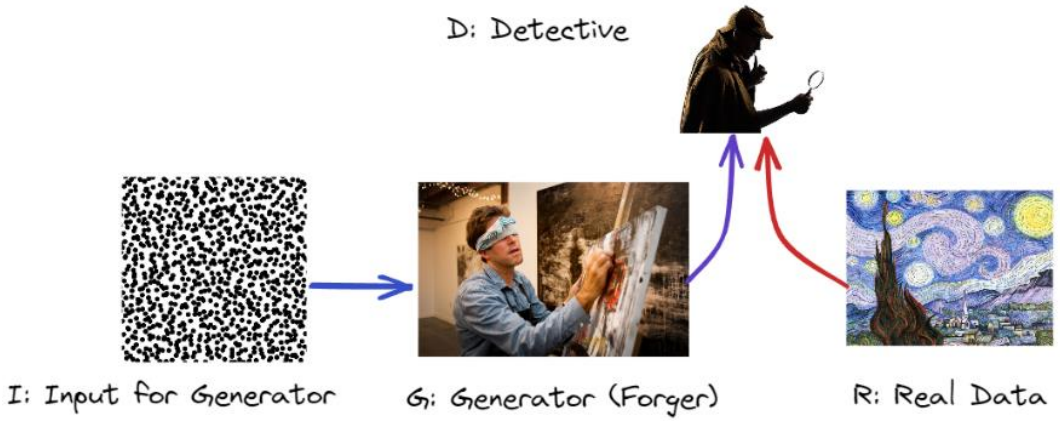
Introduction to
Generative Adversarial Networks
GANS

WORKSHOP

https://f21dl.github.io/workshop/gans/

Thursday 19th October (Week 6) in LT1 at 9-10am and EM245 and EM252 10am-5pm.



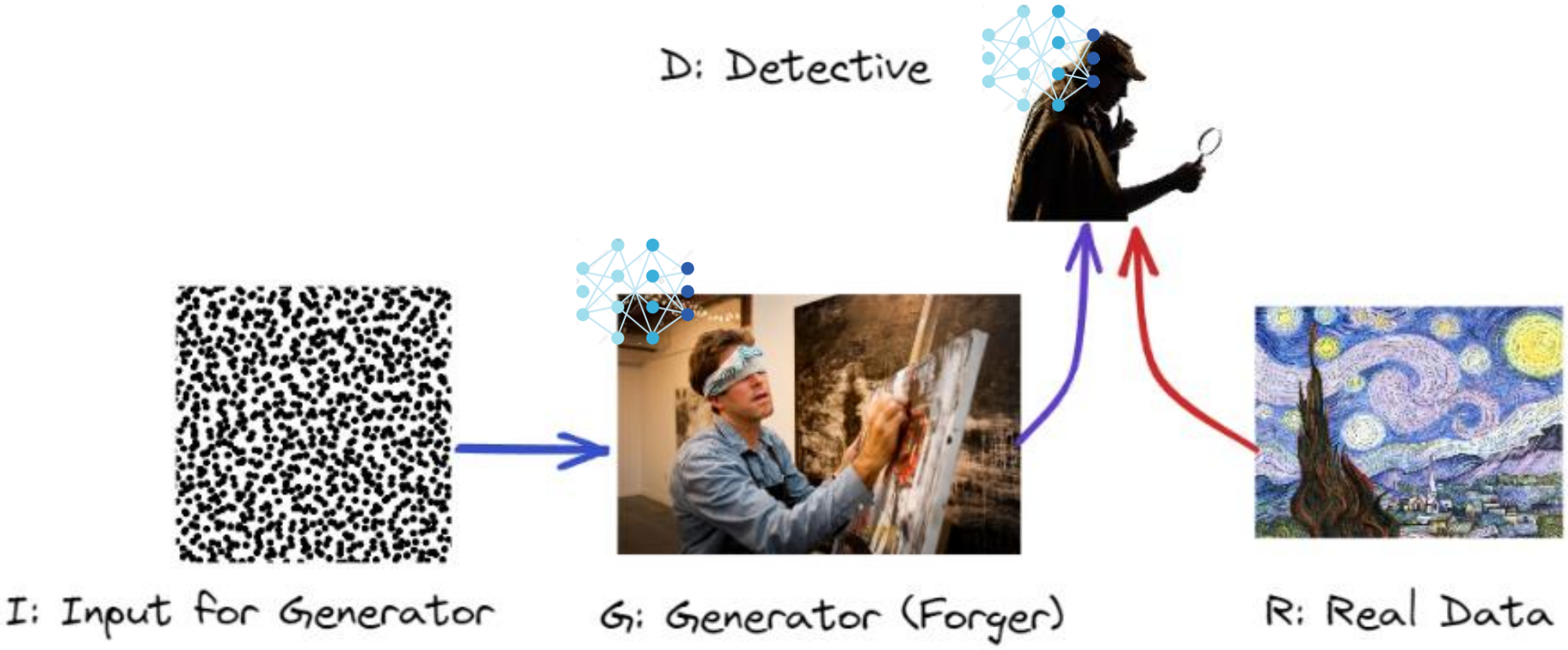
Benjamin Kenwright
Data Mining and Machine Learning
b.kenwright@hw.ac.uk

Introduction

- Generative adversarial networks (GANs) were proposed in 2014 paper [1]
- A GAN is composed of two neural networks (Generator and Discriminator)
- Generator: Takes a random distribution as input (typically Gaussian) and outputs some data (e.g., image). You can think of the random inputs as the latent representations (i.e., codings) of the image to be generated
- Discriminator: Takes either a fake image from the generator or a real image from the training set as input, and must decide if the input image is fake or real

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. Advances in neural information processing systems, 27.

