BTS | Barcelona Technology School

Advanced Data Analysis

Advanced Imagery Analytics- II
Master in Big Data Solutions 2020-2021

Ankit Tewari
Course Instructor, Advanced Data Analysis (2020-21)
ankit.tewari@bts.tech

Contents

Today's topics at a glance

Image Similarity

Discarding images with poor quality

Pretrained models are used in the following two popular ways when building new models or reusing them: Using a pretrained model as a feature extractor Fine-tuning the pretrained model

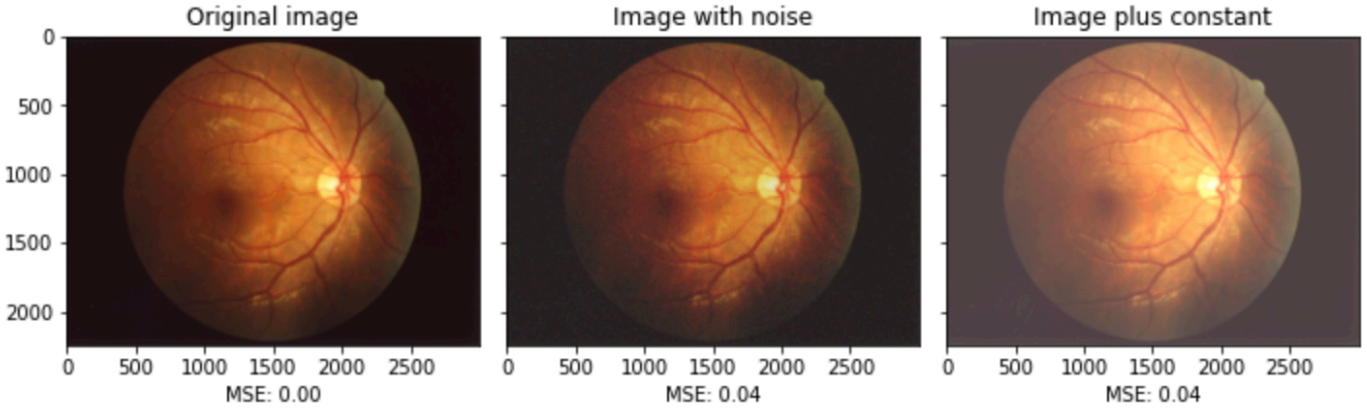


Image Similarity

Discarding images with poor quality

When comparing images, the mean squared error (MSE)- while simple to implement–is not highly indicative of perceived similarity. Structural similarity aims to address this shortcoming by taking texture into account

During this experimentation, we have computed the “Mean Squared Error (MSE)” of our input fundus image with the following-

1. The same input fundus image itself
2. The same input fundus image with an additional component of noise
3. The same input fundus image with a constant

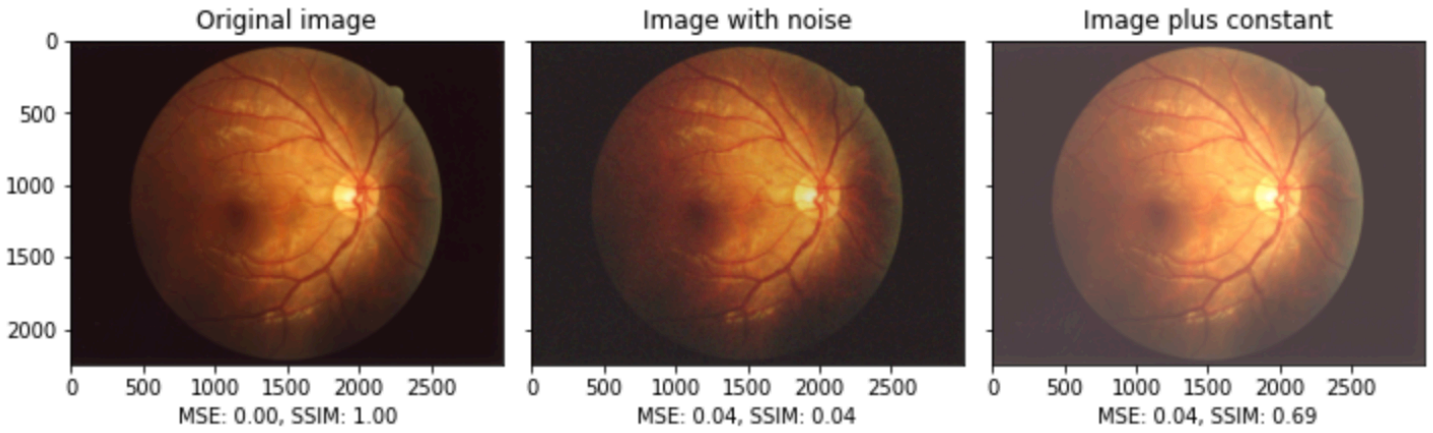


Image Similarity

Discarding images with poor quality

Pretrained models are used in the following two popular ways when building new models or reusing them: Using a pretrained model as a feature extractor Fine-tuning the pretrained model

```
img = img_as_float(imread("../Images/fundus-image.jpeg"))

noise = np.ones_like(img) * 0.2 * (img.max() - img.min())
rng = np.random.default_rng()
noise[rng.random(size=noise.shape) > 0.5] *= -1

img_noise = img + noise
img_const = img + abs(noise)

mse_none = mean_squared_error(img, img)
ssim_none = ssim(img, img, data_range=img.max() - img.min(), multichannel= True)

mse_noise = mean_squared_error(img, img_noise)
ssim_noise = ssim(img, img_noise,
                  data_range=img_noise.max() - img_noise.min(), multichannel=True)

mse_const = mean_squared_error(img, img_const)
ssim_const = ssim(img, img_const,
                  data_range=img_const.max() - img_const.min(), multichannel=True)
```

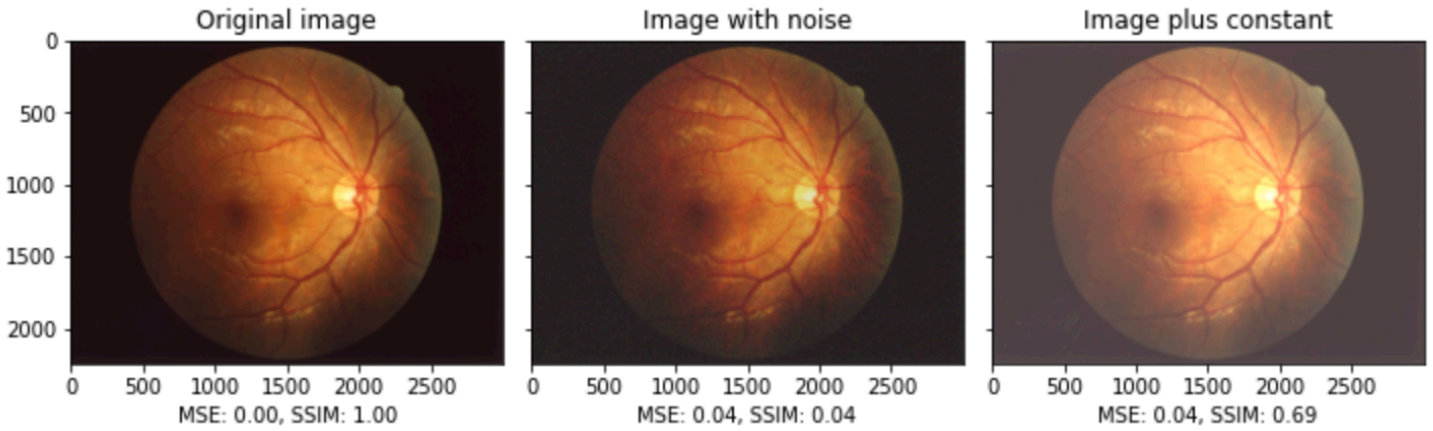


Image Similarity

Discarding images with poor quality

Pretrained models are used in the following two popular ways when building new models or reusing them: Using a pretrained model as a feature extractor Fine-tuning the pretrained model

```
from skimage.transform import resize

img_noise= img_as_float(imread("../Images/normal-eye.jpeg"))
img = img_as_float(imread("../Images/fundus-image.jpeg"))
size= (img_noise.shape[0], img_noise.shape[1])

img = resize(img, size)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4),
                         sharex=True, sharey=True)
ax = axes.ravel()

mse_none = mean_squared_error(img, img)
ssim_none = ssim(img, img, data_range=img.max() - img.min(), multichannel= True)

mse_noise = mean_squared_error(img, img_noise)
ssim_noise = ssim(img, img_noise,
                  data_range=img_noise.max() - img_noise.min(), multichannel=True)
```

