به نام خدا

نام و نام خانوادگی :محمدرضا همتی مقدم
شماره دانشجویی:99222129
تمرین :3

طبقه بندی دیتاست Architectural Heritage Elements

این تمرین در 2 بخش انجام شده است؛ بخش اول، train کردن شبکه عصبی کانولوشنی و بخش دوم، Deconvolution همان داده ها. لذا پیاده سازی این تمرین در 2 اسکریپت جداگانه تحت عنوان CNN.ipynb و Deconvolution.ipynb نوشته شده است. مراحل ابتدایی نظیر فراخوانی کتابخوانه ها و فراخوانی داده ها مشترک بوده و به صورت زیر خواهد بود:

- در مرحله نخست، کتابخانه های مورد نیاز را فراخوانی میکنیم:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,Dropout,
Dense, Conv2DTranspose
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
```

- در مرحله بعد، دیتاست مورد نیاز را که یک دیتاست آماده تحت عنوان Architectural Heritage Elements میباشد دانلود کرده و با استفاده از توابع پایتون و کتابخانه numpy، داده های train و test را جدا خواهیم کرد.

In [2]:
```
folder = 'train'
images = []
y_idx = []
k = 0
for i in os.listdir(folder):
    for filename in os.listdir(os.path.join(folder,i)):
        img = cv2.imread(os.path.join(os.path.join(folder,i),filename))
        if img is not None:
            images.append(img)
            y_idx.append(k)
    k = k + 1


n_train = len(images)
x_train = np.zeros((n_train, 64, 64, 3))
y_train = np.zeros((n_train, 1))

for i in range(n_train):
    x_train[i,:,:,:] = images[i]
    y_train[i] = y_idx[i]
```

در کد بالا، به ترتیب به داخل زیرپوشه های فولدر train رفته و تصاویر را فراخوانی و در لیست images اضافه میکنیم. در ادامه، توسط یک حلقه تکرار، تصاویر را از داخل لیست به یک آرایه پایتونی انتقال خواهیم داد. مشابه روند گفته شده را برای داده های test نیز انجام خواهیم داد.

```
In [4]:
num_class = 10
x_train = x_train.astype('float64') # change integers to 64-bit
#floating point numbers
x_test = x_test.astype('float64')
x_train /= 255 # normalize the input
x_test /= 255

y_train = to_categorical(y_train, num_class)
y_test = to_categorical(y_test, num_class)
```

این مرحله، پیش پردازش داده ها میباشد. در این مرحله تایپ داده هارا به صورت float تغییر میدهیم تا بتوان مشابه اعداد با آنها رفتار کرد. همچنین تمامی اعداد را نرمالایز خواهیم کرد. یعنی آن ها را طوری تغییر میدهیم که در بازه 0 و 1 قرار گیرند.
در ادامه، لازم است تا مقادیر لیبل هارا نیز به نوعی نرمالایز کنیم. یعنی به جای اینکه هر عکس دارای لیبلی بین 0 تا 9 باشد، به هر عکس یک بردار 10 تایی به عنوان لیبل نسبت دهیم که همه درایه های آن صفر بوده بجز درایه ای که معرف شماره کلاس است. به عنوان مثال اگر عدد داخل تصویر عدد 3 باشد، یک بردار 10 تایی خواهیم داشت که درایه سوم آن 1 بوده و سایر درایه های آن صفر میباشد
در ادامه، سایز داده ها و لیبل هایشان و همچنین نوع آنهارا مشاهده خواهیم کرد:

