IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PIX2PIX
DEEP LEARNING PROJECT

BAO Caifeng
MARTIN Pierre-Jean

SUMMARY

1. CONTEXT

1.1 Problematic

1.2 Article Presentation

1.3 Relation with the course

2. CODE

2.1 Loading the Dataset

2.2 Building the CAN model

3. RESULTS

3.1 Loss & Accuracy

3.2 Façades Visualisation

3.3 Personnal test : Pokemon

3.4 Extend : Can a bag be a Pokemon ?

CONCLUSION

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PART 1

CONTEXT

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PART 1 : CONTEXT

1.1 Problematic

Image-to-Image model

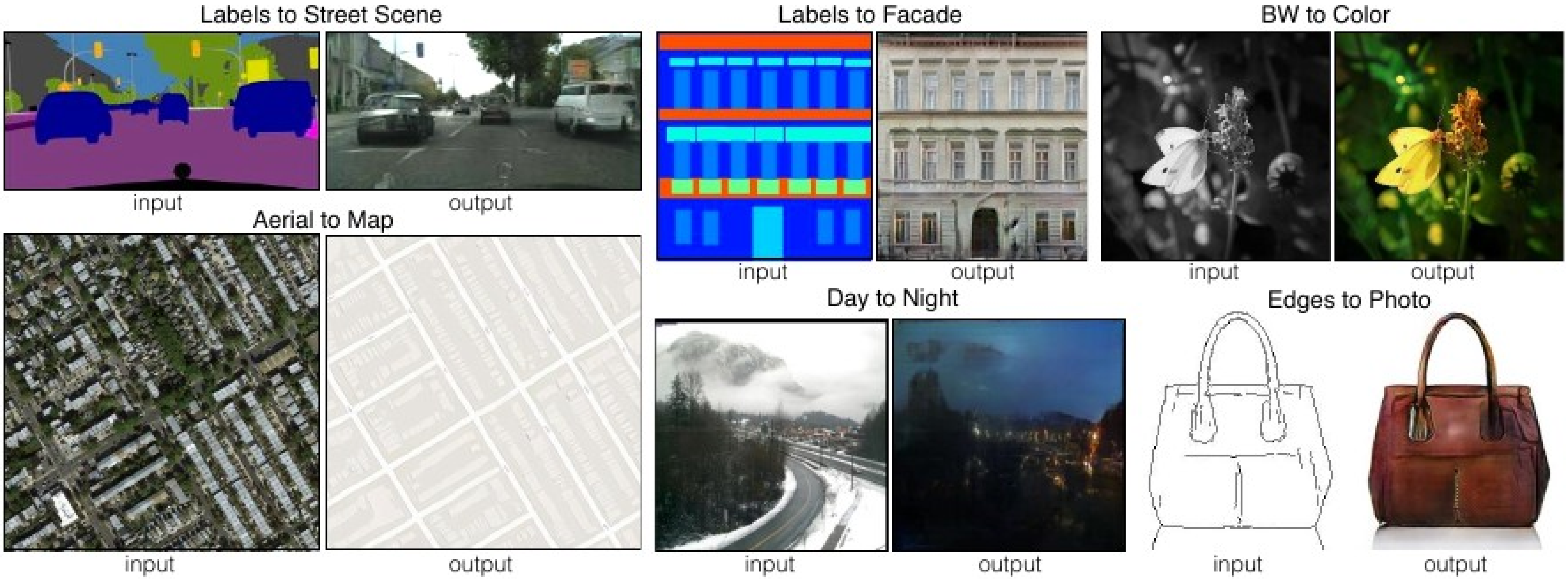
→ **Takes an Image in input and outputs an other one related.**

Setting is always the same : map pixel to pixel.
This type of Model can be used in several applications such as :

- Edges To Photos ;
- Black & White To Color ;
- Labels to Facade ;
- Day To Night ;

PART 1 : CONTEXT

1.1 Problematic

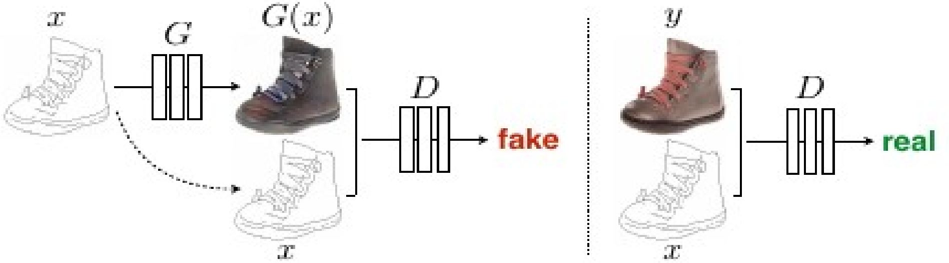


PART 1 : CONTEXT

1.2 Article Presentation

The article provided present the method.