Introduction



GAN Applications

- Generate photographs of human faces
- Generate realistic photographs
- Generate cartoon characters
- Generate examples for image datasets
- Image-to-image translation
- Text-to-image translation
- Semantic-image-to-photo translation
- Face frontal view generation
- Generate new human poses
- Face aging
- Photograph editing
- Photos to emojis
- Photo blending
- Super resolution
- Clothing translation
- Photo inpainting
- 3D object generation
- Video prediction

Example GAN

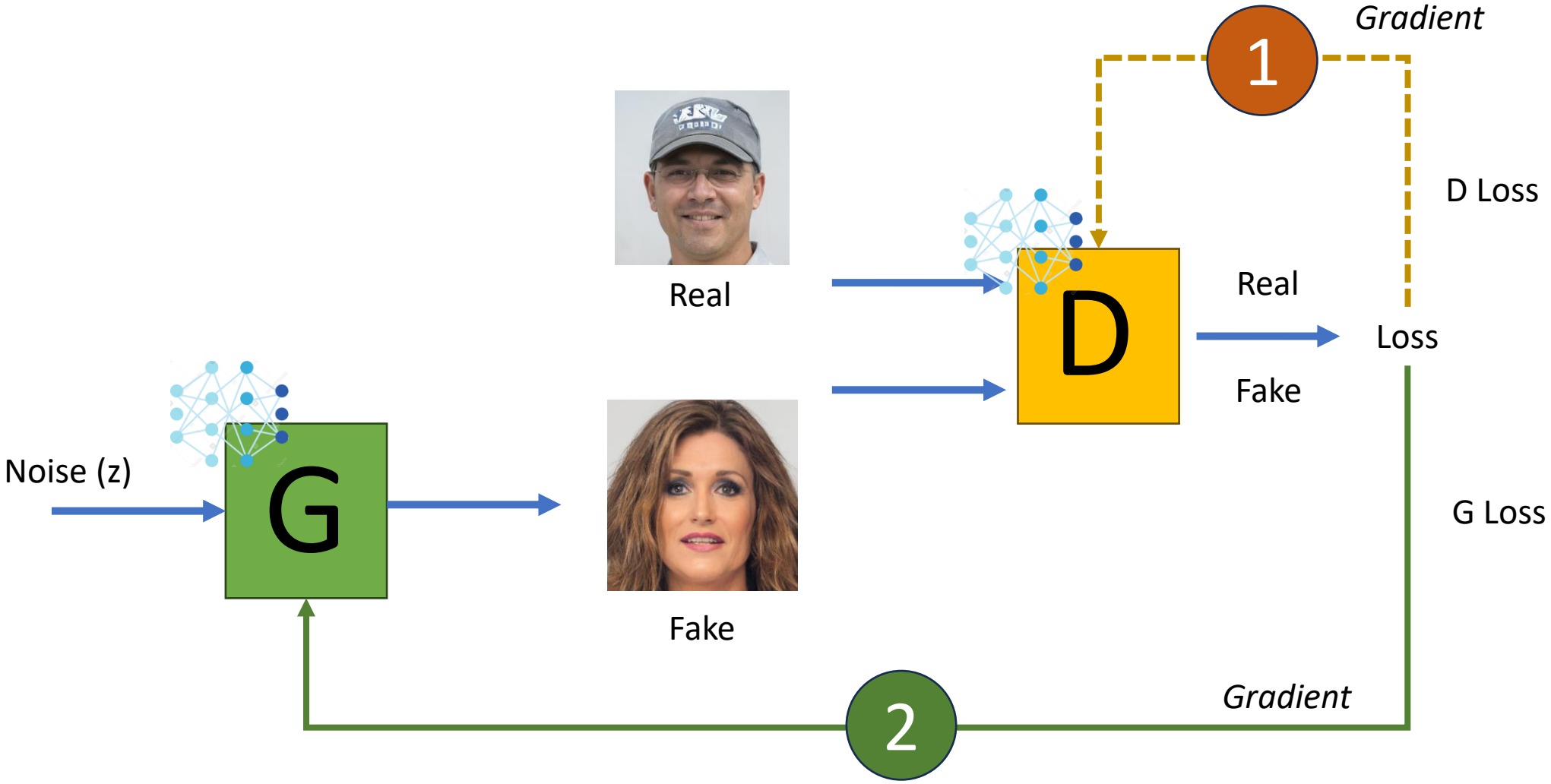
https://thispersondoesnotexist.com/

“thispersondoesnotexist.com” is a website which uses an AI GAN model to generate new faces every-time you refresh the page

GAN Training

- The generator and discriminator **have opposite goals**
  - Discriminator tries to tell fake images from real images
  - Generator tries to produce images that look real enough to trick the discriminator

- GAN is composed of two networks with different objectives, it **cannot be trained like a regular neural network**.
  - Each training iteration is divided into two phases


Gradient
1
D Loss
Real
D
Real
Loss
Fake
Noise (z)
G
Fake
G Loss
2
Gradient

Phase One

- Train **the discriminator**
  - Batch of real images is sampled from the training set and is completed with an equal number of fake images produced by the generator (labels are: **0=fake images**, **1=real images**)
- The discriminator is trained on this labelled batch for one step, using the **binary cross-entropy loss**
- **Backpropagation** only optimizes the **weights of the discriminator** during this phase

Phase Two

- Train **the generator**
  - Produce a new batch of fake images, once again the discriminator says if the images are fake or real
- This time we **do not add real images in the batch** (generator never actually sees any real images)
- The **weights of the discriminator are frozen** during this step, so the **backpropagation** only affects the **weights of the generator**

