Data Mining – Final Project

2022

Classification of Grapevine Leaves Using CNN

Student Name: Darya Farivar

Student Number: 610397141

Professor Name: Mrs.Sajedi

July - 2022

Introduction

The data that is considered in this project is related to the identification of the types of Grapevine leaves. It is not possible to separate these leaves when enlarged manually. So try with the data that automatically sorts the leaves.

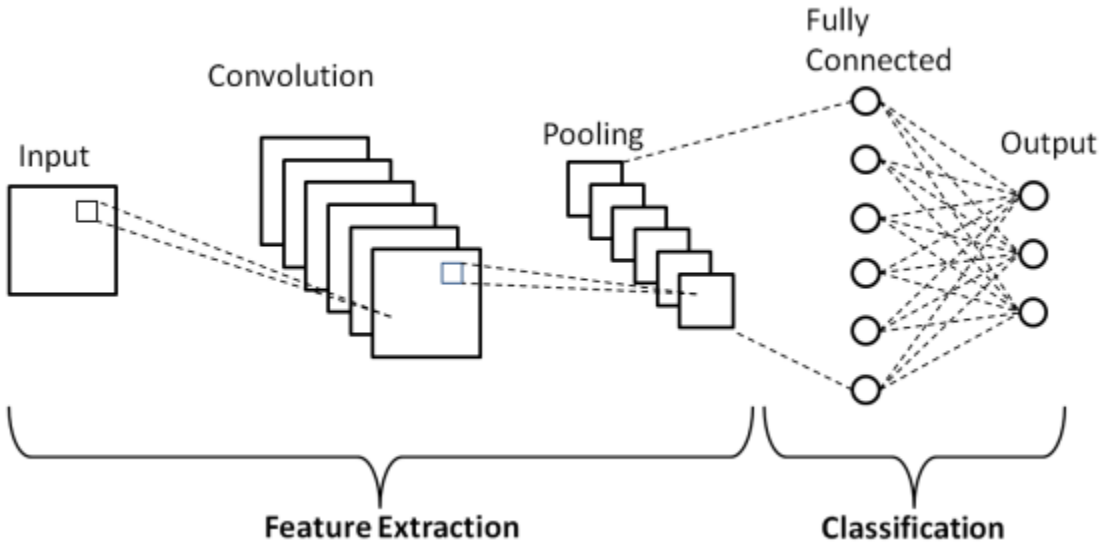
The main product of grapevines is grapes that are consumed fresh or processed. In addition, grapevine leaves are harvested once a year as a by-product. The species of grapevine leaves are important in terms of price and taste. For this purpose, images of 500 vine leaves belonging to 5 species were taken with a special self-illuminating system.

Problem Definition and Algorithm

We plan to design a machine learning model with the help of CNN so that the following 5 pages can be distinguished from each other.

	Ak	1
	Ala_Idris	2
	Buzgulu	3
	Dimnit	4
	Nazli	5

Deep Learning – which has emerged as an effective tool for analyzing big data – uses complex algorithms and artificial neural networks to train machines/computers so that they can learn from experience, classify and recognize data/images just like a human brain does. Within Deep Learning, a Convolutional Neural Network or CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. Deep Learning thus recognizes objects in an image by using a CNN. CNNs are playing a major role in diverse tasks/functions like image processing problems, computer vision tasks like localization and segmentation, video analysis, to recognize obstacles in self-driving cars, as well as speech recognition in natural language processing. As CNNs are playing a significant role in these fast-growing and emerging areas, they are very popular in Deep Learning.

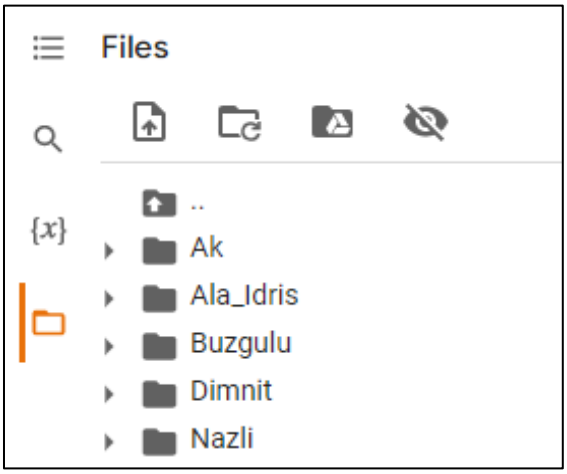


A typical neural network will have an input layer, hidden layers, and an output layer. CNNs are inspired by the architecture of the brain. Just like a neuron in the brain processes and transmits information throughout the body, artificial neurons or nodes in CNNs take inputs, processes them and sends the result as output. The image is fed as input. The input layer accepts the image pixels as input in the form of arrays. In CNNs, there could be multiple hidden layers, which perform feature extraction from the image by doing calculations. This could include convolution, pooling, rectified linear units, and fully connected layers. Convolution is the first layer that does feature extraction from an input image. The fully connected layer classifies the object and identifies it in the output layer. The Keras library in Python makes it pretty simple to build a CNN.

Building the CNN model

First of all, we import the required libraries and upload the data:

```
import numpy as np
import random
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
```



We put labels for train and test data:

```
Y_train = []
for i in range(400):
  if i < 80:
    Y_train.append(0)
  elif i < 160:
    Y_train.append(1)
  elif i < 240:
    Y_train.append(2)
  elif i < 320:
    Y_train.append(3)
  elif i < 400:
    Y_train.append(4)
Y_train = np.array(Y_train)
Y_train = Y_train.reshape(len(Y_train), 1)
print("Shape of Y_train: ", Y_train.shape)
```

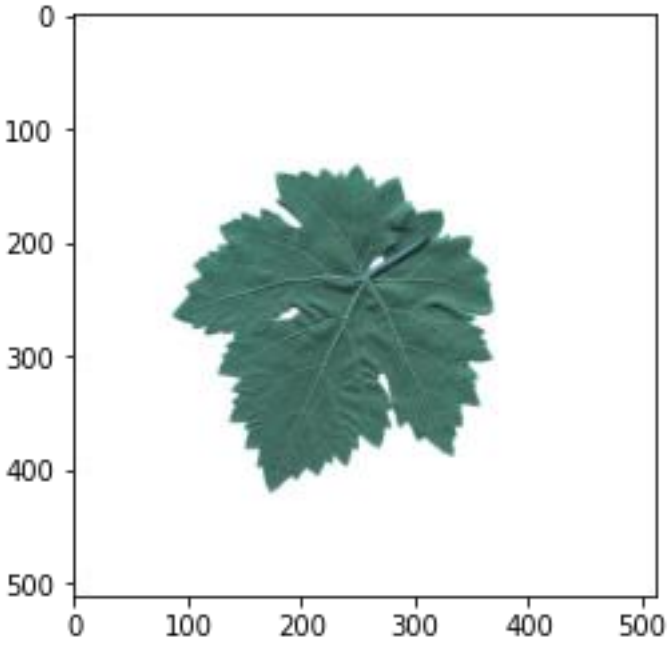
```
Shape of Y_train:  (400, 1)
```

```
X_train = []
for i in range(1,81):
  X_train.append(cv2.imread('Ak/Ak ('+str(i)+').png'))
for i in range(1,81):
  X_train.append(cv2.imread('Ala_Idris/Ala_Idris ('+str(i)+').png'))
for i in range(1,81):
  X_train.append(cv2.imread('Buzgulu/Buzgulu ('+str(i)+').png'))
for i in range(1,81):
  X_train.append(cv2.imread('Dimnit/Dimnit ('+str(i)+').png'))
for i in range(1,81):
  X_train.append(cv2.imread('Nazli/Nazli ('+str(i)+').png'))
X_train = np.array(X_train)

print('Train data shape: {}'.format(X_train.shape))
```

```
Train data shape: (400, 511, 511, 3)
```

A random image of the training data is plotted as follows:

```
idx = random.randint(0, len(X_train))
plt.imshow(X_train[idx, :])
plt.show()
```



We design and fit the model using Keras library as below:

```
model = Sequential([
    Conv2D(32, (3,3), activation = 'relu', input_shape = (511, 511, 3)),
    MaxPooling2D((2,2)),

    Conv2D(32, (3,3), activation = 'relu'),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(64, activation = 'relu'),
    Dense(1, activation = 'sigmoid')
])
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics =
['accuracy'])
model.fit(X_train, Y_train, epochs = 5, batch_size = 64)
```

```
Epoch 1/5
7/7 [==============================] - 116s 15s/step - loss: -270640.7188 - accuracy: 0.2000
Epoch 2/5
7/7 [==============================] - 101s 14s/step - loss: -2865414.7500 - accuracy: 0.2000
Epoch 3/5
7/7 [==============================] - 100s 14s/step - loss: -13435059.0000 - accuracy: 0.2000
Epoch 4/5
7/7 [==============================] - 100s 14s/step - loss: -43794664.0000 - accuracy: 0.2000
Epoch 5/5
7/7 [==============================] - 100s 14s/step - loss: -120244664.0000 - accuracy:
0.2000
<keras.callbacks.History at 0x7fb6976c9b50>
```

