Introduction to GANs

DEEP LEARNING FOR IMAGES WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

General Adversarial Networks

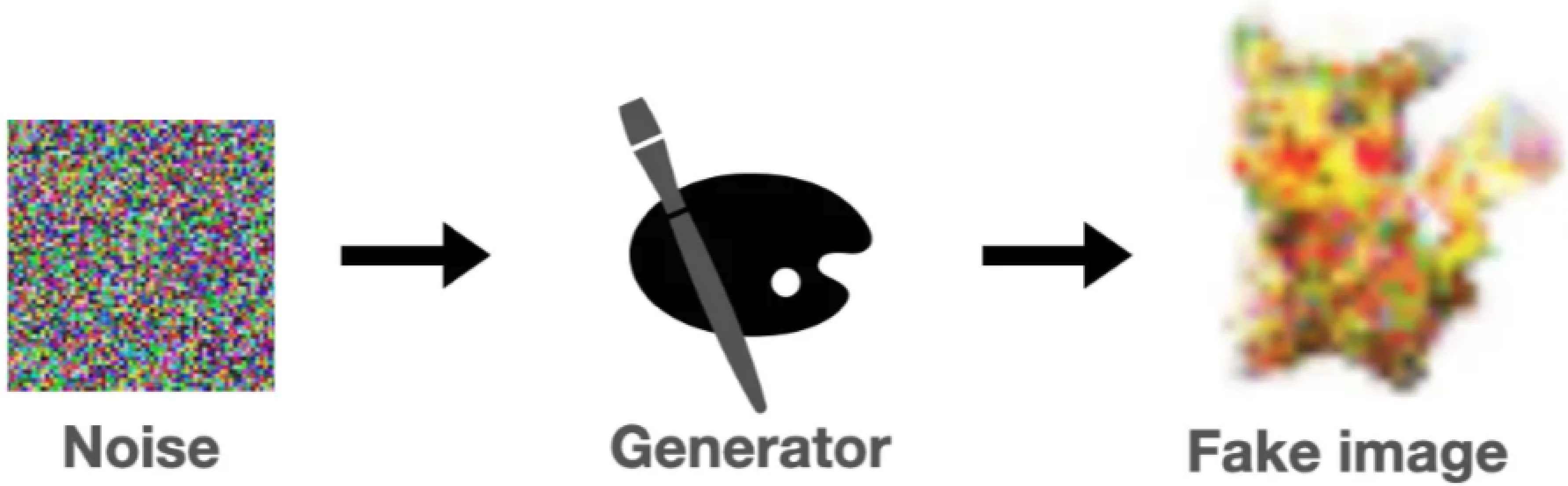
- https://thesecatsdonotexist.com
- Generative Adversarial Networks (GANs)
- Generate new samples based on training data

Pokemon Sprites Dataset

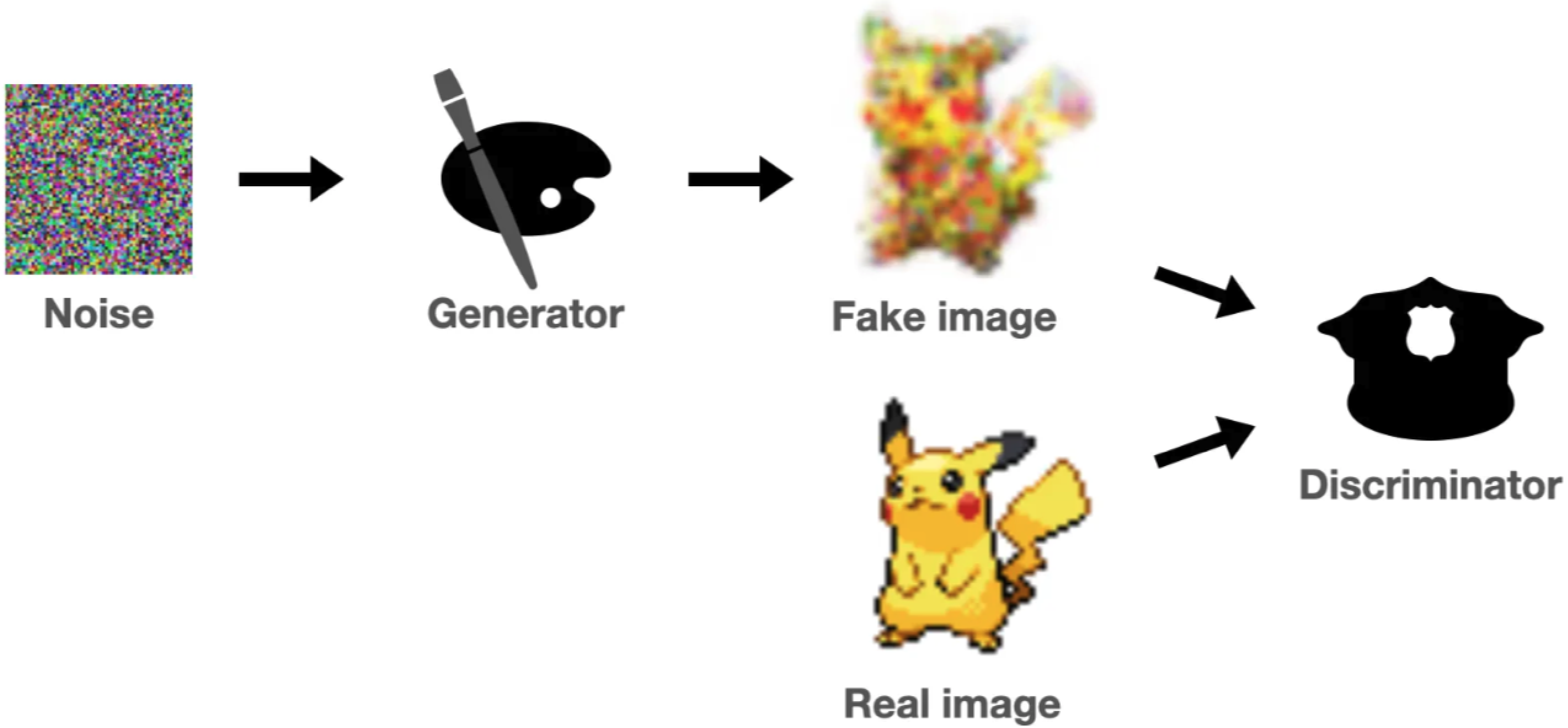
- Pokemon Sprites Dataset from PokeAPI
- About 1300 sprites of animal-like creatures from a Pokemon video game
- Goal: Generate new Pokemons!

