UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING

TECSAR
Transformative Computer Systems and Architecture Research

Real-time AI
Lecture 3:Deep Fake

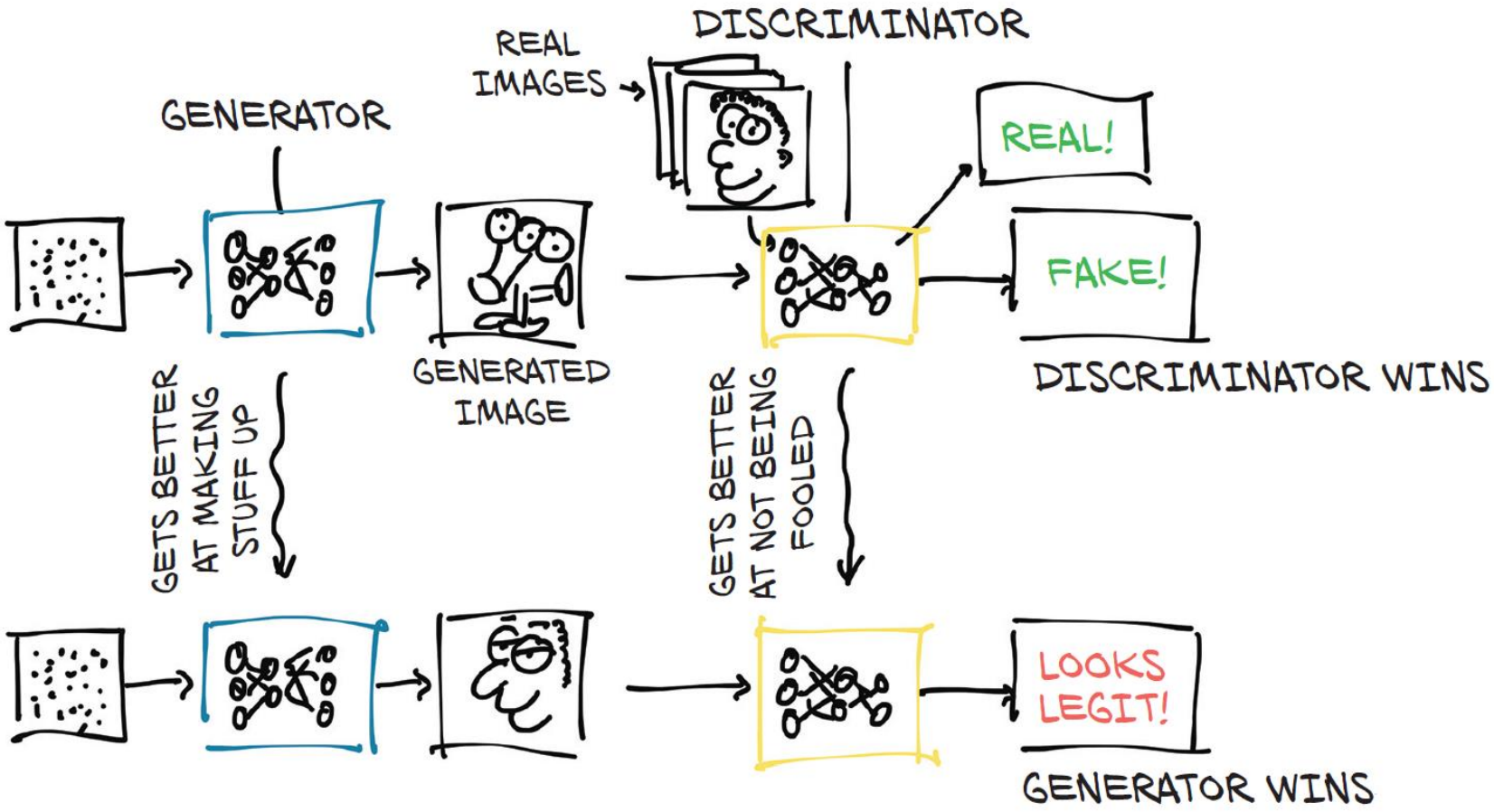
Hamed Tabkhi

Department of Electrical and Computer Engineering,

University of North Carolina Charlotte (UNCC)

htabkhiv@uncc.edu

Generative Adversarial Networks (GAN)



Generative Adversarial Networks (GAN)

