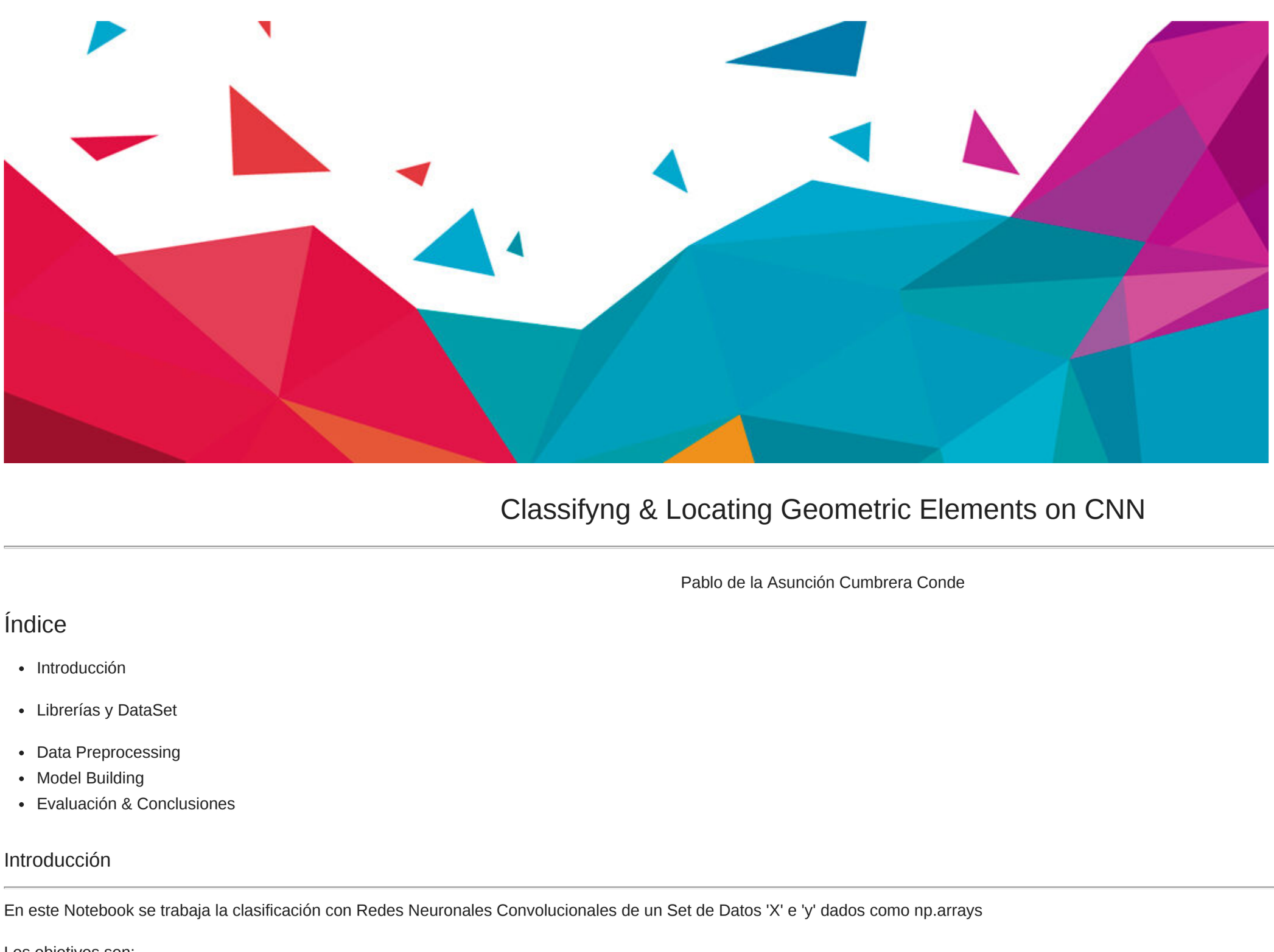


Classifyng & Locating Geometric Elements on CNN

Pablo de la Asunción Cumbrera Conde

Índice

Introducción

En este Notebook se trabaja la clasificación con Redes Neuronales Convolucionales de un Set de Datos 'X' e 'y' dados como np.arrays

Los objetivos son:

- Identificar y clasificar correctamente las figuras geométricas
- Localizar espacialmente las figuras geométricas

Librerias y DataSet

In [1]:
```python
import pickle
import joblib
import pandas as pd
import numpy as np
import tensorflow as tf
import torch as th
from pathlib import Path
from matplotlib import pyplot as plt
import visualkeras
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from tensorflow import keras
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.utils import *
```

