APT: Image Augmentation

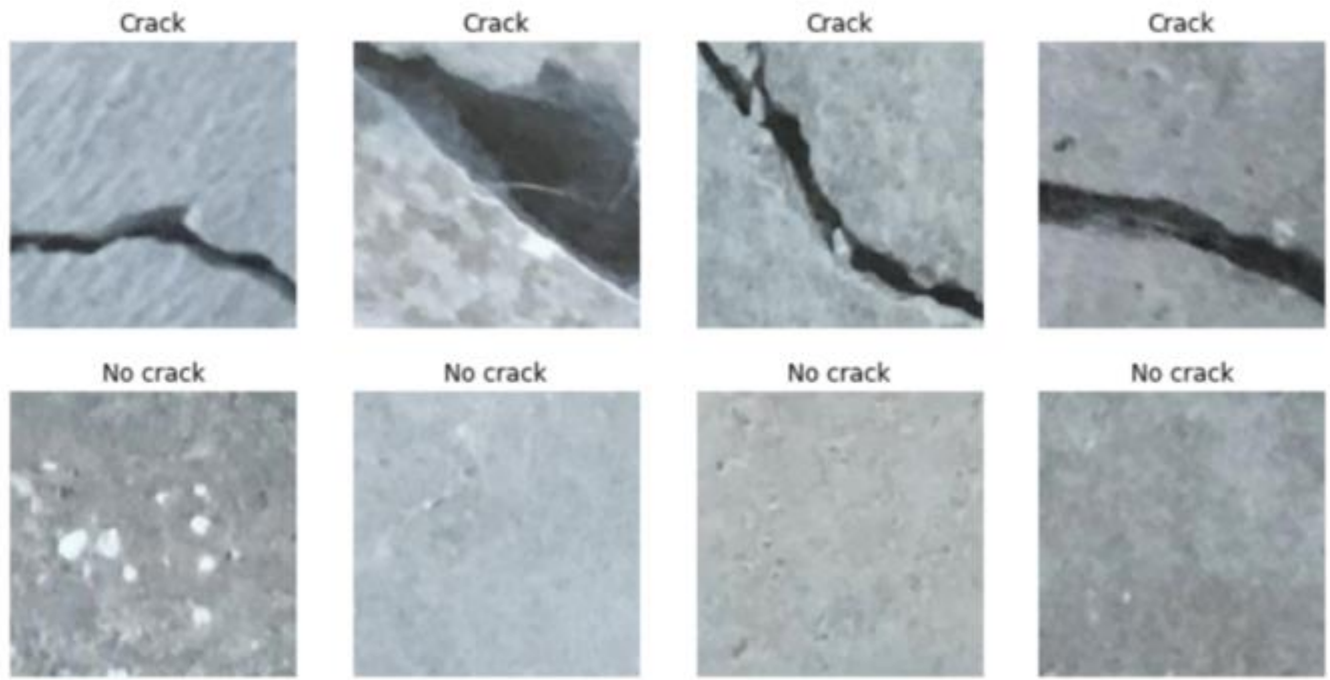
Presenter: Dr. Ha Viet Uyen Synh.

INTRODUCTION

Warm up _ Surface Crack dataset

The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016).

High-resolution images have variance in terms of surface finish and illumination conditions. No data augmentation in terms of random rotation or flipping is applied.



Surface Crack dataset

https://data.mendeley.com/datasets/5y9wdsg2zt/2

Or https://www.kaggle.com/datasets/arunrk7/surface-crack-detection

The dataset is divided into two as negative and positive crack images for image classification. Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels.

