```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
silviua_image_detection_images_path = kagglehub.dataset_download('silviua/image-detection-images')
silviua_training_imagesdownsized_path = kagglehub.dataset_download('silviua/training-imagesdownsized')
silviua_training_images_downsizedmore_path = kagglehub.dataset_download('silviua/training-images-downsizedmore')
silviua_trained_model_path = kagglehub.dataset_download('silviua/trained-model')
silviua_cuprogram_facut_path = kagglehub.dataset_download('silviua/cuprogram-facut')
silviua_images_personally_labeled_path = kagglehub.dataset_download('silviua/images-personally-labeled')
silviua_trained_model_nou_path = kagglehub.dataset_download('silviua/trained-model-nou')

print('Data source import complete.')
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn import metrics

import codecs
import glob
import json
import numpy as np

from tqdm import tqdm
import sys

from PIL import Image, ImageOps
import cv2 as cv

# Library for the Neural Network
from keras import layers
from keras import models
from keras import optimizers
from keras import losses
from keras.callbacks import ModelCheckpoint,EarlyStopping
from keras.models import load_model
from keras import backend as K

# Default image sizes
default_x = 100
default_y = 50
```

Load data, process and save as numpy zip

```
images_to_train = np.load("../input/image-detection-images/images.npy", allow_pickle=True)

image_det = images_to_train[200]
img = image_det[0] #cv.resize(np.uint8(image_det[0]), (600, 300))
img = Image.fromarray(img, 'RGB')
up_point = image_det[1][0]["points"][0]
down_point = image_det[1][0]["points"][1]
shape_x = image_det[0].shape[1]
shape_y = image_det[0].shape[0]
print(up_point, shape_y)
print (image_det[0].shape)

plt.imshow(img)
```

```
image_adjusted = np.uint8(cv.resize(np.uint8(img), (default_x, default_y)))
plt.imshow(image_adjusted)
```

+ Code + Text

```
def create_masks_for_training():
    global default_x
    global default_y
    X = np.empty((0, default_y, default_x, 3))
    y = np.empty((0, default_y, default_x, 3))

    for image in tqdm(images_to_train):
#         image = images_to_train[2]
        shape_x = image[0].shape[1]
        shape_y = image[0].shape[0]
        image_mask = np.zeros((shape_y, shape_x, 3)) # set background to 0
        for bbox in image[1]:
            up_point = bbox["points"][0]
            down_point = bbox["points"][1]
            image_mask[int(up_point['y']*shape_y):int(down_point['y']*shape_y), int(up_point['x']*shape_x):int(down_point['x'

        image_adjusted = np.uint8(cv.resize(np.uint8(image[0]), (default_x, default_y)))
        image_mask = np.uint8(cv.resize(np.uint8(image_mask), (default_x, default_y)))

        if (image_adjusted.shape != (default_y, default_x, 3)):
            continue

        image_adjusted = np.reshape(image_adjusted, (1, default_y, default_x, 3))
        image_mask = np.reshape(image_mask, (1, default_y, default_x, 3))

#         print (image_adjusted.shape, image_mask.shape)
        X = np.vstack((X, image_adjusted))
        y = np.vstack((y, image_mask))
#         fig = plt.figure()
#         fig.add_subplot(2,1,1)

#         plt.imshow(image_mask, cmap='Greys')
#         fig.add_subplot(2,1,2)

#         plt.imshow(image_adjusted)
    return X,y


X, y = create_masks_for_training()
```

```
np.savez('training_data_processed5.npz', train=X, train_labels=y)
```

```
print (X.shape, y.shape)
fig = plt.figure()
fig.add_subplot(2,1,1)

plt.imshow(y[100], cmap='Greys')
fig.add_subplot(2,1,2)

plt.imshow(Image.fromarray(np.uint8(X[100]), 'RGB'))
```

Load the processed training images

```
print (glob.glob('../input/*more/*.npz'))
```

```
def load_processed_data():
    # load training data

    X = np.empty((0, default_y, default_x, 3))
    y = np.empty((0, default_y* default_x* 1))
    training_data = glob.glob('../input/training-images-downsizedmore/training_data_processed4.npz')
    training_data.append(glob.glob('../input/images-personally-labeled/training_data_processed5.npz')[0])
    print (training_data)

    # if no data, exit
    if not training_data:
        print("Data not found, exit")
        sys.exit()

    for single_npz in training_data:
        with np.load(single_npz) as data:
            train = data['train']
            train_labels = data['train_labels']
        X = np.vstack((X, train))
        del train

        for label in tqdm(train_labels):
            img = cv.cvtColor(label.astype('uint8'), cv.COLOR_BGR2GRAY)
            y = np.vstack((y, img.ravel()))
        del train_labels
    return X, y

X, y = load_processed_data()
```

```
img_nr = 800
img = np.uint8(X[img_nr])
img_label = np.uint8(y[img_nr])
img_label = img_label.reshape(default_y, default_x)

fig = plt.figure()
fig.add_subplot(2,1,1)

plt.imshow(img_label, cmap='Greys')
fig.add_subplot(2,1,2)

plt.imshow(img)
```

```
# Blending the images with 0.3 and 0.7
test_img = img
pred = img_label
r = cv.split(test_img)[0]
g = cv.split(test_img)[1]
b = cv.split(test_img)[2]
b = b + pred
img = cv.merge([r,g,b])
plt.imshow(img)
```

```
X = (X/(float(np.amax(X[0]))))
y = (y/(float(np.amax(y))))
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3)

print (np.amax(X[0]))
print (np.amax(y[0]))
```

```
del X
del y
```

Create the model

Arhitecture

```
K.clear_session()

model = models.Sequential()
model.add(layers.Conv2D(3, (3, 3), activation = 'relu',data_format="channels_last", input_shape = (default_y, default_x, 3))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(3, (5, 5), activation = 'relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(3, (3, 3), activation = 'relu'))
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(y_train.shape[-1], activation = 'relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(y_train.shape[-1], activation = 'relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(y_train.shape[-1], activation = 'sigmoid'))
model.add(layers.Dropout(0.2))
model.summary()
```

Callbacks

```
best_model_weights = './base.model'
checkpoint = ModelCheckpoint(
    best_model_weights,
    monitor='val_loss',
    verbose=1,
    save_best_only=True,
    mode='min',
    save_weights_only=False,
    period=1
)

callbacks = [checkpoint]
```

Training method

```
model.compile(loss='binary_crossentropy', optimizer='adam')
history = model.fit(
    X_train,
    y_train,
    validation_data=(X_valid,y_valid),
    epochs = 100,
    verbose = 1,
    batch_size = 2
)
```

```
#Save the model
model.save_weights('model_wieghts_nou.h5')
model.save('model_keras_nou.h5')
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'b', label = "training loss")
plt.plot(epochs, val_loss, 'r', label = "validation loss")
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
test_img = X_valid[22].reshape(1,default_y, default_x, 3)
pred = model.predict(test_img)
pred = pred*255

test_img = test_img.reshape(default_y, default_x, 3)
pred = pred.reshape(default_y, default_x)
```

```
fig = plt.figure()
fig.add_subplot(2,1,1)

plt.imshow(pred, cmap='Greys')
fig.add_subplot(2,1,2)

plt.imshow(test_img)
```

```
image_number = 67
test_img = X_valid[image_number].reshape(1,default_y, default_x, 3)
pred = model.predict(test_img)
pred = pred*255

test_img = test_img.reshape(default_y, default_x, 3)
pred = pred.reshape(default_y, default_x)
```

```
fig = plt.figure()
fig.add_subplot(1,3,1)

plt.imshow(pred, cmap='Greys')
fig.add_subplot(1,3,2)
plt.imshow(test_img)
# Blending the images with 0.3 and 0.7
r = cv.split(test_img)[0]
g = cv.split(test_img)[1]
b = cv.split(test_img)[2]
b = b + pred
img = cv.merge([r,g,b])
fig.add_subplot(1,3,3)
plt.imshow(img)
```

Load the model for tests

```
print (glob.glob('../input/trained-model*/*'))


# path_to_model = glob.glob("../input/trained-model/model_keras.h5")[0]
path_to_model = glob.glob("../input/trained-model-nou/model_keras_nou.h5")[0]
new_model = load_model(path_to_model)


image_number = 39
test_img = X_valid[image_number].reshape(1,default_y, default_x, 3)
pred = new_model.predict(test_img)
pred = pred*255

test_img = test_img.reshape(default_y, default_x, 3)
pred = pred.reshape(default_y, default_x)
pred[pred>100] = 255
pred[pred<=100] = 0


fig = plt.figure()
fig.add_subplot(2,1,1)

plt.imshow(pred, cmap='Greys')
fig.add_subplot(2,1,2)
plt.imshow(test_img)



# Blending the images with 0.3 and 0.7
r = cv.split(test_img)[0]
g = cv.split(test_img)[1]
b = cv.split(test_img)[2]
b = b + pred
img = cv.merge([r,g,b])
plt.imshow(img)
```