In [2]:
```python
your_directory = Path('C:/Users/pablo.cumbrera/Documents/Data/Raw DATA/Geometric')
file = 'Geometric_Elements.pyc'
labels='Geometric_Elements_labels.pyc'

with open(your_directory/file,'rb') as f:
    X=pickle.load(f)
    print(f'This is X: \n\n {X}' "\n")

with open(your_directory/labels,'rb') as f:
    y=pickle.load(f)
    print(f'This is y: \n\n {y}')
```

```
This is X: 

 [[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 ...

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]

This is y: 

 [[array([0.08283513, 0.07818838, 0.08283513, 1.07818838, 1.08283513,
       1.07818838, 1.08283513, 0.07818838])
  array([1], dtype=int64)]
 [array([0.44331814, 0.73571743, 0.44331814, 1.48571743, 1.30934355,
       1.48571743, 1.30934355, 0.73571743])
  array([2], dtype=int64)]
 [array([0.20251281, 0.37490636, 0.20251281, 1.37490605, 1.20251219,
       1.37490605, 1.20251219, 0.37490636])
  array([0], dtype=int64)]
 ...
 [array([0.54042877, 0.12995149, 0.54042877, 1.12995149, 1.54042877,
       1.12995149, 1.54042877, 0.12995149])
  array([1], dtype=int64)]
 [array([0.75676932, 0.62497626, 0.75676932, 1.37497626, 1.62279472,
       1.37497626, 1.62279472, 0.62497626])
  array([2], dtype=int64)]
 [array([0.00707801, 0.23449319, 0.00707801, 1.23449288, 1.0070774 ,
       1.23449288, 1.0070774 , 0.23449319])
  array([0], dtype=int64)]]
```

In [3]:
```python
print("X shape: ",X.shape)
print("y shape: ", y.shape)
```

```
X shape:  (3000, 56, 56)
y shape:  (3000, 2)
```

In [4]:
```python
y_data = pd.DataFrame(y)
y_data.head()
y_data[0][0]
```

Out[4]:
```
array([0.08283513, 0.07818838, 0.08283513, 1.07818838, 1.08283513,
       1.07818838, 1.08283513, 0.07818838])
```

In [5]:
```python
y_data.columns = ['loc','labels']
```

In [6]:
```python
y_data.head()
```

Out[6]:

	loc	labels
0	[0.08283513, 0.07818838, 0.08283513, 1.0781883...	[1]
1	[0.44331814, 0.73571743, 0.44331814, 1.4857174...	[2]
2	[0.20251281, 0.37490636, 0.20251281, 1.3749060...	[0]
3	[0.11171338, 0.29153416, 0.11171338, 1.2915341...	[1]
4	[0.45019937, 0.32350172, 0.45019937, 1.3235014...	[0]

In [7]:
```python
X_train, X_test = train_test_split(X, shuffle=False)
y_train, y_test = train_test_split(y_data,shuffle=False)
print("X train shape: ",X_train.shape)
print("X test shape: ",X_test.shape)
print("y train shape: ", y_train.shape)
print("y test shape: ", y_test.shape)
```

```
X train shape:  (2250, 56, 56)
X test shape:  (750, 56, 56)
y train shape:  (2250, 2)
y test shape:  (750, 2)
```

In [8]:
```python
y_train_box = np.array([i for i in y_train.values[:,0]])
y_train_cat = tf.keras.utils.to_categorical(np.array([i for i in y_train.values[:,1]]))
y_test_box = np.array([i for i in y_test.values[:,0]])
y_test_cat = tf.keras.utils.to_categorical(np.array([i for i in y_test.values[:,1]]))
```

Data Preprocessing

- Exploración de imagenes 'X' asociadas a etiqueta 'y'

In [9]:
```python
class_names = []

def plot_pics(Ximg):
    numero_imagenes = 5
    fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(16, 8))
    for i in range(numero_imagenes):
        axs[i].imshow(tf.reshape(Ximg[i], [56, 56])*255,vmin=0, vmax=255,cmap='gray')

plot_pics(X)
```

3 Elementos

- Cuadrado
- Triángulo
- Círculo

In [10]:
```python
print("Elemento: Cuadrado ", y_data['labels'][0],"\n")
print("Elemento: Triángulo ", y_data['labels'][1],"\n")
print("Elemento: Círculo ", y_data['labels'][2],"\n")
print("Elemento: Cuadrado ", y_data['labels'][3],"\n")
print("Elemento: Círculo ", y_data['labels'][4],"\n")
```

```
Elemento: Cuadrado  [1] 

Elemento: Triángulo  [2] 

Elemento: Círculo  [0] 

Elemento: Cuadrado  [1] 

Elemento: Círculo  [0] 
```