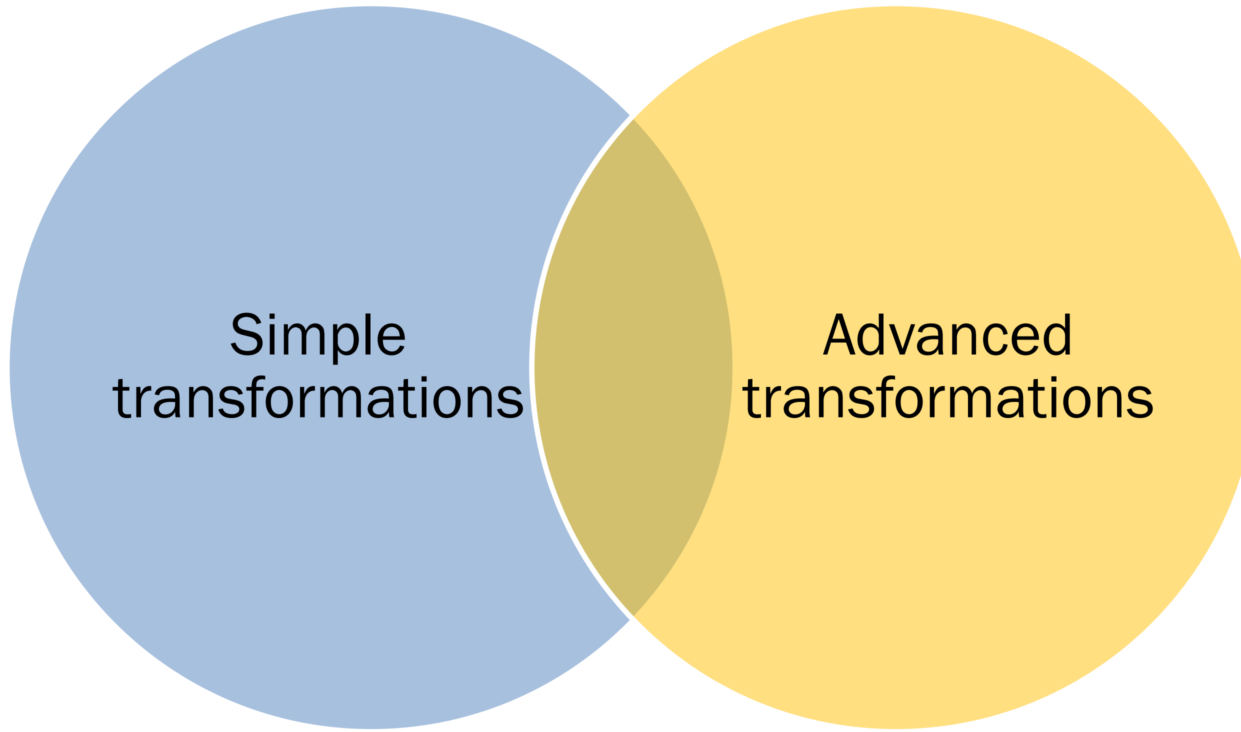
Image Augmentation

A way to increase the amount of data and make the model more robust

The problem addressed is anomaly detection, which is quite challenging since there is a small volume of data and then, the model is not enough to do all the work alone. The common scenario is to train a two-network model with the normal images available for training and evaluate its performance on the test set, that contains both normal and anomalous images.

The initial hypothesis is that the generative model should capture well the normal distribution but at the same time, it should fail on reconstructing the abnormal samples. How is it possible to verify this hypothesis? We can look at the reconstruction error, which should be higher for abnormal images, while it should be low for the normal samples.

Image Augmentation techniques



1. Simple transformations

Simple transformations

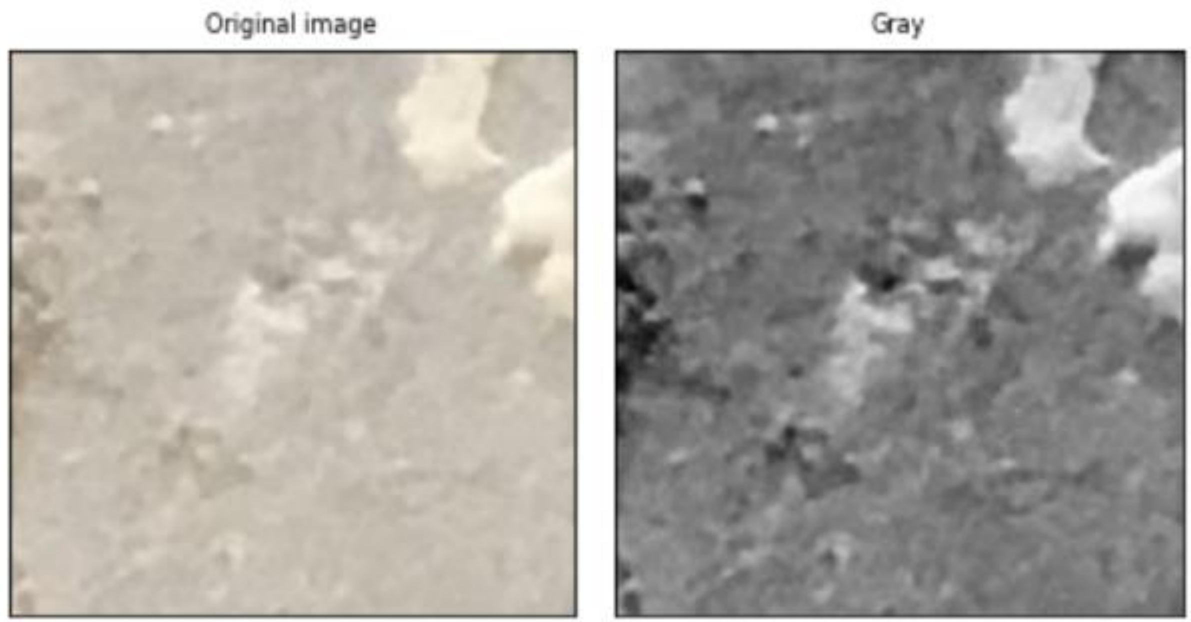
```
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
import sys
import torch
import numpy as np
import torchvision.transforms as T

plt.rcParams["savefig.bbox"] = 'tight'
orig_img = Image.open(Path('../input/surface-crack-detection/Negative/00026.jpg'))
torch.manual_seed(0)
data_path = '../input/surface-crack-detection/'
diz_class = {'Positive':'Crack','Negative':'No crack'}


np.asarray(orig_img).shape  #(227, 227, 3)
```

Resize

```
resized_imgs = [T.Resize(size=size)(orig_img) for size in [32,128]]
plot(resized_imgs,col_title=["32x32","128x128"])
```

Gray Scale

```
gray_img = T.Grayscale()(orig_img)
plot([gray_img], cmap='gray', col_title=["Gray"])
```