```
In [5]:
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)

x_train shape: (10130, 64, 64, 3)
y_train shape: (10130, 10)
x_test shape: (1404, 64, 64, 3)
y_test shape: (1404, 10)
```

همانطور که مشاهده میشود، تعداد 10130 عکس با ابعاد 64 در 64 پیکسل برای آموزش مدل و همچنین 1404 عکس با همان ابعاد برای آزمایش مدل از جنس آرایه خواهیم داشت. به ازای تمامی این عکس ها، لیبل آنها که برابر است با یک عدد بین 0 تا 9 موجود خواهد بود. درواقع درکل 11534 عکس خواهیم داشت که هریک، لیبل متناظر با خود را دارد.
تمامی مراحل بالا، برای هر دو اسکریپت مشابه خواهد بود. در ادامه، به بررسی جداگانه اهداف تمرین خواهیم پرداخت:

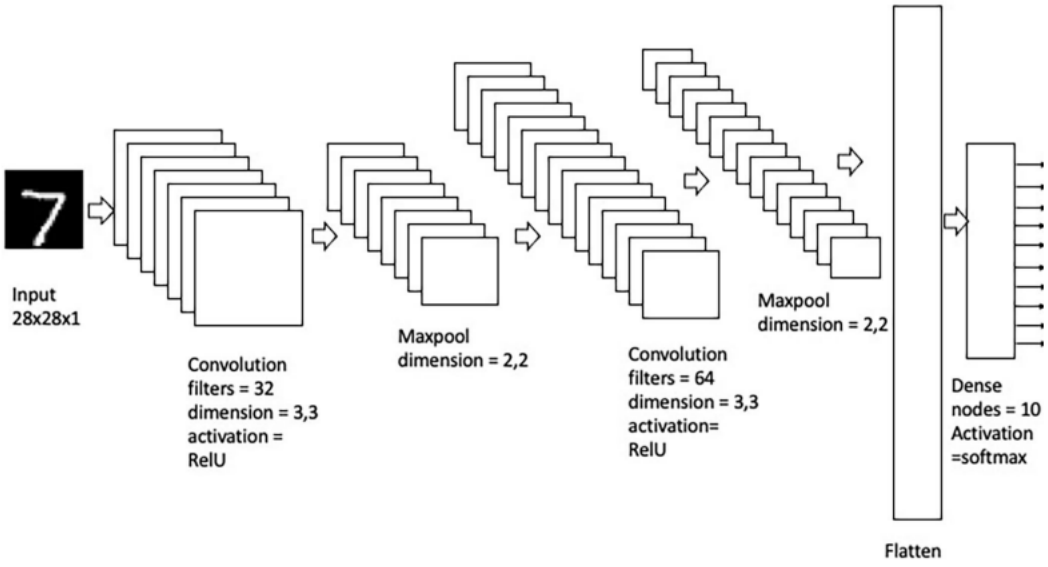
- **CNN**

در بخش اول، باید یک مدل ساخته و لایه های مورد نیاز را با آن اضافه کنیم:

از پکیج تنسورفلو و کتابخانه کراس، مدل Sequential را فراخوانی میکنیم که یک ترکیب خطی از لایه های کراس است. با توجه به شکل زیر، یک لایه ورودی، دو لایه پنهان از معماری Dense با ابعاد مختلف و یک لایه خروجی خواهیم داشت که به کمک دستورات زیر ساخته خواهد شد:

```
model = Sequential()
```

در اولین نگاه، مزیت این آن است که نیازی نیست برای ورود داده ها به مدل، لزومی ندارد صرفا از آرایه یک بعدی استفاده شود و میتوان خود تصاویر را به عنوان ورودی به مدل داد. درواقع بجای آنکه لایه پنهان با ورودی ای با سایز 12288 ست شود، میتوان لایه را با ورودی ای با سایز 64 در 64 در 3 تنظیم کرد. شکل زیر را برای طراحی شبکه عصبی پیچشی درنظر بگیرید:



شکل 2: لایه های تشکیل دهنده شبکه عصبی پیچشی (CNN)

درواقع ترتیب لایه ها در شبکه عصبی پیچشی به صورت زیر خواهد بود:

1. لایه ورودی که تصاویری با ابعاد 64 در 64 در 3 هستند
2. لایه کانولوشنی 3 در 3 که با 32 کانال. این کرنل 3 در 3 در هر مرحله روی تصویر کانوالو میشود
3. لایه maxpool که 2 در 2 بوده و وظیفه ادغام را دارد
4. مجددا لایه کانولوشنی این بار با 64 کانال
5. مجددا لایه maxpool
6. لایه Flatten که وظیفه تبدیل لایه pooling را به بردار 1 بعدی دارد
7. لایه dense خروجی با 10 گره

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape = (64,64,3)))
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(Flatten())
model.add(Dropout(0.5))
model.add(Dense(num_class, activation='softmax'))
```

خروجی کدهای بالا، مدل کانولوشنی ساخته شده به صورت زیر خواهد بود:

```
In [7]: model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten (Flatten)           (None, 12544)             0

 dropout (Dropout)           (None, 12544)             0

 dense (Dense)               (None, 10)                125450

=================================================================
Total params: 144842 (565.79 KB)
Trainable params: 144842 (565.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

مدلی با 144842 پارامتر ساخته شده است. هر مدل یادگیری عمیق، یک تابع هدف داشته و یک برآوردگر که برآوردگر یا بهینه کننده، بدنبال بهینه سازی تابع هدف میباشد. به عنوان مثال در کد زیر، بهینه گر Adam به دنبال مینیمم کردن تابع loss بوده که نوع categorical_crossentropy میباشد.

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

پس از طراحی مدل، نوبت به برازش مدل به داده های آموزشی میباشد. برای این منظور از کد زیر استفاده میکنیم:

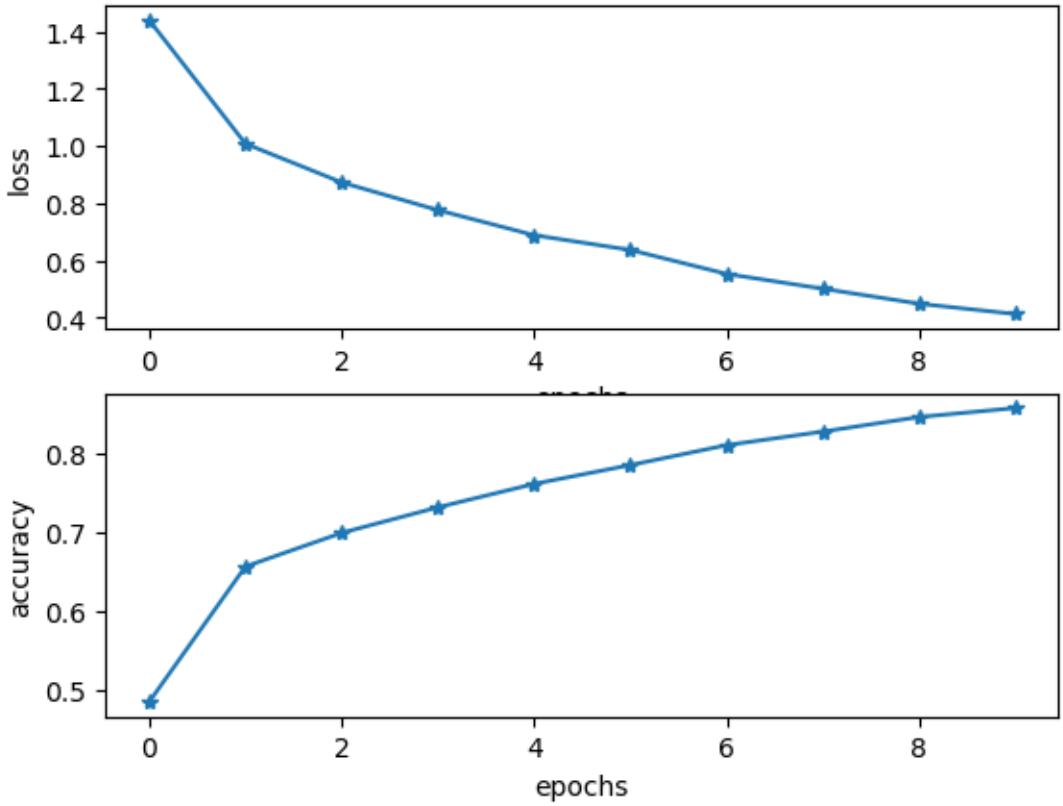
model.fit(X_train, y_train, batch_size=128, epochs=10)

ورودی اول، داده های آموزشی، ورودی دوم لیبل متناظر هر عکس، ورودی سوم مقدار batch size بوده که بیانگر تعداد تصاویری است که در هر اپوک وارد فاز آموزش میشود. ورودی سوم تعداد اپوک هایی است که داده ها پردازش میشوند. داده های ما از انجایی که از یک دیتاست تمیز و نسبتا مناسب استفاده شده، نیازی به تعداد اپوک بالا برای پردازش نخواهند داشت.

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

```
In [9]: history = model.fit(x_train, y_train, batch_size=64, epochs=10)

Epoch 1/10
WARNING:tensorflow:From E:\Software\Anaconda\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTen
sorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From E:\Software\Anaconda\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executi
ng_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

159/159 [==============================] - 17s 96ms/step - loss: 1.4393 - accuracy: 0.4839
Epoch 2/10
159/159 [==============================] - 15s 95ms/step - loss: 1.0078 - accuracy: 0.6563
Epoch 3/10
159/159 [==============================] - 15s 97ms/step - loss: 0.8725 - accuracy: 0.6991
Epoch 4/10
159/159 [==============================] - 16s 98ms/step - loss: 0.7756 - accuracy: 0.7317
Epoch 5/10
159/159 [==============================] - 16s 100ms/step - loss: 0.6880 - accuracy: 0.7615
Epoch 6/10
159/159 [==============================] - 16s 99ms/step - loss: 0.6364 - accuracy: 0.7855
Epoch 7/10
159/159 [==============================] - 16s 99ms/step - loss: 0.5526 - accuracy: 0.8109
Epoch 8/10
159/159 [==============================] - 16s 100ms/step - loss: 0.5010 - accuracy: 0.8281
Epoch 9/10
159/159 [==============================] - 16s 101ms/step - loss: 0.4486 - accuracy: 0.8467
Epoch 10/10
159/159 [==============================] - 16s 100ms/step - loss: 0.4125 - accuracy: 0.8581
```

پس از برازش مدل، همانند مدل قبلی، میتوان ارزیابی را نیز انجام داد. اما پیش از آن، روند کاهش loss و افزایش Accuracy را در طی 10 اپوک در نمودارهای زیر شاهد خواهیم بود:



شکل: نمودار های Accuracy و Loss

در نهایت به ارزیابی مدل خواهیم پرداخت:

```
score = model.evaluate(x_test, y_test)
print('loss on test data: ', score[0])
print('accuracy on test data:', score[1])
```

```
44/44 [==============================] - 1s 14ms/step - loss: 0.8904 -
accuracy: 0.7023
loss on test data:  0.8903890252113342
accuracy on test data: 0.7022792100906372
```

- **Deconvolution**

پس از روش های رایج درونیابی نظیر Bilinear و Bicubic که علیرغم سریع بودن، قابلیت انعطاف مناسبی نداشتند، در این تمرین بدنبال روش deconvolution خواهیم بود.

برای پیاده سازی این الگورتیم، لازم است تا 2 کار زیر صورت گیرد:

1. در مدلی که قرار است ساخته شود از لایه های دی-کانولوشنی نیز استفاده شود
2. در آموزش مدل، به عنوان تصاویر و لیبل شان از خود تصاویر استفاده شود

In [6]:
```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', kernel_initializer='he_normal', input_shape=(64,64,3)))
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu', kernel_initializer='he_normal'))
model.add(Conv2D(8, kernel_size=(3, 3), activation='relu', kernel_initializer='he_normal'))
model.add(Conv2DTranspose(8, kernel_size=(3,3), activation='relu', kernel_initializer='he_normal'))
model.add(Conv2DTranspose(16, kernel_size=(3,3), activation='relu', kernel_initializer='he_normal'))
model.add(Conv2DTranspose(32, kernel_size=(3,3), activation='relu', kernel_initializer='he_normal'))
model.add(Conv2D(3, kernel_size=(3, 3), activation='sigmoid', padding='same'))
```

```
WARNING:tensorflow:From E:\Software\Anaconda\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is de
precated. Please use tf.compat.v1.get_default_graph instead.
```

همانطور که در تصویر بالا مشخص است، پس از لایه های کانولوشنی، لایه های معکوس کانولوشنی نیز اضافه شده اند تا بتوان توسط آنها، به ورودی رسید.

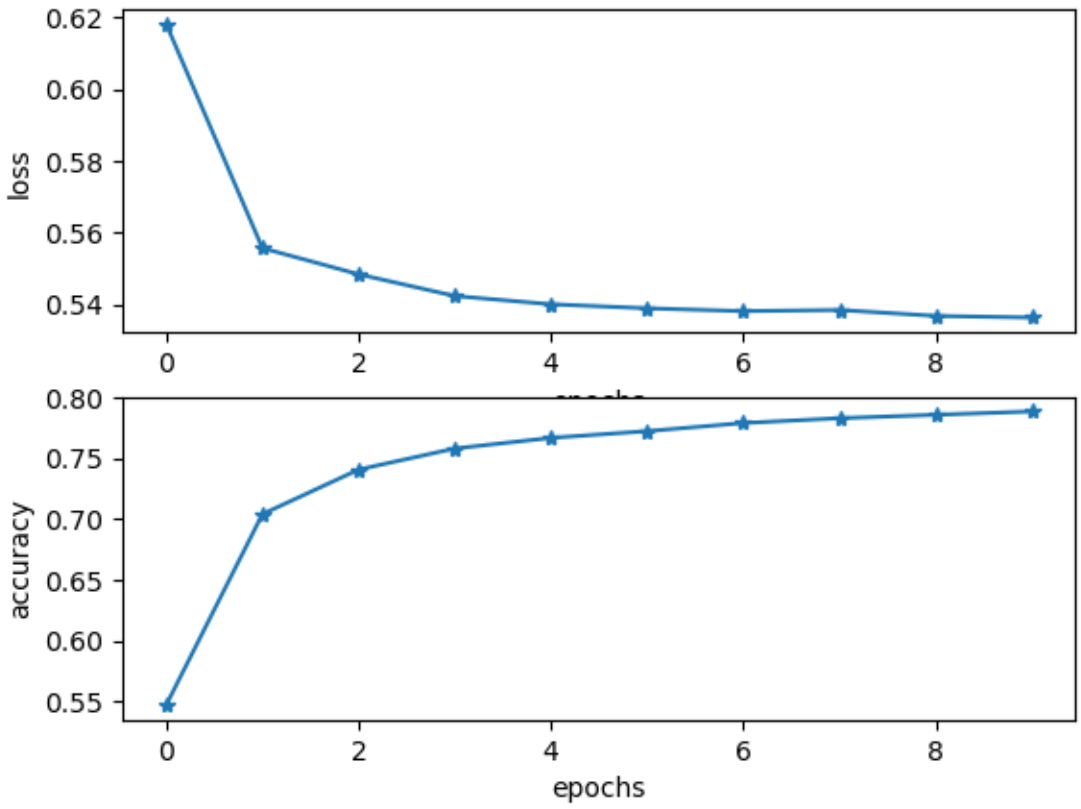
In [7]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 conv2d_1 (Conv2D)           (None, 60, 60, 16)        4624

 conv2d_2 (Conv2D)           (None, 58, 58, 8)         1160

 conv2d_transpose (Conv2DTr  (None, 60, 60, 8)         584
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 62, 62, 16)        1168
 Transpose)

 conv2d_transpose_2 (Conv2D  (None, 64, 64, 32)        4640
 Transpose)

 conv2d_3 (Conv2D)           (None, 64, 64, 3)         867

=================================================================
Total params: 13939 (54.45 KB)
Trainable params: 13939 (54.45 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

مدل جدید مدلی با 13939 پارامتر میباشد.

```
In [9]: history = model.fit(x_train, x_train,
                            epochs=10,
                            batch_size=128,
                            validation_split=0.2)

Epoch 1/10
WARNING:tensorflow:From E:\Software\Anaconda\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTen
sorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From E:\Software\Anaconda\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executi
ng_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

64/64 [==============================] - 41s 590ms/step - loss: 0.6182 - accuracy: 0.5472 - val_loss: 0.5180 - val_accuracy:
0.6263
Epoch 2/10
64/64 [==============================] - 36s 569ms/step - loss: 0.5557 - accuracy: 0.7041 - val_loss: 0.4941 - val_accuracy:
0.7014
Epoch 3/10
64/64 [==============================] - 35s 547ms/step - loss: 0.5484 - accuracy: 0.7408 - val_loss: 0.4883 - val_accuracy:
0.7244
Epoch 4/10
64/64 [==============================] - 35s 551ms/step - loss: 0.5423 - accuracy: 0.7581 - val_loss: 0.4827 - val_accuracy:
0.7327
Epoch 5/10
64/64 [==============================] - 40s 633ms/step - loss: 0.5400 - accuracy: 0.7669 - val_loss: 0.4808 - val_accuracy:
0.7435
Epoch 6/10
64/64 [==============================] - 36s 563ms/step - loss: 0.5389 - accuracy: 0.7724 - val_loss: 0.4787 - val_accuracy:
0.7531
Epoch 7/10
64/64 [==============================] - 36s 562ms/step - loss: 0.5381 - accuracy: 0.7791 - val_loss: 0.4776 - val_accuracy:
0.7667
Epoch 8/10
64/64 [==============================] - 36s 561ms/step - loss: 0.5384 - accuracy: 0.7830 - val_loss: 0.4778 - val_accuracy:
0.7691
Epoch 9/10
64/64 [==============================] - 36s 559ms/step - loss: 0.5367 - accuracy: 0.7858 - val_loss: 0.4762 - val_accuracy:
```

در ادامه، نمودار دقت و loss نیز برای این مدل آورده شده است:



شکل: نمودار های Accuracy و Loss

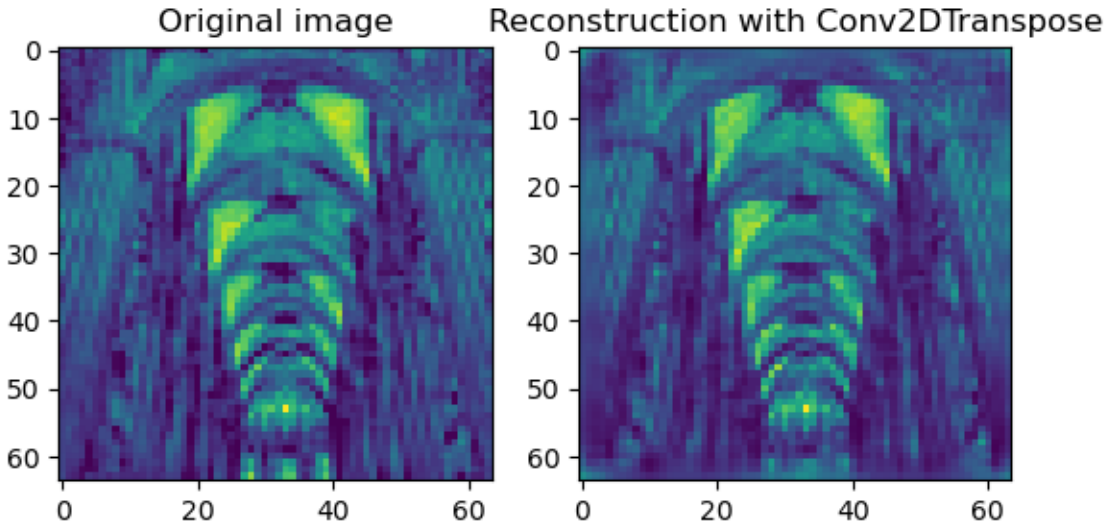
در ادامه، 4 عکس از داخل دیتاست test به صورت رندوم انتخاب کرده و قصد پیش بینی آنرا توسط مدل ساخته شده داریم. درواقع قصد داریم ببینیم که مدلی که در آن از لایه های Conv2DTranspose استفاده شده، میتواند به خوبی تصویر ورودی را حدس بزند یا خیر.

```
In [20]:
# Generate reconstructions
idx = []
num_reconstructions = 4
for i in range(num_reconstructions):
    idx.append(np.random.randint(len(x_test)))

samples = x_test[idx,:,:,:]
targets = y_test[idx,:]
reconstructions = model.predict(samples)

1/1 [==============================] - 0s 34ms/step
```

انتخاب 4 عکس رندوم و پیش بینی آن توسط مدل ساخته شده.

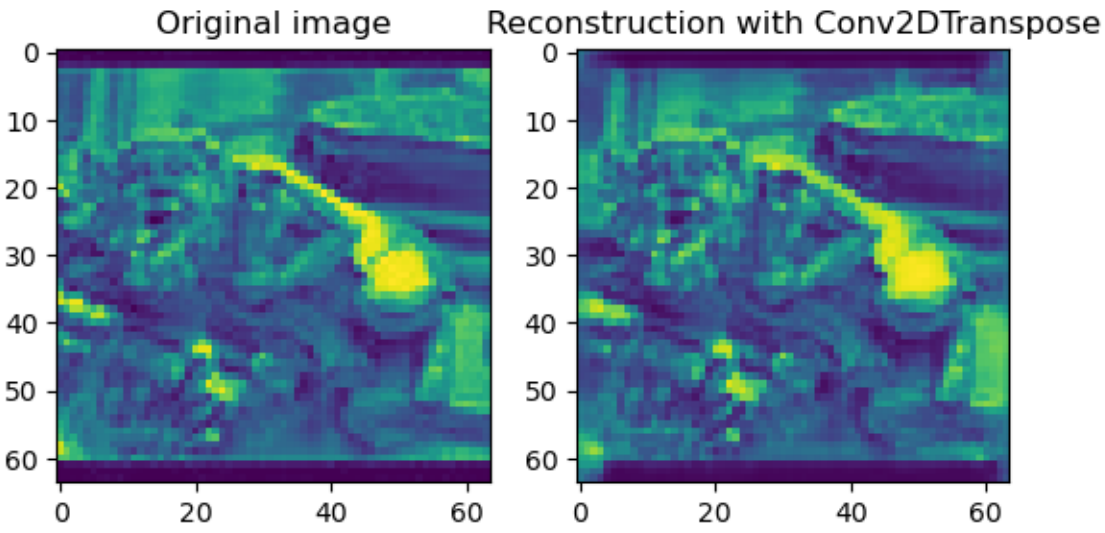
در ادامه، نتایج را مشاهده خواهیم کرد:

AHE target = [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]



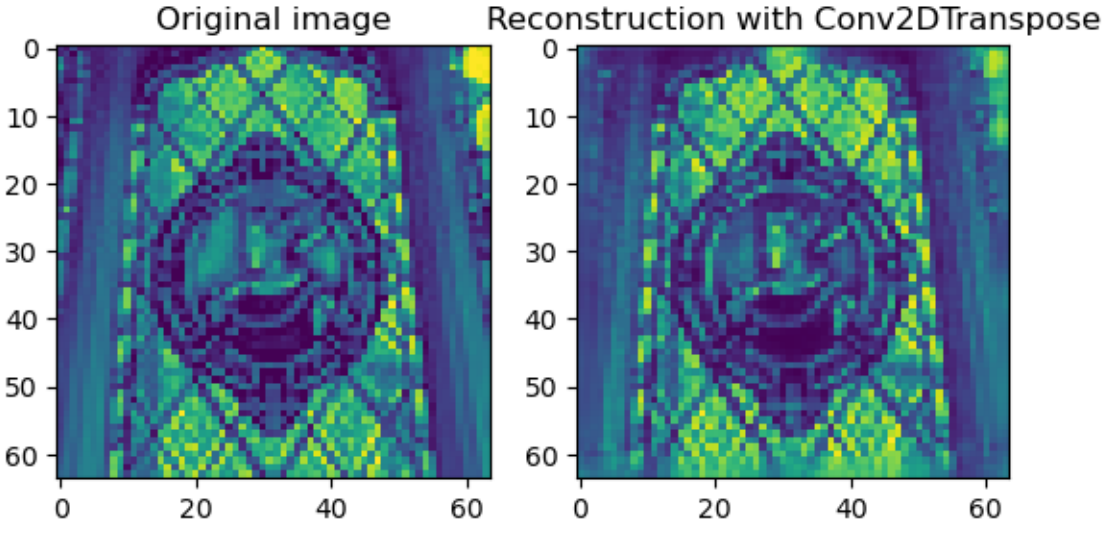
عکس اول

AHE target = [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]



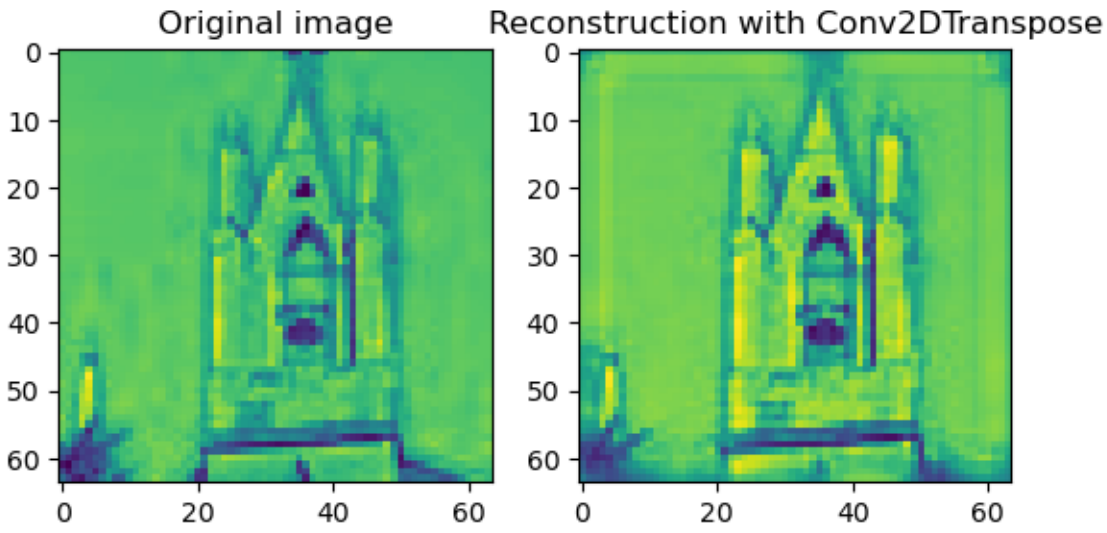
عکس دوم

AHE target = [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]



عکس سوم

AHE target = [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]



عکس چهارم

همانطور که مشخص است، تمامی خروجی ها نسبت به داده اصلی، از شباهت حداکثری برخوردار هستند