3 etiquetas asociadas a elementos

- Círculo: 0
- Cuadrado: 1
- Triángulo: 2

In [11]:
```python
class_names = ["Círculo","Cuadrado","Triángulo"]
class_names
labels = {0:'Círculo',1:'Cuadrado',2:'Triángulo'}
type(labels)
```

Out[11]: dict

Exploratory Data Analysis

Comprobamos que las etiquetas para cada elemento son correctas

In [12]:
```python
class_names = ["Círculo","Cuadrado","Triángulo"]

def plot_pics(Ximg, ylabel):
    numero_imagenes = 10
    fig, axs = plt.subplots(nrows=1, ncols=10, figsize=(16, 8))
    for i in range(numero_imagenes):
        axs[i].imshow(tf.reshape(Ximg[i], [56, 56])*255,vmin=0, vmax=255,cmap='gray')
        axs[i].title.set_text(str(class_names[int(ylabel.values[i])]))

plot_pics(X,y_data['labels'])
```

In [13]:
```python
class_names = ["Círculo","Cuadrado","Triángulo"]

def plot_pics(Ximg, ylabel):
    numero_imagenes = 10
    fig, axs = plt.subplots(nrows=1, ncols=10, figsize=(16, 8))
    for i in range(numero_imagenes):
        axs[i].imshow(tf.reshape(Ximg[i], [56, 56])*255,vmin=0, vmax=255,cmap='gray')
        axs[i].title.set_text(str(class_names[int(ylabel.values[i])]))

plot_pics(X_train,y_train['labels'])
plot_pics(X_test,y_test['labels'])
```

In [14]:
```python
X_train = np.array([i.reshape(56,56,1) for i in X_train])
X_test = np.array([i.reshape(56,56,1) for i in X_test])
```

In [15]:
```python
print("Shape tensor de X_train: ", X_train.shape,"\n")
print("Shape tensor de X_test: ", X_test.shape)
```

```
Shape tensor de X_train:  (2250, 56, 56, 1) 

Shape tensor de X_test:  (750, 56, 56, 1)
```

In [16]:
```python
#def token_create(arg):
   # return np.array([np.eye(1,size, int(i))[0] for i in arg.values])

#Creamos un token para cada Training y Test de Y

#size=3
#y_train_labels= token_create(y_train['labels'])
#y_test_labels = token_create(y_test['labels'])
```

In [17]:
```python
#y_train_loc = th.tensor(y_train['loc'])
#y_test_loc = th.tensor(y_train['loc'])
#print(type(y_train_loc))
#print(type(y_test_loc))
```