GANs architecture

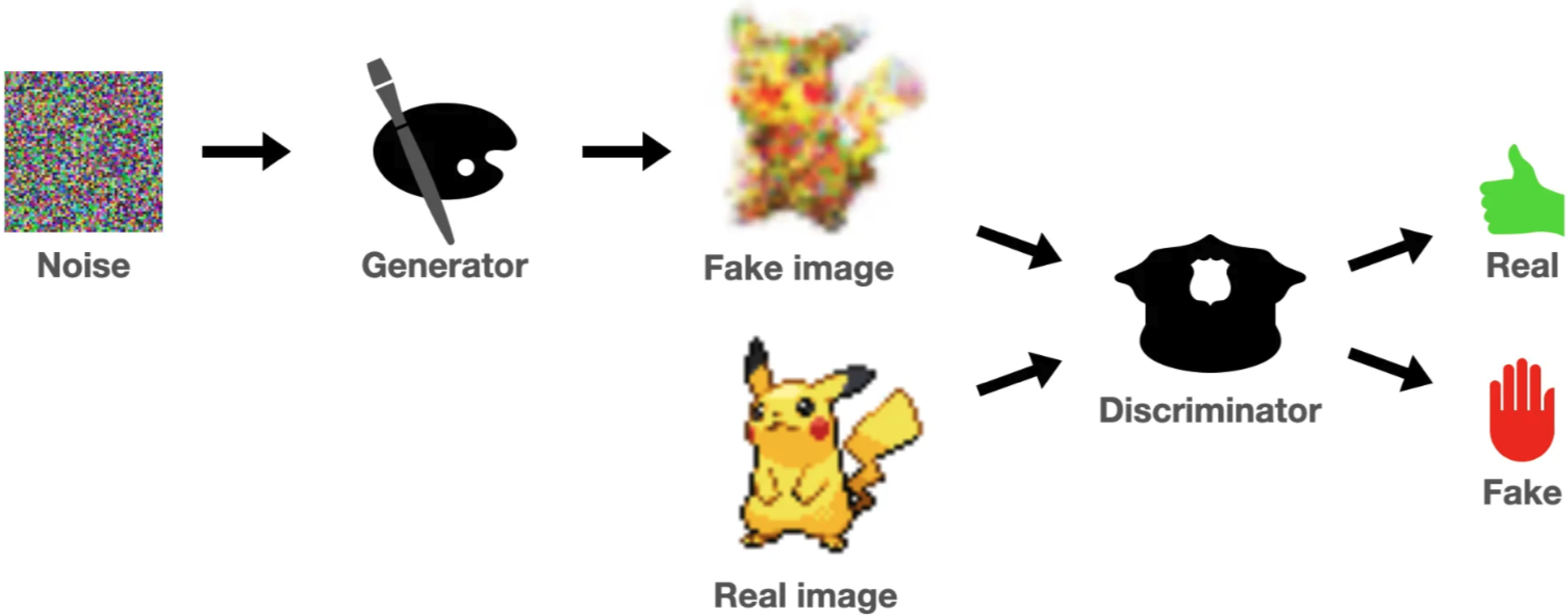
GANs architecture



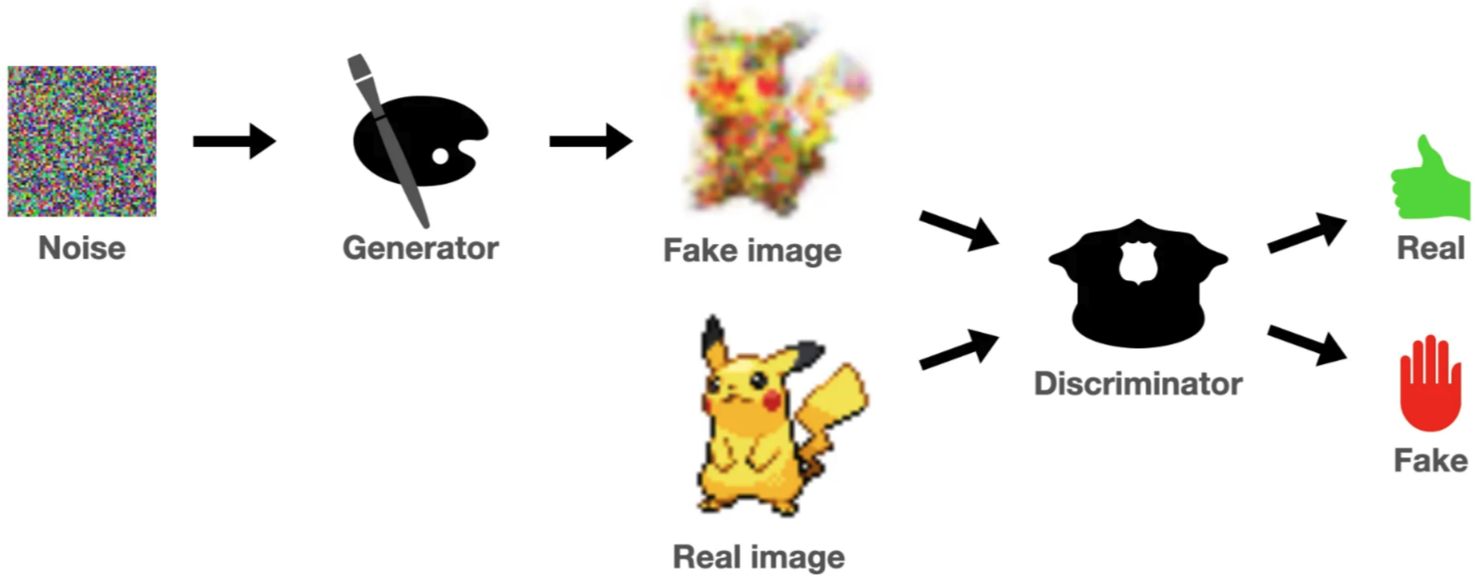
GANs architecture



GANs architecture



GANs learning process



- Generator: learns to produce realistic-looking images
- Discriminator: learns to tell the fakes from real images
- Conflicting objectives ensure each network gets better at its task
- In the end, generator should generate realistic images

Basic Generator

```python
class Generator(nn.Module):
    def __init__(self, in_dim, out_dim):
        super(Generator, self).__init__()
        self.generator = nn.Sequential(
            gen_block(in_dim, 256),
            gen_block(256, 512),
            gen_block(512, 1024),
            nn.Linear(1024, out_dim),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.generator(x)
```

- Define `Generator` class
- Sequence of generator blocks, a linear layer, and a sigmoid activation

```python
def gen_block(in_dim, out_dim):
  return nn.Sequential(
      nn.Linear(in_dim, out_dim),
      nn.BatchNorm1d(out_dim),
      nn.ReLU(inplace=True)
  )
```

- Pass input through all layers
- Input: noise of size `in_dim`
- Output: image of size `out_dim`

Basic Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, im_dim):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            disc_block(im_dim, 1024),
            disc_block(1024, 512),
            disc_block(512, 256),
            nn.Linear(256, 1),
        )

    def forward(self, x):
        return self.disc(x)
```

- Define `Discriminator` class
- Sequence of discriminator blocks and a linear layer