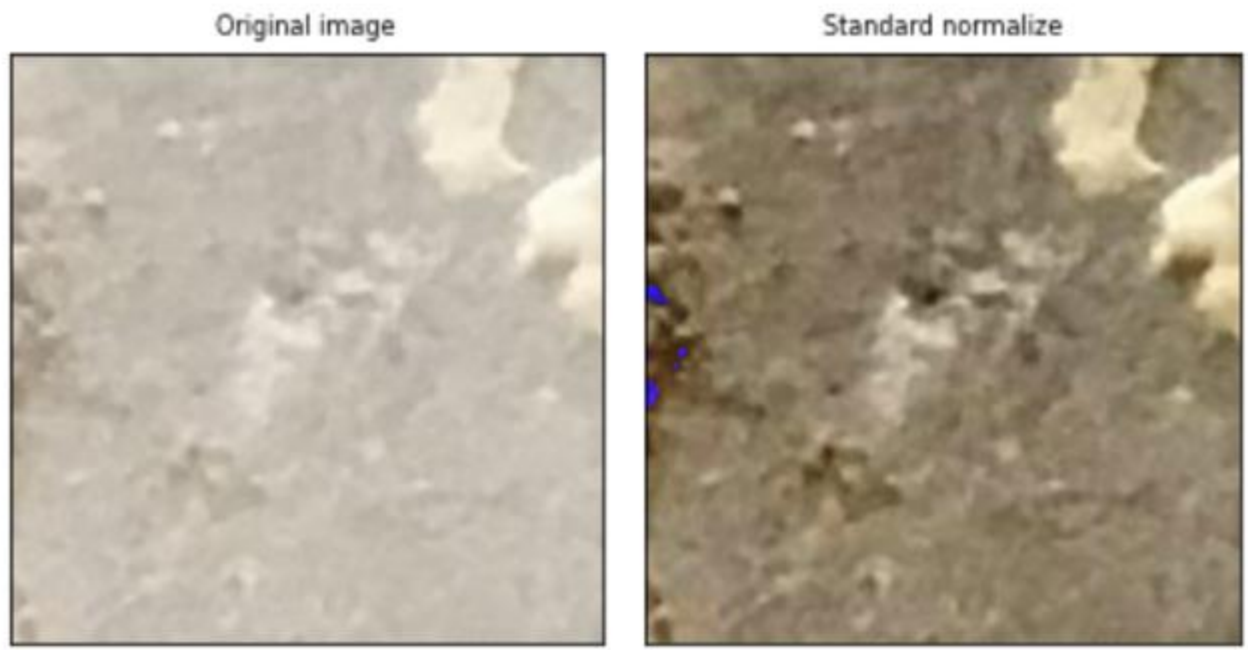


Normalize

The normalization can constitute an effective way to speed up the computations in the model based on neural network architecture and learn faster. There are two steps to normalize the images:

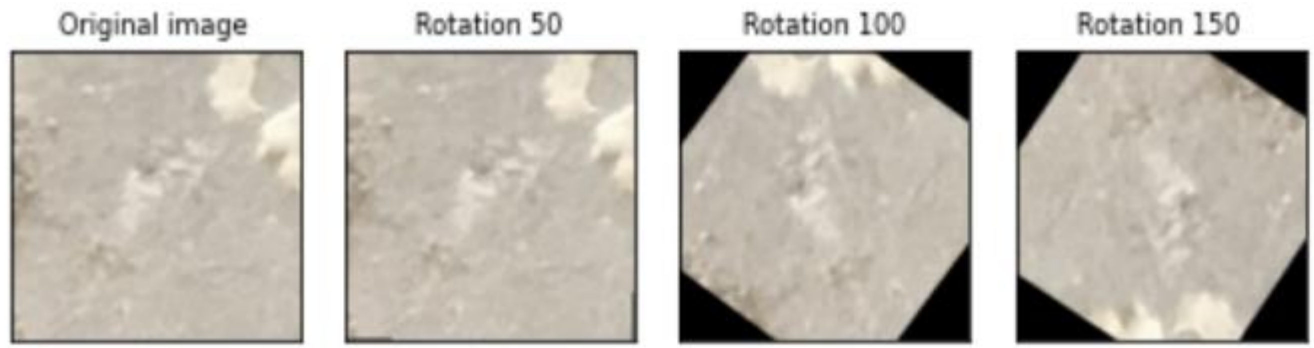
- we subtract the channel mean from each input channel
- we divide it by the channel standard deviation.

```
normalized_img = T.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5,
 0.5))(T.ToTensor()(orig_img))
normalized_img = [T.ToPILImage()(normalized_img)]
plot(normalized_img, col_title=["Standard normalize"])
```



Random Rotation

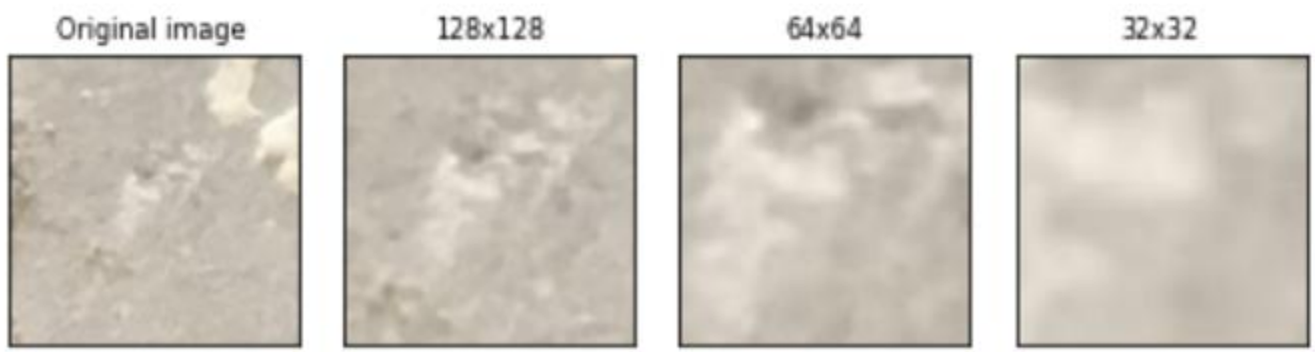
```
rotated_imgs = [T.RandomRotation(degrees=d)(orig_img) for
d in range(50,151,50)]
plot(rotated_imgs, col_title=["Rotation 50","Rotation 100"
,"Rotation 150"])
```



Center Crop

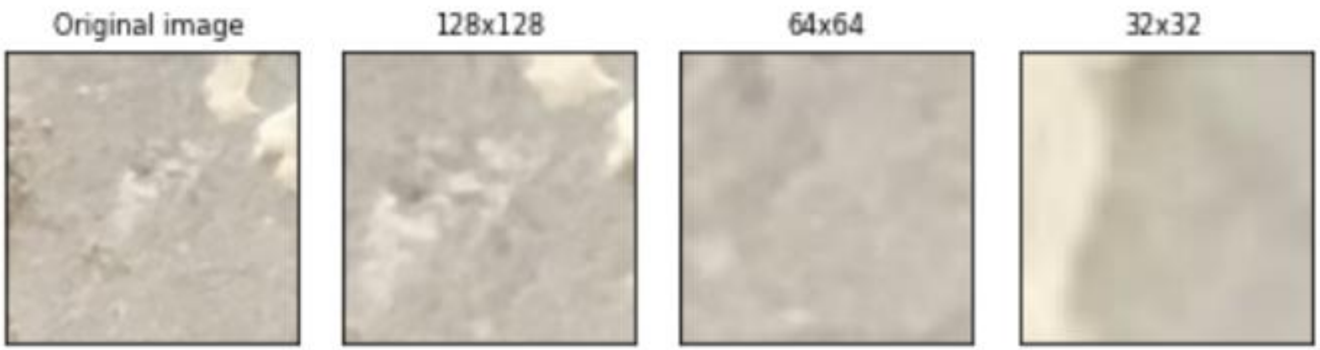
This transformation can be useful when the image has a big background in the borders that isn’t necessary at all for the classification task.

```
center_crops = [T.CenterCrop(size=size)(orig_img) for size in (128,64, 32)]
plot(center_crops,col_title=['128x128','64x64','32x32'])
```



Random drop

```
random_crops = [T.RandomCrop(size=size)(orig_img) for size in (832,704, 256)]
plot(random_crops,col_title=['832x832','704x704','256x256'])
```

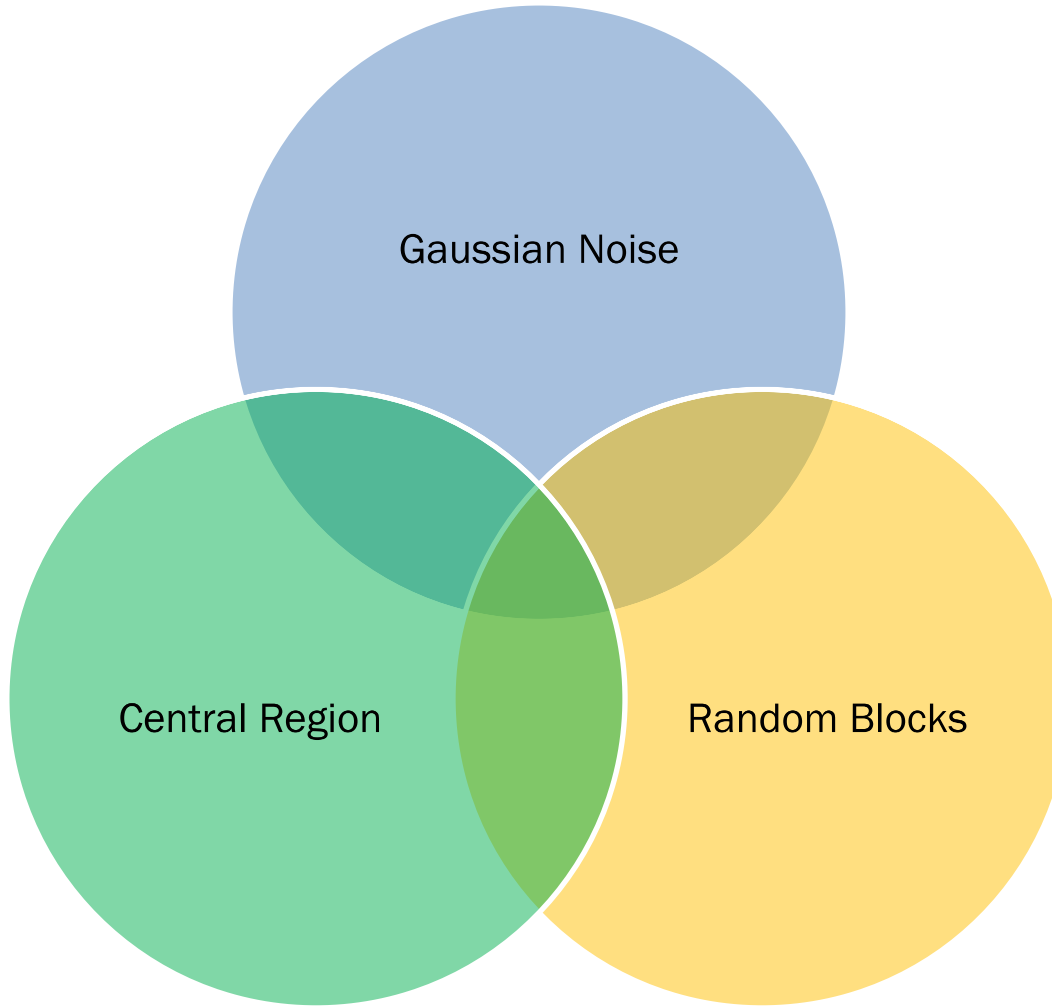


Gaussian Blur

This method can be helpful in making the image less clear and distinct and, then, this resulting image is fed into a neural network, which becomes more robust in learning patterns of the samples

```
blurred_imgs = [T.GaussianBlur(kernel_size=(51, 91), sigma=sigma)
(orig_img) for sigma in (3,7)]
plot(blurred_imgs)
```

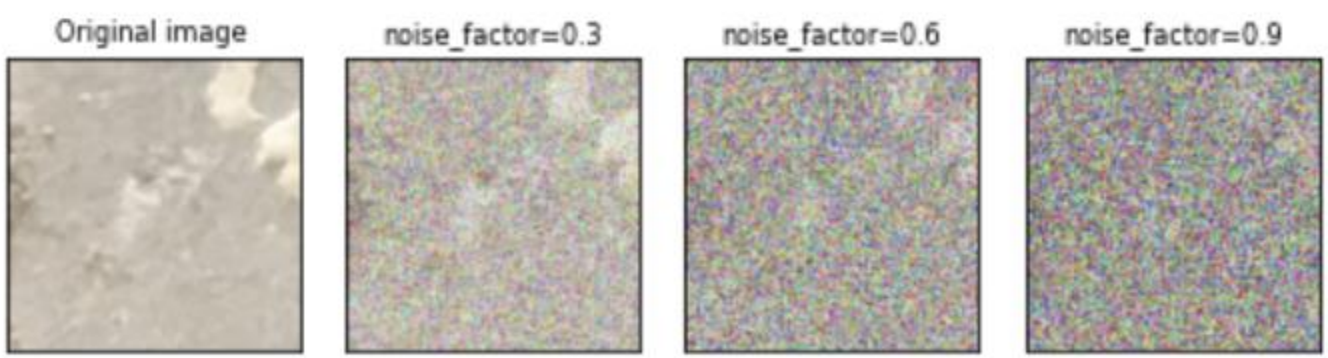
2. Advanced transformations



Gaussian Noise

The Gaussian Noise is a popular way to add noise to the whole dataset, forcing the model to learn the most important information contained in the data.

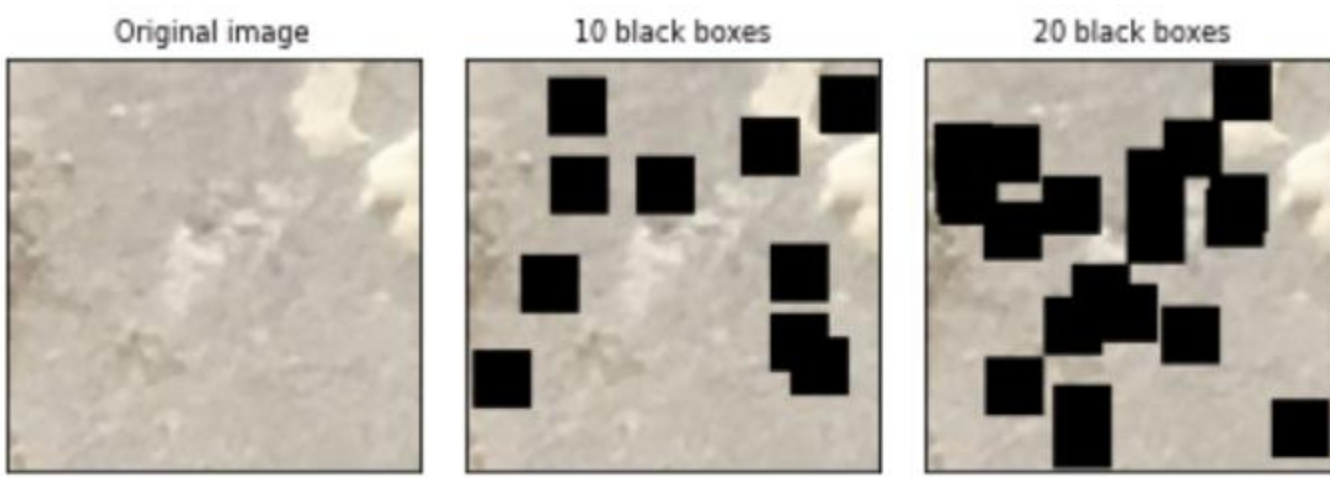
```
def add_noise(inputs,noise_factor=0.3):
     noisy = inputs+torch.randn_like(inputs) * noise_factor
     noisy = torch.clip(noisy,0.,1.)
     return noisy

noise_imgs = [add_noise(T.ToTensor()(orig_img),noise_factor) for noise_factor in (0.3,0.6,0.9)]
noise_imgs = [T.ToPILImage()(noise_img) for noise_img in noise_imgs]
plot(noise_imgs, col_title=["noise_factor=0.3","noise_factor=0.6","noise_factor=0.9"])
```



Random Blocks

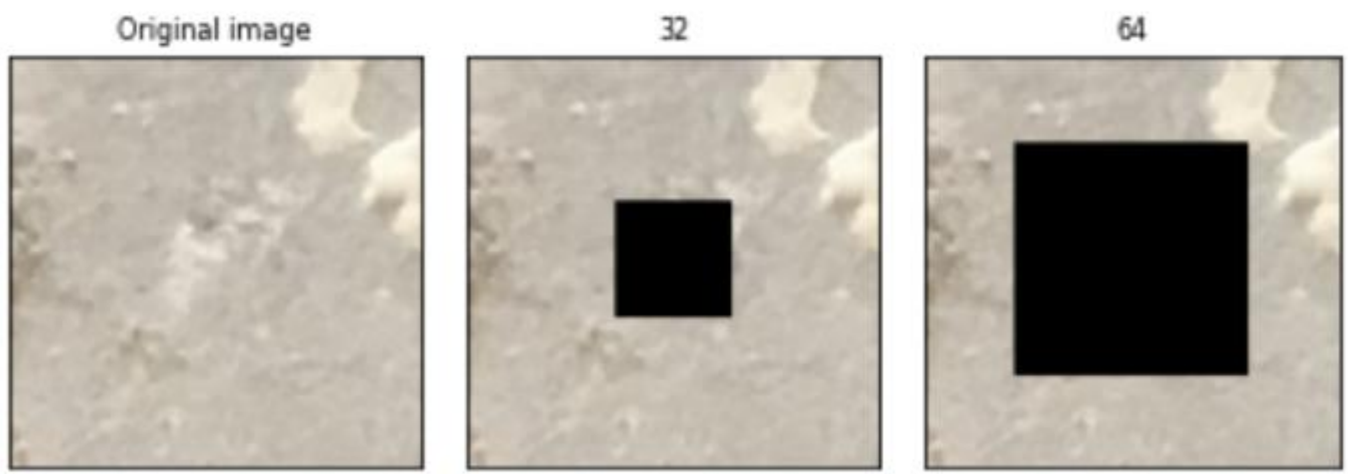
Square patches are applied as masks in the image randomly. The higher the number of these patches, the more the neural network will find challenging the problem to solve.

```
def add_random_boxes(img,n_k,size=32):
    h,w = size,size
    img = np.asarray(img)
    img_size = img.shape[1]
    boxes = []
    for k in range(n_k):
        y,x = np.random.randint(0,img_size-w,(2,))
        img[y:y+h,x:x+w] = 0
        boxes.append((x,y,h,w))
    img = Image.fromarray(img.astype('uint8'), 'RGB')
    return img

blocks_imgs = [add_random_boxes(orig_img,n_k=i) for i in (10,20)]
plot(blocks_imgs,col_title=["10 black boxes","20 black boxes"])
```



Central Region

```
def add_central_region(img,size=32):
    h,w = size,size
    img = np.asarray(img)
    img_size = img.shape[1]
    img[int(img_size/2-h):int(img_size/2+h),int(img_size/2-w):int(img_size/2+w)] = 0
    img = Image.fromarray(img.astype('uint8'), 'RGB')
    return img

central_imgs = [add_central_region(orig_img,size=s) for s in (32,64)]
plot(central_imgs,col_title=["32","64"])
```



ALBUMENTATIONS

Why Albumentations

•Albumentations **supports all common computer vision tasks** such as classification, semantic segmentation, instance segmentation, object detection, and pose estimation.

•The library provides **a simple unified API** to work with all data types: images (RBG-images, grayscale images, multispectral images), segmentation masks, bounding boxes, and key points.

•The library contains **more than 70 different augmentations** to generate new training samples from the existing data.

•Albumentations is **fast**. We benchmark each new release to ensure that augmentations provide maximum speed.

•It **works with popular deep learning frameworks** such as PyTorch and TensorFlow. By the way, Albumentations is a part of the PyTorch ecosystem.

•**Written by experts**. The authors have experience both working on production computer vision systems and participating in competitive machine learning. Many core team members are Kaggle Masters and Grandmasters.

•The library is **widely used** in industry, deep learning research, machine learning competitions, and open source projects.

Example


Original image
RGBShift
HueSaturationValue
ChannelShuffle
CLAHE
RandomContrast
RandomGamma
RandomBrightness
Blur
MedianBlur
ToGray
JpegCompression

Installation

Install the latest stable version from PyPI

pip install -U albumentations

Install the latest version from the master branch on GitHub

pip install -U git+https://github.com/albumentations-team/albumentations

Image Augmentation with Albumentations

```
from PIL import Image
import numpy as np
import albumentations as A
import matplotlib.pyplot as plt

orig_img  = Image.open("img1.jpg")
image = np.array(orig_img )
plt.imshow(image)
plt.axis('off')
plt.show()
```

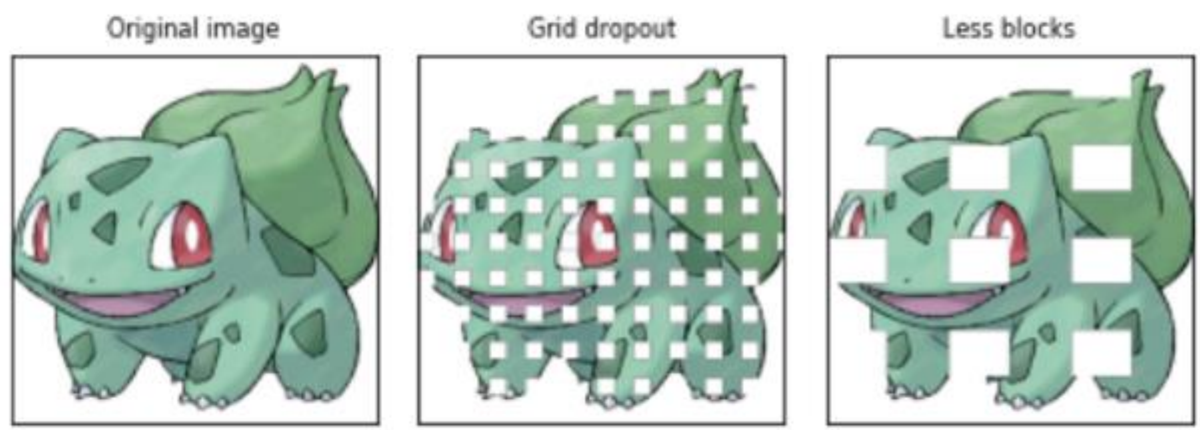
```
(256, 256, 4)
```

Resize and Crop

```
transform_resize = A.Resize(width=64, height=64)

transform_cc = A.Compose([
    A.Resize(width=128, height=128),
    A.CenterCrop(width=32, height=32),
])

transform_rc = A.Compose([
    A.Resize(width=128, height=128),
    A.RandomCrop(width=32, height=32),
])

transformed_res = transform_resize(image=image)
transformed_cc = transform_cc(image=image)
transformed_rc = transform_rc(image=image)
plot([transformed_res['image'],transformed_cc['image'],transformed_rc['image']],col_title=["Resize 64x64","Resize & Center Crop","Resize & Random Crop"])
```

Grid Dropout

```
transform_grid= A.GridDropout(p=1.0)
transformed_grid = transform_grid(image=image)
transform_grid2= A.GridDropout(p=1.0,holes_number_x=3,holes_number_y=4)
transformed_grid2 = transform_grid2(image=image)

plot([transformed_grid['image'],transformed_grid2['image']],col_title=["Grid dropout","Less blocks"])
```



Different types of blur

```
transform_blur = A.Blur(p=1.0)
transform_mblur = A.MedianBlur(p=1.0)
transform_gblur = A.GaussianBlur(sigma_limit=9, p=1.0)

transformed_blur = transform_blur(image=image)
transformed_gblur = transform_gblur(image=image)
transformed_mblur = transform_mblur(image=image)
plot([transformed_blur['image'],transformed_gblur['image'
],transformed_mblur['image']],col_title=["blur","gaussian
 blur","median blur"])
```

Gaussian noise

```
transform = A.GaussNoise(var_limit=350.0, p=1.0)

transformed_gnoise = transform(image=image)
plot([transformed_gnoise['image']],col_title=["Gaussian Noise"])
```

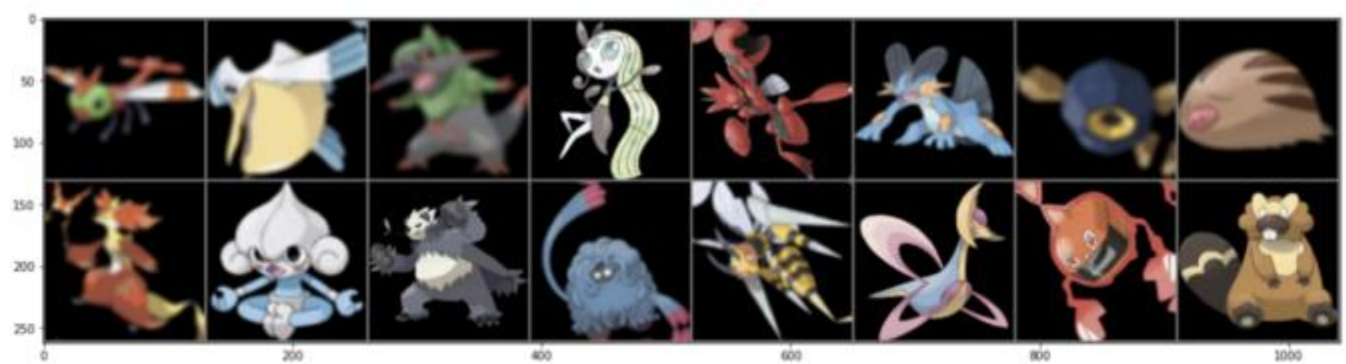
Integration with Pytorch

```
from torchvision import transforms,datasets
from torch.utils.data import Dataset, DataLoader
from albumentations.pytorch import ToTensorV2
import os
import cv2
import random
import torchvision

dataset_directory = "../input/pokemon-images-
dataset/pokemon/pokemon"
pokemon_filepaths = sorted([os.path.join(dataset_directory, f
) for f in os.listdir(dataset_directory)])
correct_images_filepaths = [i for i in pokemon_filepaths if c
v2.imread(i) is not None]

random.seed(42)
random.shuffle(correct_images_filepaths)
n = len(correct_images_filepaths)
n_train = int(n*0.8)
train_images_filepaths = correct_images_filepaths[:n_train]
test_images_filepaths = correct_images_filepaths[n_train:]
print(len(train_images_filepaths), len(test_images_filepaths)
)
```

```python
class PokemonDataset(Dataset):
    def __init__(self, images_filepaths, transform=None):
        self.images_filepaths = images_filepaths
        self.transform = transform

    def __len__(self):
        return len(self.images_filepaths)

    def __getitem__(self, idx):
        image_filepath = self.images_filepaths[idx]
        image = cv2.imread(image_filepath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform is not None:
            image = self.transform(image=image)["image"]
        return image
```

```python
train_transform = A.Compose(    [
        A.Resize(height=128, width=128),
        A.Rotate(),
        A.GaussianBlur(sigma_limit=9, p=0.5),
        A.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),  ToTensorV2(),    ])

test_transform = A.Compose(    [
        A.Resize(height=128, width=128),
        A.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),  ToTensorV2(),    ])

train_dataset = PokemonDataset(images_filepaths=train_images_filepaths, transform=train_transform)
test_dataset = PokemonDataset(images_filepaths=test_images_filepaths, transform=test_transform)
train_loader = DataLoader(dataset=train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=16, shuffle=False)
```

```python
def show_img(img):
    plt.figure(figsize=(20,16))
    img = img * 0.5 + 0.5
    npimg = np.clip(img.numpy(), 0., 1.)
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

data = iter(train_loader)
images = data.next()
show_img(torchvision.utils.make_grid(images))
```



Part A

STYLE TRANSFERRING

1. What’s Style Transferring

Style Transferring will use neural representation to separate, recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images.

Neural style transfer is a way to generate images in the style of another image. The neural-style algorithm takes a content-image (a style image) as input and returns the content image as if it printed using the artistic style of the style image.

VGG-19 model

The VGG model was introduced by Simonyan and Zisserman. VGG-19 is trained on more than a million images from the ImageNet database.

This model has 19 layers deep neural network, which can classify images into 1000 objects categories.

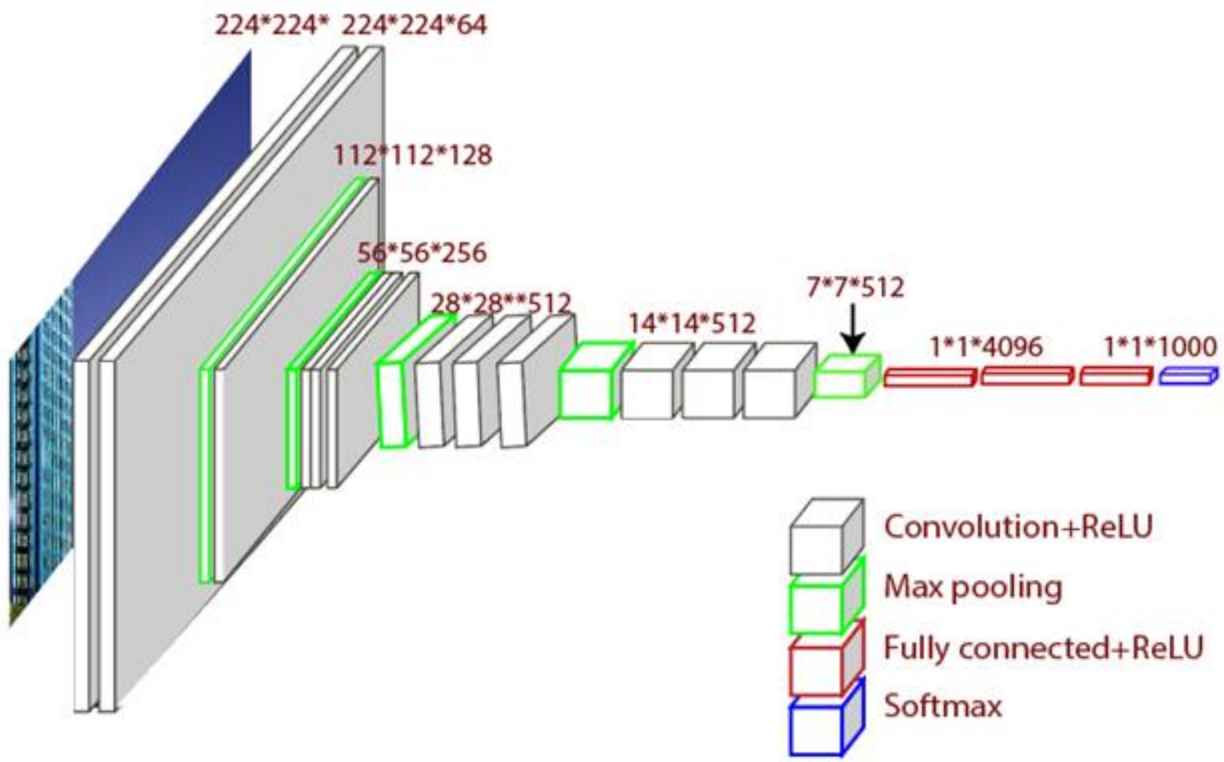
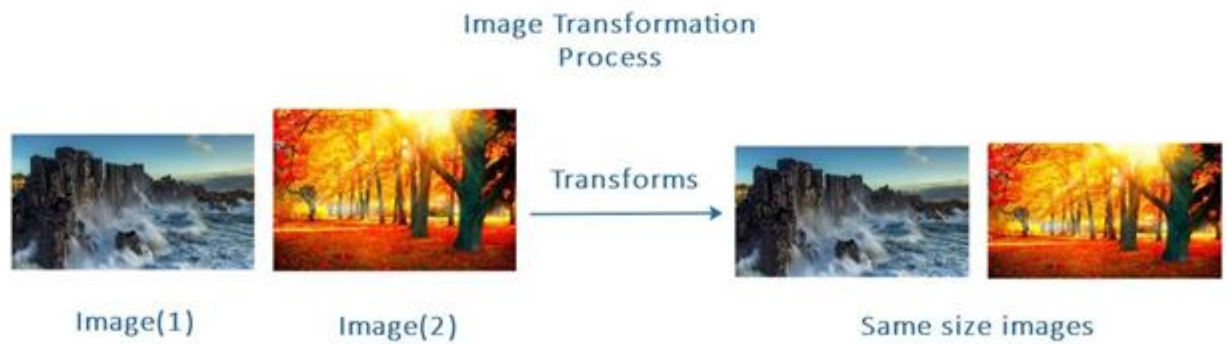


Image loading and transformation

We have a content image, and style image and the target image will be the combination of both these images. Not every image needs to have the same size or pixel. To make the images equal, we will also apply the image transformation process.



Feature Extraction

Gram Matrix

Optimization process

Any Questions?