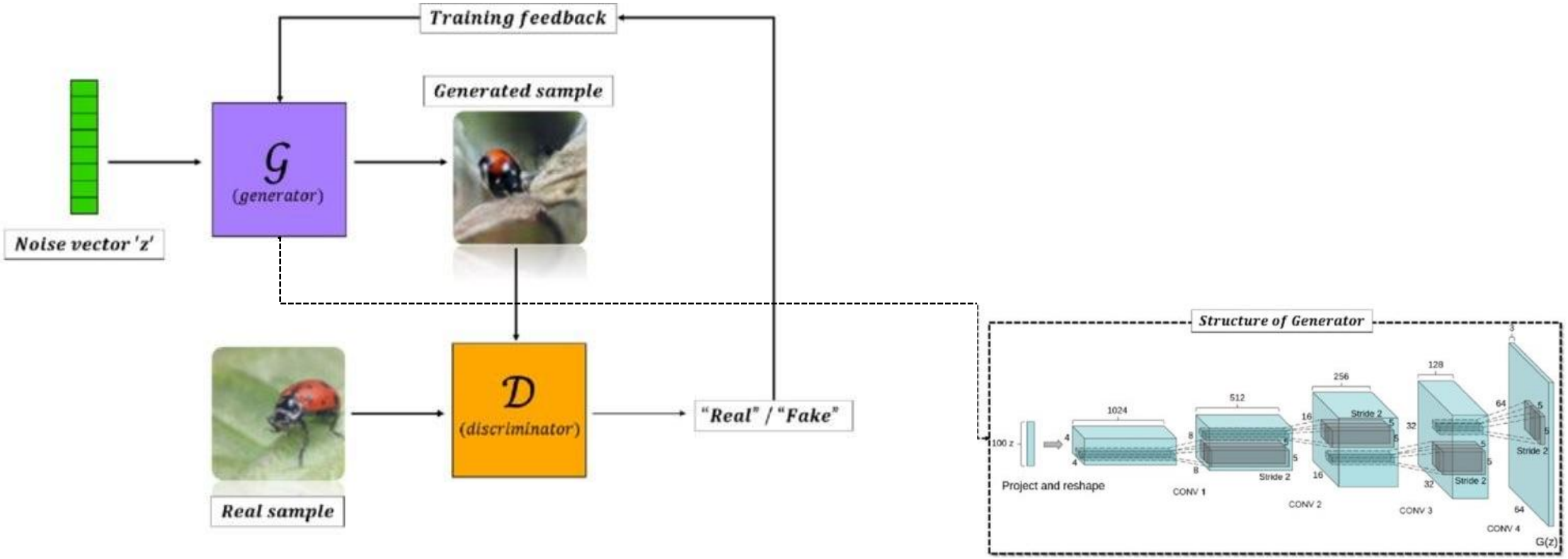
Common Problems

- **Vanishing Gradients**: when the discriminator doesn't provide enough information for the generator to make progress (original GAN paper proposed a modification to the minmax loss to deal with vanishing gradient [2])
- **Mode Collapse**: when the generator starts producing the same output (or a small set of outputs) over and over again. How can this happen? Suppose the generator gets better at producing convincing one type of result. It will fool the discriminator a bit more and will encourage it to produce more types of this image. Gradually it will forget how to produce anything else.
- **GANs sensitive to the hyperparameters**: may need to spend a lot of effort fine-tuning them

[2] https://developers.google.com/machine-learning/gan/loss/

Deep Convolution GANs

Deep Convolutional GANs (**DCGANs**) 2015



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

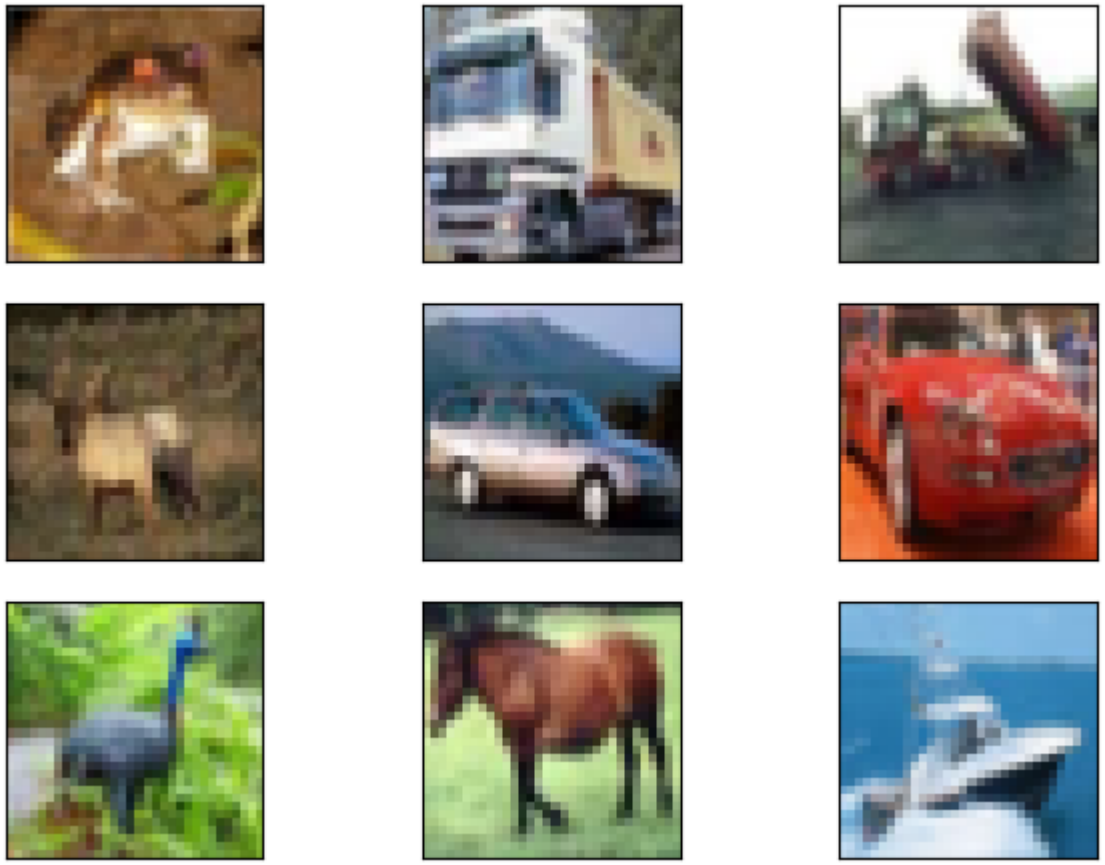
Deep Convolution GANs

Vanilla GAN to DCGAN

Outline of steps for building a stable Convolutional GAN

1. Replace any pooling layers with stridded convolutions (in the discriminator) and transpose convolutions (in the generator)
2. Use Batch Normalization in both the generator and the discriminator, except in the generator's output layer and the discriminator's input layer
3. Remove fully connected hidden layers for deep architectures
4. Use ReLU activation in the generator for all layers except the output layer, which should use tanh
5. Use leaky ReLU activation in the discriminator for all layers

Example: Preparing Dataset cifar10



```
import tensorflow as tf # pip install tensorflow
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np


# using Keras to load dataset
(X_train, y_train),(X_test, y_test) = keras.datasets.cifar10.load_data()
print("X_train shape=", X_train.shape," X_test shape =", X_test.shape)

fig = plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.xticks([])
    plt.yticks([])


# scale the pixel intensities from 0 to 255 to [0,1] range
X_train = X_train.astype("float32")/255.0

# create create a dataset to iterate through images
batch_size=128
dataset=tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset=dataset.batch(batch_size, drop_remainder=True).prefetch(1)

plt.show()
```

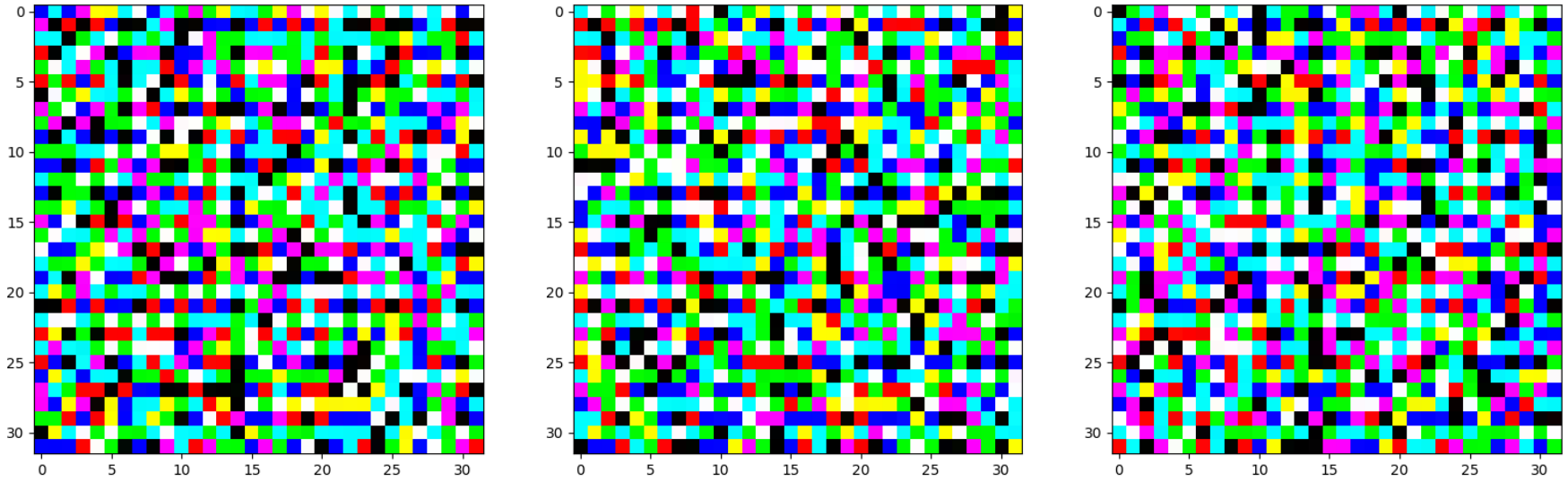
Example: The Generator

```
# coding size - the dimension of the input vector for the generator
codingSize = 100

def buildGenerator( codingSize=100 ):
    generator = tf.keras.Sequential()

    # latent variable as input
    generator.add(keras.layer.Dens(1024, activation="relu", input_shape=(codingSize,) ) )
    generator.add(keras.layer.BatchNormalization())
    generator.add(keras.layers.Dense(1024, activation="relu") )
    generator.add(keras.layers.BatchNormalization())
    generator.add(keras.layers.Dense(128*8*8, ativation="relu") )
    generator.add(keras.layers.Reshape((8,8,128) ) )
    assert generator.output_shape == (None, 8, 8, 128) # None is the batch size

    generator.add(keras.layers.Conv2DTranspose(filters=128,kernel_size=2, strides=2, activation="relu", padding="same") )
    assert generator.output_shape == (None,16,16,128)
    generator.add(keras.layers.BatchNormalization() )

    generator.add(keras.layers.Conv2DTranspose(filters=3, kernel_size=2, strides=2, activation="tanh", padding="same") )
    assert generator.output_shape == (None, 32, 32, 3)
    generator.add(keras.layers.BatchNormalization() )

    return generator
```

Example: The Generator (Plot)

```
generator = buildGenerator()
nbrImgs = 3

def plotGeneratedImages( nbrImgs, titleadd="" ):
    noise = tf.random.normal( [nbrImgs, 100] )
    imgs = generator.predict(noise)

    fig = plt.figure(figsize=(40,10))
    for i, img in enumerate(imgs):
        ax = fig.add_subplot(1,nbrImgs,i+1)
        ax.imshow( (img*255).astype(np.uint8) )
    fig.suptitle( "Gen images"+titleadd, fontsize=25 )
    plt.show()
```

```
plotGeneratedImages( nbrImgs )
```



Example: The Discriminator

```
# the discriminator
def buildDiscriminator():
    discriminator = tf.keras.Sequential()

    discriminator.add(keras.layers.Conv2D(filters=64,kernel_size=3,strides=2,activation=keras.layers.LeakyReLU(0.2), padding="same",
                                          input_shape=(32,32,3) ) )
    discriminator.add(keras.layers.Conv2D(filters=128,kernel_size=3, strides=2, activation=keras.layers.LeakyReLU(0.2),padding="same") )
    discriminator.add(keras.layers.Conv2D(filters=128, kernel_size=3, strides=2, activation=keras.layers.LeakyReLU(0.2), padding="same") )
    discriminator.add(keras.layers.Conv2D(filters=256, kernel_size=3, strides=2, activation=keras.layers.LeakyReLU(0.2), padding="same" ) )

    # classifier
    discriminator.add(keras.layers.Flatten() )
    discriminator.add(keras.layers.Dropout(0.4) )
    discriminator.add(keras.layers.Dense(1,activation="sigmoid") )
    return discriminator

discriminator = buildDiscriminator()
# compile our model
opt = keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)
discriminator.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
discriminator.trainable = False
```

Example: Training the GAN

```
gan = keras.models.Sequential( [ generator, discriminator ] )
# compile the gan
opt = keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)
gan.compile( loss="binary_crossentropy", optimizer=opt )

# Comine images into 'gif' (store animations)
from PIL import Image
import cv2 # pip install opencv-python
images = []

def animatedGif():
    noise_1 = tf.random.normal(shape=[4,codingSize] )
    imgs = generator.predict(noise_1)
    img0 = (imgs[0]*255).astype(np.uint8)
    img1 = (imgs[1]*255).astype(np.uint8)
    img2 = (imgs[2]*255).astype(np.uint8)
    img3 = (imgs[3]*255).astype(np.uint8)

    img = cv2.hconcat([img0, img1, img2, img3])
    img = Image.fromarray(np.uint8(img)).convert("RGB")
    return img
```

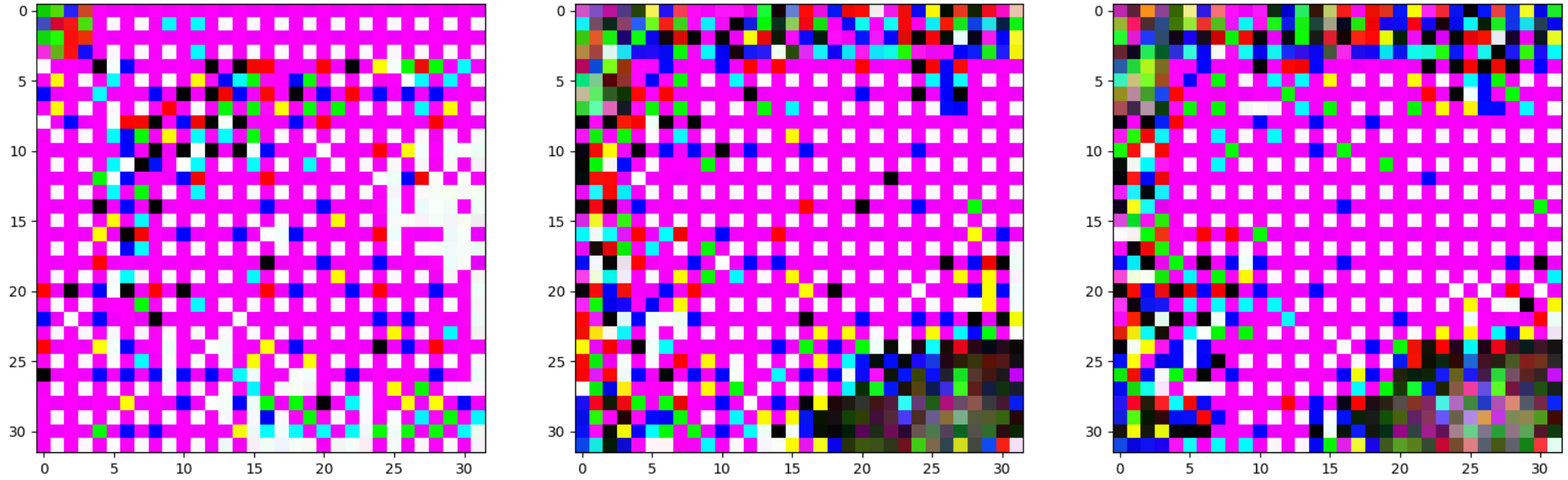
Example: Train the GAN (Loop)

```
print('----')
def trainGAN(gan, dataset, batchSize, codingsSize, nEpochs):
    generator, discriminator = gan.layers
    for epoch in range( nEpochs ):
        for X_batch in dataset:
            # phase 1 - training discriminator
            noise = tf.random.normal(shape=[batchSize, codingsSize] )
            generatedImages = generator.predict(noise)
            X_fake_and_real = tf.concat( [ generatedImages, X_batch], axis=0 )
            y1 = tf.constant( [[0.0]]*batchSize + [[1.0]]*batchSize )
            discriminator.trainable=True
            d_loss_accuracy = discriminator.train_on_batch( X_fake_and_real, y1 )

            # phase 2 - training the generator
            noise = tf.random.normal( shape=[batchSize, codingsSize] )
            y2 = tf.constant( [[1.0]] * batchSize )
            discriminator.trainable = False
            g_loss = gan.train_on_batch( noise, y2 )

        print("epoch:",epoch," d_loss_accuracy:",d_loss_accuracy, "g_loss:", g_loss )
        plotGeneratedImages( 3, titleadd=":Epoch{}".format(epoch) )
        # create animated gif
        img = animatedGif()
        images.append(img)
        print("---")
```

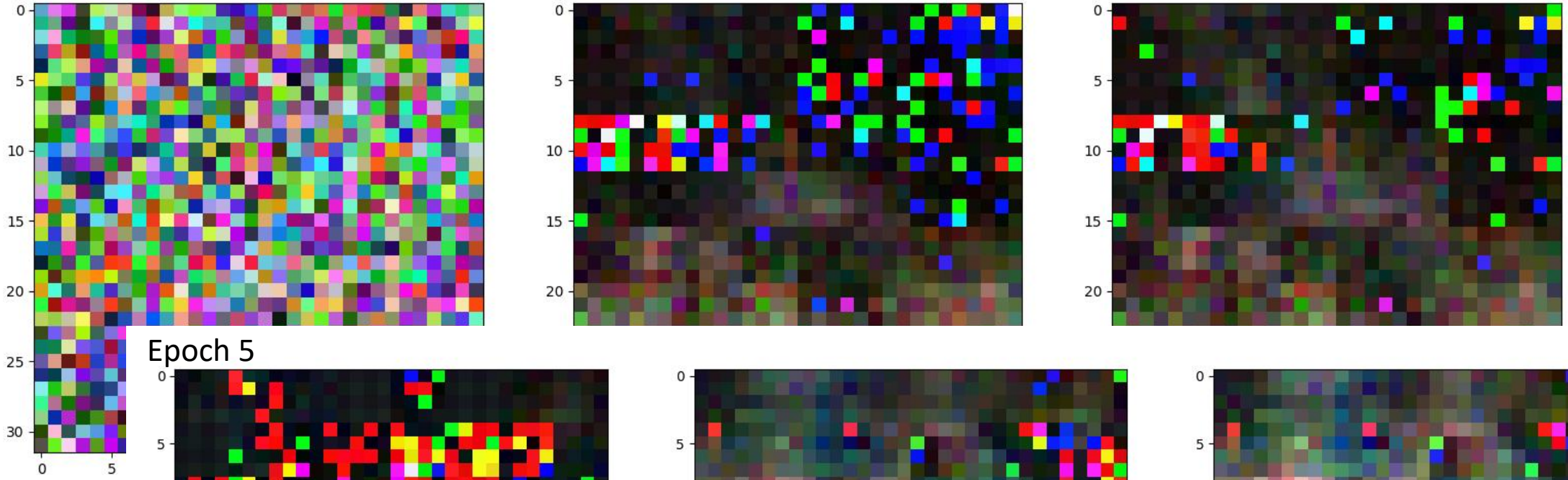
Example: Train the GAN (Result)

```
nEpochs = 100
trainGAN( gan, dataset, batchSize, codingSize, nEpochs )

# create gif - images at each epoch
images[0].save('./genImages.gif', save_all=True, append_Images=images[1:], optimize=False, duration=500, loop=0)
```

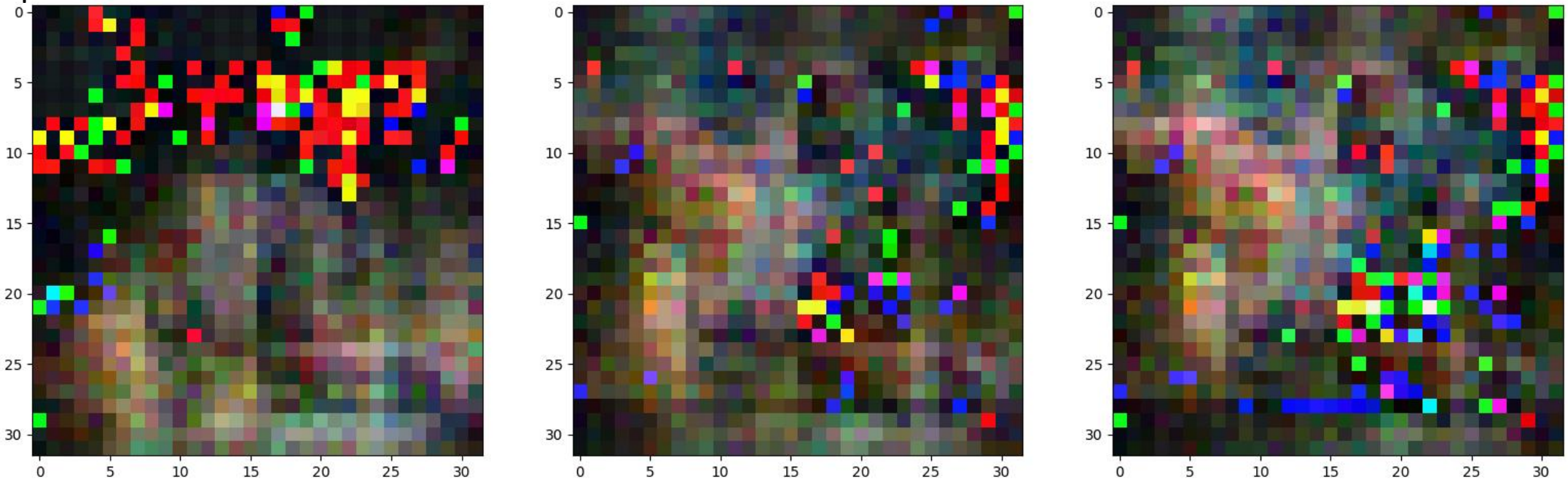
Epoch 1



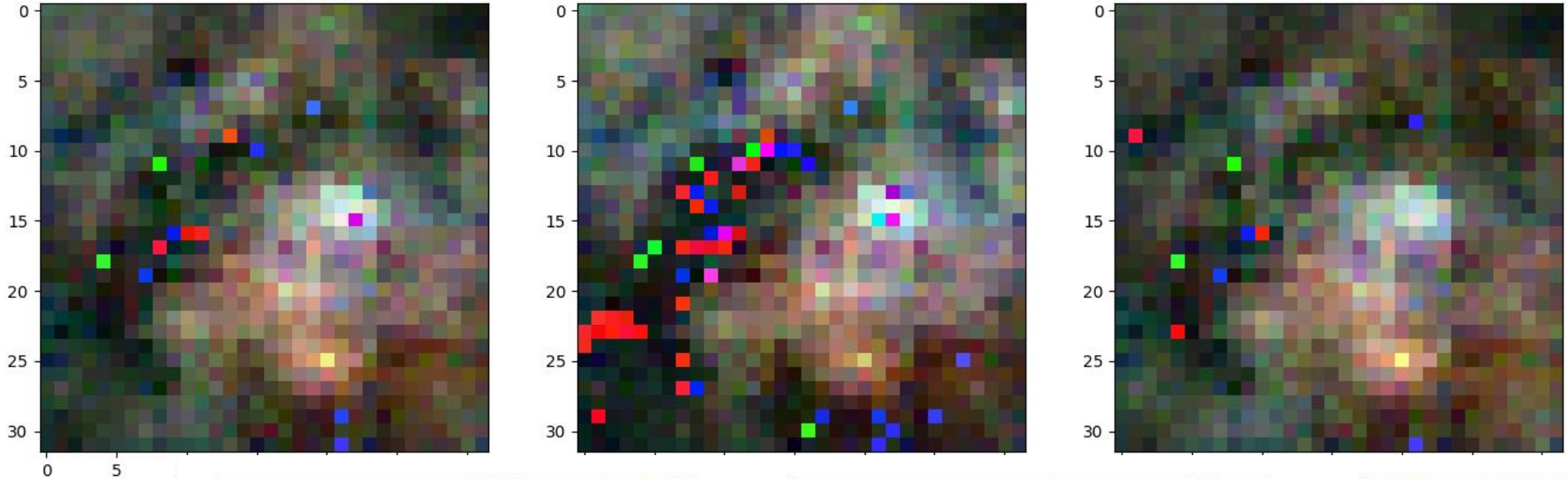
Epoch 4



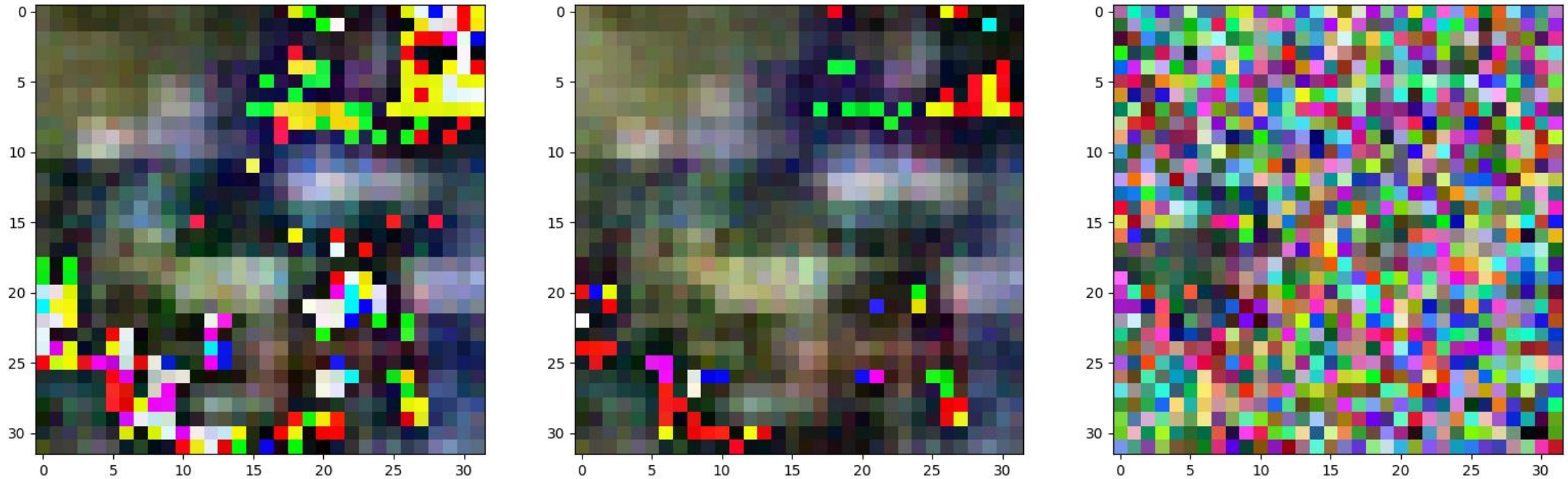
Epoch 5



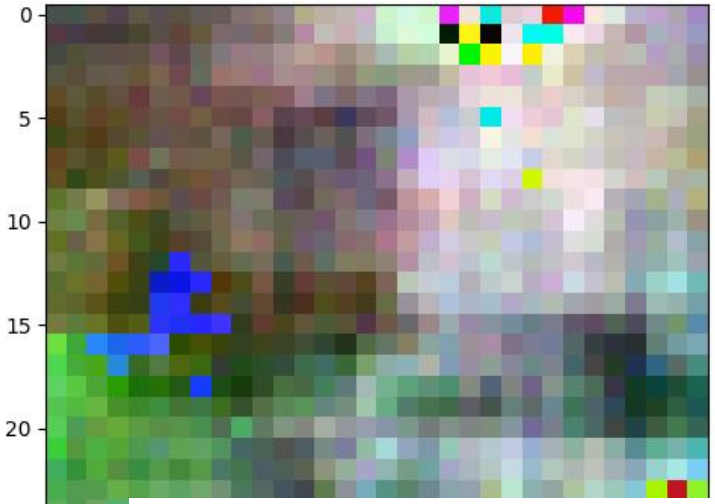
Epoch 6



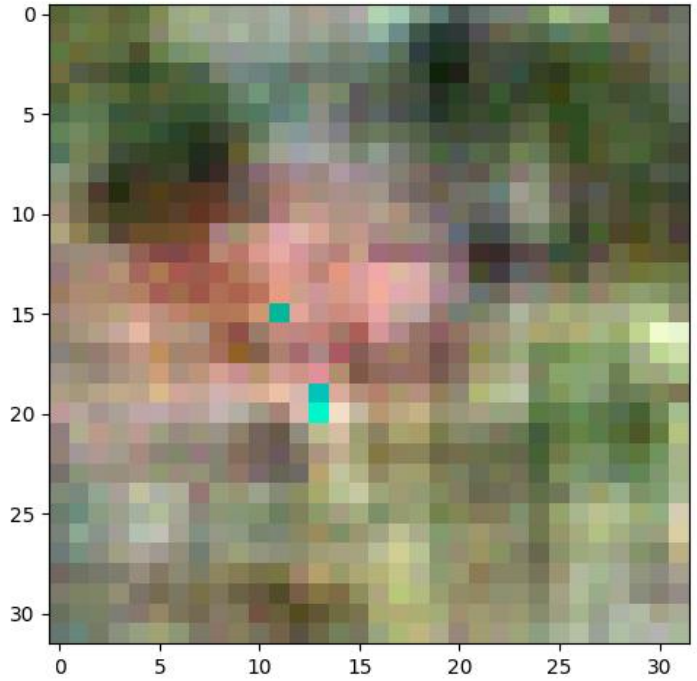
Epoch 9

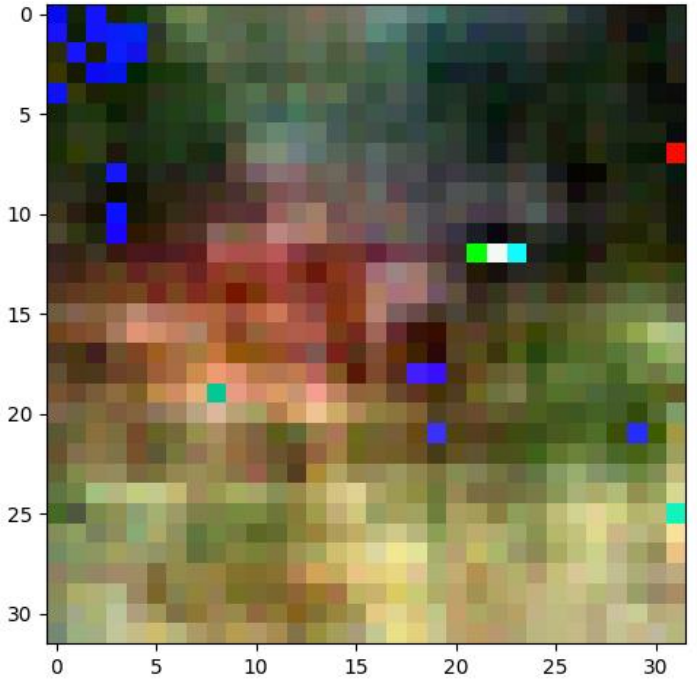


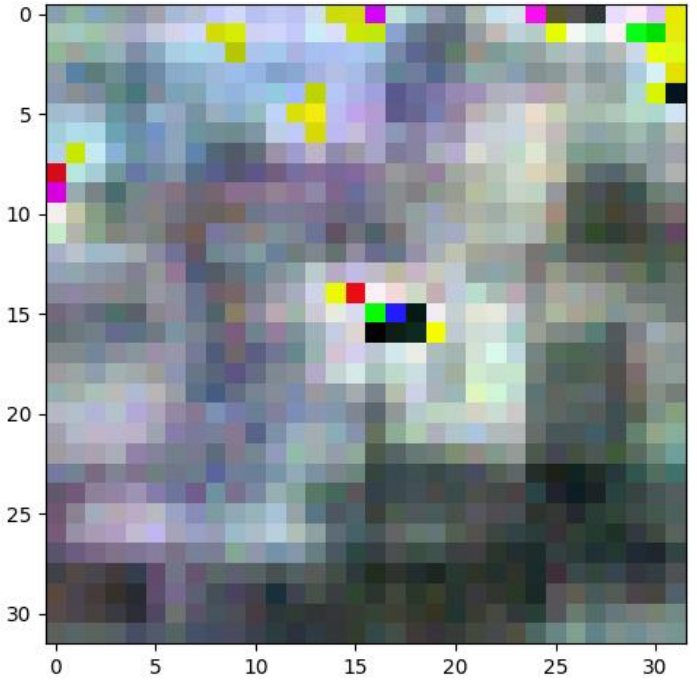
Epoch 12



Epoch 14







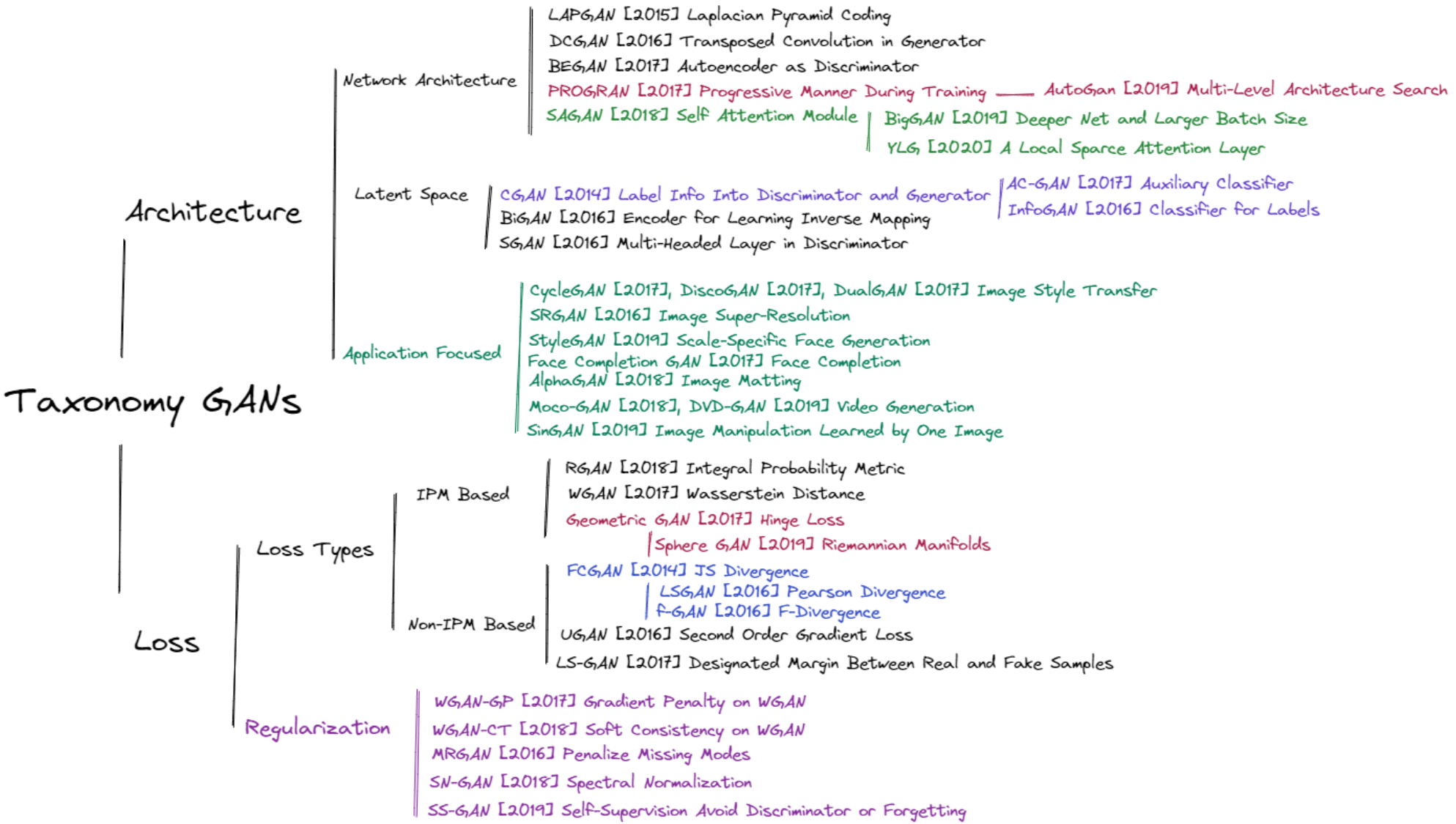
Example: AttGAN

AttGAN – Arbitrary Facial Attribute Editing (Only Change What You Want)

https://github.com/elvisyjlin/AttGAN-PyTorch

[Bald, Bangs, Black_Hair, Blond_Hair, Brown_Hair, Bushy_Eyebrows, Eyeglasses, Male, Mouth_Slightly_Open, Mustache, No_Beard, Pale_Skin]

Recent GANs (Taxonomy)



Recent GANs (Survey Paper)

Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy

ZHENGWEI WANG, School of Computer Science and Statistics, Trinity Collge Dublin, Ireland
QI SHE, ByteDance AI Lab, Beijing, China
TOMÁS E. WARD, Insight Centre for Data Analytics, School of Computing, Dublin City University, Ireland

Generative adversarial networks (GANs) have been extensively studied in the past few years. Arguably their most significant impact has been in the area of computer vision where great advances have been made in challenges such as plausible image generation, image-to-image translation, facial attribute manipulation, and similar domains. Despite the significant successes achieved to date, applying GANs to real-world problems still poses significant challenges, three of which we focus on here. These are as follows: (1) the generation of high quality images, (2) diversity of image generation, and (3) stabilizing training. Focusing on the degree to which popular GAN technologies have made progress against these challenges, we provide a detailed review of the state-of-the-art in GAN-related research in the published scientific literature. We further structure this review through a convenient taxonomy we have adopted based on variations in GAN architectures and loss functions. While several reviews for GANs have been presented to date, none have considered the status of this field based on their progress toward addressing practical challenges relevant to computer vision. Accordingly, we review and critically discuss the most popular architecture-variant, and loss-variant GANs, for tackling these challenges. Our objective is to provide an overview as well as a critical analysis of the status of GAN research in terms of relevant progress toward critical computer vision application requirements. As we do this we also discuss the most compelling applications in computer vision in which GANs have demonstrated considerable success along with some suggestions for future research directions. Codes related to the GAN-variants studied in this work is summarized on https://github.com/sheqi/GAN_Review.

GANs in NLP

Article **explores the use of a GAN** for **NLP** tasks
(Proposes a GAN architecture)

https://arxiv.org/abs/1905.01976

TextKD-GAN: Text Generation using Knowledge Distillation and Generative Adversarial Networks

Md. Akmal Haidar and Mehdi Rezagholizadeh
{md.akmal.haidar, mehdi.rezagholizadeh}@huawei.com

Huawei Noah's Ark Lab, Montreal Research Center, Montreal, Canada

Abstract. Text generation is of particular interest in many NLP applications such as machine translation, language modeling, and text summarization. Generative adversarial networks (GANs) achieved a remarkable success in high quality image generation in computer vision, and recently, GANs have gained lots of interest from the NLP community as well. However, achieving similar success in NLP would be more challenging due to the discrete nature of text. In this work, we introduce a method using knowledge distillation to effectively exploit GAN setup for text generation. We demonstrate how autoencoders (AEs) can be used for providing a continuous representation of sentences, which

Questions

Ideas

Starter Code
https://f21dl.github.io/workshop/gans/

- Cartoon dataset
- Fonts/text
- Frames from a film (e.g., convert the frames to single images)
- Tuning/speed-ups (noise, ..)
- Products/designs
- Street maps
- Game level designs
- Instead of human faces focus on other objects, hands, feet, chairs, glasses, ..

Thursday 19th October (Week 6) in LT1 at 9-10am and EM245 and EM252 10am-5pm.

Try Starter Code
https://f21dl.github.io/workshop/gans/

Experiment