```
def disc_block(in_dim, out_dim):
  return nn.Sequential(
      nn.Linear(in_dim, out_dim),
      nn.LeakyReLU(0.2)
  )
```

- Pass input through all layers
- Input: image of size `in_dim`
- Output: classification of size `1`

Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH

Deep Convolutional GAN

DEEP LEARNING FOR IMAGES WITH PYTORCH

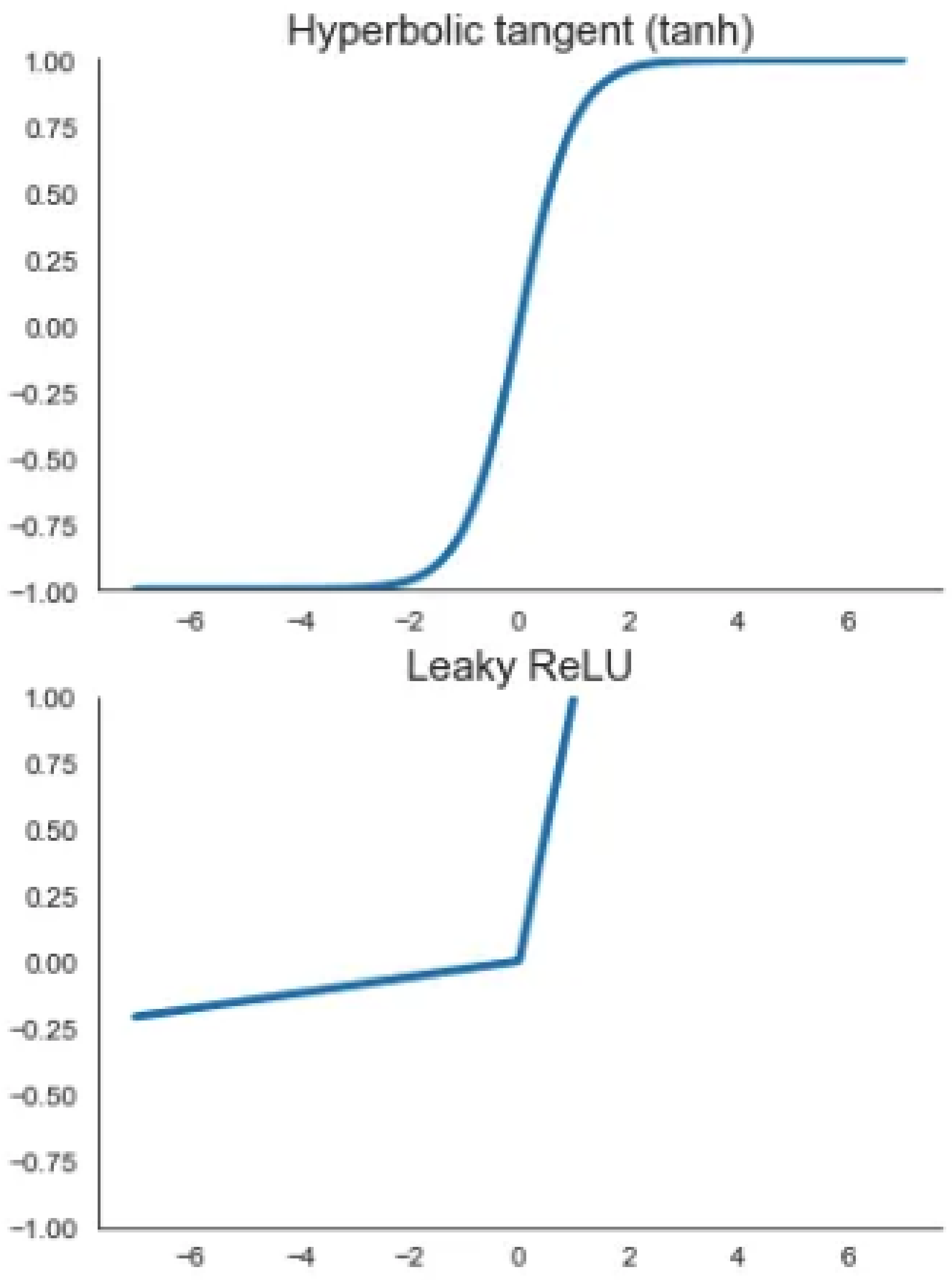
Michal Oleszak
Machine Learning Engineer

Deep Convolutional GAN intuition

- In discriminator, replace linear layers with convolutions