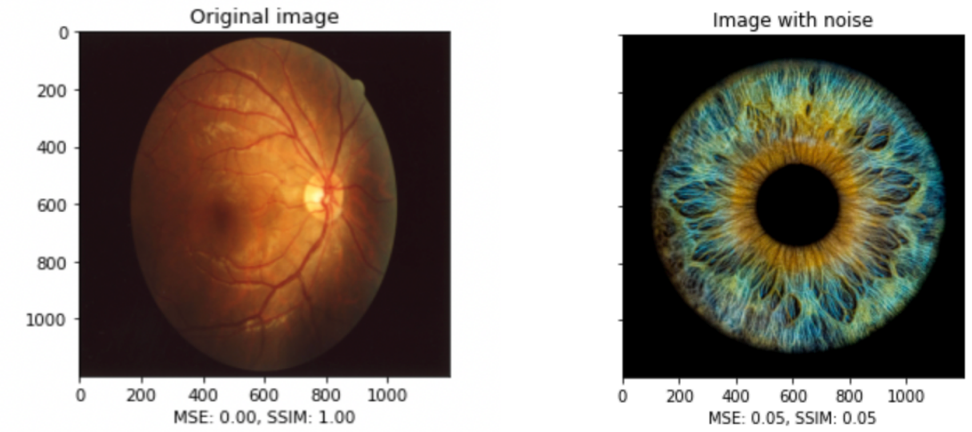


Image Restoration

Deconvoluting Images for Image Sharpening

Mathematically, Deconvolution is an operation that is reverse of Convolution. For example, convolution can be used to apply a filter, and it may be possible to recover the original signal using deconvolution.

Deconvolution is a computationally intensive image processing technique that is being increasingly utilized for improving the contrast and resolution of digital images captured in the microscope. The foundations are based upon a suite of methods that are designed to remove or reverse the blurring present in microscope images induced by the limited aperture of the objective.

In order to develop a better understanding, consider the following equation-

$$f * g = h$$

Here, *h* is some recorded signal, and *f* is some signal that we wish to recover, but has been convolved with a filter or distortion function *g*, before we recorded it.

If we know *g*, or at least know the form of *g*, then we can perform deterministic deconvolution. However, if we do not know *g* in advance, then we need to estimate it using statistical estimation methods.

Image Restoration

Deconvoluting Images for Image Sharpening

Mathematically, Deconvolution is an operation that is reverse of Convolution. For example, convolution can be used to apply a filter, and it may be possible to recover the original signal using deconvolution.

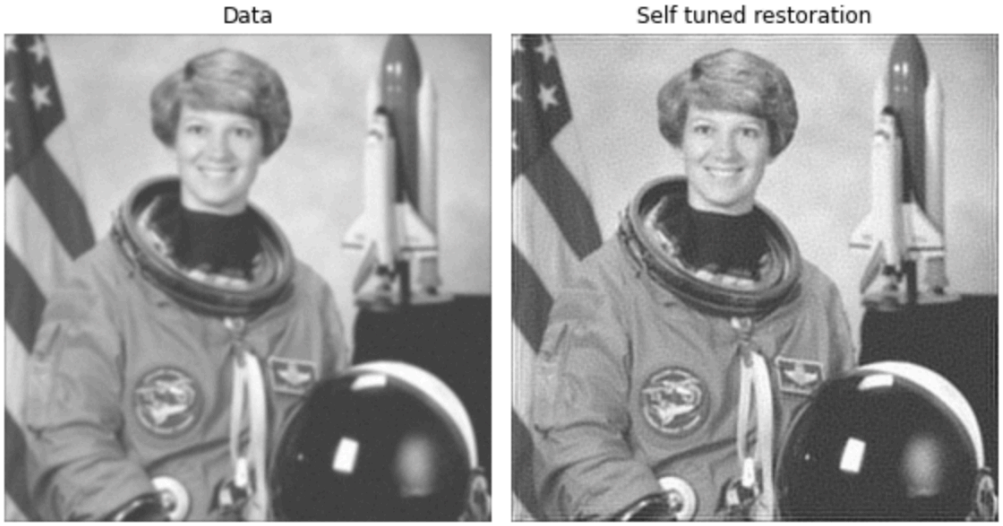
```
from skimage import color, data, restoration

rng = np.random.default_rng()

astro = color.rgb2gray(data.astronaut())
from scipy.signal import convolve2d as conv2
psf = np.ones((5, 5)) / 25
astro = conv2(astro, psf, 'same')
astro += 0.1 * astro.std() * rng.standard_normal(astro.shape)

deconvolved, _ = restoration.unsupervised_wiener(astro, psf)
```

Image Restoration

Deconvoluting Images for Image Sharpening

Mathematically, Deconvolution is an operation that is reverse of Convolution. For example, convolution can be used to apply a filter, and it may be possible to recover the original signal using deconvolution.

```
from skimage import color, data, restoration

rng = np.random.default_rng()

astro = color.rgb2gray(data.astronaut())
from scipy.signal import convolve2d as conv2
psf = np.ones((5, 5)) / 25
astro = conv2(astro, psf, 'same')
astro += 0.1 * astro.std() * rng.standard_normal(astro.shape)

deconvolved, _ = restoration.unsupervised_wiener(astro, psf)
```



Object Detection

Detecting Faces

Object detection is the process of locating some specific feature(s) of interest within an image. For example, detecting the position of left and right eyes as specific features in an image of human faces. Some other applications of object detection are-

1. Security systems: facial unlock systems (by means of detecting and comparing faces) allowing only the owners to unlock the access;
2. Autonomous driving: identifying pedestrians or even obstacles;

In the context of facial detection, one of the first methods used were Haar Classifiers. Haar Classifiers generally consist of the following four stages–

1. Calculating Haar Features
2. Creating Integral Images
3. Using Adaboost
4. Implementing Cascading Classifiers

Read more about the Haar Classifiers here

Object Detection

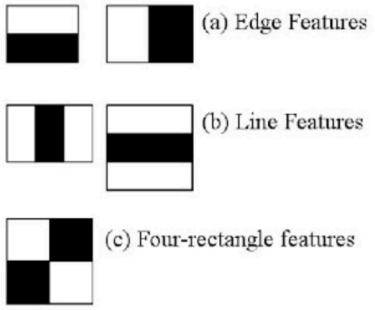
Detecting Faces

Object detection is the process of locating some specific feature(s) of interest within an image. For example, detecting the position of left and right eyes as specific features in an image of human faces. Some other applications of object detection are-

1. Security systems: facial unlock systems (by means of detecting and comparing faces) allowing only the owners to unlock the access;
2. Autonomous driving: identifying pedestrians or even obstacles;

In the context of facial detection, one of the first methods used were Haar Classifiers. Haar Classifiers generally consist of the following four stages–

1. Calculating Haar Features: The first step is to collect the Haar features. A **Haar feature** is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. Here are some examples of Haar features below.



