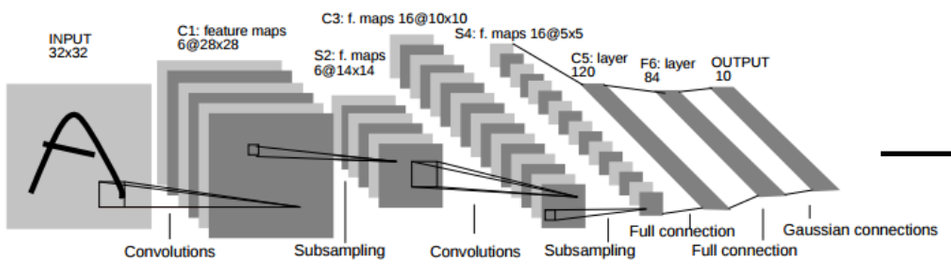
Safety in AI:
Robustness metrics for Deep Learning predictions

Davide Posillipo - Data Science Retreat (Batch 15)

https://github.com/DavidePosillipo/AI_Safety
https://www.linkedin.com/in/davide-posillipo
dav.posillipo@gmail.com

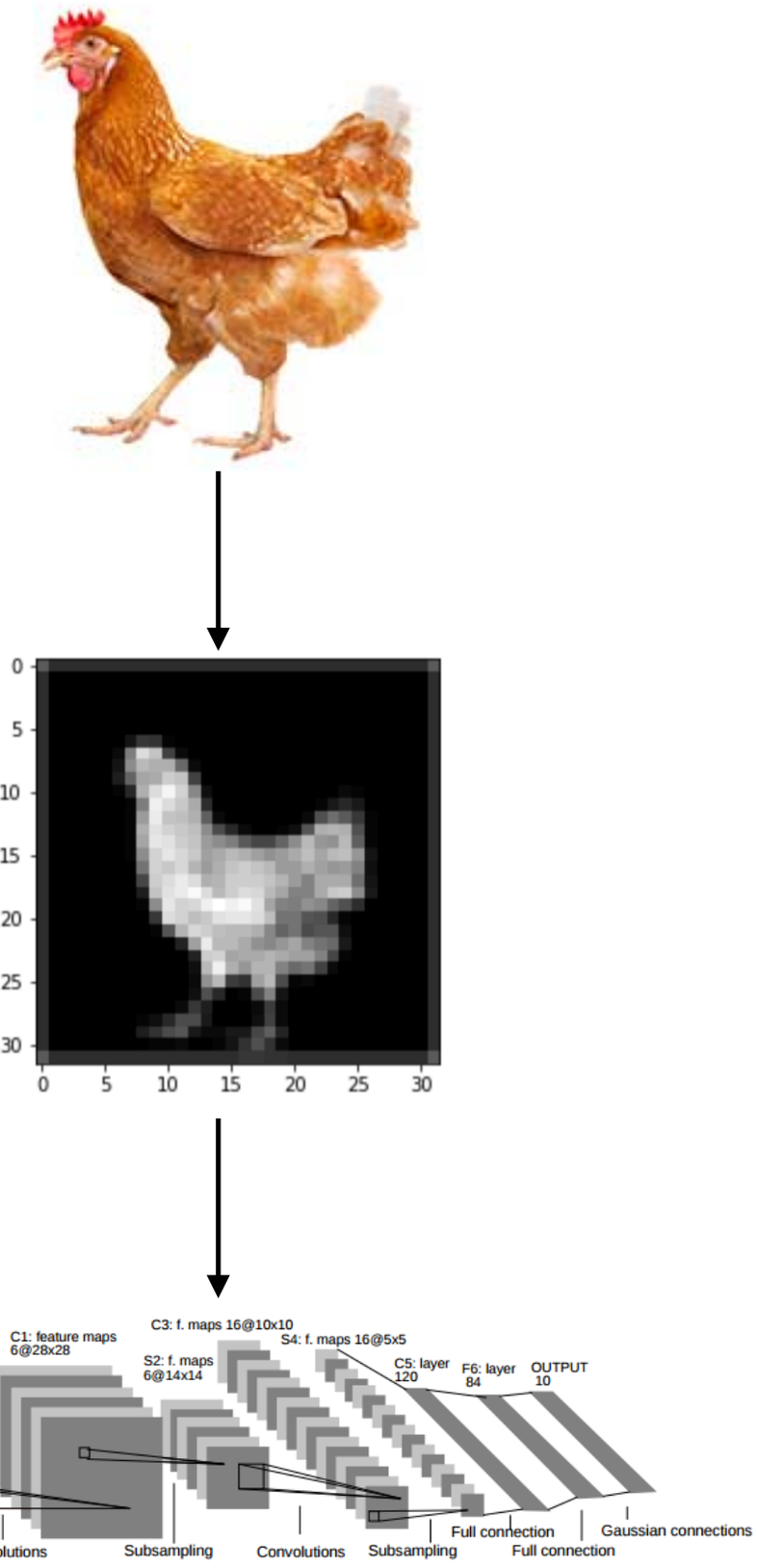
MNIST: a simple case

- MNIST: solved problem
- LeNet: Deep Convolutional Neural Network, 99% accuracy



Predicted digit

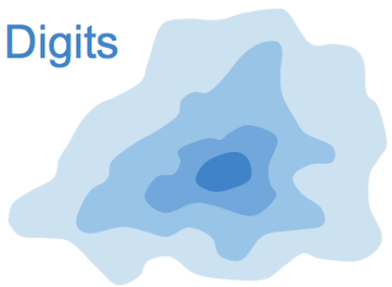
What happens if we pass a chicken to LeNet?



The chicken comes from a different distribution.