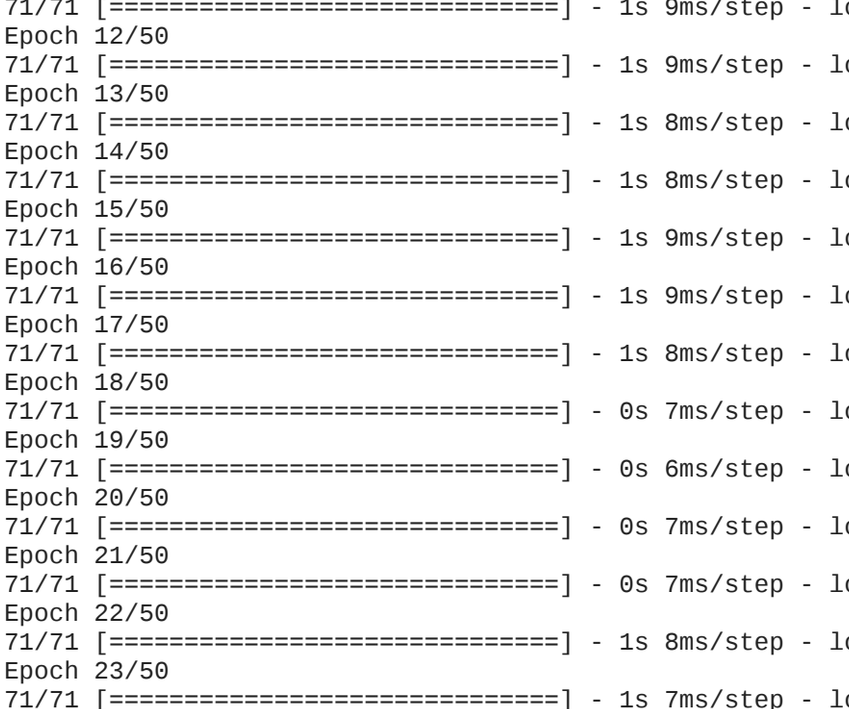
Model Building

In [18]:
```python
#Input Layer

model_input = Input(shape=(56,56,1))

#Block 1

x = Conv2D(filters=8, kernel_size=(2,2), strides=(2,2), activation='relu', padding= 'same')(model_input)
x = Conv2D(filters=8, kernel_size=(2,2), strides=(2,2), activation='relu', padding= 'same')(x)

#Block 5

flat = Flatten()(x)
y = Dense(300, activation='relu')(flat)
y = Dense(100, activation='relu')(y)

# Output Layer

output1 = Dense(3, activation='softmax')(y)
output2 = Dense(8,)(y)
outputs=[output1,output2]

model = Model(inputs = model_input, outputs = outputs)

print(model.summary())

# Compile the model

opt1 = keras.optimizers.Adam(learning_rate=0.002)
opt2 = keras.optimizers.RMSprop(learning_rate=0.002)

model.compile(optimizer=opt2,loss=['categorical_crossentropy','mse'], metrics=['accuracy'])

# Callback

early_stopping = EarlyStopping(monitor='loss',patience=5)
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 56, 56, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 28, 28, 8)    40          input_1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 14, 14, 8)    264         conv2d[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1568)         0           conv2d_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 300)          470700      flatten[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 100)          30100       dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 3)            303         dense_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 8)            808         dense_1[0][0]
==================================================================================================
Total params: 502,215
Trainable params: 502,215
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

In [19]:
```python
visualkeras.layered_view(model)
```

Out[19]:



In [27]:
```python
#Entremos el modelo
history = model.fit(X_train, [y_train_cat,y_train_box], epochs=50,callbacks=[early_stopping])
```

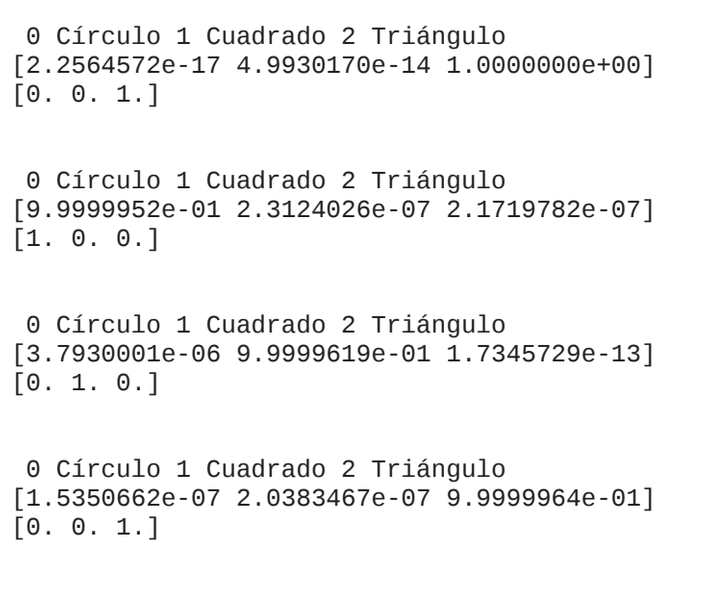
```
Epoch 1/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0030 - dense_2_loss: 7.2656e-06 - dense_3_loss: 0.0029 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4862
Epoch 2/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0027 - dense_2_loss: 8.2589e-06 - dense_3_loss: 0.0027 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4702
Epoch 3/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0026 - dense_2_loss: 7.5550e-06 - dense_3_loss: 0.0026 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4867
Epoch 4/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0026 - dense_2_loss: 5.0779e-06 - dense_3_loss: 0.0026 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4884
Epoch 5/50
71/71 [==============================] - 1s 10ms/step - loss: 0.0024 - dense_2_loss: 5.0477e-06 - dense_3_loss: 0.0024 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4853
Epoch 6/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0025 - dense_2_loss: 5.3402e-06 - dense_3_loss: 0.0025 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4680
Epoch 7/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0023 - dense_2_loss: 8.2272e-06 - dense_3_loss: 0.0023 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4716
Epoch 8/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0023 - dense_2_loss: 6.8697e-06 - dense_3_loss: 0.0022 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4938
Epoch 9/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0021 - dense_2_loss: 6.6367e-06 - dense_3_loss: 0.0021 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4933
Epoch 10/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0021 - dense_2_loss: 5.3415e-06 - dense_3_loss: 0.0021 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4662
Epoch 11/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0021 - dense_2_loss: 5.7152e-06 - dense_3_loss: 0.0020 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4942
