-1],
        [2,-6,8,-6,2],
        [-2,8,-12,8,-2],
        [2,-6,8,-6,2],
        [-1,2,-2,2,-1]])/12.
```

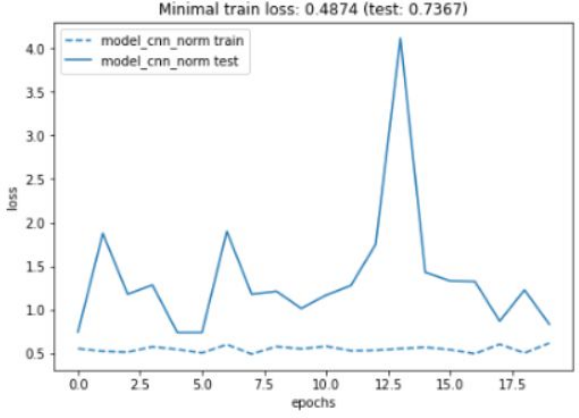
```
def forward(self, x):
  out = torch.nn.functional.conv2d(x, self.kernel, stride=1, padding=2)
  return out
```

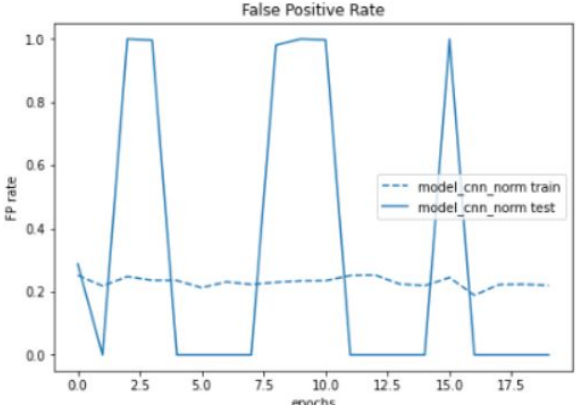
CNN + BN + High-Pass filtering

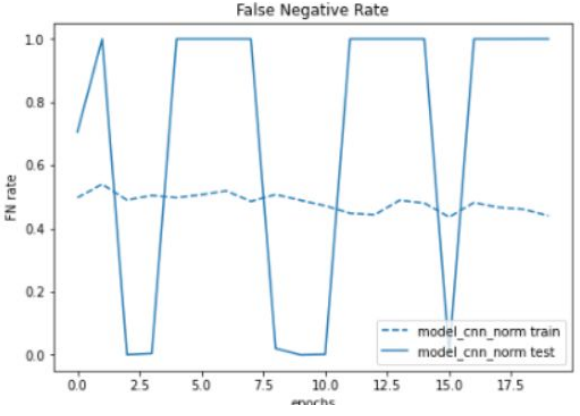
Layer (type)	Output Shape	Param #
HPF-1	[-1, 1, 256, 256]	0
Conv2d-2	[-1, 8, 256, 256]	208
BatchNorm2d-3	[-1, 8, 256, 256]	16
Tanh-4	[-1, 8, 256, 256]	0
AvgPool2d-5	[-1, 8, 128, 128]	0
Conv2d-6	[-1, 16, 128, 128]	3,216
BatchNorm2d-7	[-1, 16, 128, 128]	32
Tanh-8	[-1, 16, 128, 128]	0
AvgPool2d-9	[-1, 16, 64, 64]	0
Conv2d-10	[-1, 32, 64, 64]	12,832
BatchNorm2d-11	[-1, 32, 64, 64]	64
ReLU-12	[-1, 32, 64, 64]	0
AvgPool2d-13	[-1, 32, 32, 32]	0
Conv2d-14	[-1, 64, 32, 32]	18,496
BatchNorm2d-15	[-1, 64, 32, 32]	128
ReLU-16	[-1, 64, 32, 32]	0
AvgPool2d-17	[-1, 64, 16, 16]	0
Conv2d-18	[-1, 128, 16, 16]	73,856
BatchNorm2d-19	[-1, 128, 16, 16]	256
ReLU-20	[-1, 128, 16, 16]	0
AvgPool2d-21	[-1, 128, 1, 1]	0
Squeeze-22	[-1, 128]	0
Linear-23	[-1, 32]	4,128
ReLU-24	[-1, 32]	0
Linear-25	[-1, 2]	66

CNN + BN + High-Pass filtering

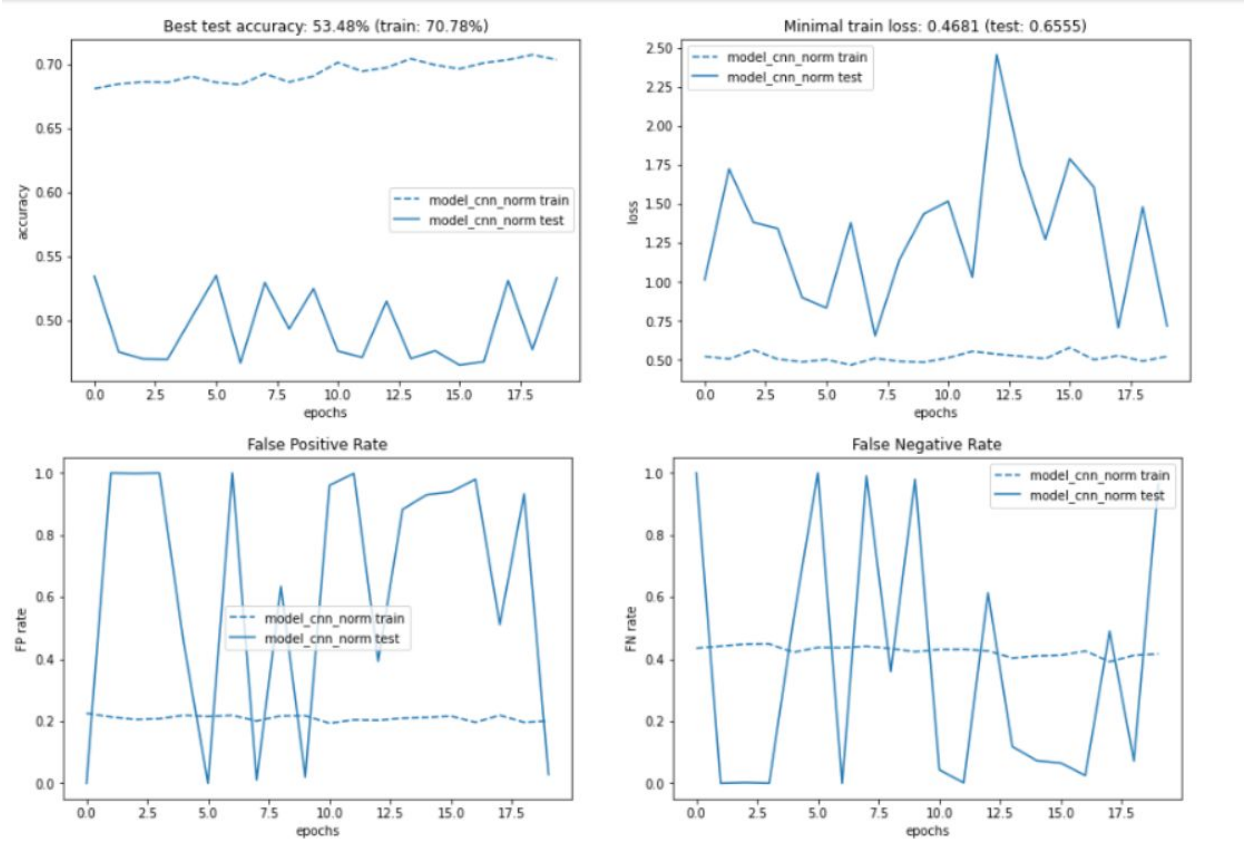








… + Samper!


Best test accuracy: 53.48% (train: 70.78%)
accuracy
epochs
model_cnn_norm train
model_cnn_norm test
Minimal train loss: 0.4681 (test: 0.6555)
loss
epochs
model_cnn_norm train
model_cnn_norm test
False Positive Rate
FP rate
epochs
model_cnn_norm train
model_cnn_norm test
False Negative Rate
FN rate
epochs
model_cnn_norm train
model_cnn_norm test

CNN + BN + High-Pass filtering, but regularized

Layer (type)	Output Shape	Param #
HPF-1	[-1, 1, 256, 256]	0
Conv2d-2	[-1, 8, 256, 256]	200
BatchNorm2d-3	[-1, 8, 256, 256]	16
Tanh-4	[-1, 8, 256, 256]	0
AvgPool2d-5	[-1, 8, 128, 128]	0
Conv2d-6	[-1, 16, 128, 128]	3,200
BatchNorm2d-7	[-1, 16, 128, 128]	32
Tanh-8	[-1, 16, 128, 128]	0
AvgPool2d-9	[-1, 16, 64, 64]	0
Conv2d-10	[-1, 32, 64, 64]	512
BatchNorm2d-11	[-1, 32, 64, 64]	64
ReLU-12	[-1, 32, 64, 64]	0
AvgPool2d-13	[-1, 32, 32, 32]	0
Conv2d-14	[-1, 64, 32, 32]	2,048
BatchNorm2d-15	[-1, 64, 32, 32]	128
ReLU-16	[-1, 64, 32, 32]	0
AvgPool2d-17	[-1, 64, 1, 1]	0
Squeeze-18	[-1, 64]	0
ReLU-19	[-1, 64]	0
Linear-20	[-1, 2]	130

CNN + BN + High-Pass filtering, but regularized