We label the test data and ask the model to make a prediction:

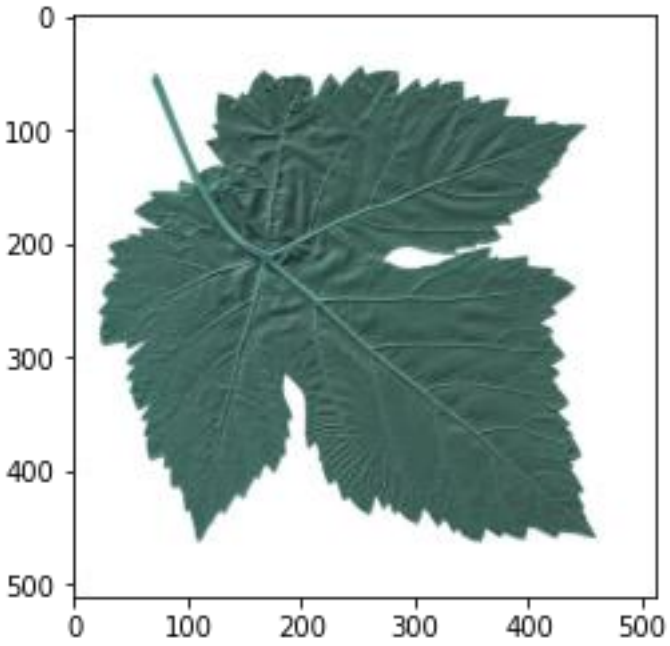
```
Y_test = []
for i in range(100):
  if i < 20:
    Y_test.append(0)
  elif i < 40:
    Y_test.append(1)
  elif i < 60:
    Y_test.append(2)
  elif i < 80:
    Y_test.append(3)
  elif i < 100:
    Y_test.append(4)
Y_test = np.array(Y_test)
Y_test = Y_test.reshape(len(Y_test), 1)
print("Shape of Y_test: ", Y_test.shape)
```

```
Shape of Y_test:  (100, 1)
```

```
X_test = []
for i in range(81,101):
  X_test.append(cv2.imread('Ak/Ak ('+str(i)+').png'))
for i in range(81,101):
  X_test.append(cv2.imread('Ala_Idris/Ala_Idris ('+str(i)+').png'))
for i in range(81,101):
  X_test.append(cv2.imread('Buzgulu/Buzgulu ('+str(i)+').png'))
for i in range(81,101):
  X_test.append(cv2.imread('Dimnit/Dimnit ('+str(i)+').png'))
for i in range(81,101):
  X_test.append(cv2.imread('Nazli/Nazli ('+str(i)+').png'))
X_test = np.array(X_test)

print('Train data shape: {}'.format(X_test.shape))
```

```
Train data shape: (100, 511, 511, 3)
```

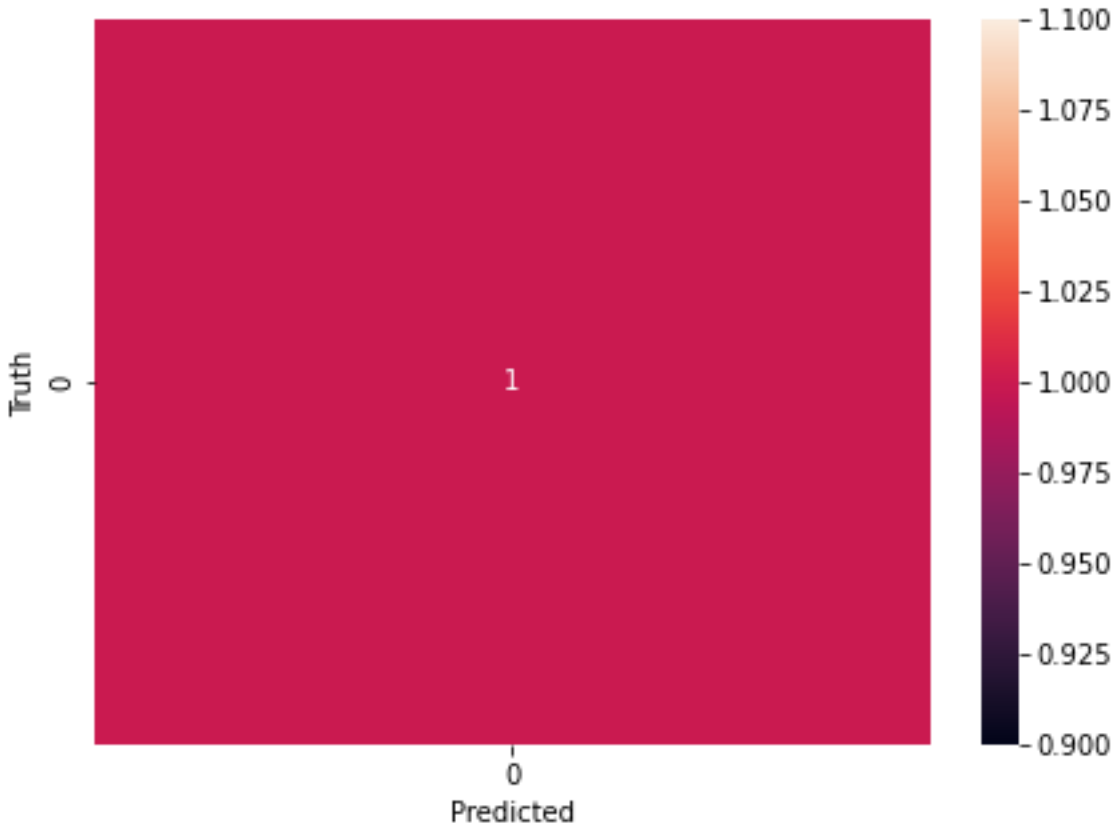
```
idx2 = random.randint(0, len(Y_test))
plt.imshow(X_test[idx2, :])
plt.show()

y_pred = model.predict(X_test[idx2, :].reshape(1, 511, 511, 3))

if(y_pred == 0):
    pred = 'Ak'
elif(y_pred == 1):
    pred = 'Ala_Idris'
elif(y_pred == 2):
    pred = 'Buzgulu'
elif(y_pred == 3):
    pred = 'Dimnit'
elif(y_pred == 4):
    pred = 'Nazli'

print("Our model says it is a :", pred)
```




```
Our model says it is a : Ala_Idris
```

Results and Confusion Matrix

	precision	recall	f1-score	support
1	1.00	1.00	1.00	1
accuracy			1.00	1
macro avg	1.00	1.00	1.00	1
weighted avg	1.00	1.00	1.00	1

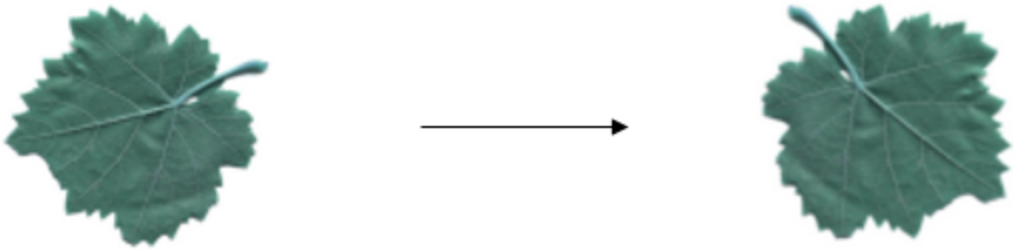


Data Augmentation

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularize and helps reduce overfitting when training a machine learning model.

Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.

Convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called invariance. More specifically, a CNN can be invariant to translation, viewpoint, size or illumination.



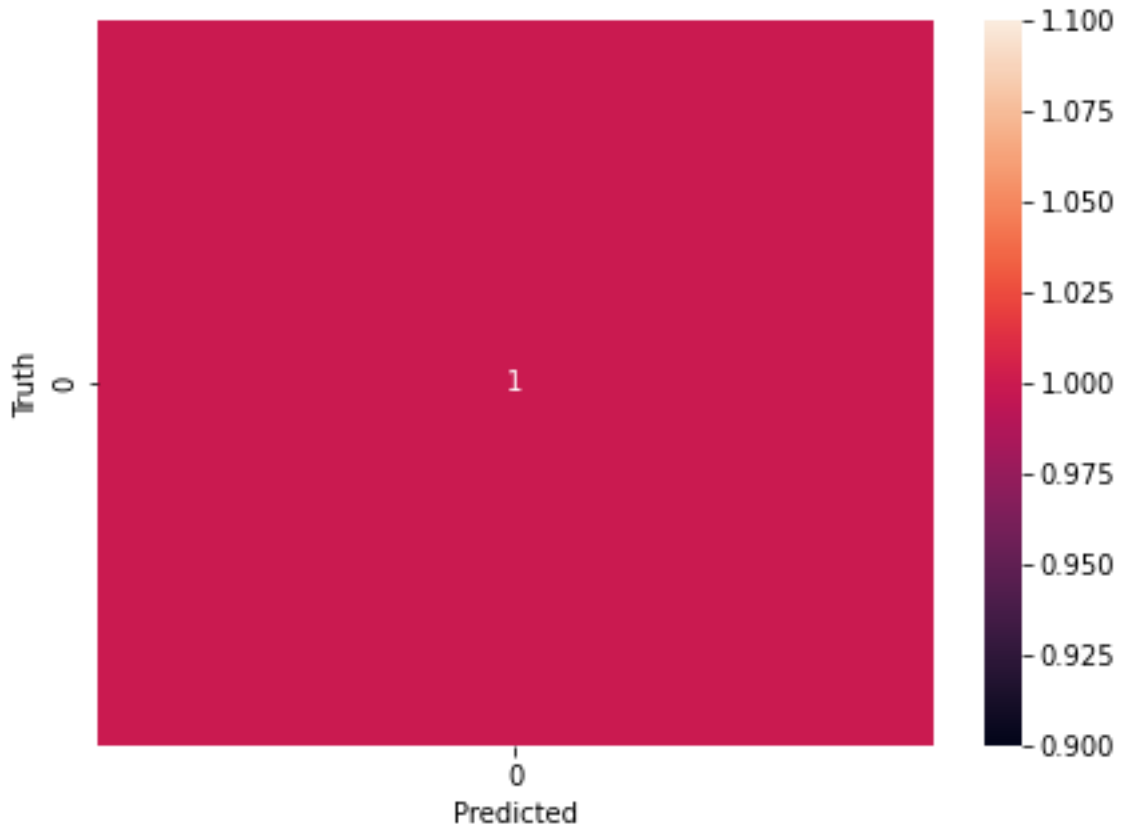
We doubled the data size:

```
Train data shape: (800, 511, 511, 3)
Shape of Y_train_augmented:  (800, 1)
```

Applying CNN on Augmented Data

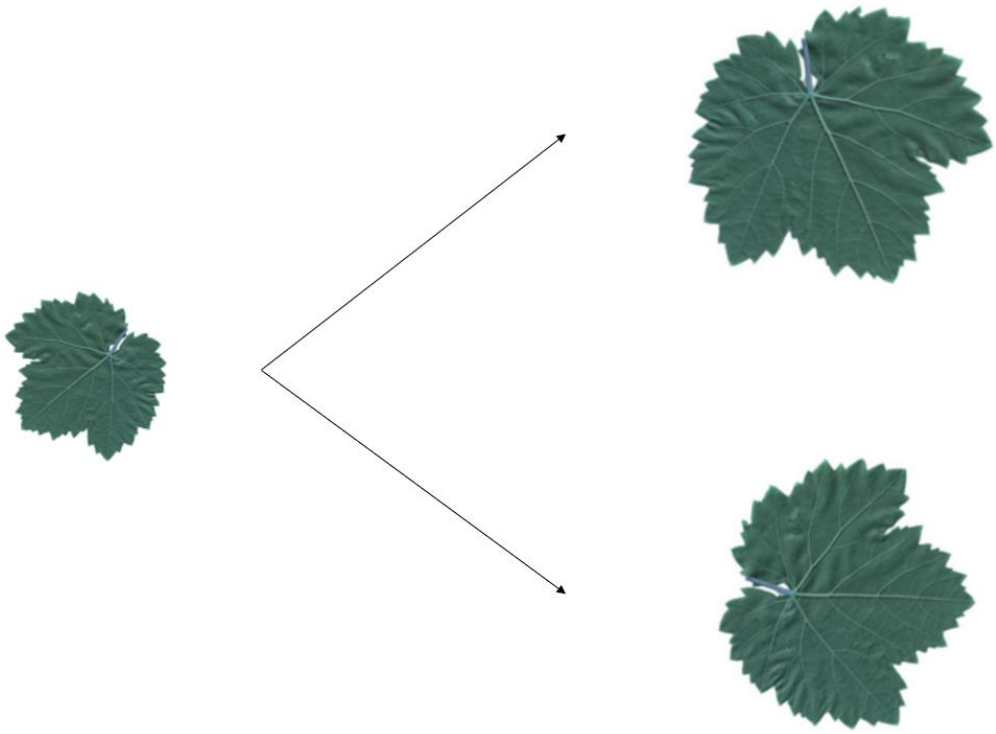
The results and confusion matrix:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	1
accuracy			1.00	1
macro avg	1.00	1.00	1.00	1
weighted avg	1.00	1.00	1.00	1



Applying CNN on Augmented Data #2

This time, we apply the data augmented as follows:



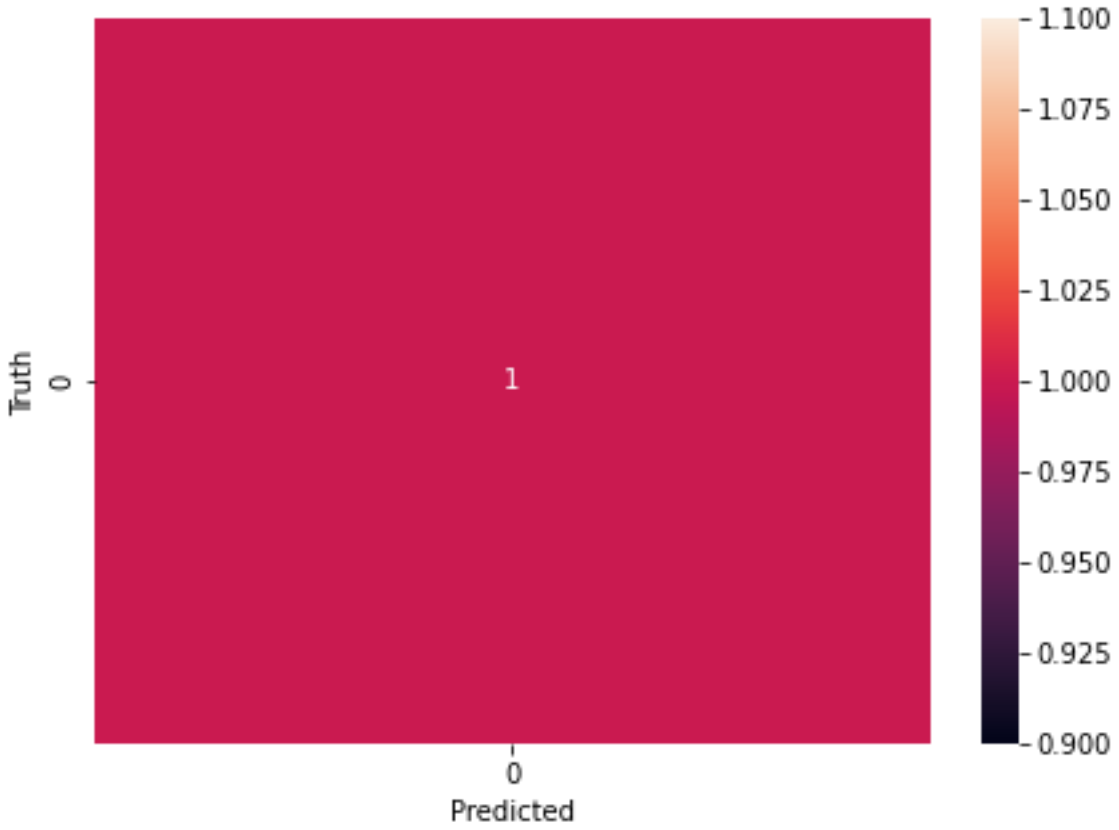
```
Train data shape: (2000, 511, 511, 3)

Shape of Y_train_augmented_1: (2000, 1)
```

```
model.fit(X_train_augmented_1, Y_train_augmented_1, epochs = 5, batch_size
 = 64)
```

```
Epoch 1/5
32/32 [==============================] - 532s 17s/step - loss: -
40690413568.0000 - accuracy: 0.2000
Epoch 2/5
32/32 [==============================] - 521s 16s/step - loss: -
123440930816.0000 - accuracy: 0.2000
Epoch 3/5
32/32 [==============================] - 523s 16s/step - loss: -
301287079936.0000 - accuracy: 0.2000
Epoch 4/5
32/32 [==============================] - 529s 17s/step - loss: -
634264092672.0000 - accuracy: 0.2000
Epoch 5/5
32/32 [==============================] - 527s 16s/step - loss: -
1186602418176.0000 - accuracy: 0.2000
<keras.callbacks.History at 0x7fb6966e6490>
```

The results and confusion matrix:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	1
accuracy			1.00	1
macro avg	1.00	1.00	1.00	1
weighted avg	1.00	1.00	1.00	1



Convolutional Autoencoder for Image Denoising

Autoencoder is an unsupervised artificial neural network that is trained to copy its input to output. In the case of image data, the autoencoder will first encode the image into a lower-dimensional representation, then decodes that representation back to the image.

In the context of computer vision, denoising autoencoders can be seen as very powerful filters that can be used for automatic pre-processing. For example, a denoising autoencoder could be used to automatically pre-process an image, improving its quality for an OCR algorithm and thereby increasing OCR accuracy.

With tensorflow.keras, we try to do this on our own data.

10-fold cross validation

10-fold cross validation would perform the fitting procedure a total of ten times, with each fit being performed on a training set consisting of 90% of the total training set selected at random, with the remaining 10% used as a hold out set for validation.

If we run the following code ten times and implement it on the model, we will reach this evaluation.

```
X_train = []
X_test = []
Y_train = []
Y_test = []

numbers = list(range(1,101))

XXX,YYY = train_test_split(numbers, test_size=0.1, random_state=42)

for i in range(1,101):
  if i in XXX:
    X_train.append(cv2.imread('Ak/Ak ('+str(i)+').png'))
    Y_train.append(i)
    Y_train.append(0)
  elif i in YYY:
    X_test.append(cv2.imread('Ak/Ak ('+str(i)+').png'))
    Y_test.append(i)
    Y_test.append(0)

for i in range(1,101):
  if i in XXX:
    X_train.append(cv2.imread('Ala_Idris/Ala_Idris ('+str(i)+').png'))
    Y_train.append(i)
    Y_train.append(1)
  elif i in YYY:
    X_test.append(cv2.imread('Ala_Idris/Ala_Idris ('+str(i)+').png'))
    Y_test.append(i)
    Y_test.append(1)

for i in range(1,101):
  if i in XXX:
    X_train.append(cv2.imread('Buzgulu/Buzgulu ('+str(i)+').png'))
    Y_train.append(i)
    Y_train.append(2)
```

```
  elif i in YYY:
    X_test.append(cv2.imread('Buzgulu/Buzgulu ('+str(i)+').png'))
    Y_test.append(i)
    Y_test.append(2)

for i in range(1,101):
  if i in XXX:
    X_train.append(cv2.imread('Dimnit/Dimnit ('+str(i)+').png'))
    Y_train.append(i)
    Y_train.append(3)
  elif i in YYY:
    X_test.append(cv2.imread('Dimnit/Dimnit ('+str(i)+').png'))
    Y_test.append(i)
    Y_test.append(3)

for i in range(1,101):
  if i in XXX:
    X_train.append(cv2.imread('Nazli/Nazli ('+str(i)+').png'))
    Y_train.append(i)
    Y_train.append(4)
  elif i in YYY:
    X_test.append(cv2.imread('Nazli/Nazli ('+str(i)+').png'))
    Y_test.append(i)
    Y_test.append(4)

def to_matrix(l, n):
  return [l[i:i+n] for i in range(0, len(l), n)]

Y_train = to_matrix(Y_train,2)
Y_test = to_matrix(Y_test,2)

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)

print('Train data shape: {}'.format(X_train.shape))
print('Train data shape: {}'.format(X_test.shape))
print("Shape of X_test: ", Y_train.shape)
print("Shape of Y_test: ", Y_test.shape)
```

```
Train data shape: (450, 511, 511, 3)
Train data shape: (50, 511, 511, 3)
Shape of X_test:  (450, 2)
Shape of Y_test:  (50, 2)
```

References

https://keras.io/examples/vision/autoencoder/
https://www.sciencedirect.com/science/article/abs/pii/S0263224121013142?via%3Dihub