→ **We use a cGAN model with a Generator and a Discriminator**



- Minimax objective function : $G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$

- With : $$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))],$$

PART 1 : CONTEXT

1.3 Relation With the Course

The cGAN is part of the Generative Models

→ **Like GAN but more conditionnal**

Normal GAN : $G : z \rightarrow y$

cGAN : $G : \{x, z\} \rightarrow y.$

→ **Add labels to the generator to make it better.**

PART 2
CODE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PART 2 : CODE

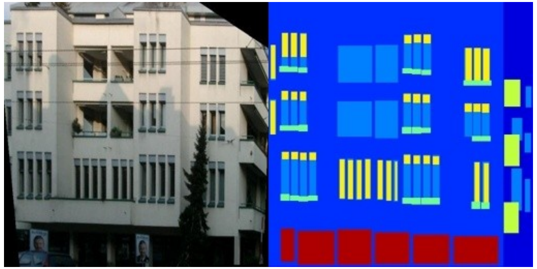
2.1 Loading the Dataset

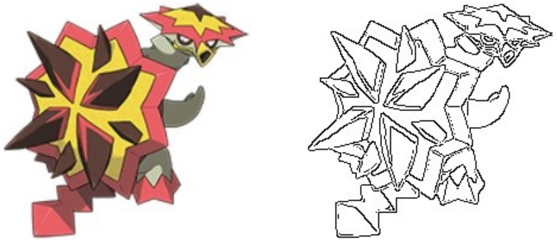
Class DataLoader

→Prepare the dataset, resize the shapes, normalize the pixels, set the flags (train & test),…

2 methods : load_data & load_batch

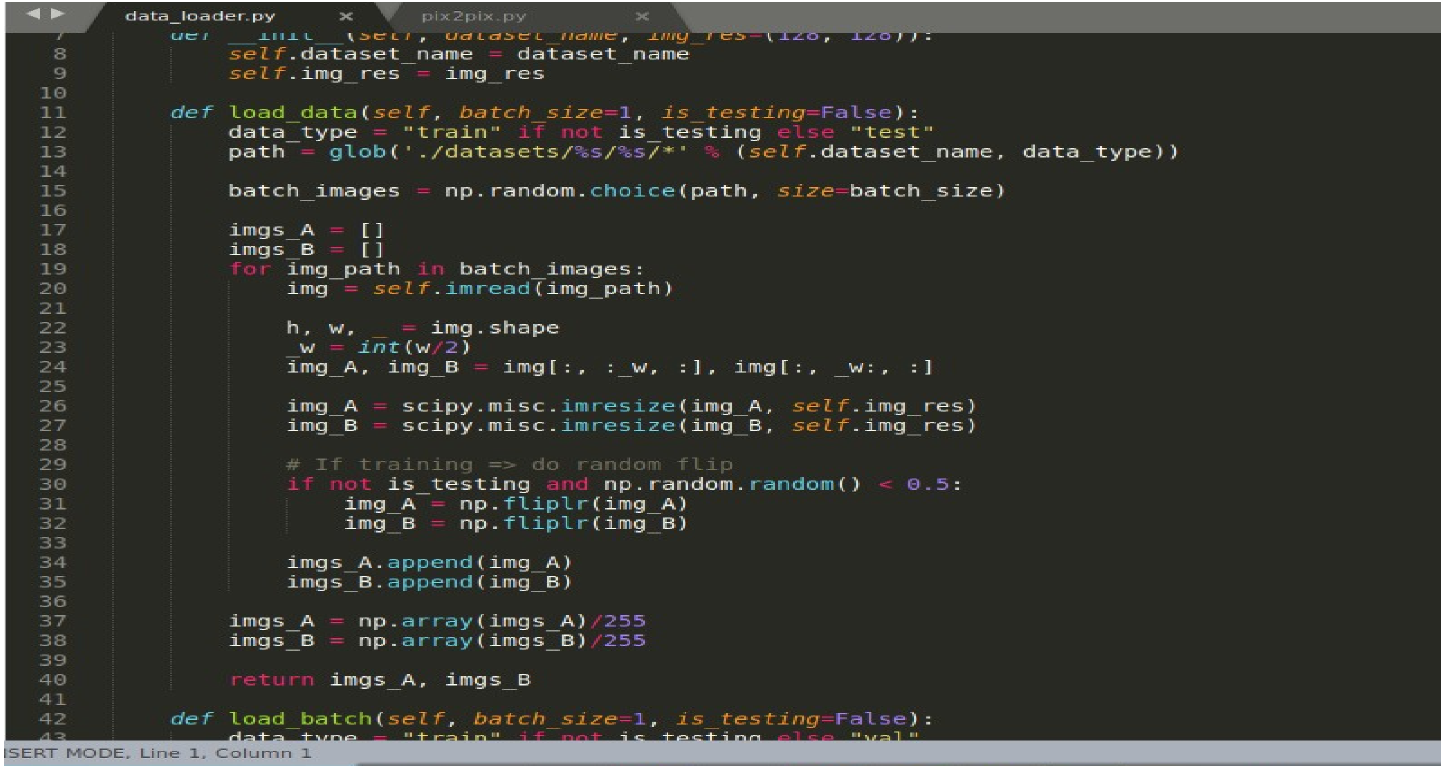
→Separe the images, concatene it into one array for the training.