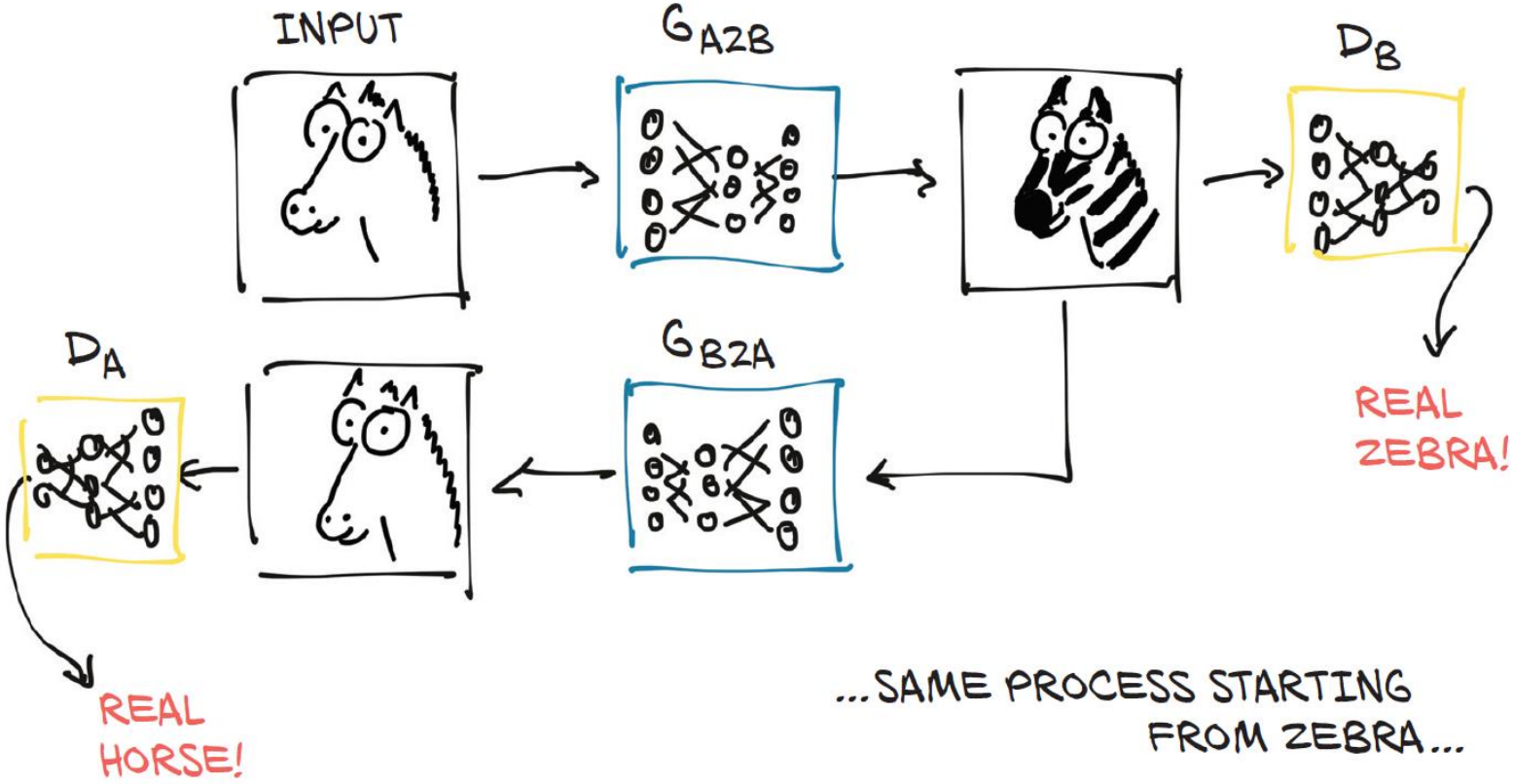
- *GAN game*, where two networks, one acting as the painter and the other as the art historian, compete to outsmart each other at creating and detecting forgeries.
- GAN stands for *generative adversarial network*.
- *Generative* means something is being created (in this case, fake masterpieces).
- *Adversarial* means the two networks are competing to outsmart the other.
- These networks are one of the most original outcomes of recent deep learning research.
- For our example: the *generator* network takes the role of the painter in our scenario, tasked with producing realistic-looking images, starting from an arbitrary input.
- For our example: the *discriminator* network is the amoral art inspector, needing to tell whether a given image was fabricated by the generator or belongs in a set of real images.