Object Detection

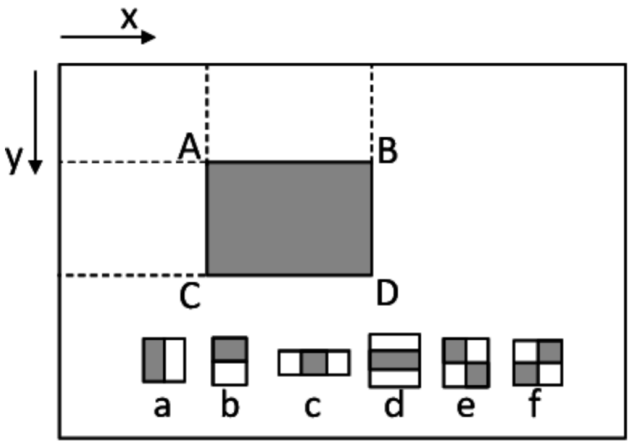
Detecting Faces

Object detection is the process of locating some specific feature(s) of interest within an image. For example, detecting the position of left and right eyes as specific features in an image of human faces. Some other applications of object detection are-

1. Security systems: facial unlock systems (by means of detecting and comparing faces) allowing only the owners to unlock the access;
2. Autonomous driving: identifying pedestrians or even obstacles;

In the context of facial detection, one of the first methods used were Haar Classifiers. Haar Classifiers generally consist of the following four stages–

2. Creating Integral Images: Without going into too much of the mathematics behind it (check out the paper if you're interested in that), integral images essentially speed up the calculation of these Haar features. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.



Object Detection

Detecting Faces

Object detection is the process of locating some specific feature(s) of interest within an image. For example, detecting the position of left and right eyes as specific features in an image of human faces. Some other applications of object detection are-

1. Security systems: facial unlock systems (by means of detecting and comparing faces) allowing only the owners to unlock the access;
2. Autonomous driving: identifying pedestrians or even obstacles;

In the context of facial detection, one of the first methods used were Haar Classifiers. Haar Classifiers generally consist of the following four stages–

3. Using Adaboost: It's important to note that nearly all of the Haar features will be irrelevant when doing object detection, because the only features that are important are those of the object. However, how do we determine the best features that represent an object from the hundreds of thousands of Haar features? This is where Adaboost comes into play. Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of "weak classifiers" to create a "strong classifier" that the algorithm can use to detect objects.

Object Detection

Detecting Faces

Object detection is the process of locating some specific feature(s) of interest within an image. For example, detecting the position of left and right eyes as specific features in an image of human faces. Some other applications of object detection are-

1. Security systems: facial unlock systems (by means of detecting and comparing faces) allowing only the owners to unlock the access;
2. Autonomous driving: identifying pedestrians or even obstacles;

In the context of facial detection, one of the first methods used were Haar Classifiers. Haar Classifiers generally consist of the following four stages–

4. Implementing Cascading Classifiers: The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners.

Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative). Stages are designed to reject negative samples as fast as possible, because a majority of the windows do not contain anything of interest.

It's important to maximize a low false negative rate, because classifying an object as a non-object will severely impair your object detection algorithm.

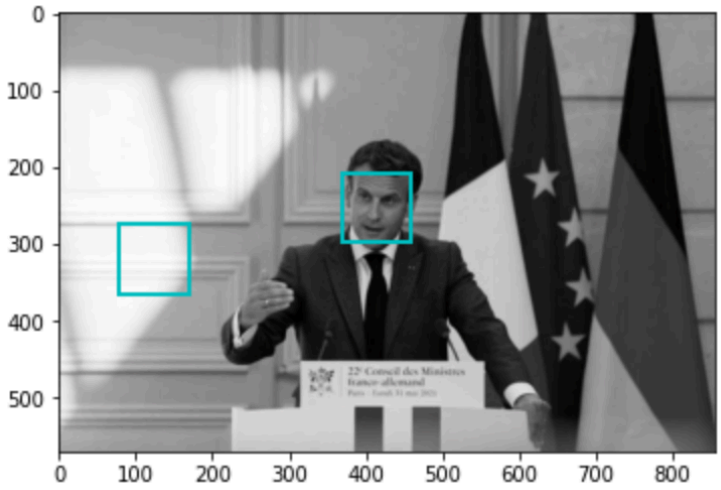
Read more about how Cascade Classifiers are different from CNN here

Object Detection

Detecting Faces

The main idea behind cascade of classifiers is to create classifiers of medium accuracy and ensemble them into one strong classifier instead of just creating a strong one.

```
from skimage import data
from skimage.feature import Cascade
import cv2 as cv
import matplotlib.pyplot as plt
from matplotlib import patches

trained_file = data.lbp_frontal_face_cascade_filename()
detector = Cascade(trained_file)
img = cv.imread("../Images/emanuel-macron.jpeg", 2)

detected = detector.detect_multi_scale(img=img,
                                       scale_factor=1.2,
                                       step_ratio=1,
                                       min_size=(60, 60),
                                       max_size=(123, 123))
```

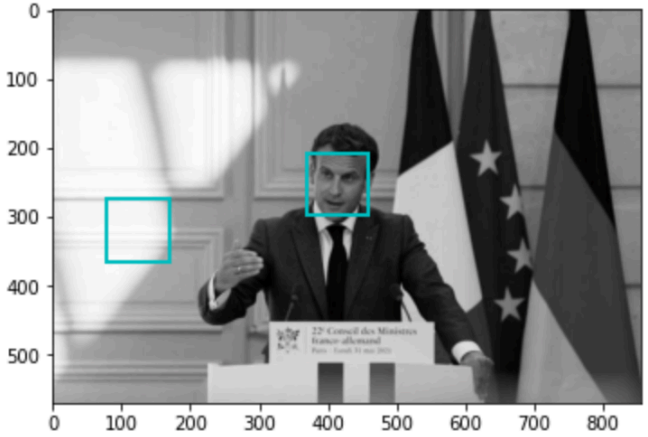


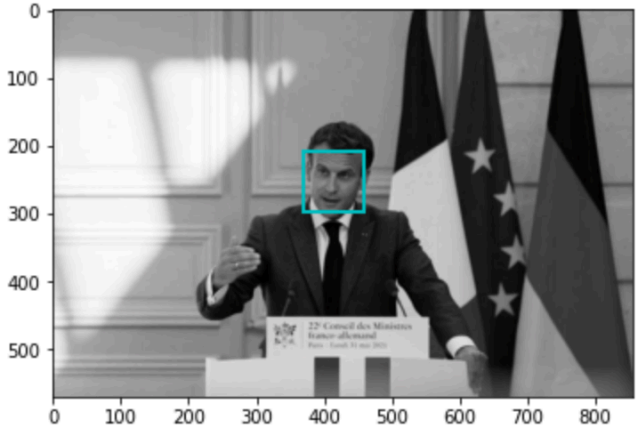
Object Detection

Detecting Faces

The main idea behind cascade of classifiers is to create classifiers of medium accuracy and ensemble them into one strong classifier instead of just creating a strong one.

```
from skimage import data
from skimage.feature import Cascade
import cv2 as cv
import matplotlib.pyplot as plt
from matplotlib import patches

trained_file = data.lbp_frontal_face_cascade_filename()
detector = Cascade(trained_file)
img = cv.imread("../Images/emanuel-macron.jpeg", 2)
detected = detector.detect_multi_scale(img=img,
                                       scale_factor=1.2,
                                       step_ratio=1,
                                       min_size=(60, 60),
                                       max_size=(123, 123), min_neighbour_number= 6)
```





Exposure Management

Improving Contrast

Just like histogram equalisation which we had studied previous, we may use the following to play with contrast-
Logarithmic Correction: It transforms the input image pixelwise according to the equation O = gain x log(1 + I) after scaling each pixel to the range 0 to 1.
Gamma Correction: It is also known as Power Law Transform. This function transforms the input image pixelwise according to the equation $O = I^{gamma}$ after scaling each pixel to the range 0 to 1.