PART 2 : CODE

2.1 Loading the Dataset



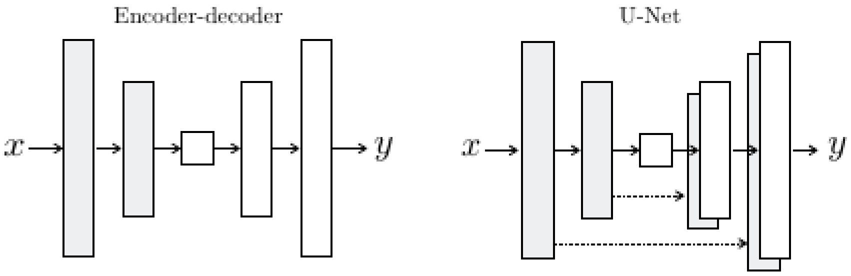
```
    def __init__(self, dataset_name, img_res=(128, 128)):
        self.dataset_name = dataset_name
        self.img_res = img_res

    def load_data(self, batch_size=1, is_testing=False):
        data_type = "train" if not is_testing else "test"
        path = glob('./datasets/%s/%s/*' % (self.dataset_name, data_type))

        batch_images = np.random.choice(path, size=batch_size)

        imgs_A = []
        imgs_B = []
        for img_path in batch_images:
            img = self.imread(img_path)

            h, w, _ = img.shape
            _w = int(w/2)
            img_A, img_B = img[:, :_w, :], img[:, _w:, :]

            img_A = scipy.misc.imresize(img_A, self.img_res)
            img_B = scipy.misc.imresize(img_B, self.img_res)

            # If training => do random flip
            if not is_testing and np.random.random() < 0.5:
                img_A = np.fliplr(img_A)
                img_B = np.fliplr(img_B)

            imgs_A.append(img_A)
            imgs_B.append(img_B)

        imgs_A = np.array(imgs_A)/255
        imgs_B = np.array(imgs_B)/255

        return imgs_A, imgs_B

    def load_batch(self, batch_size=1, is_testing=False):
        data_type = "train" if not is_testing else "val"
```

```
ISERT MODE, Line 1, Column 1
```

Class Pix2Pix

→Builds the architectures, trains the models, predicts and saves predictions

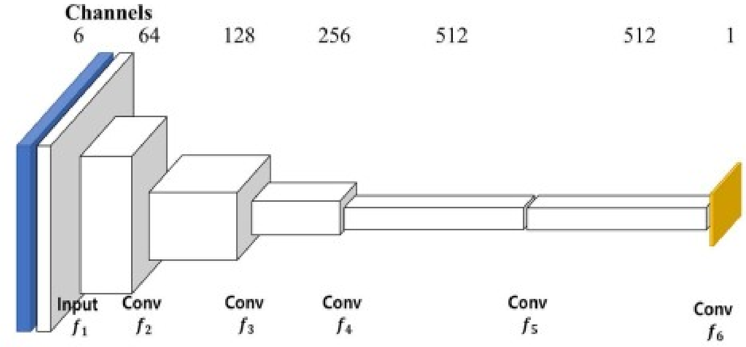
The generator :