Epoch 12/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0019 - dense_2_loss: 6.7879e-06 - dense_3_loss: 0.0019 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4822
Epoch 13/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0018 - dense_2_loss: 4.0386e-06 - dense_3_loss: 0.0018 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4836
Epoch 14/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0018 - dense_2_loss: 1.0341e-05 - dense_3_loss: 0.0018 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4751
Epoch 15/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0018 - dense_2_loss: 5.3709e-06 - dense_3_loss: 0.0018 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4760
Epoch 16/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0017 - dense_2_loss: 6.1832e-06 - dense_3_loss: 0.0017 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4853
Epoch 17/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0016 - dense_2_loss: 4.9026e-06 - dense_3_loss: 0.0015 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4716
Epoch 18/50
71/71 [==============================] - 0s 7ms/step - loss: 0.0015 - dense_2_loss: 7.9022e-06 - dense_3_loss: 0.0015 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4662
Epoch 19/50
71/71 [==============================] - 0s 6ms/step - loss: 0.0016 - dense_2_loss: 5.2635e-06 - dense_3_loss: 0.0016 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4733
Epoch 20/50
71/71 [==============================] - 0s 7ms/step - loss: 0.0014 - dense_2_loss: 4.5123e-06 - dense_3_loss: 0.0014 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4893
Epoch 21/50
71/71 [==============================] - 0s 7ms/step - loss: 0.0014 - dense_2_loss: 6.9168e-06 - dense_3_loss: 0.0013 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4778
Epoch 22/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0014 - dense_2_loss: 3.6353e-06 - dense_3_loss: 0.0014 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4791
Epoch 23/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0013 - dense_2_loss: 4.5034e-06 - dense_3_loss: 0.0013 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4756
Epoch 24/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0013 - dense_2_loss: 4.6244e-06 - dense_3_loss: 0.0013 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4947
Epoch 25/50
71/71 [==============================] - 0s 7ms/step - loss: 0.0012 - dense_2_loss: 5.5632e-06 - dense_3_loss: 0.0012 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4800
Epoch 26/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0012 - dense_2_loss: 4.9069e-06 - dense_3_loss: 0.0011 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4813
Epoch 27/50
71/71 [==============================] - 0s 7ms/step - loss: 0.0011 - dense_2_loss: 4.1076e-06 - dense_3_loss: 0.0011 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4631
Epoch 28/50
71/71 [==============================] - 1s 7ms/step - loss: 0.0011 - dense_2_loss: 2.8875e-05 - dense_3_loss: 0.0011 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4844
Epoch 29/50
71/71 [==============================] - 1s 8ms/step - loss: 0.0012 - dense_2_loss: 6.1448e-06 - dense_3_loss: 0.0011 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4769
Epoch 30/50
71/71 [==============================] - 1s 9ms/step - loss: 9.8564e-04 - dense_2_loss: 2.4921e-05 - dense_3_loss: 9.6071e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4853
Epoch 31/50
71/71 [==============================] - 1s 9ms/step - loss: 0.0010 - dense_2_loss: 4.5726e-06 - dense_3_loss: 0.0010 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4716
Epoch 32/50
71/71 [==============================] - 1s 9ms/step - loss: 9.1724e-04 - dense_2_loss: 8.3287e-06 - dense_3_loss: 9.0892e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4796
Epoch 33/50
71/71 [==============================] - 1s 15ms/step - loss: 0.0010 - dense_2_loss: 6.4721e-06 - dense_3_loss: 0.0010 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4951   - ET
A: 0s - loss: 0.0012 - dense_2_loss: 1.0341e-05 - dense_3_loss: 0.0012 - dense_2_accuracy: 1.0000 - dense_3_acc
Epoch 34/50
71/71 [==============================] - 1s 15ms/step - loss: 8.9691e-04 - dense_2_loss: 3.9908e-06 - dense_3_loss: 8.9292e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4662
Epoch 35/50
71/71 [==============================] - 1s 15ms/step - loss: 8.7675e-04 - dense_2_loss: 3.3144e-06 - dense_3_loss: 8.7344e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4938
Epoch 36/50
71/71 [==============================] - 1s 14ms/step - loss: 9.1929e-04 - dense_2_loss: 4.0339e-06 - dense_3_loss: 9.1525e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.476
4: 0s - loss: 9.3007e-04 - dense_2_loss: 4.9689e-06 - dense_3_loss: 9.2510e-04 - dense_2_accuracy: 1.0000 - dense_3_accu
Epoch 37/50
71/71 [==============================] - 1s 14ms/step - loss: 8.2139e-04 - dense_2_loss: 6.1635e-06 - dense_3_loss: 8.1522e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4689
Epoch 38/50
71/71 [==============================] - 1s 14ms/step - loss: 7.9326e-04 - dense_2_loss: 4.0571e-06 - dense_3_loss: 7.8920e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4742
Epoch 39/50
71/71 [==============================] - 1s 15ms/step - loss: 7.8818e-04 - dense_2_loss: 5.3817e-06 - dense_3_loss: 7.8280e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4796
Epoch 40/50
71/71 [==============================] - 1s 14ms/step - loss: 7.4981e-04 - dense_2_loss: 7.9586e-06 - dense_3_loss: 7.4186e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4711
Epoch 41/50
71/71 [==============================] - 1s 11ms/step - loss: 7.1834e-04 - dense_2_loss: 3.7599e-06 - dense_3_loss: 7.1458e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4756
Epoch 42/50
71/71 [==============================] - 0s 7ms/step - loss: 7.6520e-04 - dense_2_loss: 3.6230e-06 - dense_3_loss: 7.6157e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4644
Epoch 43/50
71/71 [==============================] - 0s 7ms/step - loss: 7.0895e-04 - dense_2_loss: 4.2607e-06 - dense_3_loss: 7.0468e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4742
Epoch 44/50
71/71 [==============================] - 1s 7ms/step - loss: 7.0224e-04 - dense_2_loss: 4.2212e-06 - dense_3_loss: 6.9802e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4538
Epoch 45/50
71/71 [==============================] - 1s 7ms/step - loss: 6.5697e-04 - dense_2_loss: 3.6060e-06 - dense_3_loss: 6.5336e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4676
Epoch 46/50
71/71 [==============================] - 1s 7ms/step - loss: 6.2121e-04 - dense_2_loss: 8.1302e-06 - dense_3_loss: 6.1308e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4644
Epoch 47/50
71/71 [==============================] - 1s 8ms/step - loss: 6.4415e-04 - dense_2_loss: 4.0283e-06 - dense_3_loss: 6.4012e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4560
Epoch 48/50
71/71 [==============================] - 1s 8ms/step - loss: 5.9725e-04 - dense_2_loss: 1.4715e-05 - dense_3_loss: 5.8253e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4716
Epoch 49/50
71/71 [==============================] - 1s 8ms/step - loss: 5.7323e-04 - dense_2_loss: 2.9170e-06 - dense_3_loss: 5.7031e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4427
A: 0s - loss: 2.9727e-04 - dense_2_loss: 4.7721e-06 - dense_3_loss: 2.9250e-04 - dense_2_accuracy: 1.0000 - dense_3_accura
Epoch 50/50
71/71 [==============================] - 1s 8ms/step - loss: 5.7380e-04 - dense_2_loss: 3.1670e-06 - dense_3_loss: 5.7063e-04 - dense_2_accuracy: 1.0000 - dense_3_accuracy: 0.4636
```

In [28]:
```python
#accuracy= model.evaluate(X_test ,[y_test_cat,yn_train_box])
#print ('Model accuracy:' ,accuracy*100, '%')
preds = model.predict(X_test)
print(f'Predicted Values para ELEMENTOS\n\n {preds[0][:3]}\n\nReal Values para ELEMENTOS\n\n {y_test_cat[:3]}\n\n')
print(f'Predicted Values para LOCALIZACION\n\n {preds[1][:3]}\n\nReal Values para LOCALIZACION\n\n {y_test_box[:3]}')
```