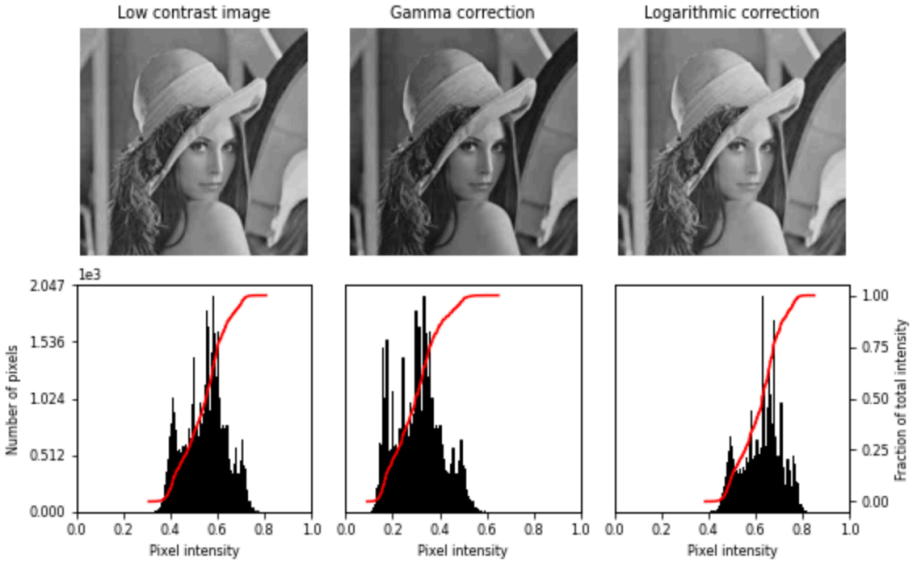
```
from skimage import data, img_as_float
from skimage import exposure

img = imread("../Images/low-contrast-image.jpeg")
gamma_corrected = exposure.adjust_gamma(img, 2)
logarithmic_corrected = exposure.adjust_log(img, 1)
```

Exposure Management

Improving Contrast

Just like histogram equalisation which we had studied previous, we may use the following to play with contrast-
Logarithmic Correction: It transforms the input image pixelwise according to the equation O = gain x log(1 + I) after scaling each pixel to the range 0 to 1.
Gamma Correction: It is also known as Power Law Transform. This function transforms the input image pixelwise according to the equation $O = I^{gamma}$ after scaling each pixel to the range 0 to 1.

```
from skimage import data, img_as_float
from skimage import exposure

img = imread("../Images/low-contrast-image.jpeg")
gamma_corrected = exposure.adjust_gamma(img, 2)
logarithmic_corrected = exposure.adjust_log(img, 1)
```



Exposure Management

Histogram Matching

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.

Histogram matching can be used as a lightweight normalisation for image processing, such as feature matching, especially in circumstances where the images have been taken from different sources or in different conditions (i.e. lighting).

Exposure Management

Histogram Matching

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.

Histogram matching can be used as a lightweight normalisation for image processing, such as feature matching, especially in circumstances where the images have been taken from different sources or in different conditions (i.e. lighting).

```
from skimage import data
from skimage import exposure
from skimage.exposure import match_histograms

reference = data.coffee()
image = data.chelsea()

matched = match_histograms(image, reference)

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3),
                                    sharex=True, sharey=True)
```

Exposure Management

Histogram Matching

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.

Histogram matching can be used as a lightweight normalisation for image processing, such as feature matching, especially in circumstances where the images have been taken from different sources or in different conditions (i.e. lighting).

```
from skimage import data
from skimage import exposure
from skimage.exposure import match_histograms

reference = data.coffee()
image = data.chelsea()

matched = match_histograms(image, reference)

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3),
                                    sharex=True, sharey=True)
```

Exposure Management

Histogram Matching

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.

In order to illustrate the effect of the histogram matching, we plot for each RGB channel, the histogram and the cumulative histogram. Clearly, the matched image has the same cumulative histogram as the reference image for each channel.

```
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(8, 8))


for i, img in enumerate((image, reference, matched)):
    for c, c_color in enumerate(('red', 'green', 'blue')):
        img_hist, bins = exposure.histogram(img[..., c], source_range='dtype')
        axes[c, i].plot(bins, img_hist / img_hist.max())
        img_cdf, bins = exposure.cumulative_distribution(img[..., c])
        axes[c, i].plot(bins, img_cdf)
        axes[c, 0].set_ylabel(c_color)

axes[0, 0].set_title('Source')
axes[0, 1].set_title('Reference')
axes[0, 2].set_title('Matched')

plt.tight_layout()
plt.show()
```

Exposure Management

Histogram Matching

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.

In order to illustrate the effect of the histogram matching, we plot for each RGB channel, the histogram and the cumulative histogram. Clearly, the matched image has the same cumulative histogram as the reference image for each channel.

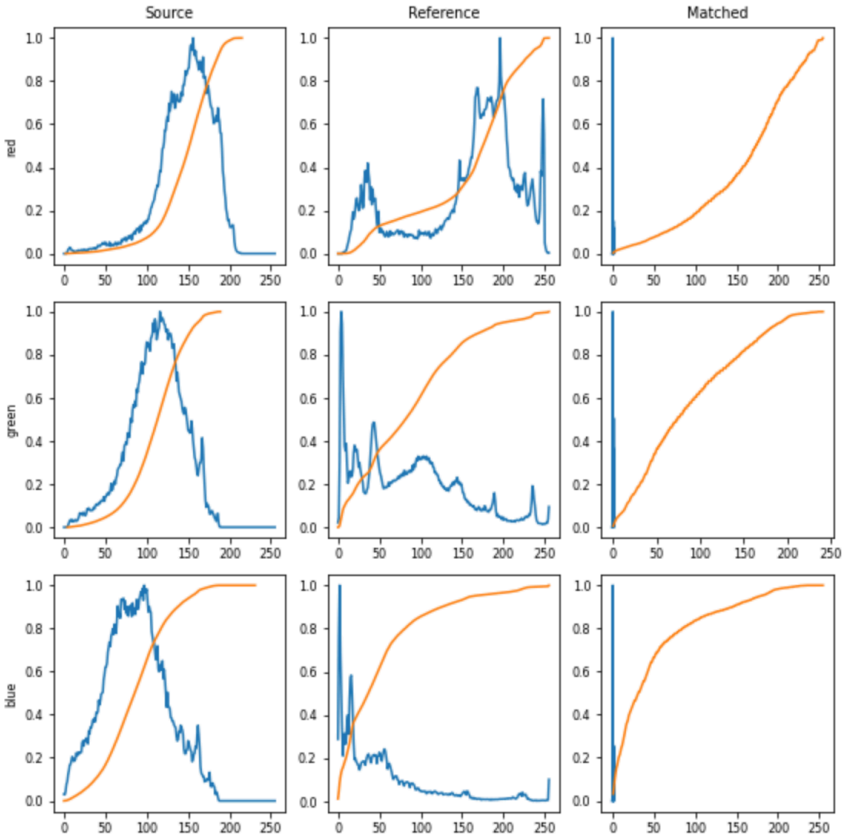


Image Recovery

Recovering Rotational Differences: Polar Transforms

Phase correlation is an efficient method for determining translation offset between pairs of similar images. However this approach relies on a near absence of rotation/scaling differences between the images, which are typical in real-world examples.

Image Recovery

Recovering Rotational Differences: Polar Transforms

Phase correlation is an efficient method for determining translation offset between pairs of similar images. However this approach relies on a near absence of rotation/scaling differences between the images, which are typical in real-world examples.

In the example below, we consider the simple case of two images that only differ with respect to rotation around a common centre point. By remapping these images into polar space, the rotation difference becomes a simple translation difference that can be recovered by phase correlation.

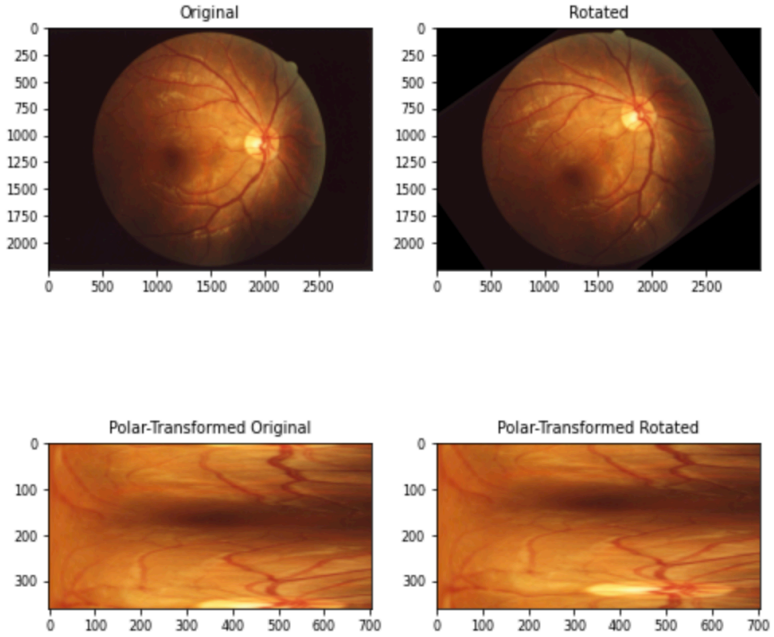
```
from skimage import data
from skimage.registration import phase_cross_correlation
from skimage.transform import warp_polar, rotate, rescale
from skimage.util import img_as_float

radius = 705
angle = 35
image = imread("../Images/fundus-image.jpeg")
image = img_as_float(image)
rotated = rotate(image, angle)
image_polar = warp_polar(image, radius=radius, multichannel=True)
rotated_polar = warp_polar(rotated, radius=radius, multichannel=True)

shifts, error, phasediff = phase_cross_correlation(image_polar, rotated_polar)
print("Expected value for counterclockwise rotation in degrees: "
      f"{angle}")
print("Recovered value for counterclockwise rotation: "
      f"{shifts[0]}")
```

Image Recovery

Recovering Rotational Differences: Polar Transforms

Phase correlation is an efficient method for determining translation offset between pairs of similar images. However this approach relies on a near absence of rotation/scaling differences between the images, which are typical in real-world examples.

In the example below, we consider the simple case of two images that only differ with respect to rotation around a common centre point. By remapping these images into polar space, the rotation difference becomes a simple translation difference that can be recovered by phase correlation.




```
Expected value for counterclockwise rotation in degrees: 35
Recovered value for counterclockwise rotation: 35.0
```

Image Recovery

Recovering Rotational Differences: Log Polar Transforms

Phase correlation is an efficient method for determining translation offset between pairs of similar images. However this approach relies on a near absence of rotation/scaling differences between the images, which are typical in real-world examples.

In another example below, the images differ by both rotation and scaling (note the axis tick values). By remapping these images into log-polar space, we can recover rotation as before, and now also scaling, by phase correlation.

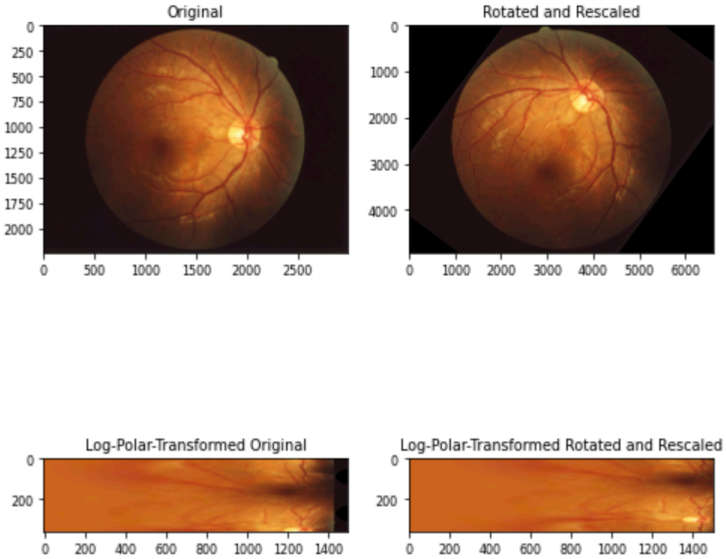
```
radius = 1500
angle = 53.7
scale = 2.2
image = imread("../Images/fundus-image.jpeg")
image = img_as_float(image)
rotated = rotate(image, angle)
rescaled = rescale(rotated, scale, multichannel=True)
image_polar = warp_polar(image, radius=radius,
                         scaling='log', multichannel=True)
rescaled_polar = warp_polar(rescaled, radius=radius,
                            scaling='log', multichannel=True)

# setting upsample_factor can increase precision
shifts, error, phasediff = phase_cross_correlation(image_polar, rescaled_polar,
                                                   upsample_factor=20)
shiftr, shiftc = shifts[:2]

# Calculate scale factor from translation
klog = radius / np.log(radius)
shift_scale = 1 / (np.exp(shiftc / klog))

print(f"Expected value for cc rotation in degrees: {angle}")
print(f"Recovered value for cc rotation: {shiftr}")
print()
print(f"Expected value for scaling difference: {scale}")
print(f"Recovered value for scaling difference: {shift_scale}")
```

Image Recovery

Recovering Rotational Differences: Log Polar Transforms

Phase correlation is an efficient method for determining translation offset between pairs of similar images. However this approach relies on a near absence of rotation/scaling differences between the images, which are typical in real-world examples.

In another example below, the images differ by both rotation and scaling (note the axis tick values). By remapping these images into log-polar space, we can recover rotation as before, and now also scaling, by phase correlation.




```
Expected value for cc rotation in degrees: 53.7
Recovered value for cc rotation: 51.75

Expected value for scaling difference: 2.2
Recovered value for scaling difference: 1.0017078748501664
```

Flood Fill

How can we do selective colouring?

Flood fill is an algorithm to identify and/or change adjacent values in an image based on their similarity to an initial seed point. The conceptual analogy is the ‘paint bucket’ tool in many graphic editors.

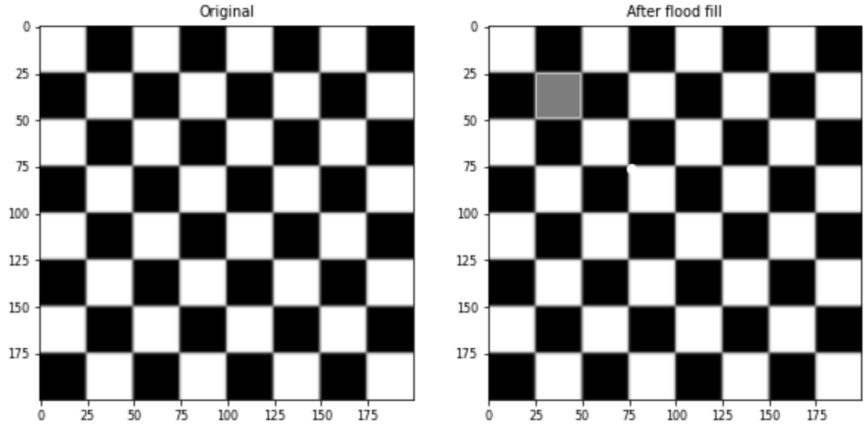
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, filters, color, morphology
from skimage.segmentation import flood, flood_fill


checkers = data.checkerboard()


filled_checkers = flood_fill(checkers, (36, 36), 127)
```

Flood Fill

How can we do selective colouring?

Flood fill is an algorithm to identify and/or change adjacent values in an image based on their similarity to an initial seed point. The conceptual analogy is the ‘paint bucket’ tool in many graphic editors.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, filters, color, morphology
from skimage.segmentation import flood, flood_fill


checkers = data.checkerboard()


filled_checkers = flood_fill(checkers, (36, 36), 127)
```