- In generator, we can use transposed convolutions
- Training GANs is often unstable, more adjustments are needed

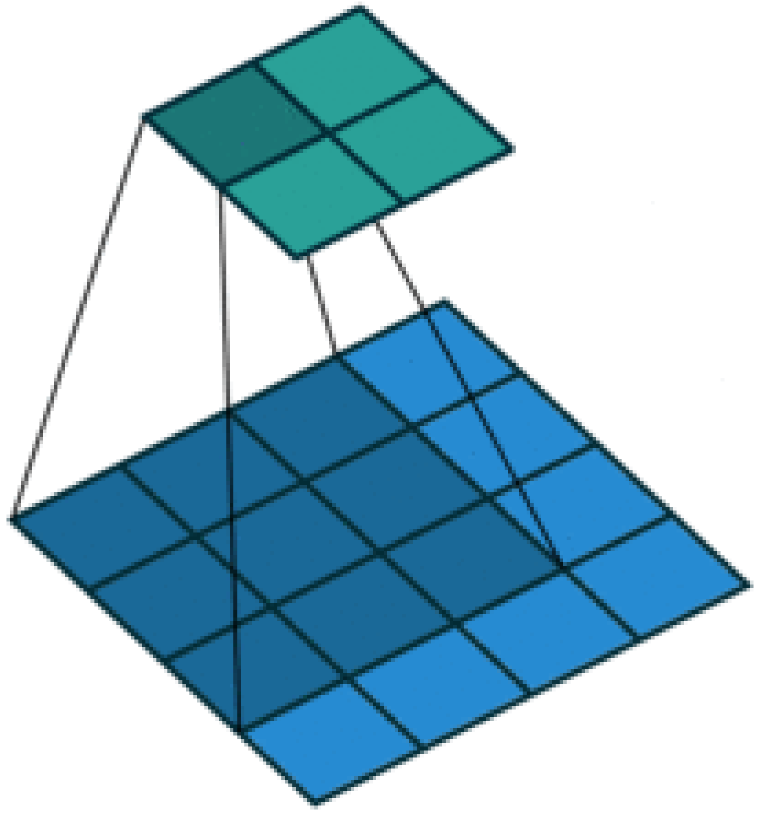
DCGAN guidelines

- Deep Convolutional GAN (DCGAN)
- DCGAN guidelines:
  - Use only strided convolutions
  - Don't use any linear or pooling layers
  - Use batch normalization
  - Use ReLU activations in the generator (except last layer which uses tanh)
  - Use Leaky ReLU activation in the discriminator



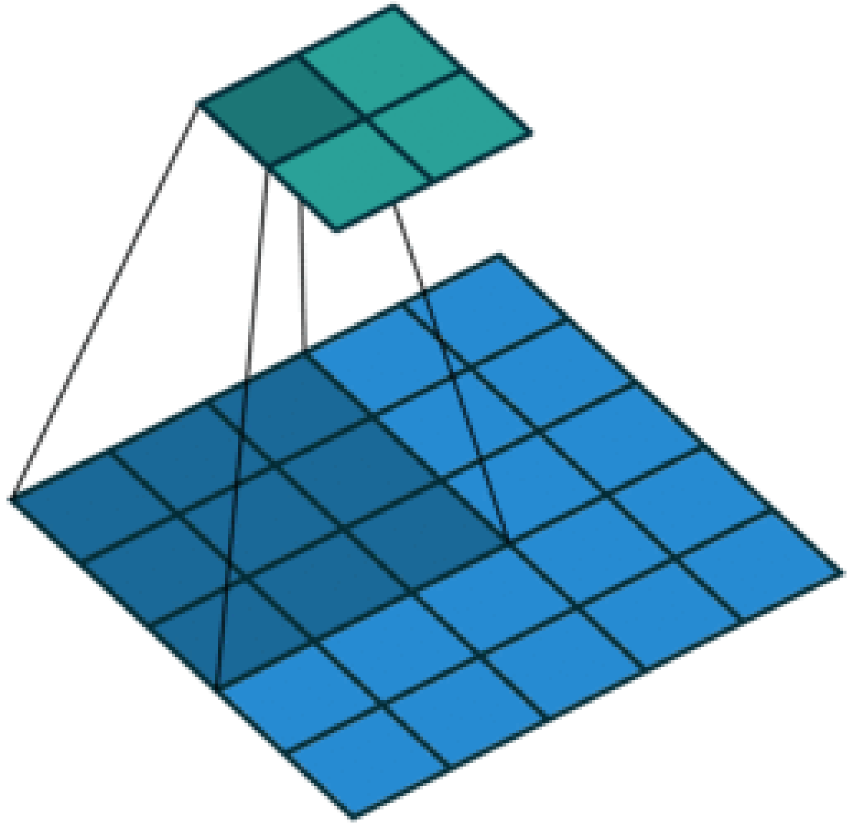
Strided convolution

Convolution with the stride of 1:



```
nn.Conv2d(..., stride=1)
```

Convolution with the stride of 2:



```
nn.Conv2d(..., stride=2)
```

Convolutional generator block

```
def dc_gen_block(
    in_dim, out_dim, kernel_size, stride
):
    return nn.Sequential(
        nn.ConvTranspose2d(
            in_dim,
            out_dim,
            kernel_size,
            stride=stride,
        ),
        nn.BatchNorm2d(out_dim),
        nn.ReLU()
    )
```

Generator block consists of:

- Strided transposed convolution
- Batch normalization
- ReLU activation

Deep Convolutional Generator

```
class DCGenerator(nn.Module):
    def __init__(self, in_dim, kernel_size=4, stride=2):
        super(Generator, self).__init__()
        self.in_dim = in_dim
        self.gen = nn.Sequential(
            dc_gen_block(in_dim, 1024, kernel_size, stride),
            dc_gen_block(1024, 512, kernel_size, stride),
            dc_gen_block(512, 256, kernel_size, stride),
            nn.ConvTranspose2d(256, 3, kernel_size, stride=stride),
            nn.Tanh()
        )

    def forward(self, x):
        x = x.view(len(x), self.in_dim, 1, 1)
        return self.gen(x)
```

Convolutional discriminator block

```
def dc_disc_block(
    in_dim, out_dim, kernel_size, stride
):
    return nn.Sequential(
        nn.Conv2d(
            in_dim,
            out_dim,
            kernel_size,
            stride=stride,
        ),
        nn.BatchNorm2d(out_dim),
        nn.LeakyReLU(0.2),
    )
```

Discriminator block consists of:

- Strided convolution
- Batch normalization
- Leaky ReLU activation

Deep Convolutional Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, kernel_size=4, stride=2):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            dc_disc_block(3, 512, kernel_size, stride),
            dc_disc_block(512, 1024, kernel_size, stride),
            nn.Conv2d(1024, 1, kernel_size, stride=stride),
        )

    def forward(self, x):
        x = self.disc(x)
        return x.view(len(x), -1)
```

Let's practice!

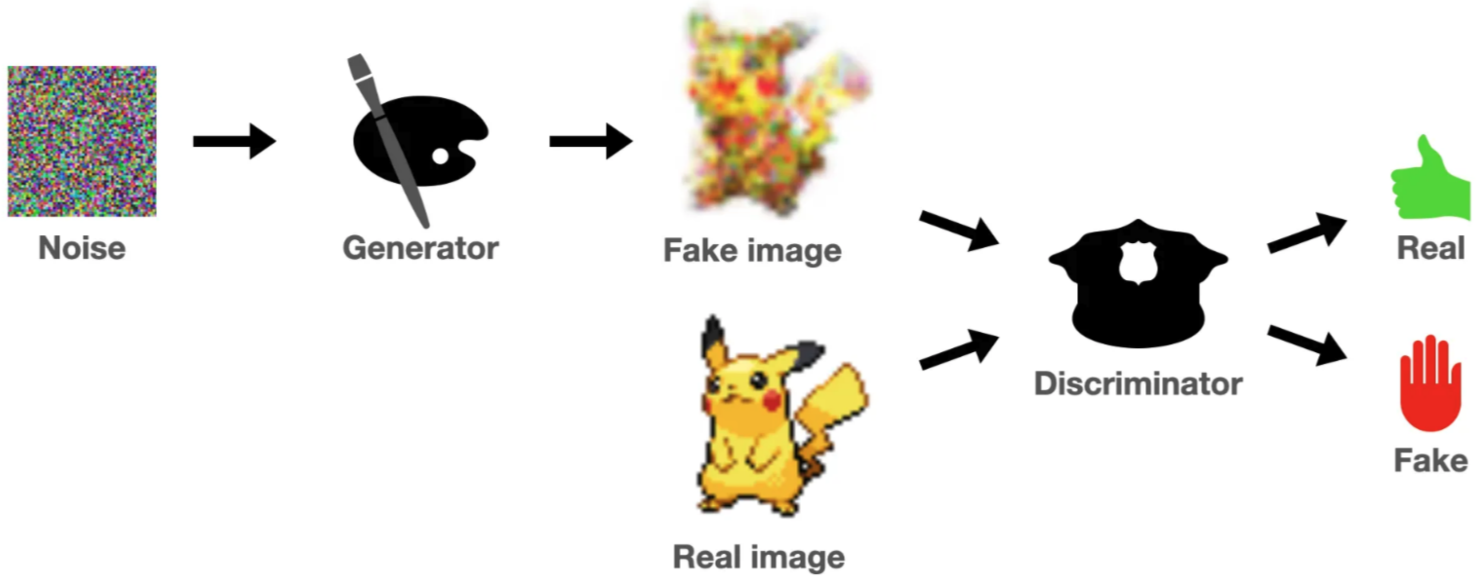
DEEP LEARNING FOR IMAGES WITH PYTORCH

Training GANs

DEEP LEARNING FOR IMAGES WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

Generator objective



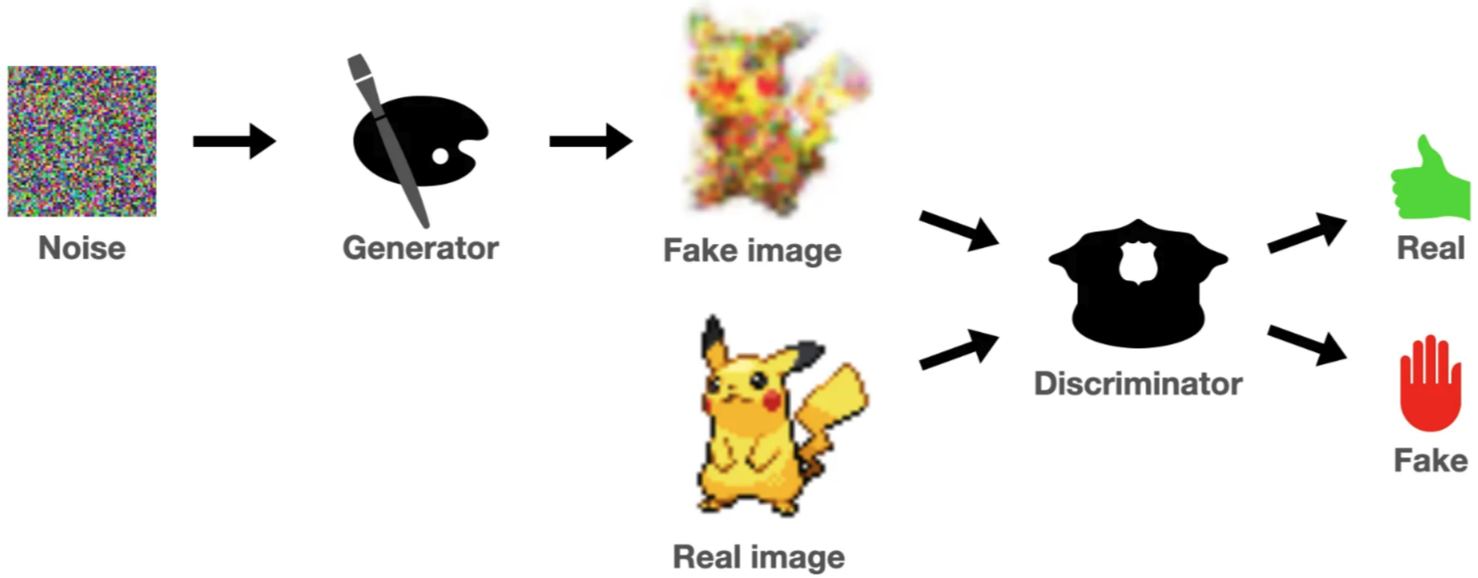
- Objective: Generate fakes that fool the discriminator
- Idea: Use the discriminator to inform us about generator's performance
- Generator's output classified by the discriminator as:
  - Real (label `1`) - good, small loss
  - Fake (label `0`) - bad, large loss

Generator loss

```python
def gen_loss(gen, disc, num_images, z_dim):
    noise = torch.randn(num_images, z_dim)
    fake = gen(noise)
    disc_pred = disc(fake)
    criterion = nn.BCEWithLogitsLoss()
    gen_loss = criterion(
        disc_pred, torch.ones_like(disc_pred)
    )
    return gen_loss
```

- Define random noise
- Generate fake image
- Get discriminator's prediction on the fake image
- Use binary cross-entropy (BCE) criterion
- Generator loss: BCE between discriminator predictions and a tensor of ones

Discriminator objective



- Objective: Correctly classify fakes and real images
- Generator's outputs should be classified as fake (label `0`)
- Real images should be classified as real (label `1`)

Discriminator loss

```
def disc_loss(gen, disc, real, num_images, z_dim):
    criterion = nn.BCEWithLogitsLoss()
    noise = torch.randn(num_images, z_dim)
    fake = gen(noise)
    disc_pred_fake = disc(fake)
    fake_loss = criterion(
        disc_pred_fake,
        torch.zeros_like(disc_pred_fake)
    )
    disc_pred_real = disc(real)
    real_loss = criterion(
        disc_pred_real,
        torch.ones_like(disc_pred_real)
    )
    disc_loss = (real_loss + fake_loss) / 2
    return disc_loss
```

- Define binary cross-entropy criterion
- Generate input noise for generator
- Generate fakes
- Get discriminator's predictions for fake images
- Calculate the fake loss component
- Get discriminator's predictions for real images
- Calculate the real loss component
- Final loss is the average between the real and fake loss components

GAN training loop

```
for epoch in range(num_epochs):
    for real in dataloader:
        cur_batch_size = len(real)

        disc_opt.zero_grad()
        disc_loss = disc_loss(
            gen, disc, real, cur_batch_size, z_dim=16)
        disc_loss.backward()
        disc_opt.step()

        gen_opt.zero_grad()
        gen_loss = gen_loss(
            gen, disc, cur_batch_size, z_dim=16)
        gen_loss.backward()
        gen_opt.step()
```

- Loop over epochs and real data batches and compute current batch size
- Reset discriminator optimizer's gradients
- Compute discriminator loss
- Compute discriminator gradients and perform the optimization step
- Reset generator optimizer's gradients
- Compute generator loss
- Compute generator gradients and perform the optimization step

Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH

Evaluating GANs

DEEP LEARNING FOR IMAGES WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

Generating images

```python
num_images_to_generate = 9
noise = torch.randn(num_images_to_generate, 16)
with torch.no_grad():
    fake = gen(noise)
print(f"Generated shape: {fake.shape}")
```

```
Generated shape: torch.Size([9, 3, 96, 96])
```

```python
for i in range(num_images_to_generate):
    image_tensor = fake[i, :, :, :]
    image_permuted = image_tensor.permute(1, 2, 0)
    plt.imshow(image_permuted)
    plt.show()
```

- Create random noise tensor
- Pass noise to generator
- Iterate over number of images
- Slice fake to select i-th image
- Rearrange the image dimensions
- Plot the image

GAN generations

Fréchet Inception Distance

- Inception: Image classification model
- Fréchet distance: Distance measure between two probability distributions
- Fréchet Inception Distance:
  1. Use Inception to extract features from both real and fake images samples
  2. Calculate means and covariances of the features for real and fake images
  3. Calculate Fréchet distance between the real and the fake normal distributions
- Low FID = fakes similar to training data and diverse
- FID < 10 = good

FID in PyTorch

```python
from torchmetrics.image.fid import \
FrechetInceptionDistance

fid = FrechetInceptionDistance(feature=64)

fid.update(
  (fake * 255).to(torch.uint8), real=False)
fid.update(
  (real * 255).to(torch.uint8), real=True)

fid.compute()
```

```
tensor(7.5159)
```

- Import `FrechetInceptionDistance`
- Instantiate the FID metric
- Update the metric with fake images:
  - Multiply by 255
  - Parse to `torch.uint8`
- Similarly, update the metric with real images
- Compute metric value

Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH

Wrap-up

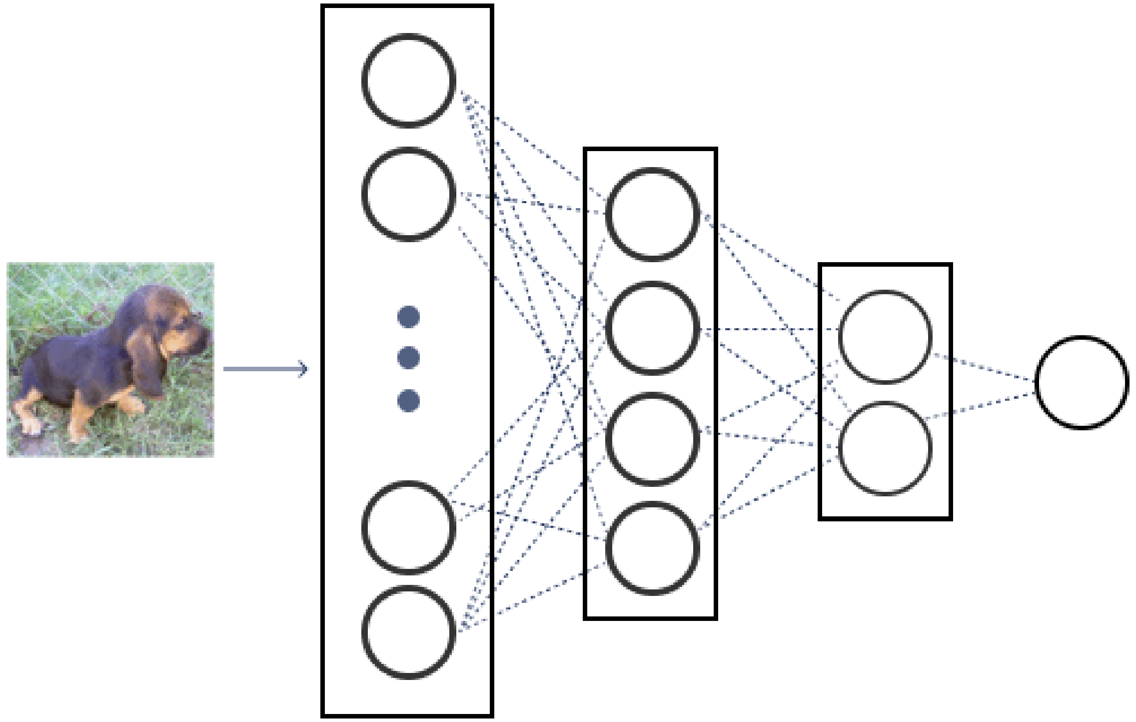
DEEP LEARNING FOR IMAGES WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

What you learned

1. Image Classification with CNNs

- Binary classification
- Multi-class classification
- Convolutional neural networks
- Leverage pre-trained models



What you learned

2. Object Recognition

- Bounding boxes
- Models: R-CNN, Faster R-CNN
- Non-max suppression (NMS)
- Evaluation: IoU

What you learned

3. Image Segmentation

- Segmentation masks
- Instance segmentation: Mask R-CNN
- Semantic segmentation: U-Net
- Panoptic segmentation

What you learned

4. Image Generation with GANs

- Basic GAN
- Deep Convolutional GAN (DCGAN)
- Model training
- Evaluation: FID

Congratulations and good luck!

DEEP LEARNING FOR IMAGES WITH PYTORCH