```
Predicted Values para ELEMENTOS

 [[4.5513005e-08 1.0000000e+00 4.2069090e-10]
 [6.7167667e-07 9.9999893e-01 2.9842849e-07]
 [9.9999774e-01 2.2734512e-06 4.0574395e-08]]

Real Values para ELEMENTOS

 [[0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]]


Predicted Values para LOCALIZACION

 [[0.27243719 0.24684115 0.27271378 1.2672071  1.2859747  1.2682313
  1.2854664  0.24688779]
 [0.12800676 0.26782715 0.12886131 1.2785552  1.1375482  1.2806842
  1.1369749  0.2678957 ]
 [0.6543788  0.6225803  0.65468943 1.6168582  1.6508658  1.6156328
  1.6514202  0.6226222 ]]

Real Values para LOCALIZACION

 [[0.27286373 0.24494664 0.27286373 1.24494664 1.27286373 1.24494664
  1.27286373 0.24494664]
 [0.1417609  0.28551696 0.1417609  1.28551696 1.1417609  1.28551696
  1.1417609  0.28551696]
 [0.62707206 0.59685115 0.62707206 1.59685084 1.62707144 1.59685084
  1.62707144 0.59685115]]
```

In [29]:
```python
a = np.array(y_test_box[0])
a_r = a.reshape(4,2)
a_r = np.concatenate((a_r,a_r[:1]),axis = 0)
b = preds[1][0]

b_r =b.reshape(4,2)
b_r = np.concatenate((b_r,b_r[:1]),axis = 0)

f = plt.figure()
a = f.add_subplot(111)
a.plot(a_r[:,0],a_r[:,1],label='Example of real')
a.plot(b_r[:,0],b_r[:,1],label= ' Example of prediction')
a.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x2038efb9fa0>



Evaluación & Conclusiones

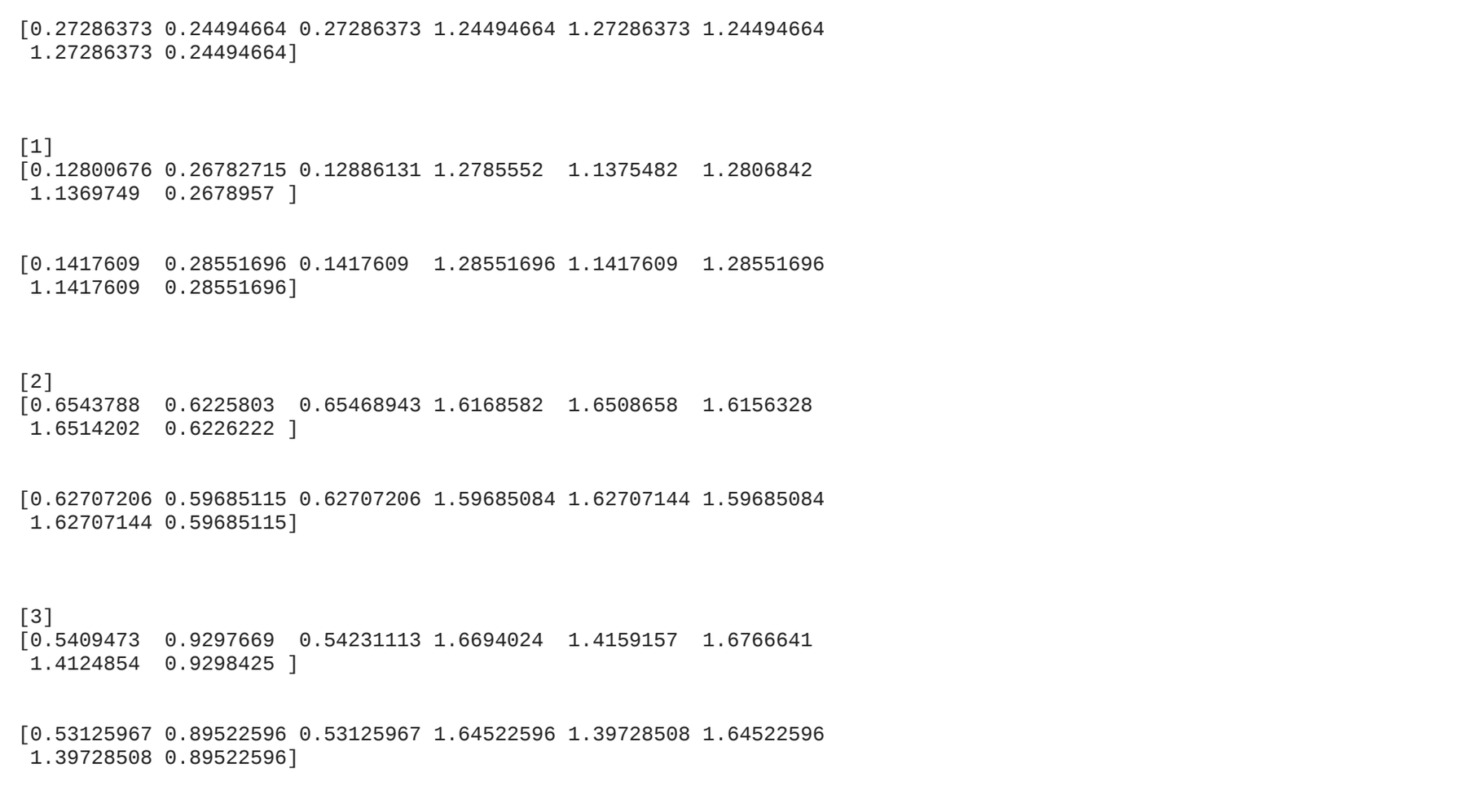
- Evaluación CATEGORÍA

In [30]:
```python
numero_predicciones=10
for i in range (numero_predicciones):
    print(" 0 Círculo 1 Cuadrado 2 Triángulo")
    print(preds[0][i])
    print(y_test_cat[i])
    print (" \n ")
```

```
 0 Círculo 1 Cuadrado 2 Triángulo
[4.5513005e-08 1.0000000e+00 4.2069090e-10]
[0. 1. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[6.716767e-07 9.999989e-01 2.984285e-07]
[0. 1. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[9.9999774e-01 2.2734512e-06 4.0574395e-08]
[1. 0. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[2.2564572e-17 4.9930170e-14 1.0000000e+00]
[0. 0. 1.]


 0 Círculo 1 Cuadrado 2 Triángulo
[9.9999952e-01 2.3124026e-07 2.1719782e-07]
[1. 0. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[3.7930001e-06 9.9999619e-01 1.7345729e-13]
[0. 1. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[1.5350662e-07 2.0383467e-07 9.9999964e-01]
[0. 0. 1.]


 0 Círculo 1 Cuadrado 2 Triángulo
[9.9999917e-01 8.3946725e-07 6.6242296e-09]
[1. 0. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[9.9999666e-01 3.2227038e-06 1.0715558e-07]
[1. 0. 0.]


 0 Círculo 1 Cuadrado 2 Triángulo
[3.6650233e-06 9.9999630e-01 1.3379486e-08]
[0. 1. 0.]
```

- Evaluación Coordenadas

In [31]:
```python
numero_preds=5

fig, axs = plt.subplots(nrows=1, ncols=numero_preds, figsize=(16, 8))

for i in range(numero_preds):

    a = y_test_box[i]
    a_r = a.reshape(4,2)
    a_r = np.concatenate((a_r,a_r[:1]),axis = 0)

    b = preds[1][i]
    b_r =b.reshape(4,2)
    b_r = np.concatenate((b_r,b_r[:1]),axis = 0)

    axs[i].plot(a_r[:,0],a_r[:,1],label='Real Values')
    axs[i].plot(b_r[:,0],b_r[:,1],label= 'Predicted Values')
    axs[i].legend()


plt.show()
```



In [32]:
```python
numero_predicciones=10
for i in range (numero_predicciones):
    print([i])
    print(preds[1][i])
    print("\n")
    print(y_test_box[i])
    print (" \n \n")
```

```
[0]
[0.27243719 0.24684115 0.27271378 1.2672071  1.2859747  1.2682313
 1.2854664  0.24688779]


[0.27286373 0.24494664 0.27286373 1.24494664 1.27286373 1.24494664
 1.27286373 0.24494664]


[1]
[0.12800676 0.26782715 0.12886131 1.2785552  1.1375482  1.2806842
 1.1369749  0.2678957 ]


[0.1417609  0.28551696 0.1417609  1.28551696 1.1417609  1.28551696
 1.1417609  0.28551696]


[2]
[0.6543788  0.6225803  0.65468943 1.6168582  1.6508658  1.6156328
 1.6514202  0.6226222 ]


[0.62707206 0.59685115 0.62707206 1.59685084 1.62707144 1.59685084
 1.62707144 0.59685115]


[3]
[0.5409473  0.9297669  0.54231113 1.6694024  1.4159157  1.6766641
 1.4124854  0.9298425 ]


[0.53125967 0.89522596 0.53125967 1.64522596 1.39728508 1.64522596
 1.39728508 0.89522596]


[4]
[0.47423443 0.5330367  0.47489604 1.5638207  1.4972789  1.5669649
 1.4956738  0.5330636 ]


[0.44999481 0.52084356 0.44999481 1.52084325 1.44999419 1.52084325
 1.44999419 0.52084356]


[5]
[0.43606806 0.6855854  0.43639103 1.6841518  1.4284495  1.6804401
 1.4304649  0.6856572 ]


[0.44107999 0.69903112 0.44107999 1.69903112 1.44107999 1.69903112
 1.44107999 0.69903112]


[6]
[0.40831256 0.4725284  0.40942943 1.2227471  1.2698181  1.2208034
 1.2715188  0.4726202 ]


[0.42634203 0.48163631 0.42634203 1.23163631 1.29236743 1.23163631
 1.29236743 0.48163631]


[7]
[0.03675526 0.34190035 0.0376884  1.334725   1.0317875  1.3342824
 1.0324496  0.34196323]


[0.01786506 0.30380992 0.01786506 1.30380962 1.01786445 1.30380962
 1.01786445 0.30380992]


[8]
[0.492442   0.38707638 0.49271172 1.3311311  1.4521439  1.325202
 1.4553686  0.3871207 ]


[0.49596877 0.39165474 0.49596877 1.39165443 1.49596816 1.39165443
 1.49596816 0.39165474]


[9]
[0.07512698 0.29275241 0.07596272 1.2980518  1.0789282  1.2987959
 1.0790792  0.29259443]


[0.07432167 0.27434435 0.07432167 1.27434435 1.07432167 1.27434435
 1.07432167 0.27434435]
```

In [33]:
```python
##################
```

In [ ]:

In [ ]: