

## Lecture 11:

# Convolution and Image Classification & Segmentation



Markus Hohle

University California, Berkeley

Bayesian Data Analysis and  
Machine Learning for Physical  
Sciences

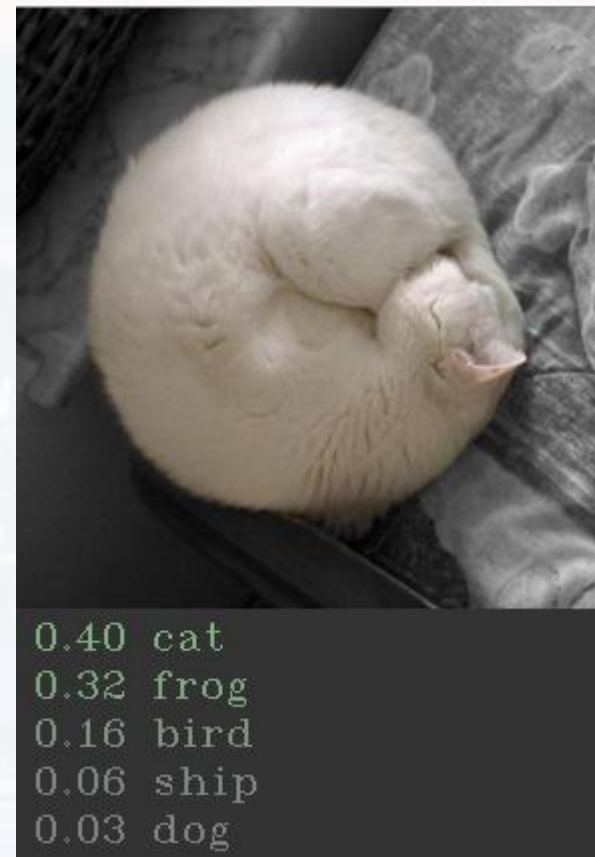


## Course Map

Module 1	Maximum Entropy and Information, Bayes Theorem
Module 2	Naive Bayes, Bayesian Parameter Estimation, MAP
Module 3	MLE, Lin Regression
Module 4	Model selection I: Comparing Distributions
Module 5	Model Selection II: Bayesian Signal Detection
Module 6	Variational Bayes, Expectation Maximization
Module 7	Hidden Markov Models, Stochastic Processes
Module 8	Monte Carlo Methods
Module 9	Machine Learning Overview, Supervised Methods & Unsupervised Methods
Module 10	ANN: Perceptron, Backpropagation, SGD
<b>Module 11</b>	<b>Convolution and Image Classification and Segmentation</b>
Module 12	RNNs and LSTMs
Module 13	RNNs and LSTMs + CNNs
Module 14	Transformer and LLMs
Module 15	Graphs & GNNs



## Part III







```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

**PyTorch & Cuda**

**Generative Adversarial Network (GAN)**



```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

## Outline

**PyTorch & Cuda**

Generative Adversarial Network (GAN)



problem: even for moderate setups, computational time becomes the limiting factor

self made ANN

keras/tensor flow

pytorch

pytorch on GPU (Cuda)

speed

```
512/512 [=====] - 4072s 8s/step - loss: 3.4534 - accuracy: 0.3338
Epoch 4/5
512/512 [=====] - ETA: 0s - loss: 3.1926 - accuracy: 0.3980    saved ../data/segmentation
pics/checkpoints//.3
512/512 [=====] - 3363s 7s/step - loss: 3.1926 - accuracy: 0.3980
Epoch 5/5
512/512 [=====] - ETA: 0s - loss: 3.0071 - accuracy: 0.4318    saved ../data/segmentation
pics/checkpoints//.4
512/512 [=====] - 3616s 7s/step - loss: 3.0071 - accuracy: 0.4318
```

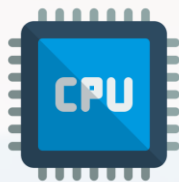
**Lenovo T14, NVIDIA GeForce MX450:** (simple LSTM)

**Keras (CPU):** 300 sec

PyTorch (CPU): 11 sec

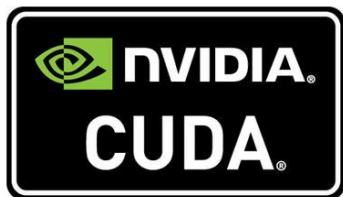
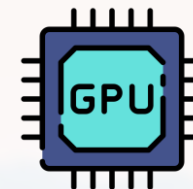
**PyTorch (GPU):** 3 sec





training AI

- mainly matrix operations
- GPUs are a lot better at it!