Class Pix2Pix

→**Builds the architectures, trains the models, predicts and saves predictions**

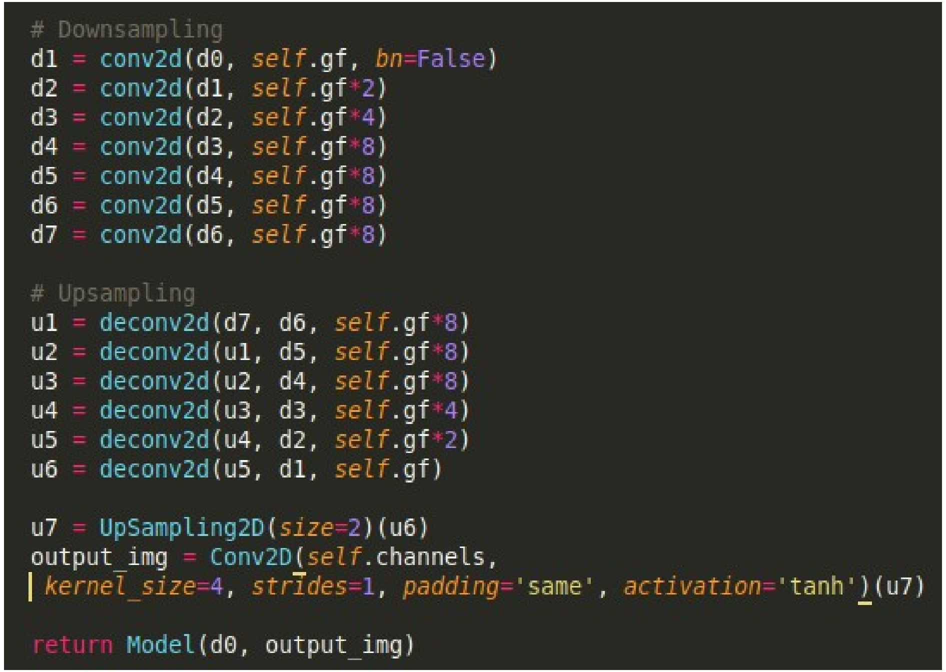
The Discriminator ;



PART 2 : CODE

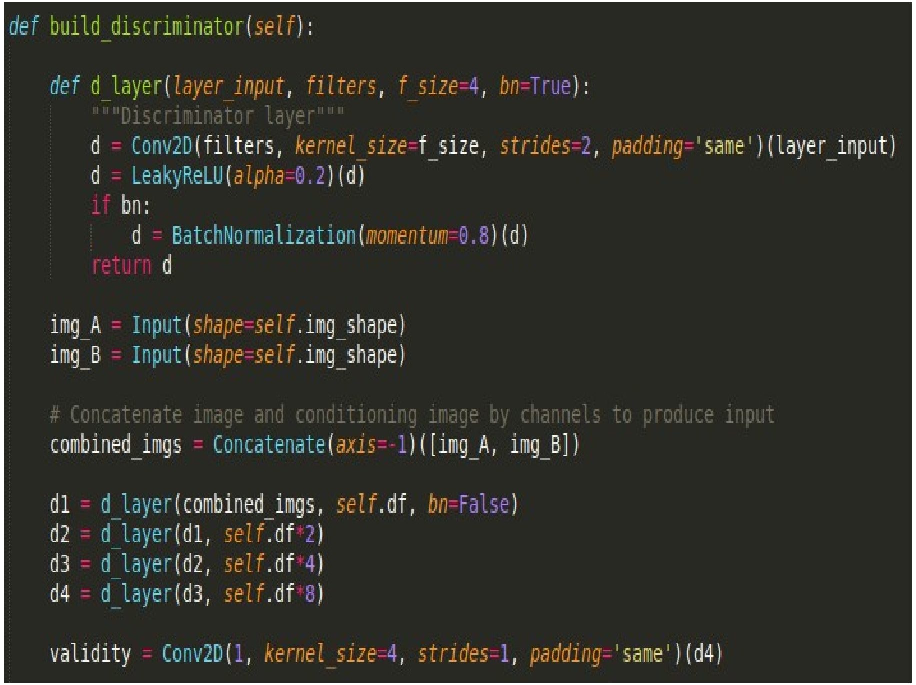
2.2 Building the Model

GENERATOR



```
# Downsampling
d1 = conv2d(d0, self.gf, bn=False)
d2 = conv2d(d1, self.gf*2)
d3 = conv2d(d2, self.gf*4)
d4 = conv2d(d3, self.gf*8)
d5 = conv2d(d4, self.gf*8)
d6 = conv2d(d5, self.gf*8)
d7 = conv2d(d6, self.gf*8)

# Upsampling
u1 = deconv2d(d7, d6, self.gf*8)
u2 = deconv2d(u1, d5, self.gf*8)
u3 = deconv2d(u2, d4, self.gf*8)
u4 = deconv2d(u3, d3, self.gf*4)
u5 = deconv2d(u4, d2, self.gf*2)
u6 = deconv2d(u5, d1, self.gf)

u7 = UpSampling2D(size=2)(u6)
output_img = Conv2D(self.channels,
 kernel_size=4, strides=1, padding='same', activation='tanh')(u7)

return Model(d0, output_img)
```

DISCRIMINATOR



```
def build_discriminator(self):

    def d_layer(layer_input, filters, f_size=4, bn=True):
        """Discriminator layer"""
        d = Conv2D(filters, kernel_size=f_size, strides=2, padding='same')(layer_input)
        d = LeakyReLU(alpha=0.2)(d)
        if bn:
            d = BatchNormalization(momentum=0.8)(d)
        return d

    img_A = Input(shape=self.img_shape)
    img_B = Input(shape=self.img_shape)

    # Concatenate image and conditioning image by channels to produce input
    combined_imgs = Concatenate(axis=-1)([img_A, img_B])

    d1 = d_layer(combined_imgs, self.df, bn=False)
    d2 = d_layer(d1, self.df*2)
    d3 = d_layer(d2, self.df*4)
    d4 = d_layer(d3, self.df*8)

    validity = Conv2D(1, kernel_size=4, strides=1, padding='same')(d4)
```