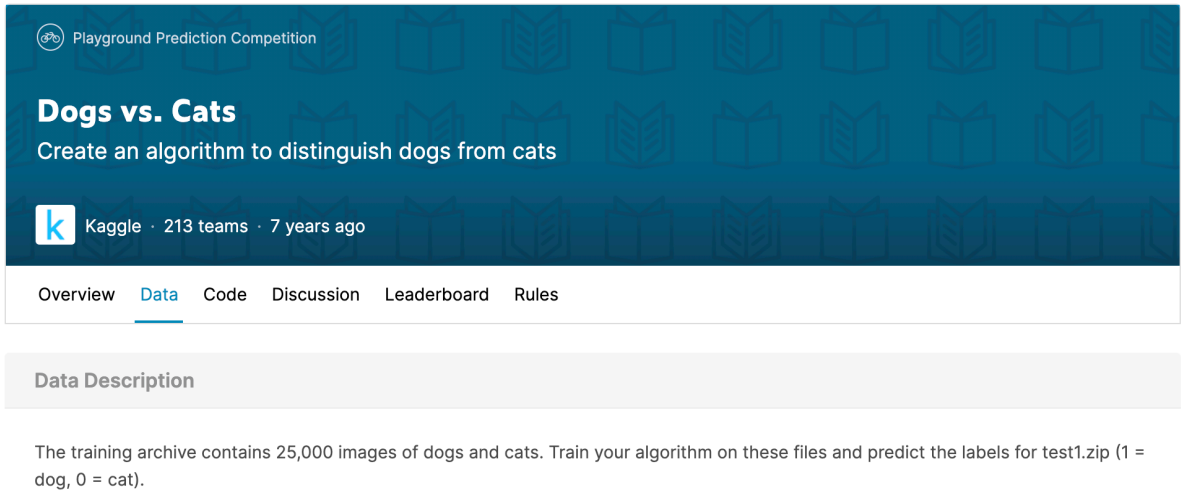


Learning from Images

What do we mean by data streams?

Let us begin a fresh perspective to understand the concept of learning from images using the ideas of deep learning. We will be using a dataset called “Dogs vs Cats” which is about distinguishing the dogs from the cats.

While solving the problem, we will be proposing many learning models architectures. While each architecture will be focused on providing a better understanding of the data by building a more accurate model, we will simultaneously be attempting to apply some of the concepts that we learned along the way during previous two sessions.



Click here to get the access the data from the Kaggle official competition page

Learning from Images

Beginning with a Basic Convolutional Neural Network Model

Let us propose a basic CNN model with only convolutional layers followed by max pooling. Our objective while developing this model is to examine how successfully our base model learns to discriminate between the classes.

```
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(),
              metrics=['accuracy'])

model.summary()
```

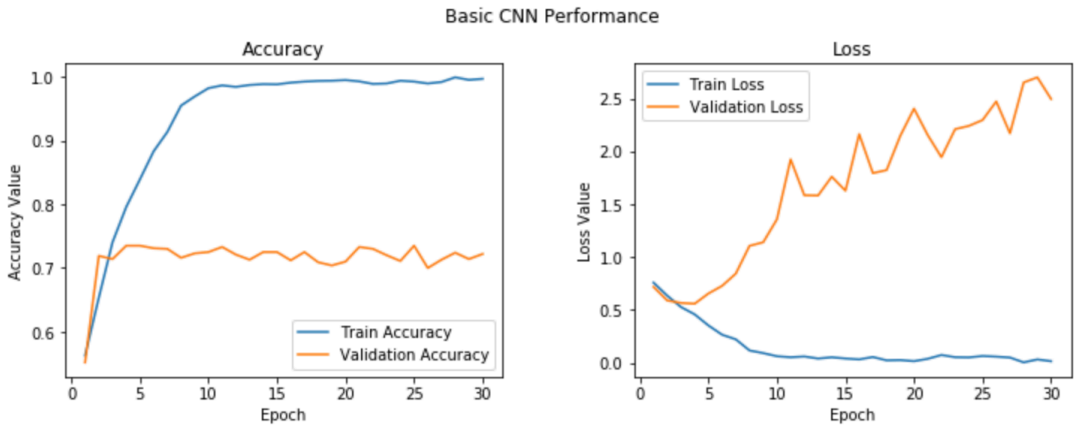
```
Epoch 25/30
3000/3000 [==============================] - 8s - loss: 0.0640 - acc: 0.9923 - val_loss: 2.2968 - val_acc: 0.7350
Epoch 26/30
3000/3000 [==============================] - 8s - loss: 0.0580 - acc: 0.9893 - val_loss: 2.4733 - val_acc: 0.7000
Epoch 27/30
3000/3000 [==============================] - 8s - loss: 0.0492 - acc: 0.9917 - val_loss: 2.1719 - val_acc: 0.7130
Epoch 28/30
3000/3000 [==============================] - 8s - loss: 0.0035 - acc: 0.9990 - val_loss: 2.6527 - val_acc: 0.7240
Epoch 29/30
3000/3000 [==============================] - 8s - loss: 0.0314 - acc: 0.9950 - val_loss: 2.7014 - val_acc: 0.7140
Epoch 30/30
3000/3000 [==============================] - 8s - loss: 0.0147 - acc: 0.9967 - val_loss: 2.4963 - val_acc: 0.7220
```

Learning from Images

Beginning with a Basic Convolutional Neural Network Model

Let us propose a basic CNN model with only convolutional layers followed by max pooling. Our objective while developing this model is to examine how successfully our base model learns to discriminate between the classes.

```
Model Performance metrics:
------------------------------
Accuracy: 0.776
Precision: 0.7769
Recall: 0.776
F1 Score: 0.7758

Model Classification report:
------------------------------
             precision    recall  f1-score   support

        cat       0.76      0.80      0.78       500
        dog       0.79      0.75      0.77       500

avg / total       0.78      0.78      0.78      1000
```

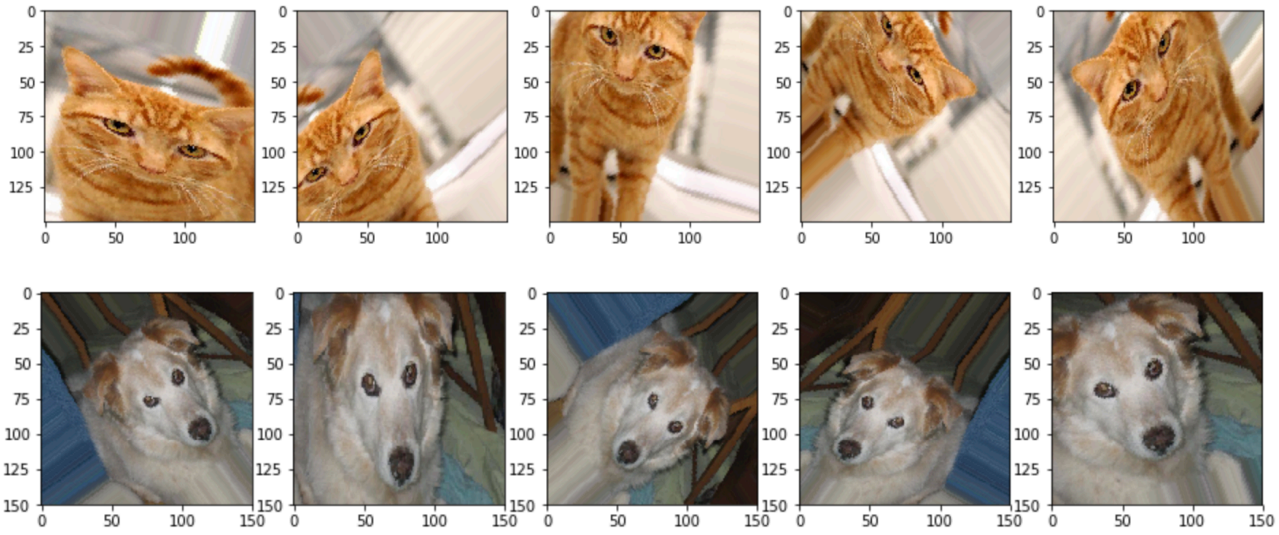


Learning from Images

Introducing Image Augmentations

Now, before we propose our next model with an increasing degree of sophistication, let us have a quick look on the image augmentation that we will perform on the dataset to improve feature extraction.

```
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
```



Learning from Images

Introducing Image Augmentations

Now, before we propose our next model with an increasing degree of sophistication, let us have a quick look on the image augmentation that we will perform on the dataset to improve feature extraction. Here is our next model-

```
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

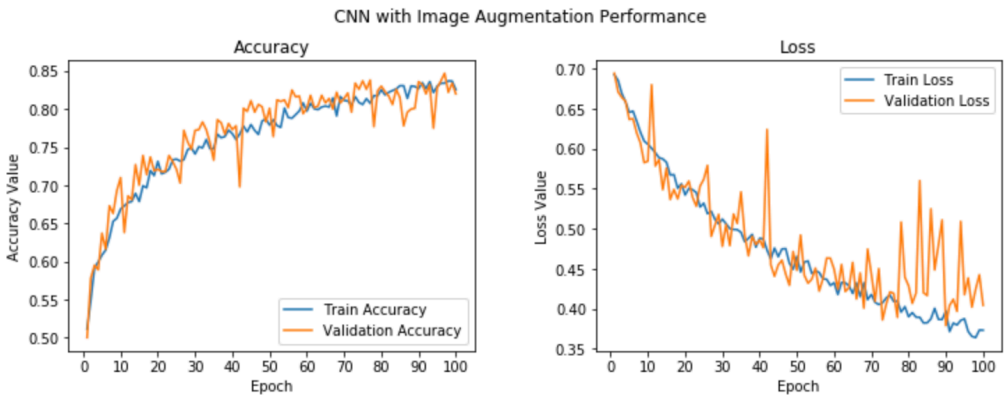
```
Epoch 95/100
100/100 [==============================] - 11s - loss: 0.3876 - acc: 0.8300 - val_loss: 0.4177 - val_acc: 0.8280
Epoch 96/100
100/100 [==============================] - 11s - loss: 0.3717 - acc: 0.8340 - val_loss: 0.4386 - val_acc: 0.8370
Epoch 97/100
100/100 [==============================] - 11s - loss: 0.3656 - acc: 0.8340 - val_loss: 0.4022 - val_acc: 0.8470
Epoch 98/100
100/100 [==============================] - 11s - loss: 0.3641 - acc: 0.8370 - val_loss: 0.4244 - val_acc: 0.8220
Epoch 99/100
100/100 [==============================] - 11s - loss: 0.3735 - acc: 0.8367 - val_loss: 0.4425 - val_acc: 0.8340
Epoch 100/100
100/100 [==============================] - 11s - loss: 0.3733 - acc: 0.8257 - val_loss: 0.4046 - val_acc: 0.8200
```

Learning from Images

Introducing Image Augmentations

Now, before we propose our next model with an increasing degree of sophistication, let us have a quick look on the image augmentation that we will perform on the dataset to improve feature extraction. Here is our next model-

```
Model Performance metrics:
------------------------------
Accuracy: 0.844
Precision: 0.844
Recall: 0.844
F1 Score: 0.844

Model Classification report:
------------------------------
              precision    recall  f1-score   support

         cat       0.84      0.84      0.84       500
         dog       0.84      0.84      0.84       500

avg / total       0.84      0.84      0.84      1000
```



Learning from Images

Introducing Transfer Learning

```
vgg = vgg16.VGG16(include_top=False, weights='imagenet',
                                     input_shape=input_shape)

output = vgg.layers[-1].output
output = keras.layers.Flatten()(output)

vgg_model = Model(vgg.input, output)
vgg_model.trainable = False

for layer in vgg_model.layers:
    layer.trainable = False

vgg_model.summary()
```

	Layer Type	Layer Name	Layer Trainable
0	<keras.engine.topology.InputLayer object at 0x7f26c86b2518>	input_1	False
1	<keras.layers.convolutional.Conv2D object at 0x7f277c9fc080>	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0x7f26c86b26d8>	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0x7f26c86e6c88>	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0x7f26c867dc18>	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0x7f26c8690f28>	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0x7f26c869e5c0>	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0x7f26c863f828>	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0x7f26c863f128>	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0x7f26c86607b8>	block3_conv3	False
10	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83d7d68>	block3_pool	False
11	<keras.layers.convolutional.Conv2D object at 0x7f26c83fd358>	block4_conv1	False
12	<keras.layers.convolutional.Conv2D object at 0x7f26c83fddd8>	block4_conv2	False
13	<keras.layers.convolutional.Conv2D object at 0x7f26c839da20>	block4_conv3	False
14	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83ac1d0>	block4_pool	False
15	<keras.layers.convolutional.Conv2D object at 0x7f26c834e978>	block5_conv1	False
16	<keras.layers.convolutional.Conv2D object at 0x7f271a15eb38>	block5_conv2	False
17	<keras.layers.convolutional.Conv2D object at 0x7f26c8371d68>	block5_conv3	False
18	<keras.layers.pooling.MaxPooling2D object at 0x7f26c8314b00>	block5_pool	False
19	<keras.layers.core.Flatten object at 0x7f26c828bda0>	flatten_1	False

Learning from Images

Introducing Transfer Learning

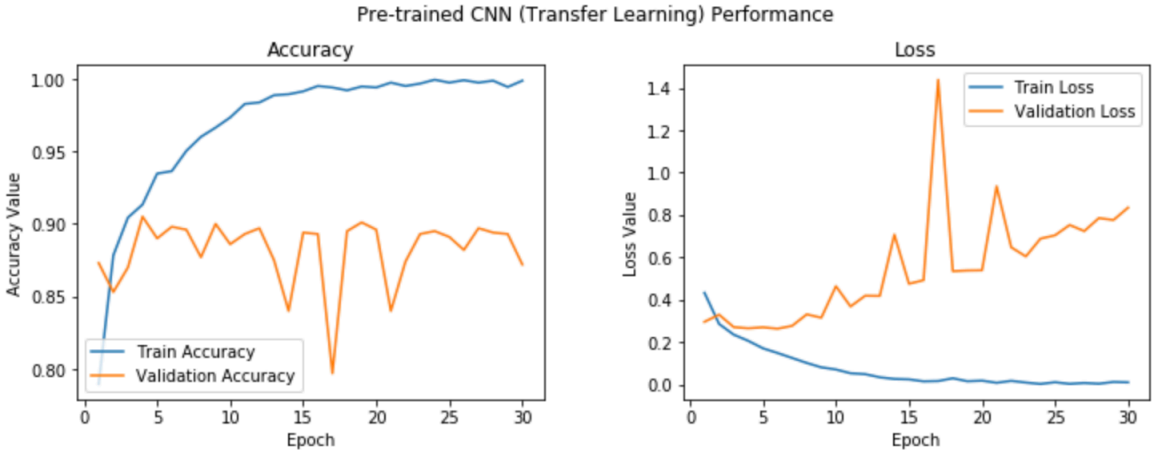
```
input_shape = vgg_model.output_shape[1]

model = Sequential()
model.add(InputLayer(input_shape=(input_shape,)))
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

```
Epoch 25/30
3000/3000 [==============================] - 1s 286us/step - loss: 0.0108 - acc: 0.9973 - val_loss: 0.7039 - val_acc: 0.8910
Epoch 26/30
3000/3000 [==============================] - 1s 285us/step - loss: 0.0030 - acc: 0.9990 - val_loss: 0.7526 - val_acc: 0.8820
Epoch 27/30
3000/3000 [==============================] - 1s 285us/step - loss: 0.0073 - acc: 0.9973 - val_loss: 0.7237 - val_acc: 0.8970
Epoch 28/30
3000/3000 [==============================] - 1s 285us/step - loss: 0.0037 - acc: 0.9987 - val_loss: 0.7852 - val_acc: 0.8940
Epoch 29/30
3000/3000 [==============================] - 1s 287us/step - loss: 0.0121 - acc: 0.9943 - val_loss: 0.7760 - val_acc: 0.8930
Epoch 30/30
3000/3000 [==============================] - 1s 287us/step - loss: 0.0102 - acc: 0.9987 - val_loss: 0.8344 - val_acc: 0.8720
```

Learning from Images

Introducing Transfer Learning

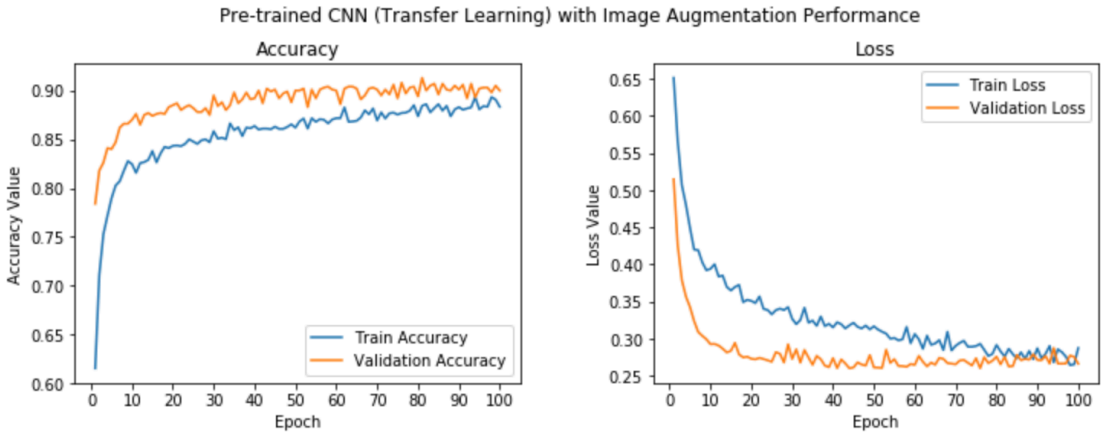
```
Model Performance metrics:
------------------------------
Accuracy: 0.888
Precision: 0.8898
Recall: 0.888
F1 Score: 0.8879

Model Classification report:
------------------------------
              precision    recall  f1-score   support

         cat       0.92      0.85      0.88       500
         dog       0.86      0.92      0.89       500

avg / total       0.89      0.89      0.89      1000
```



Learning from Images

Introducing Transfer Learning with Image Augmentation

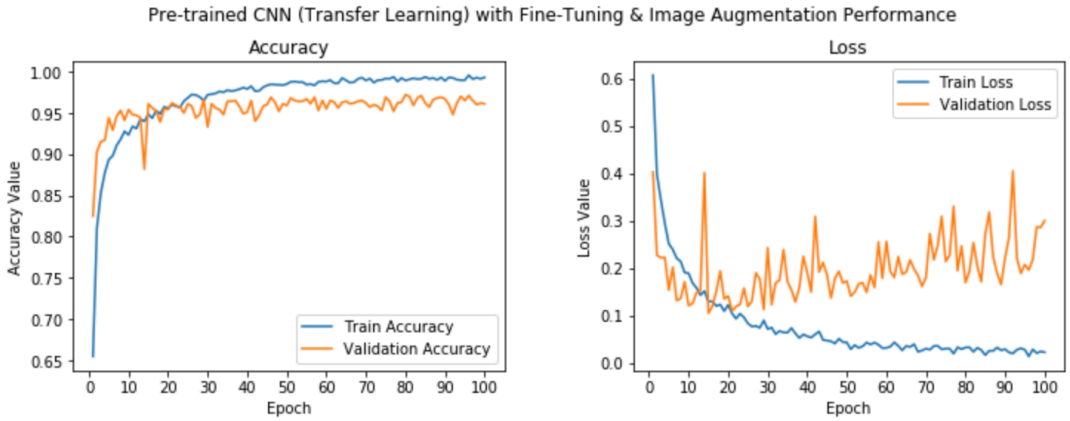
```
Model Performance metrics:
------------------------------
Accuracy: 0.898
Precision: 0.8981
Recall: 0.898
F1 Score: 0.898

Model Classification report:
------------------------------
              precision    recall  f1-score   support

         cat       0.89      0.91      0.90       500
         dog       0.90      0.89      0.90       500

avg / total       0.90      0.90      0.90      1000
```

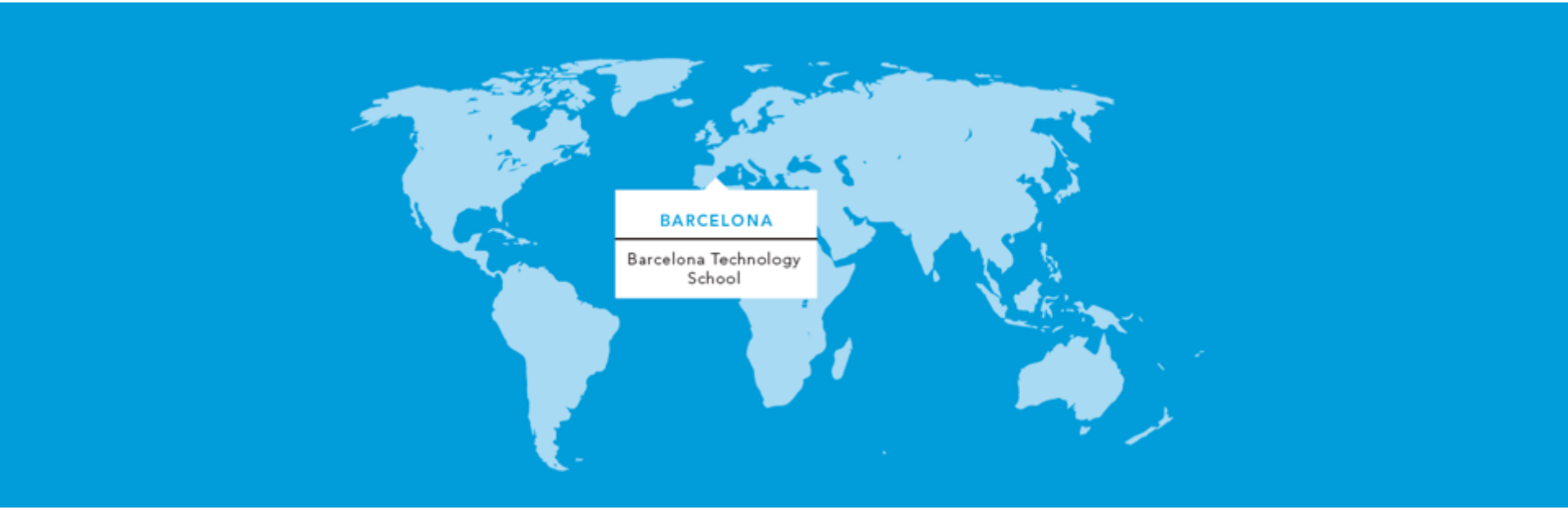


Learning from Images

Introducing Transfer Learning with Image Augmentation and Fine Tuning

```
Model Performance metrics:
------------------------------
Accuracy: 0.961
Precision: 0.9611
Recall: 0.961
F1 Score: 0.961

Model Classification report:
------------------------------
             precision    recall  f1-score   support

        cat       0.97      0.95      0.96       500
        dog       0.95      0.97      0.96       500

avg / total       0.96      0.96      0.96      1000
```




BARCELONA
Barcelona Technology School