CUDA is the link of your GPU to Python (PyTorch)  
check, if graphic card is on [list](#)

check your graphics device:

- Windows command shell prompt
- type `nvidia-smi`
- press *Enter*

```
(base) C:\Users\MMH_user>nvidia-smi
Tue Jan 16 20:09:26 2024

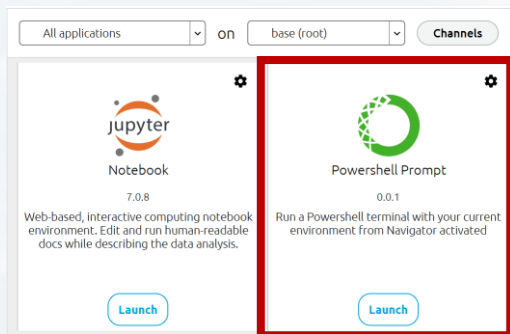
+-----+
| NVIDIA-SMI 537.79                  Driver Version: 537.79          CUDA Version: 12.2 |
+-----+-----+
| GPU   Name                               TCC/WDDM    Bus-Id      Disp.A   Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap       Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0     NVIDIA GeForce MX450             WDDM        00000000:01:00.0 Off      0%          Default |
| N/A   45C    P0      N/A / 9W           0MiB / 2048MiB |              N/A    |
+-----+-----+

Processes:
 GPU   GI   CI        PID   Type   Process name                      GPU Memory
   ID   ID                                  Usage
+-----+
No running processes found
```



## Installing CUDA

conda environment



```
C:\WINDOWS\System32\Win... x + v  
(base) PS C:\Users\MMH_user> |
```

```
C:\WINDOWS\System32\Win... x + v  
(base) PS C:\Users\MMH_user> conda create --name CUDAenv |
```

```
C:\WINDOWS\System32\Win... x + v  
(base) PS C:\Users\MMH_user> conda activate CUDAenv|
```

```
(base) PS C:\Users\MMH_user> conda activate CUDAenv  
(CUDAenv) PS C:\Users\MMH_user> conda install -c pytorch pytorch  
Channels:  
- pytorch
```





## Installing CUDA

cuda toolkit

```
(CUDAenv) PS C:\Users\MMH_user> conda install -c anaconda cudatoolkit
```

check libraries

→ type: conda list

```
(CUDAenv) PS C:\Users\MMH_user> conda list
# packages in environment at C:\Users\MMH_user\anaconda3\envs\CUDAenv:
#
# Name                                Version                                Build                                Channel
blas                                  1.0                                    mkl
bzip2                                 1.0.8                                h2bbff1b_6
ca-certificates                       2024.7.2                              haa95532_0
cudatoolkit                           11.8.0                                hd77b12b_0
expat                                  2.6.2                                hd77b12b_0
filelock                              3.13.1                                py312haa95532_0
intel-openmp                          2023.1.0                              h59b6b97_46320
```



## Installing CUDA

usually, a few libraries are missing

check again graphics card:	type in anaconda	<b>nvidia-smi</b>
check libraries:	type in anaconda	<b>conda list cudnn</b> <b>conda list cudatoolkit</b> <b>conda list torch</b>
	if not:	<b>conda install &lt;library&gt;</b>
check Python:	type in anaconda	<b>python</b> <b>import torch</b> <b>torch.cuda.is_available()</b>
open Spyder	run in Spyder	<b>pip install</b> (see the commented line in CheckMyCuda.py)



## Installing CUDA

usually, a few libraries are missing

`CheckMyCuda.py`

```
import torch
```

```
def test_cuda():
```

```
    print("PyTorch version: ", torch.__version__)
```

```
    print("CUDA version: ", torch.version.cuda)
```

```
    print("CUDA Available: ", torch.cuda.is_available())
```

```
    if torch.cuda.is_available():
```

```
        print("Number of GPUs: ", torch.cuda.device_count())
```

```
        print("GPU Name: ", torch.cuda.get_device_name(0))
```

```
if __name__ == "__main__":
```

```
    test_cuda()
```

```
PyTorch version: 2.3.1+cu118
CUDA version: 11.8
CUDA Available: True
Number of GPUs: 1
GPU Name: NVIDIA GeForce MX450
```





The key part in PyTorch is to set all matrices and the model **to the device (CPU or GPU)**

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
print("Using device:", device)
```

```
In [13]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
        ....: print("Using device:", device)  
Using device: cuda
```

**Congratulation! If you see this, you are ready to go!**



The key part in PyTorch is to set all matrices and the model **to the device (CPU or GPU)**

```
In [13]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
...: print("Using device:", device)
Using device: cuda
```

Torch objects like **model** or **torch.tensor** have the property **.to**

```
TrainX = torch.tensor(TrainX, dtype = torch.float32)
TrainY = torch.tensor(TrainY, dtype = torch.float32)
```

```
TrainX = TrainX.to(device)
TrainY = TrainY.to(device)
```

```
model = model.to(device)
```

turning numpy array into torch.tensor

allocating objects to the device



The key part in PyTorch is to set all matrices and the model **to the device (CPU or GPU)**

```
In [13]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        ...: print("Using device:", device)
Using device: cuda
```

When running the training, we need to **synchronize** between GPU (for training the model) and CPU (for everything else)...

```
torch.cuda.synchronize()
```

```
#training loop
for epoch in range(n_epochs):
    outputs = model(TrainX)

    ... #do stuff...
```

```
torch.cuda.synchronize()
```





The key part in PyTorch is to set all matrices and the model **to the device (CPU or GPU)**

```
In [13]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        ...: print("Using device:", device)
Using device: cuda
```

When running the training, we need to **synchronize** between GPU (for training the model) and CPU (for everything else)...

...and later detach the model from the GPU

```
PredY = model(TestX).detach().to('cpu').numpy()
```

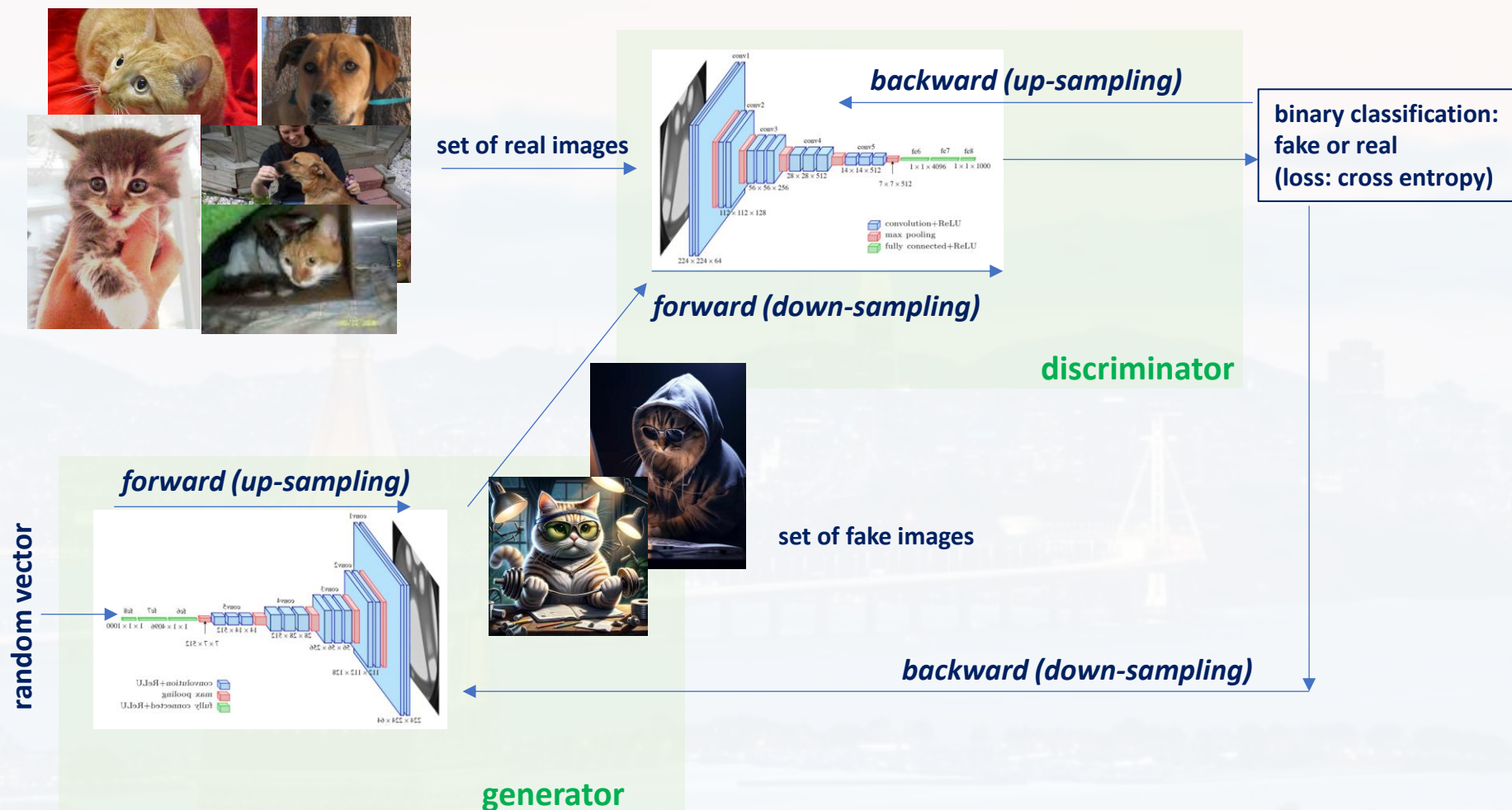


```
0.40 cat
0.32 frog
0.16 bird
0.06 ship
0.03 dog
```

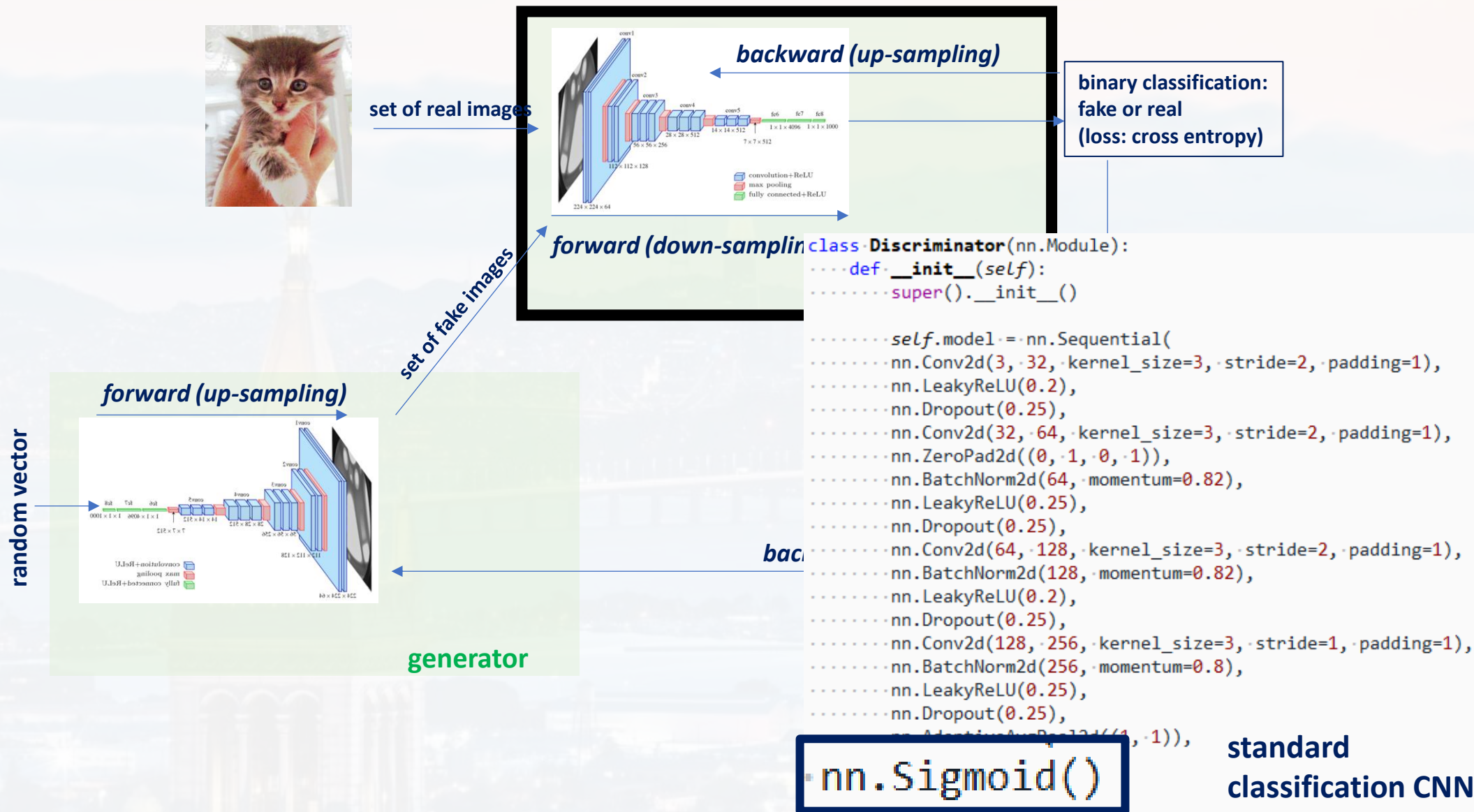
## Outline

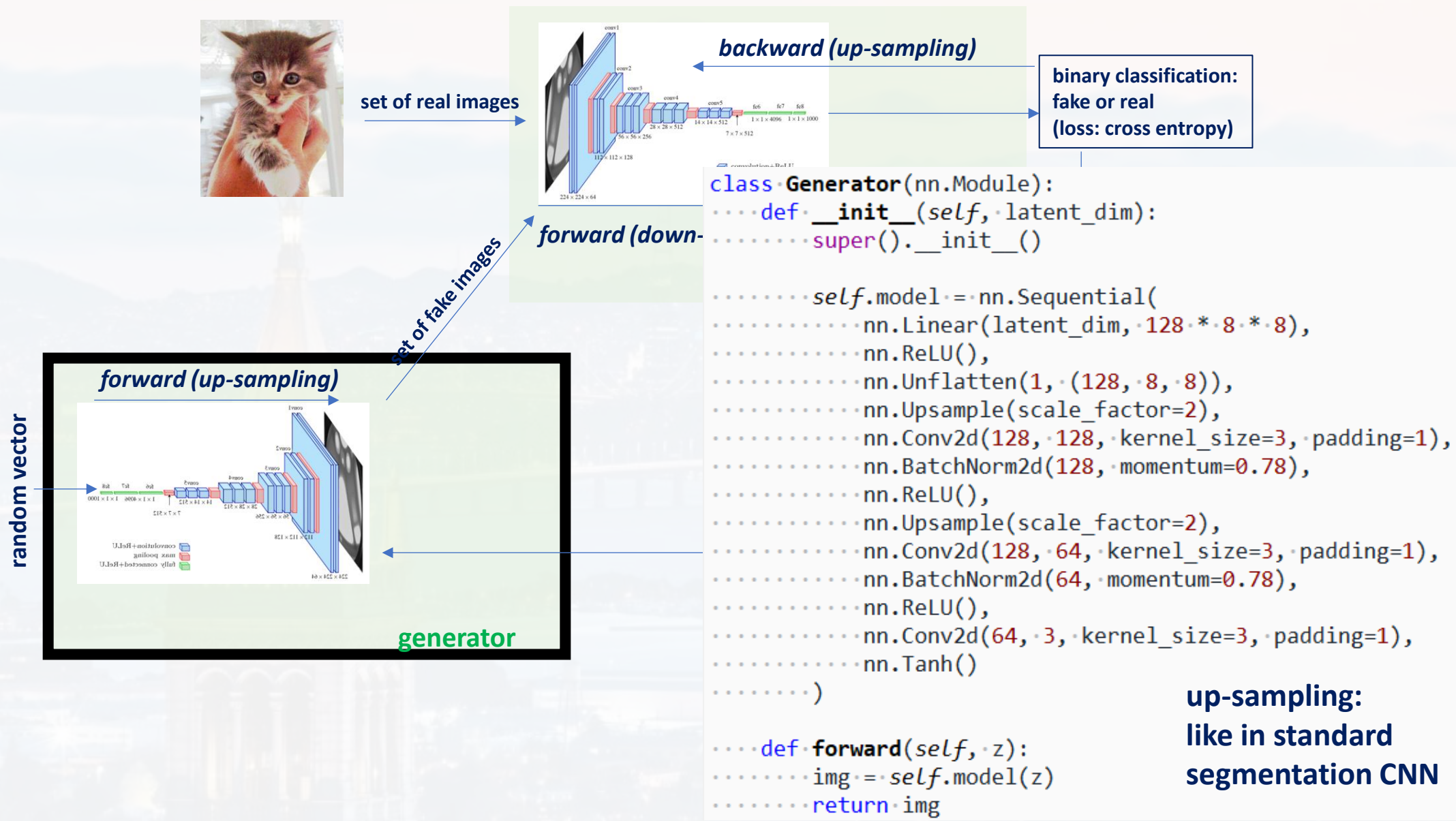
PyTorch & Cuda

**Generative Adversarial Network (GAN)**











$x$ : real image

$z$ : latent vector (random numbers)

$p_D(x)$ : probability that discriminator  $D$  classifies  $x$  as real image

$G(z)$ : fake image generated from generator  $G$

$\mathcal{L}_D$ : loss function of discriminator  $D$

$\mathcal{L}_G$ : loss function of generator  $G$

(mean) binary cross entropy (BCE):  
over all  $N$  images

$$\mathcal{L}_D = -\frac{1}{N} \sum_{i=1}^N \log[p_D(x_i)] + \log[1 - p_D(G(z_i))]$$

$$\mathcal{L}_G = -\frac{1}{N} \sum_{i=1}^N \log[1 - p_D(G(z_i))]$$

ideal case for  $D$ :

$$\mathcal{L}_D = -\log[1] - \log[1 - 0] = 0$$

worst case for  $D$ :

$$\mathcal{L}_D = -\log[1] - \log[1 - \mathbf{0.5}] = \mathbf{0.69}$$

ideal case for  $G$ :

$$\mathcal{L}_D = \log[1 - \mathbf{0.5}] = \mathbf{-0.69}$$

worst case for  $G$ :

$$\mathcal{L}_D = \log[1 - 0] = 0$$

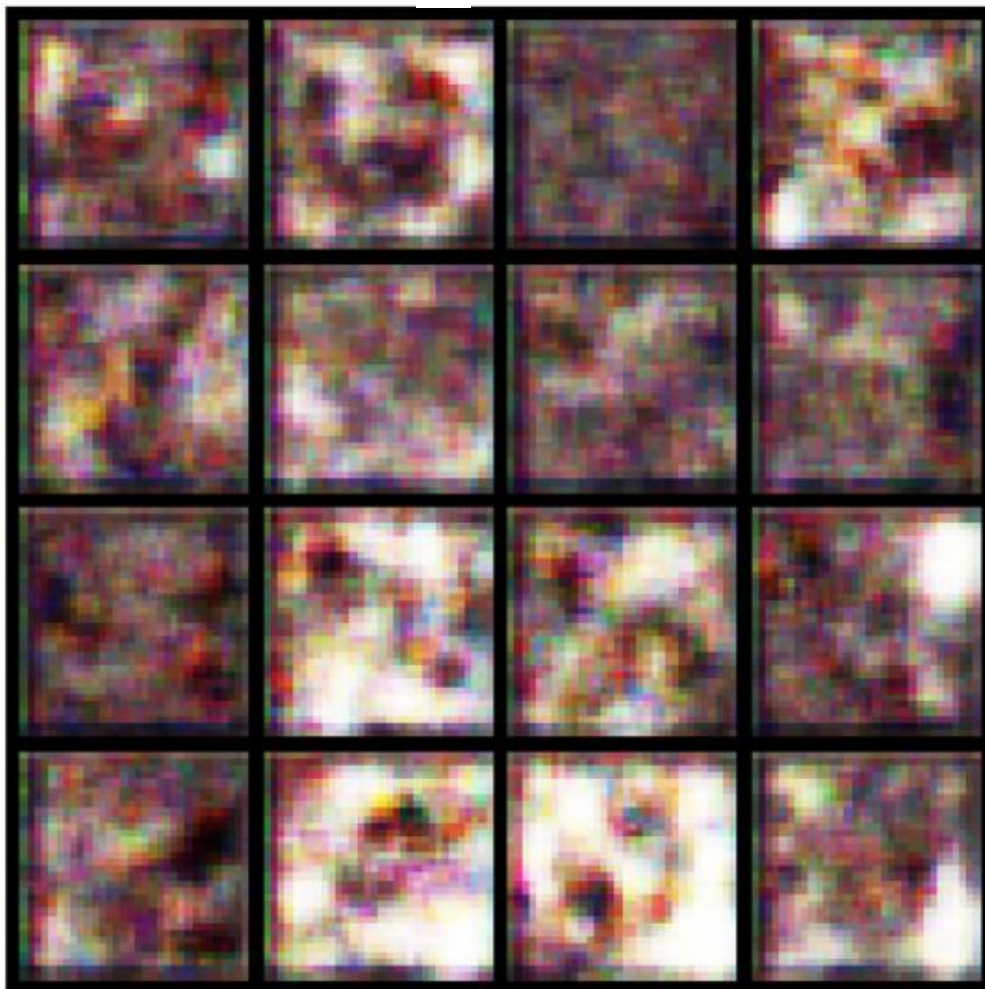




check out

`GAN architecture.ipynb`

after 1 epoch(s)



after 10 epoch(s)



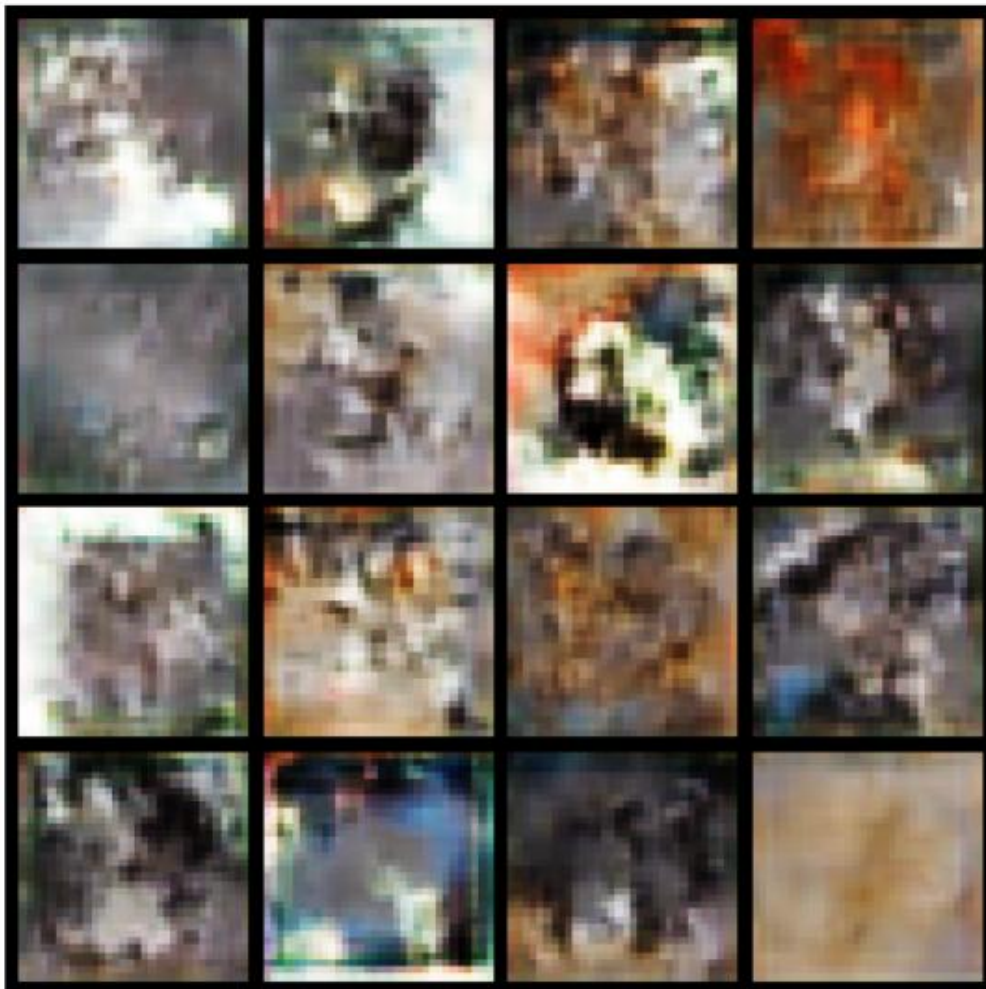




check out

`GAN architecture.ipynb`

after 30 epoch(s)



after 55 epoch(s)





Thank you very much for your attention!