Class Pix2Pix

→**Builds the architectures, trains the models, predicts and saves predictions**

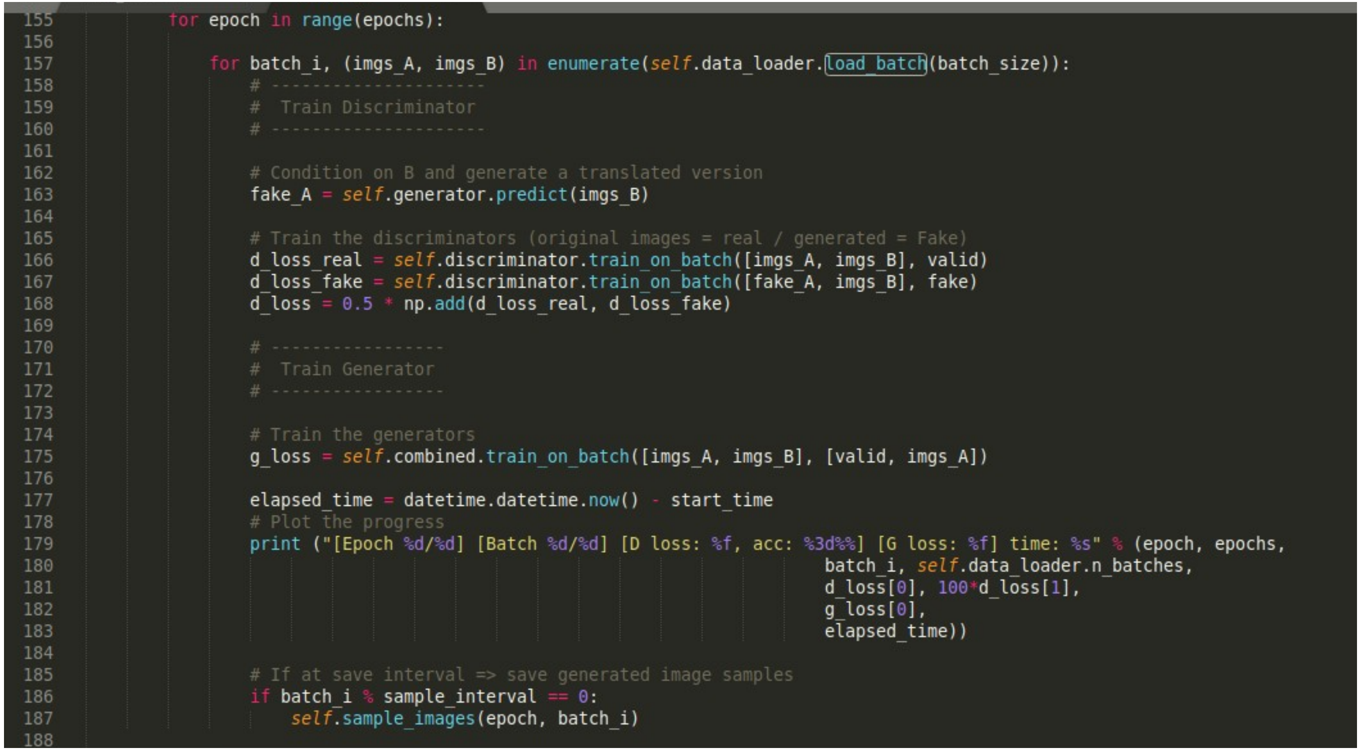
The training :

Train the discriminator and the generator batch to batch using train_on_batch() function.