Generative Adversarial Networks (GAN)

- Note that “Discriminator wins” or “Generator wins” shouldn’t be taken literally— there’s no explicit tournament between the two.
- Both networks are trained based on the outcome of the other network, which drives the optimization of the parameters of each network.
- This technique has proven itself able to lead to generators that produce realistic images from nothing but noise and a conditioning signal, like an attribute (for example, for faces: young, female, glasses on) or another image.
- In other words, a well-trained generator learns a plausible model for generating images that look real even when examined by humans.

CycleGAN

- An interesting evolution of this concept is the CycleGAN.
- A CycleGAN can turn images of one domain into images of another domain (and back)



CycleGAN

- The first generator learns to produce an image conforming to a target distribution (zebras, in this case) starting from an image belonging to a different
- distribution (horses), so that the discriminator can’t tell if the image produced from a
- horse photo is actually a genuine picture of a zebra or not. At the same time—and
- here’s where the *Cycle* prefix in the acronym comes in—the resulting fake zebra is sent
- through a different generator going the other way (zebra to horse, in our case), to be
- judged by another discriminator on the other side. Creating such a cycle stabilizes the
- training process considerably, which addresses one of the original issues with GANs.

CycleGAN

- The fun part is that at this point, we don't need matched horse/zebra pairs as ground truths (good luck getting them to match poses!). It's enough to start from a collection of unrelated horse images and zebra photos for the generators to learn their task,
- It is going beyond a purely supervised setting!
- The implications of this model go even further than this: the generator learns how to selectively change the appearance of objects in the scene without supervision about what's what.
- There's no signal indicating that manes are manes and legs are legs, but they get translated to something that lines up with the anatomy of the other animal.

A network that turns horses into zebras

- The CycleGAN network has been trained on a dataset of (unrelated) horse images and zebra images extracted from the ImageNet dataset. The network learns to take an image of one or more horses and turn them all into zebras, leaving the rest of the image as unmodified as possible.
- Let's instantiate the class with default parameters (code/p1ch2/3_cyclegan.ipynb):

```
netG = ResNetGenerator()
```

- The `netG` model has been created, but it contains random weights.
- We would run a generator model that had been pretrained on the horse2zebra dataset, whose training set contains two sets of 1068 and 1335 images of horses and zebras, respectively.
- We can load those into `ResNetGenerator` using the model's `load_state_dict` method:

```
model_path = '../data/p1ch2/horse2zebra_0.4.0.pth'
model_data = torch.load(model_path)
netG.load_state_dict(model_data)
```

A network that turns horses into zebras

- Note that this is fully equivalent to what happened when we loaded `resnet101` from `torchvision`, but the `torchvision.resnet101` function hid the loading from us.
- Let's put the network in `eval` mode, as we did for `resnet101`:

```
netG.eval()
```

-
```
# Out[4]:
ResNetGenerator(
(model): Sequential(
...
)
```

- First, we need to import `PIL` and `torchvision`:

```
# In[5]:
from PIL import Image
from torchvision import transforms
```

- Then we define a few input transformations to make sure data enters the network with the right shape and size:

```
preprocess = transforms.Compose([transforms.Resize(256),
transforms.ToTensor()])
```

A network that turns horses into zebras

```
img = Image.open("../data/p1ch2/horse.jpg")
Img
```

- let's pass it through preprocessing and turn it into a properly shaped variable:

```
img_t = preprocess(img)
batch_t = torch.unsqueeze(img_t, 0)
```

- At this point, `batch_t` can be sent to our model:

```
batch_out = netG(batch_t)
```

- `batch_out` is now the output of the generator, which we can convert back to an image:

```
out_t = (batch_out.data.squeeze() + 1.0) / 2.0
out_img = transforms.ToPILImage()(out_t)
```

A network that turns horses into zebras

```
out_img
```

- Many other fun generators have been developed using adversarial training or other approaches.
- Some of them are capable of creating credible human faces of nonexistent individuals.
- Others can translate sketches into real-looking pictures of imaginary landscapes.
- Generative models are also being explored for producing real-sounding audio, credible text, and enjoyable music.