Chicken

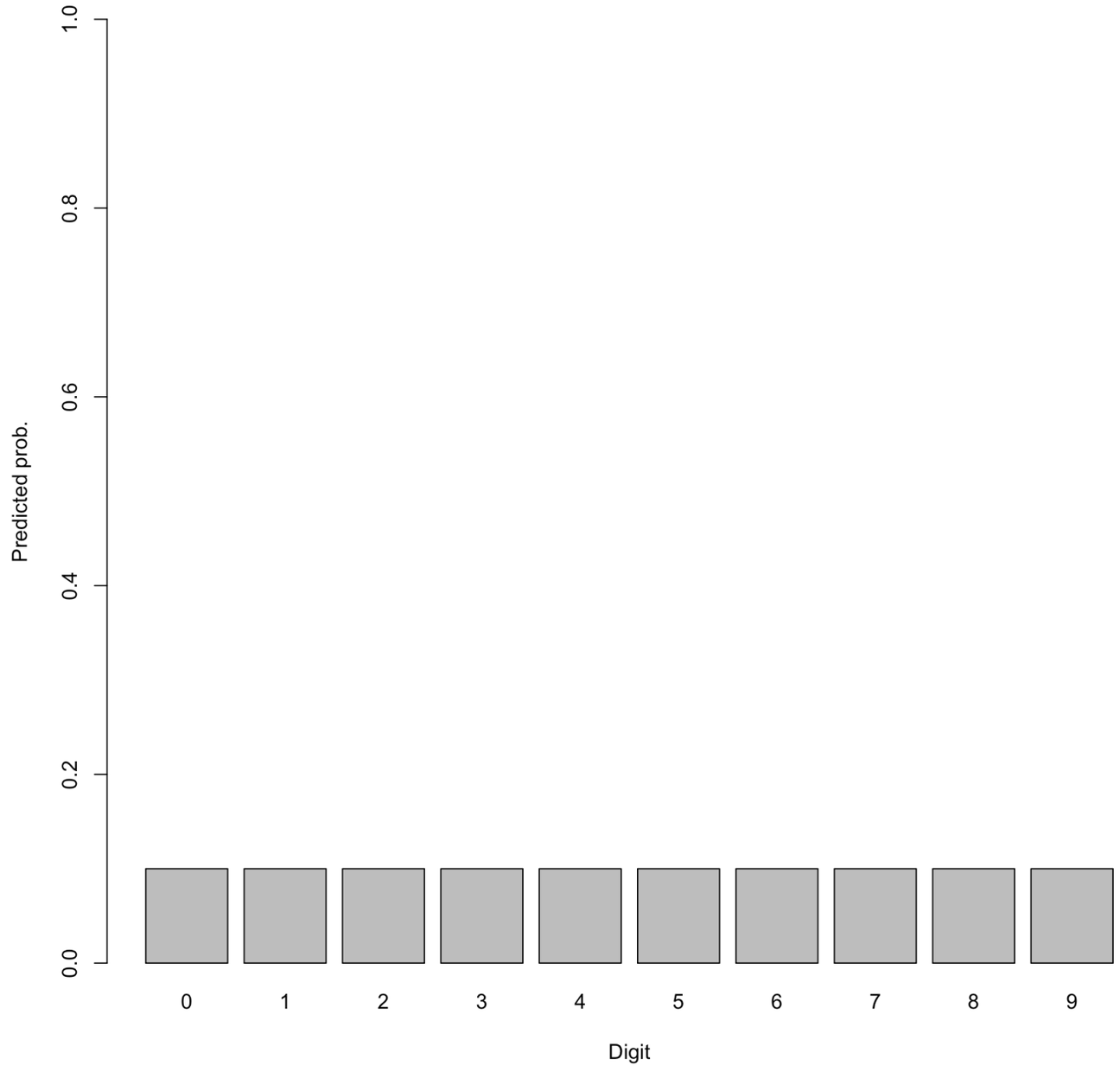
✖

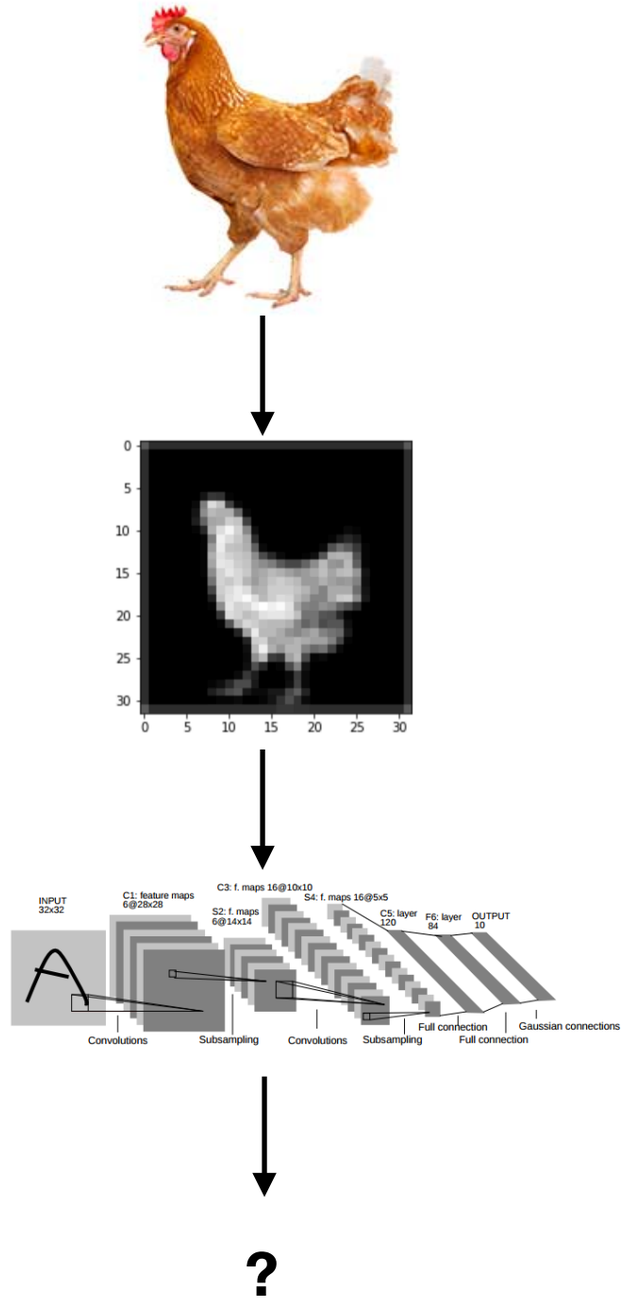


What should the net do?

What happens if we pass a chicken to LeNet?

We **expect** this output:

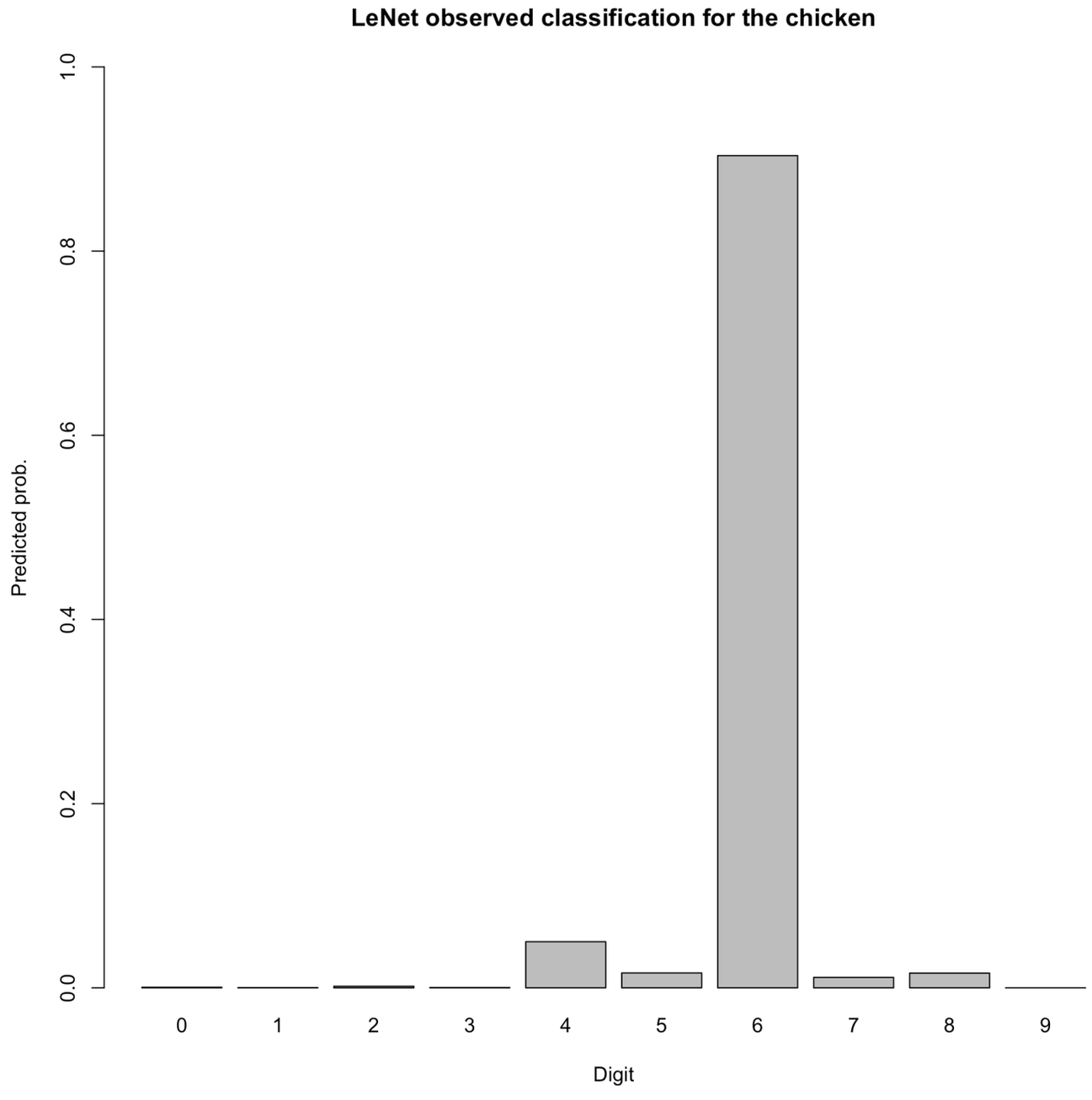


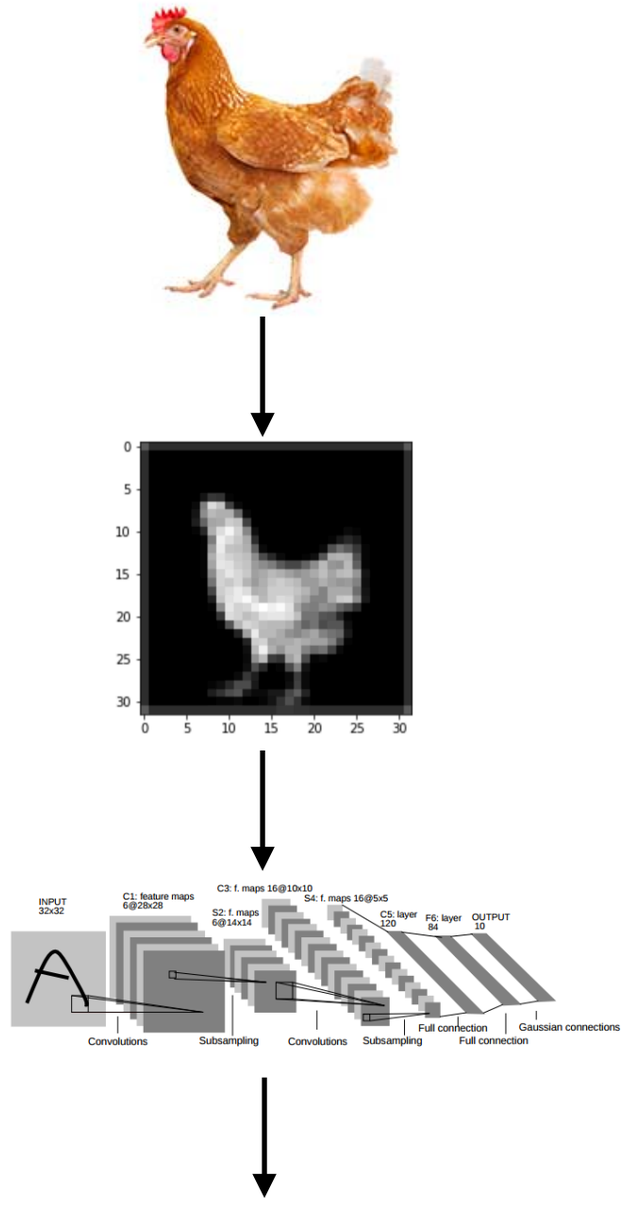


?

What happens if we pass a chicken to LeNet?

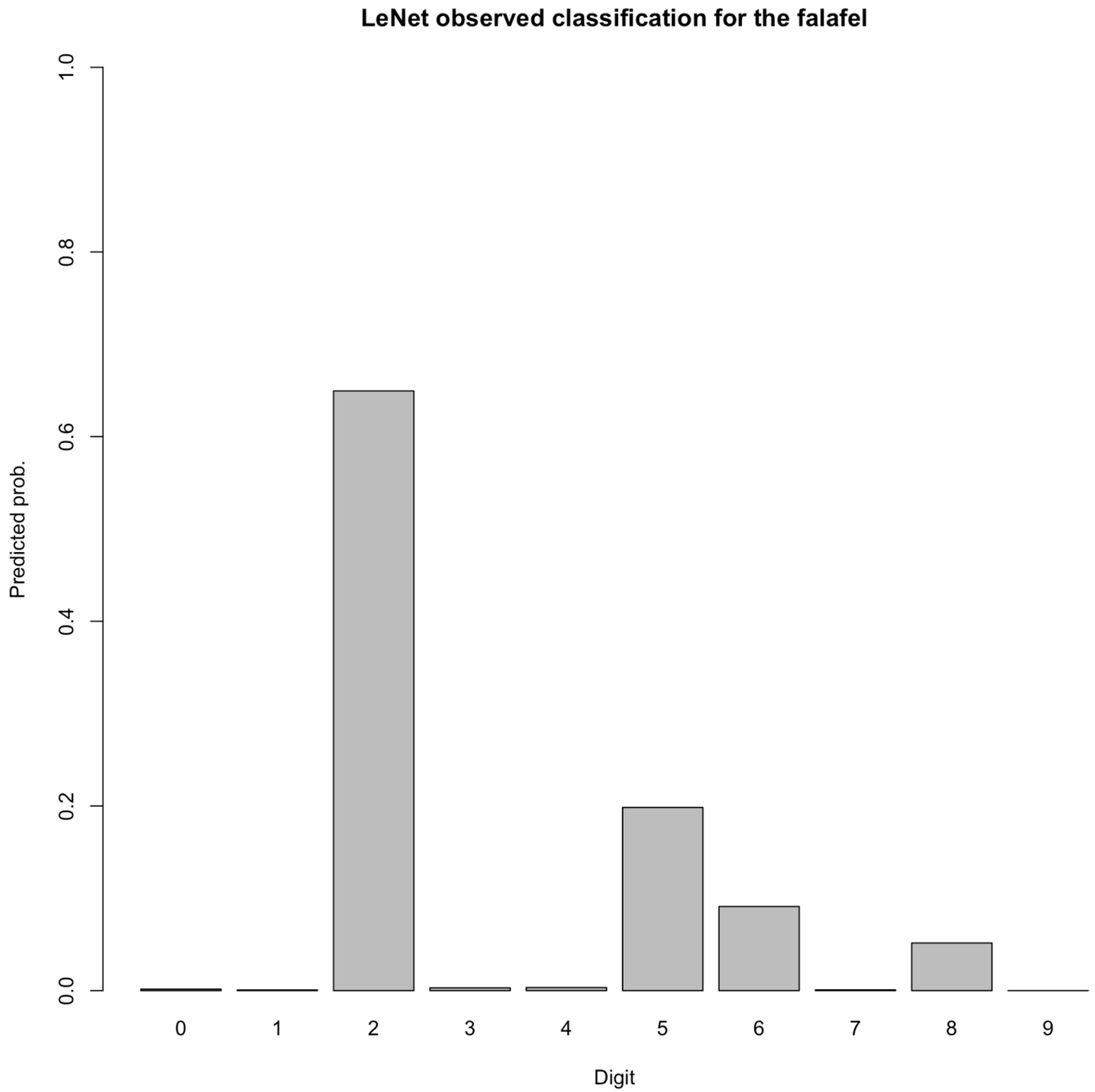
… but we **obtain** this result:

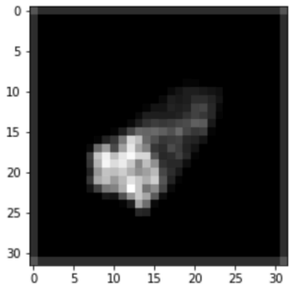


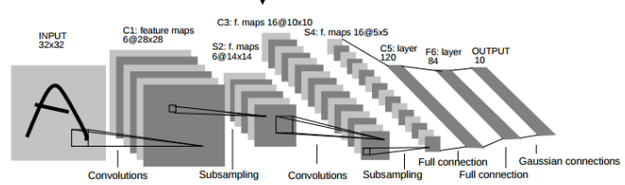


6, with probability 0.9

…and the same happens with a falafel….







2, with probability 0.65

A less obvious example: Fashion MNIST

63.4% of examples are classified with a probability greater than 0.99

74.3% of examples are classified with a probability greater than 0.95

88.9% of examples are classified with a probability greater than 0.75

LeNet (trained for MNIST) produces “confident” predictions for the Fashion MNIST

Can we trust deep learning predictions?

- There are scenarios where it's crucial to avoid meaningless predictions (e.g. healthcare, high precision manufacturing, autonomous driving…): **safety in AI**

- No prediction better than random guessing

- Need of some **robustness metrics**

```
if robustness_metrics < threshold:
    raise NotReliableClassification("Not reliable classification", y_predicted)
else:
    return y_predicted
```

Robustness metrics: how should it be?

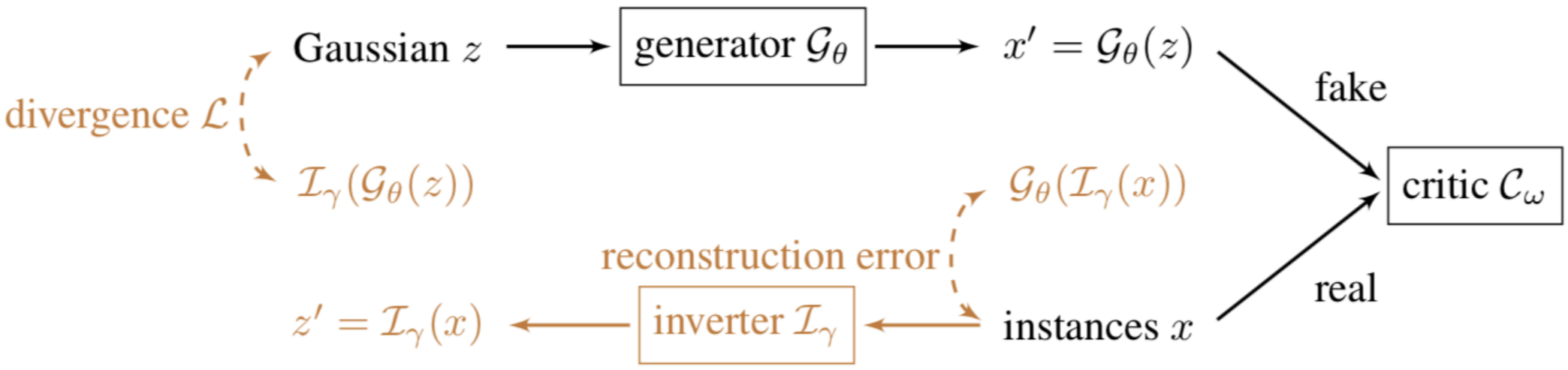
A **robustness metrics** should tell us if the prediction is reliable or if it is likely a random guessing

- Computable without labels
- Computable in streaming
- Low false positive rate, but also low false negative rate
  - High effectiveness in detecting the anomaly points (low false negative rate)
  - Low rate of discarded “good predictions” (low false positive rate)

Generative Adversarial Network (Wasserstein GAN + Inverter)

- Train a Generator (from random noise to the input space)
- Train a Discriminator (it tries to understand if a point is true or generated by the Generator)
    - The Generator learns how to fool the Discriminator
- Train an Inverter (from the input space to the latent representation)
    - The Inverter learns how the Generator maps from the random noise to the input space

Generative Adversarial Network (Wasserstein GAN + Inverter)



WGAN Loss Function

$$\min_\theta \max_\omega \mathbb{E}_{x\sim p_x(x)}[\mathcal{C}_\omega(x)] - \mathbb{E}_{z\sim p_z(z)}[\mathcal{C}_\omega(\mathcal{G}_\theta(z))]$$

Inverter Loss Function

$$\min_\gamma \mathbb{E}_{x\sim p_x(x)}\|\mathcal{G}_\theta(\mathcal{I}_\gamma(x)) - x\| + \lambda \cdot \mathbb{E}_{z\sim p_z(z)}[\mathcal{L}(z, \mathcal{I}_\gamma(\mathcal{G}_\theta(z)))]$$

WGAN + Inverter: robustness metrics

- Given x, find the closest point to x in the latent representation that is able to fool the classifier

$$z^* = \underset{\tilde{z}}{\operatorname{argmin}} \|\tilde{z} - \mathcal{I}_\gamma(x)\| \text{ s.t. } f(\mathcal{G}_\theta(\tilde{z})) \neq f(x)$$

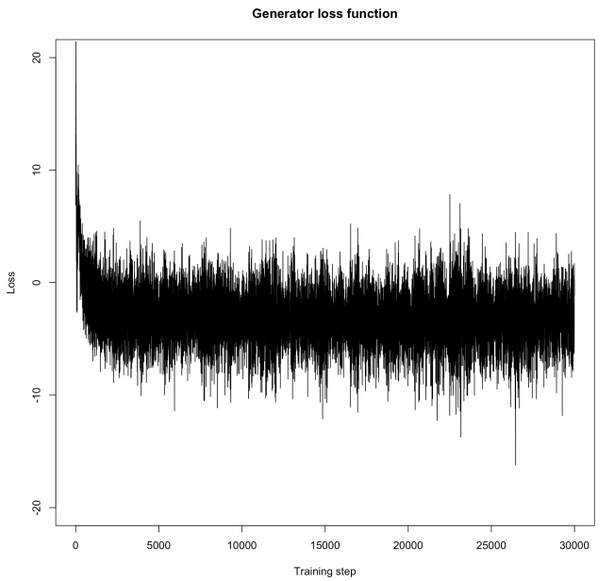
- Robustness metrics: the distance (delta_z) between these two points, in the latent representation
  - If delta_z is “small”, the classifier is not confident about its prediction
  - If delta_z is “big”, the classifier is confident about the prediction

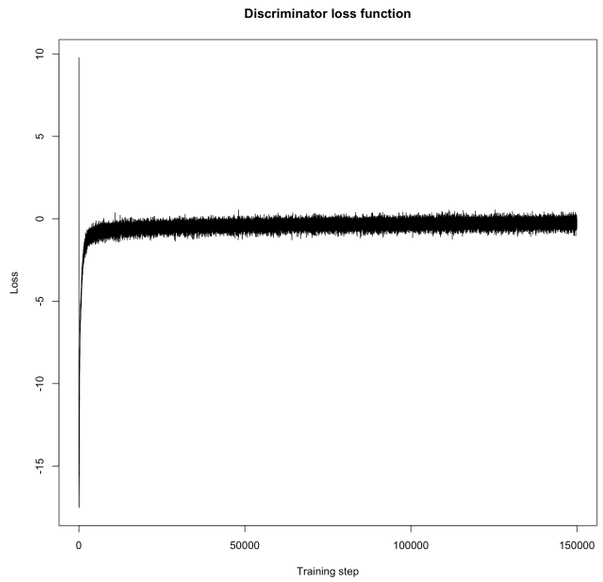
Rule: if delta_z < threshold then do not predict

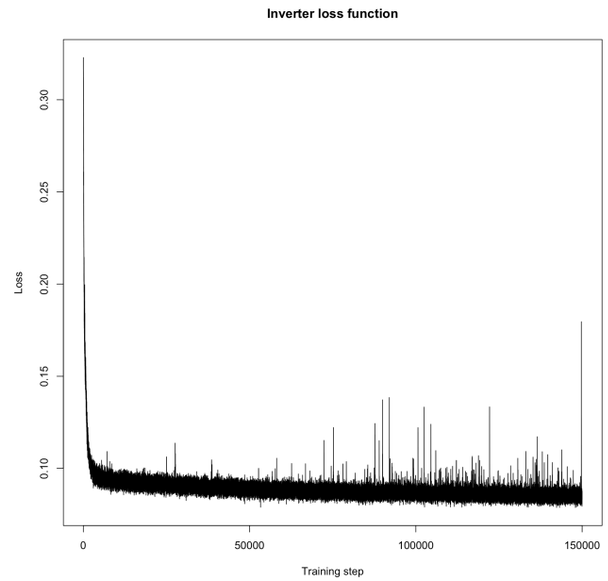
WGAN + Inverter: Architecture

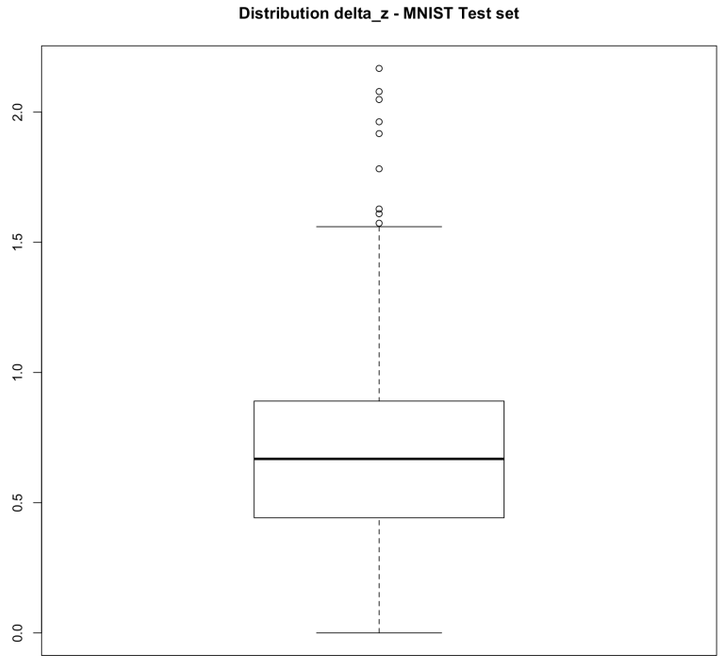
- **PyTorch** implementation
- Generator: 1 Fully Connected + 4 Transposed Conv. Layers (strides = 2), ReLu activation func.
- Discriminator (“Critic”): 4 Conv. Layers (strides = 2) + 1 Fully Connected, ReLu activation func.
- Inverter: 4 Conv. Layers (strides = 2) + 2 Fully Connected, ReLu activation func.
- Adam optimizers for the three nets

WGAN + Inverter: Training





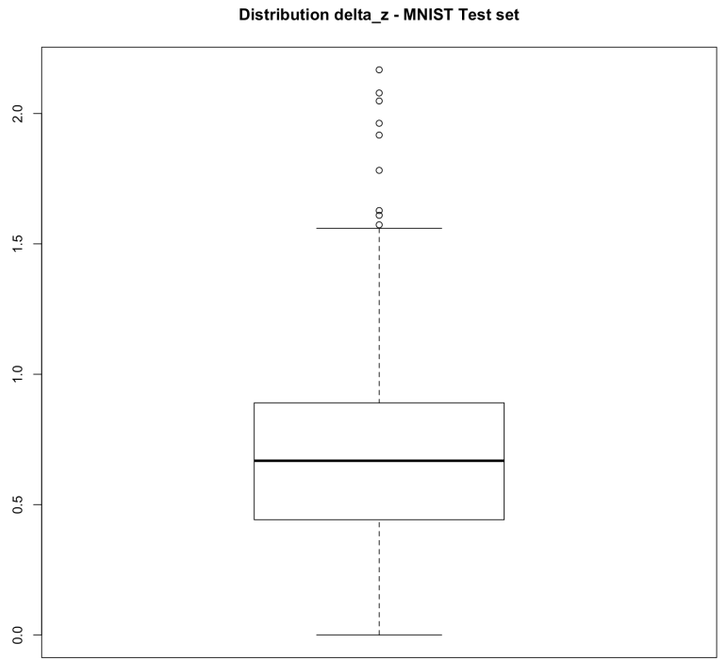




Distribution of delta_z computed on 500 test set input points

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0000	0.4421	0.6681	0.6858	0.8901	2.1676

WGAN + Inverter: Results



Distribution of delta_z computed on 500 test set input points

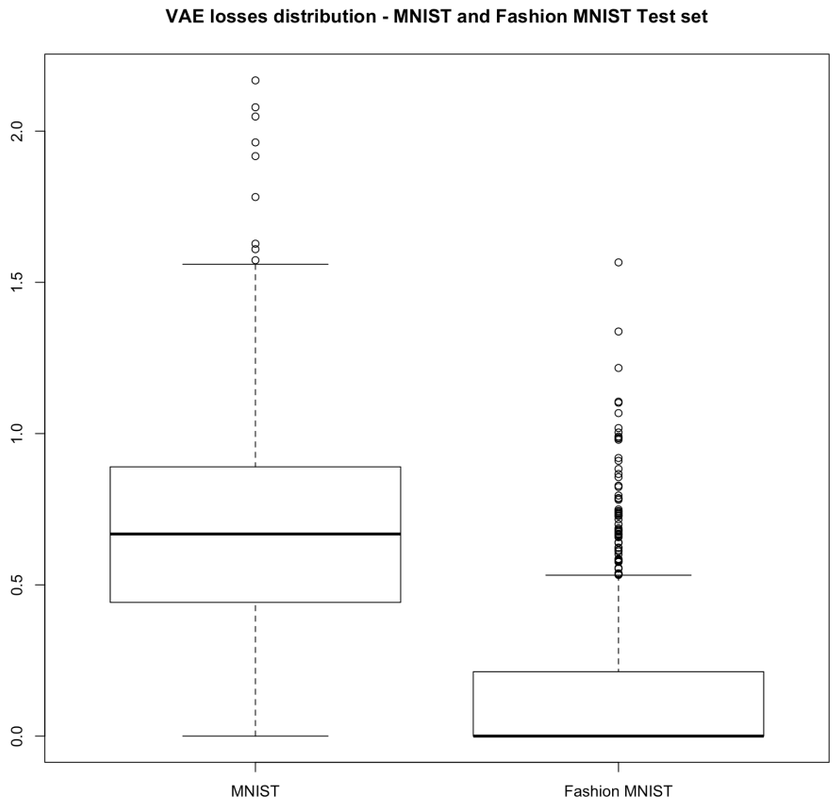
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.4421  0.6681  0.6858  0.8901  2.1676
```

5° percentile: 5.3e-02

Delta_z for the chicken: 1.8e-06 (< 5.3e-02)
Delta_z for the falafel: 5.4e-07 (< 5.3e-02)

Using the 5° percentile test delta_z as threshold, we could get a 5% as false positive rate and discard the meaningless predictions for the chicken and the falafel picture.

WGAN + Inverter: Results with Fashion MNIST



Distribution of delta_z computed on 500 test set input points

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.0000  0.4421  0.6681  0.6858  0.8901  2.1676
```

5° percentile: 5.3e-02

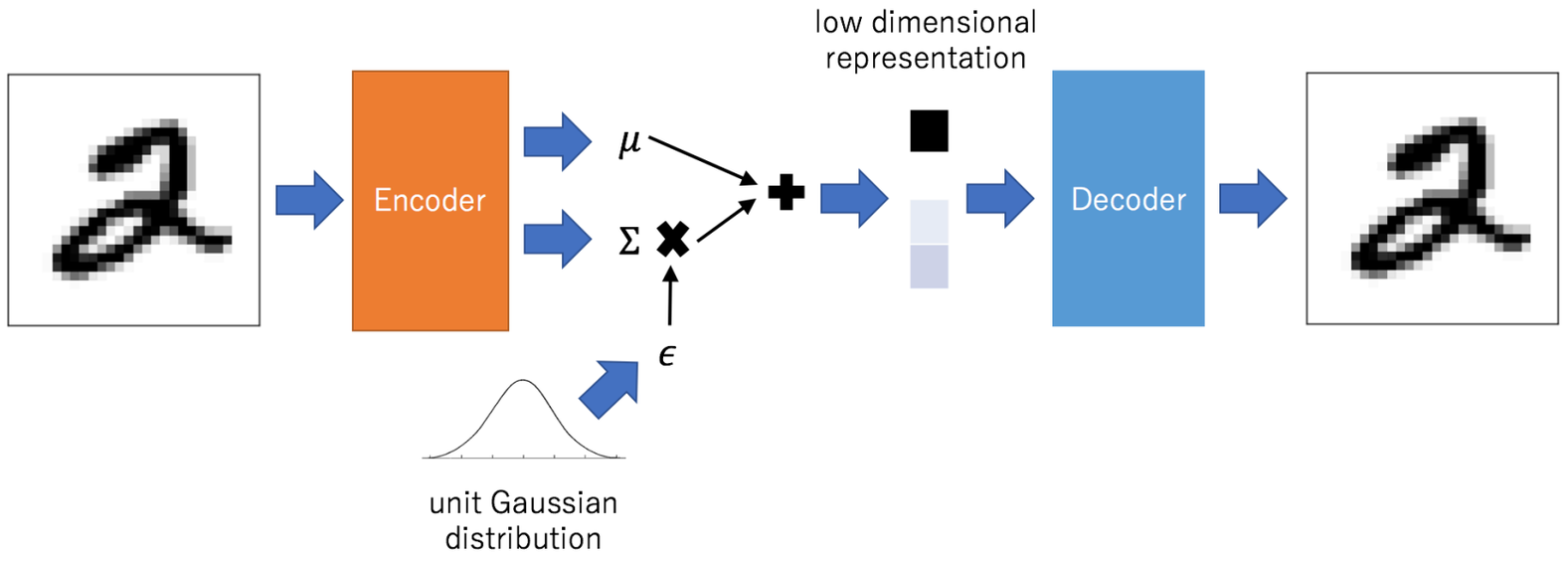
Using the 5° percentile test delta_z as threshold, we could get a 5% as false positive rate **but a 34.4% of meaningless predictions (false negative rate)!**

We need to reduce the false negative rate!

Variational Autoencoder

Understanding the most important features of the input space while learning how to build it from “scratch”.

An “unlikely” input will have troubles in the encoding-decoding process = high loss value.



Variational Autoencoder: robustness metrics

The loss function of a variational autoencoder is an indirect measure of the probability that an observation comes from the same distribution that generated the training set.

Robustness metrics: loss function

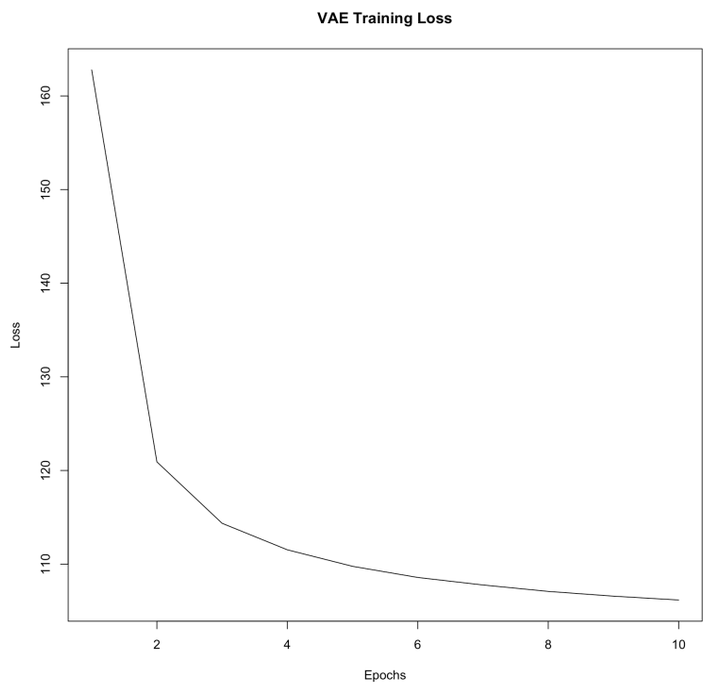
- “Big” loss = Not robust prediction
- “Small” loss = Robust prediction

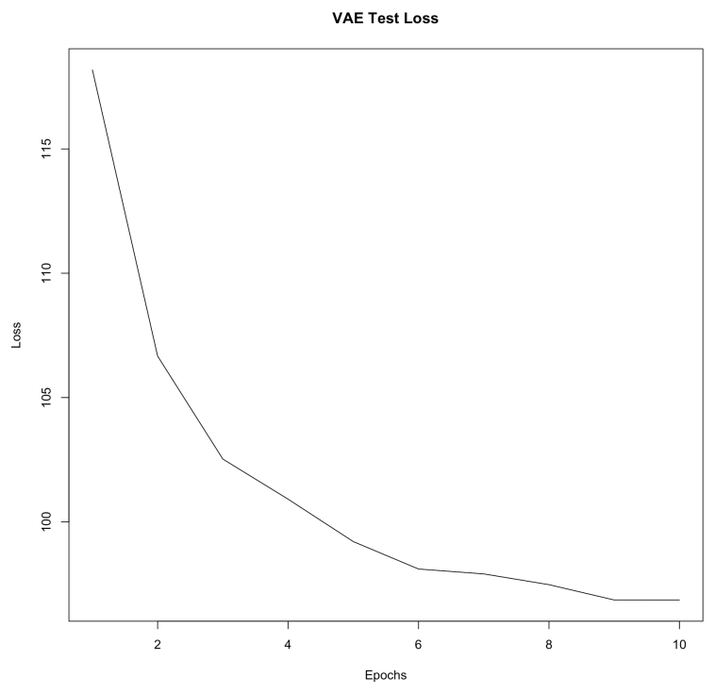
Rule: if VAE_loss > threshold then do not predict

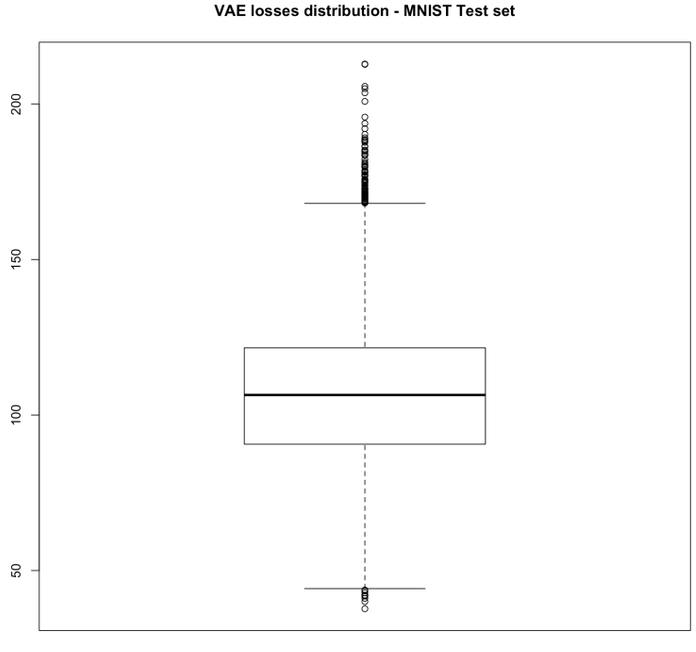
Variational Autoencoder: Architecture

- **PyTorch** implementation
- Encoder: 2 layers fully connected neural network
- Decoder: 2 layers fully connected neural network
- Adam optimizer

Variational Autoencoder: Training



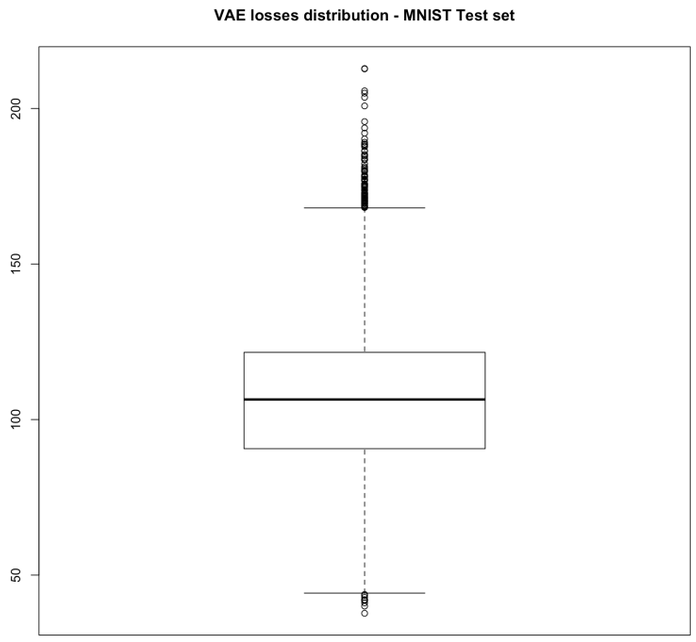




Distribution of VAE losses computed on 10.000 test set input points

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
37.71	90.63	106.45	105.38	121.60	212.85

Variational Autoencoder: Results



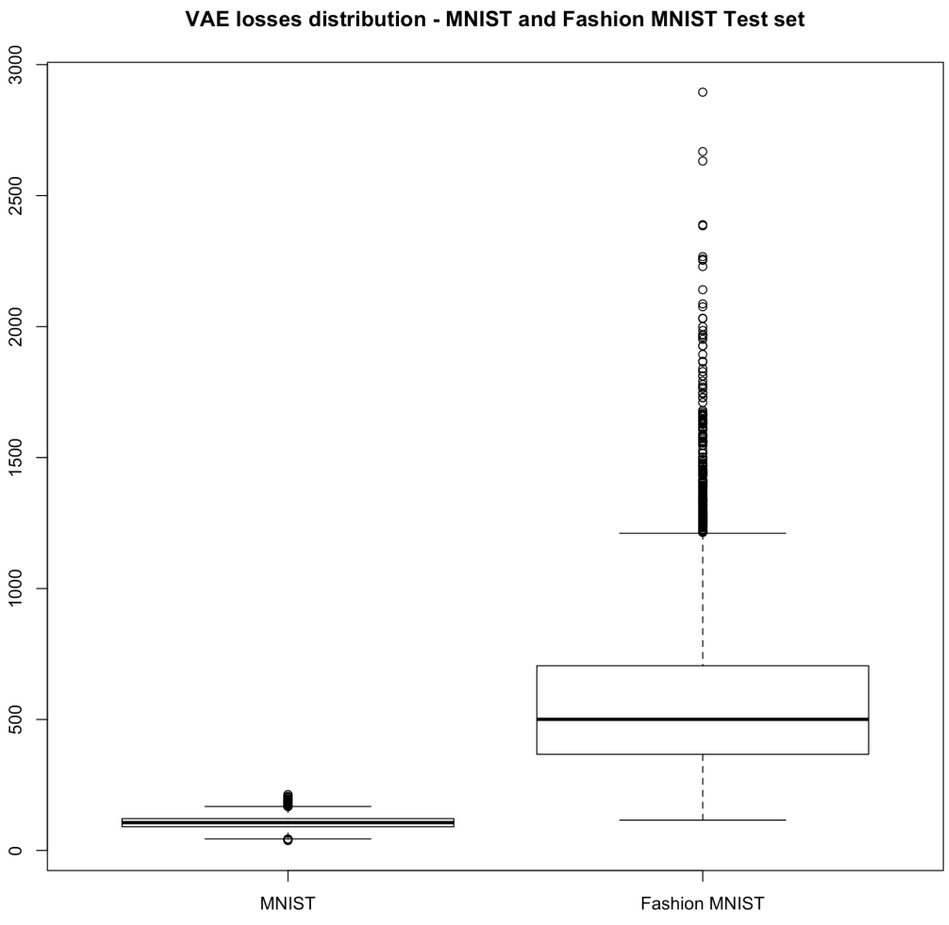
Distribution of VAE losses computed on 10.000 test set input points

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
37.71	90.63	106.45	105.38	121.60	212.85

VAE loss for the chicken: 309.4 (>212)
VAE loss for the falafel: 225.3 (>212)

Using the maximum test loss as threshold, we could get a 0% as false positive rate and discard the meaningless predictions for the chicken and the falafel picture.

Variational Autoencoder: Results with Fashion MNIST



Distribution of VAE losses computed on 10.000 test set input points

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
37.71   90.63  106.45  105.38  121.60  212.85
```

Using the maximum test loss as threshold, we could get a 0% as false positive rate and a 3.55% of meaningless predictions (false negative rate)

Using the 95° percentile test loss as threshold, we could get a 5% as false positive rate and a 0.25% of meaningless predictions (false negative rate)

Variational Autoencoder: robustness metrics into production

- Train your classifier
- Train a VAE on your training set
- Get the distribution of the VAE losses on your test set
- Define a threshold more or less “conservative”
- Implement a “conditional classifier”:

```
def conditional_classifier_VAE(input, threshold, VAE, loss_function, classifier):
    y_pred = classifier(input)
    recon, mu, logvar = VAE(input)
    input_loss = loss_function(recon, input, mu, logvar)

    if input_loss > threshold:
        raise NotReliableClassification("Not reliable classification", y_pred)
    else:
        return y_pred
```

Acknowledgements

Emilien Dupont (for the chicken idea)

Orobix s.r.l. (for the hints about the general problem and a possible solution)

Nunatac s.r.l. (for the invaluable support)

Adam Green (a precious mentor)

Chris Armbruster (the best director ever)

Sandra Meneses (for her help with the cloud computing)

Lisa Heße (for her support and trust)

All the guys of DSR batch 15… great friends!

Thank you for your attention!