→**Save the loss / accuracy at each batch and print it to the screen.**

PART 2 : CODE

2.2 Building the Model

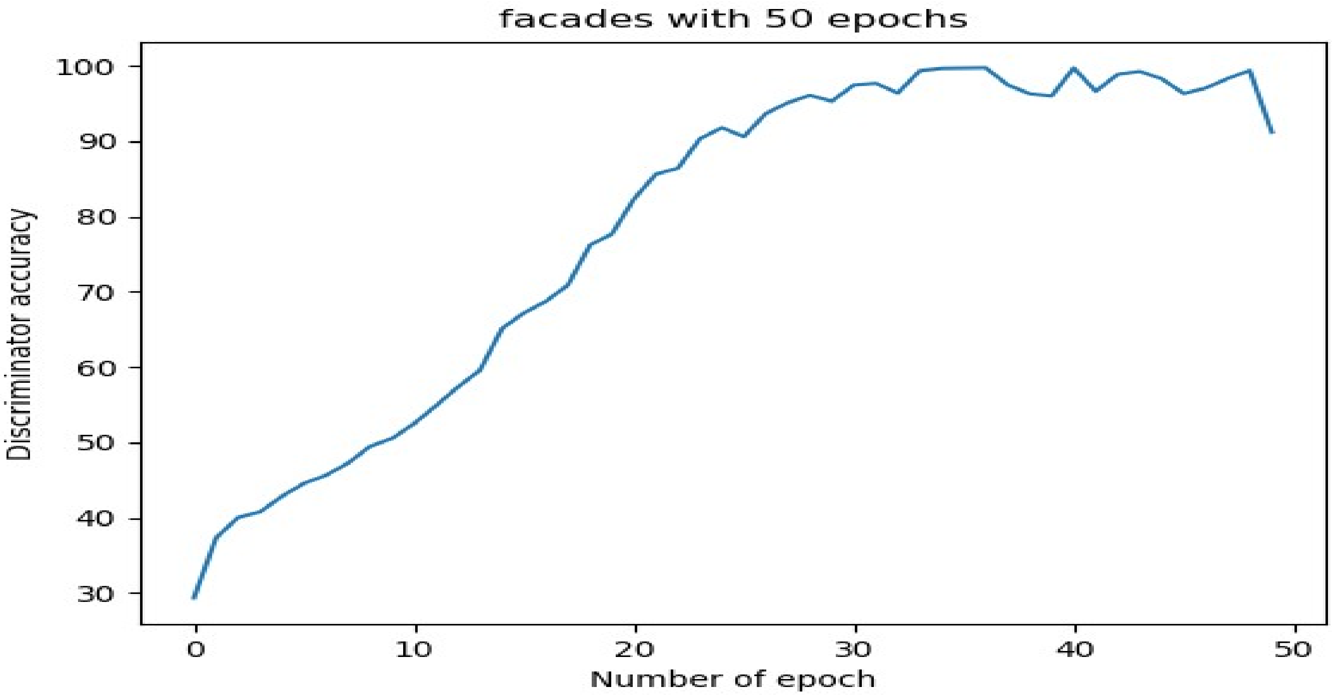


```
        for epoch in range(epochs):

            for batch_i, (imgs_A, imgs_B) in enumerate(self.data_loader.load_batch(batch_size)):
                # ---------------------
                #  Train Discriminator
                # ---------------------

                # Condition on B and generate a translated version
                fake_A = self.generator.predict(imgs_B)

                # Train the discriminators (original images = real / generated = Fake)
                d_loss_real = self.discriminator.train_on_batch([imgs_A, imgs_B], valid)
                d_loss_fake = self.discriminator.train_on_batch([fake_A, imgs_B], fake)
                d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

                # -----------------
                #  Train Generator
                # -----------------

                # Train the generators
                g_loss = self.combined.train_on_batch([imgs_A, imgs_B], [valid, imgs_A])

                elapsed_time = datetime.datetime.now() - start_time
                # Plot the progress
                print ("[Epoch %d/%d] [Batch %d/%d] [D loss: %f, acc: %3d%%] [G loss: %f] time: %s" % (epoch, epochs,
                                                                        batch_i, self.data_loader.n_batches,
                                                                        d_loss[0], 100*d_loss[1],
                                                                        g_loss[0],
                                                                        elapsed_time))

                # If at save interval => save generated image samples
                if batch_i % sample_interval == 0:
                    self.sample_images(epoch, batch_i)

```

PART 3
RESULTS

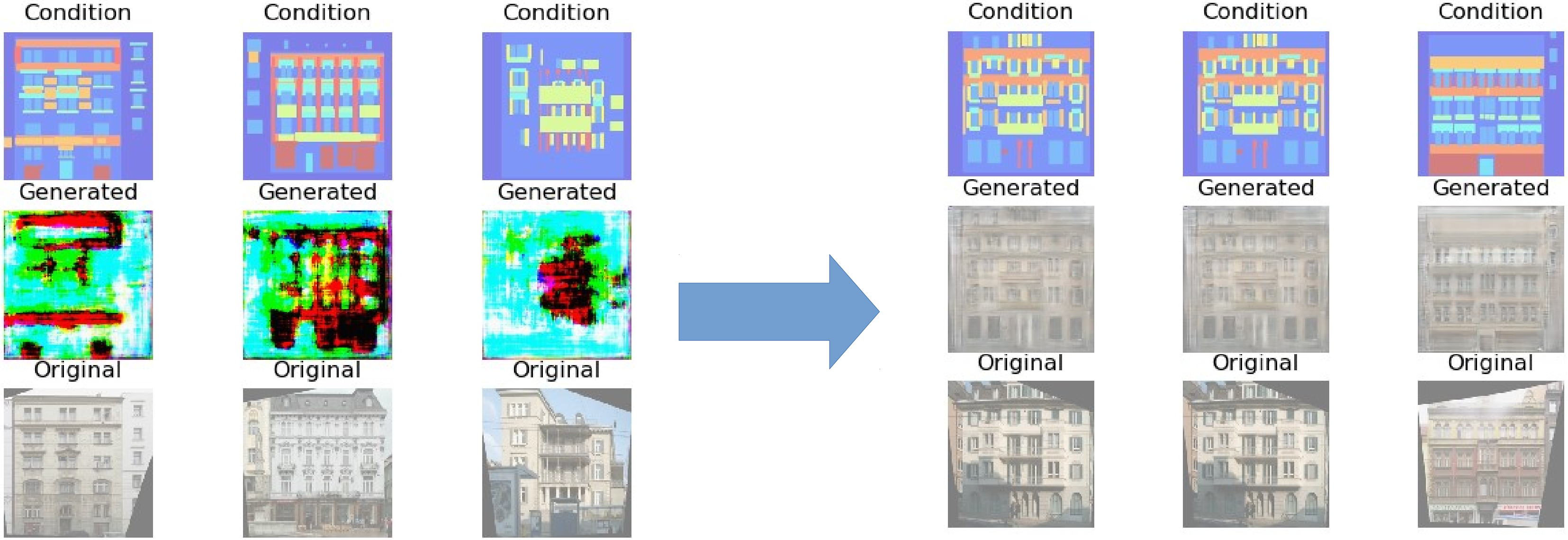
PART 3 : RESULTS

3.1 Accuracy



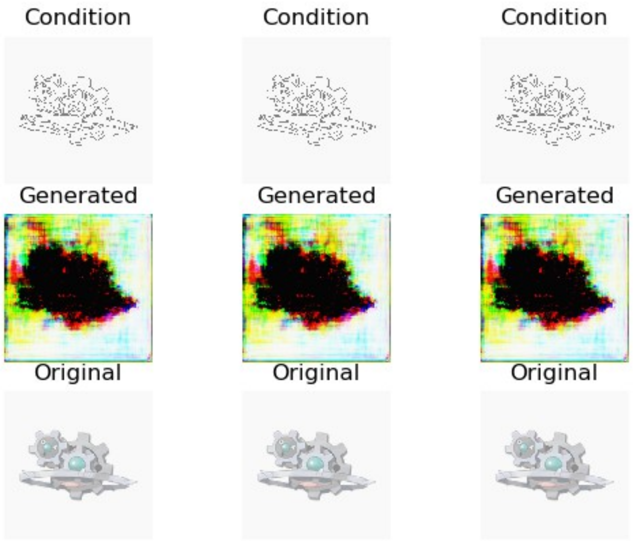
PART 3 : RESULTS

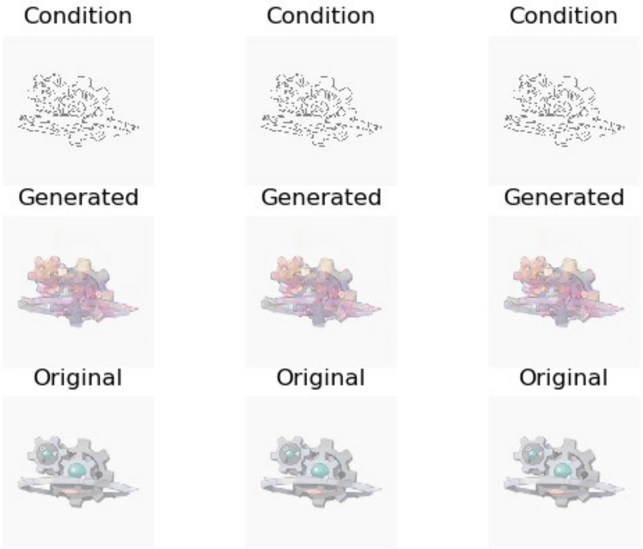
3.2 Facades Visualistation



PART 3 : RESULTS

3.3 Pokemon Visualistation



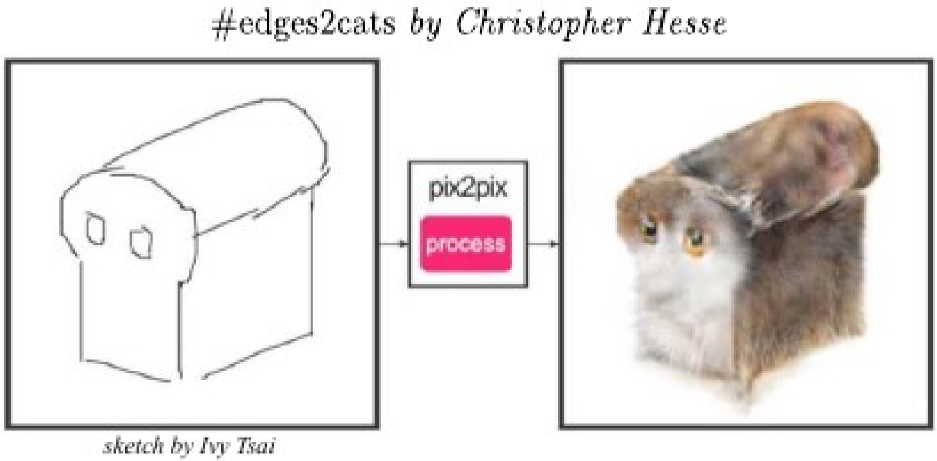


+ video !

PART 3 : RESULTS

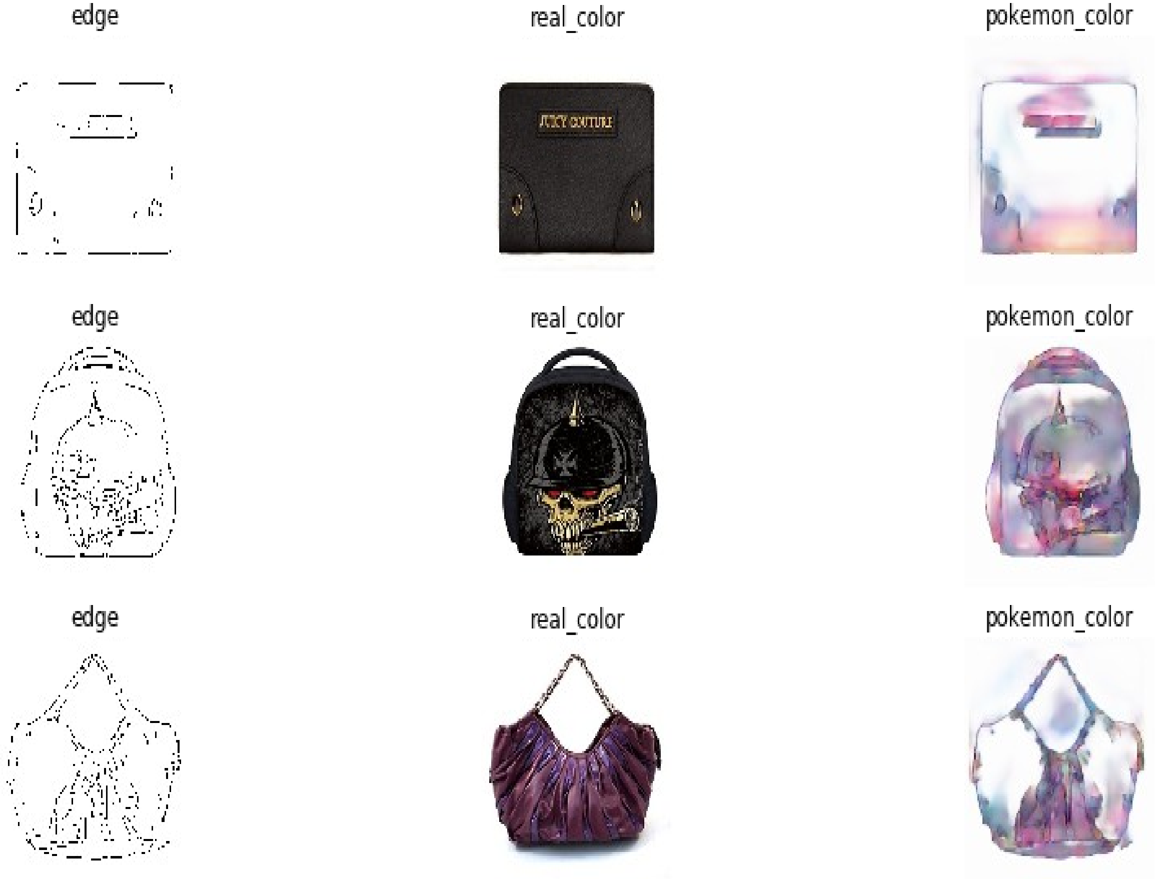
3.4 Can a bag be a pokemon ?

Funny application : Bread2Cat !



PART 3 : RESULTS

3.4 Can a bag be a pokemon ?



CONCLUSION

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

SOURCE

ARTICLE :

IMAGE-TO-IMAGE TRANSLATION WITH CONDITIONAL ADVERSARIAL NETWORKS

PHILLIP ISOLA, JUN-YAN ZHU, TINGHUI ZHOU, ALEXEI A. EFROS

BERKELEY AI RESEARCH (BAIR) LABORATORY, UC BERKELEY

28 NOV 2018

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom