
بسم الله الرحمن الرحيم

موضوع پروژه:

پردازش تصاویر دیتاست هدی

گردآورنده:

سبحان قنبری

مقدمه:

دیتاست هدی لیستی از تصاویر اعداد فارسیست که از روی اعداد دستنویس نوشته شده توسط شرکت کننده های کنکور ثبت شده است.

این لیست شامل 60000 عکس با سایزهای مختلف است.

چالشی که برای استفاده از CNN با آن مواجه می شویم یکسان نبودن سایز تصاویر است.

در ابتدا باید سایز تصاویر را یکسان کنیم و بعد لیست را به داده های تنسور تبدیل کنیم.

برای استفاده از انکدر و دیکدر به عکس ها نویز استفاده کردیم که بتوان از آن استفاده کرد

Setup:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Model
```

Dataset:

```
from HodaDatasetReader import read_hoda_cdb
from HodaDatasetReader import read_hoda_dataset
```

دیتاست هدی را میتوان از سایت های مختلف دانلود کرد

Reading Dataset:

```
(train_data,_)=read_hoda_cdb('D:\python\HodaDatasetReader-master\DigitDB\Train
60000.cdb')
(test_data,_)=read_hoda_cdb('D:\python\HodaDatasetReader-master\DigitDB/Test
20000.cdb')
```

دیتاستی که دانلود کردیم می توانیم از مسیر دانلود تعریف کنیم. در این قسمت نیازی به برچسب عکس ها نداریم

Convert to same size:

```
import cv2

def pad_images(images, padding_value=0):
    rows = [img.shape[0] for img in images]
    cols = [img.shape[1] for img in images]
    max_rows, max_cols = max(rows), max(cols)

    padded_images = []
    for img in images:
        new_img = np.ones((max_rows+2, max_cols+5), dtype=img.dtype) *
padding_value
        new_img[:img.shape[0], :img.shape[1],] = img
        padded_images.append(new_img)



    return padded_images

pad_image = pad_images(train_data)
```

```
def pad_test(images, padding_value=0):
    rows = [img.shape[0] for img in images]
    cols = [img.shape[1] for img in images]
    max_rows, max_cols = max(rows), max(cols)

    padded_images = []
    for img in images:
        new_img = np.ones((max_rows, max_cols+2), dtype=img.dtype) *
padding_value
        new_img[:img.shape[0], :img.shape[1],] = img
        padded_images.append(new_img)



    return padded_images
pad_test = pad_tests(test_data)
```

این دو تابع لیستی از تصاویر را با یک مقدار padding مشخص شده ایجاد می کنند. این تابع ها از کتابخانه NumPy برای ایجاد تصاویر جدید با اندازه بزرگ ترین تصویر در لیست استفاده می کند و در اطراف لبه ها قرار می دهد. مقدار padding به عنوان یک آرگومان مشخص می شود و در صورت عدم ارائه مقدار، به صورت پیش فرض 0 می شود.

در ابتدا بر روی ردیف و ستون پیمایش می کند و برای بزرگترین مقدار ردیف و ستون درون لیست مقدار تعریف می کند و سپس با اضافه کردن مقدار مشخص سایز عکس را تغییر می دهد

Processing:

```
def preprocess(array):
   #normalize the supplied array and reshape it
   array= np.array(array)
   array= array.astype("float64") / 255.0
   return array

processed_images=preprocess(pad_image)
processed_test=preprocess(pad_test)
```

تابع preprocess یک آرایه را به عنوان ورودی می گیرد و دو عملیات را انجام می دهد: ابتدا آرایه را به یک آرایه NumPy تبدیل می کند، سپس با تقسیم هر عنصر بر 255.0 مقادیر را ساده می کند. در نهایت، آرایه پردازش شده را برمی گرداند.این هدف برای پیش پردازش و آماده سازی تصاویر برای یادگیری ماشین است.

```
print('x_train is {}'.format(processed_images.shape))
print('x_test is {}'.format(processed_test.shape))
```

```
x_train is (60000, 64, 56)
x_test is (20000, 64, 56)
```

Add noise:

```
def add_noise(array):
    #add random noise to each image in the supplied array

    noise_factor = 0.4
    noisy_array = array + noise_factor * np.random.normal(
        loc=0.0, scale=1.0, size=array.shape
    )
    return np.clip(noisy_array, 0.0, 1.0)
```

گاهی اوقات در تصاویری مانند تصاویر پزشکی نویز به وجود میاید. در این بخش به تصاویر نویز اضافه می کنیم تا بتوانیم با استفاده از autoencoder رفع نویز کنیم.

در ابعاد تصاویر و با ضریب رندوم به تصاویر نویز اضافه کردیم

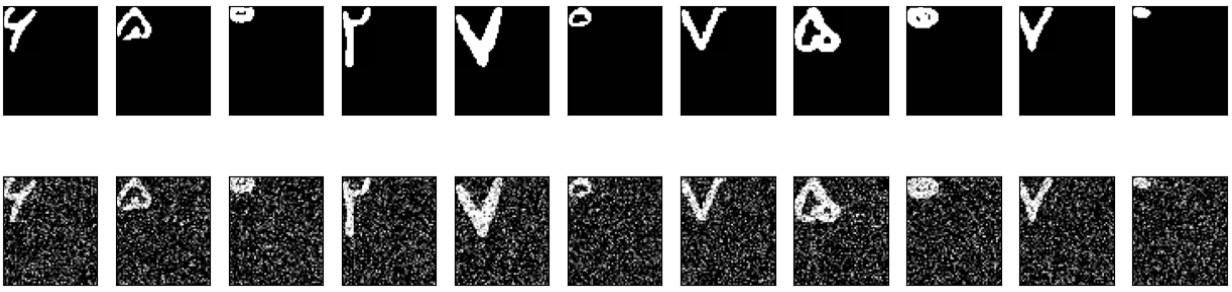
نویز اضافه شده به هر تصویر توسط متغیر noise_factor تعیین می شود که به طور پیش فرض روی 0.4 تنظیم شده است. سپس از تابع np.clip برای اطمینان از اینکه مقادیر پیکسل تصاویر نویزدار حاصل در محدوده 0.0 تا 1.0 هستند استفاده می شود.

Display images:

```
def Display(array1, array2):
#Display 10 random images
n = 14
images1 = array1[0:11]
images2 = array2[0:11]

plt.figure(figsize=(20, 4))
for i, (image1, image2) in enumerate(zip(images1, images2)):
    ax = plt.subplot(2,n , i + 1)
    plt.imshow(image1, cmap="gray")
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(image2, cmap="gray")
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show
```

یک تابع تعریف می کنیم که دو ورودی دریافت می کند و یازده ورودی اول را به نمایش می گذارد.یک کادر با عرض 20 و ارتفاع 4 در نظر میگیریم. در ادامه دو آرایه ورودی را دو به دو با هم ترکیب می کنیم و به صورت زیرهم نمایش می دهیم و به تصاویر رنگ gray می دهیم.

```
noisy_train_data =[add_noise(img) for img in processed_images]

noisy_test_data = [add_noise(img) for img in processed_test]


Display(processed_images, noisy_train_data)
```



برای اطمینان از یک سایز بودن تصاویر میتوانیم از کد زیر استفاده کنیم:

```
uv_row= set()
uv_col= set()
for i in range(len(processed_test)):
    uv_row.add(processed_test[i].shape[0])
    uv_col.add(processed_test[i].shape[1])
dist_row = dict()
dist_col = dict()
for xr in uv_row:
    dist_row[xr] = 0
    for xc in uv_col:
        dist_col[xc] = 0
for i in range(len(processed_images)):
    dist_row[processed_images[i].shape[0]]+=1
    dist_col[processed_images[i].shape[1]]+=1
print(dist_row, end="-")
print(dist_col, end="-")
```

output: `{64: 60000}-{56: 60000}-`

Model:

```
input = layers.Input(shape=(64, 56, 1))

#Encoder
x = layers.Conv2D(32,(3,3), activation="relu", padding="same")(input)
x = layers.MaxPooling2D((2,2), padding="same")(x)
x = layers.Conv2D(32,(3,3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2,2), padding="same")(x)
x = layers.Conv2D(32,(3,3), activation="relu", padding="same")(x)
```

```
#Decoder
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

#Autoencoder
autoencoder = Model(input, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

ورودی را مطابق با تصاویر64*56 انتخاب می کنیم.در قسمت انکدر در لایه اول کانولوشن دو بعدی قرار می دهیم. تعداد نورون هارا 32 می گذاریم و kernel 3*3 انتخاب می کنیم. در لایه دوم فقط کاهش ابعاد با MaxPooling انجام می دهیم و به همین ترتیب ادامه می دهیم. تابع فعال ساز را relu انتخاب می کنیم به علت ساده سازی ای که در قسمت پردازش انجام دادیم و ارایه هارا بین 0 و 1 قرار دادیم.تابع فعال ساز relu اعداد را بین صفر و یک برمیگرداند. در لایه اخر یک نورون قرار میدهیم و تابع فعال ساز را sigmoid می گذاریم.در قسمت دیکدر به هین صورت بدون نیاز با کاهش ابعاد عملیات کانولوشن را به صورت برعکس انجام می دهیم

Output:

```
Layer (type)                 Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 64, 56, 1)]       0

 conv2d_8 (Conv2D)           (None, 64, 56, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 32, 28, 32)       0
 2D)
```

```
 conv2d_9 (Conv2D)           (None, 32, 28, 32)        9248

 max_pooling2d_5 (MaxPooling  (None, 16, 14, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 14, 32)        9248

 conv2d_transpose_4 (Conv2DT  (None, 32, 28, 32)       9248
 ranspose)

 conv2d_transpose_5 (Conv2DT  (None, 64, 56, 32)       9248
 ranspose)

 conv2d_11 (Conv2D)          (None, 64, 56, 1)         289

=================================================================
Total params: 37,601
Trainable params: 37,601
Non-trainable params: 0
```

نتیجه گیری:

بیشتر مواقع برای مدلسازی چالش هایی وجود دارد که باید ابتدا آن هارا رفع کرد.سپس تعداد لایه ها,نورون ها و تابع فعال ساز را به گونه ای انتخاب می کنیم که دچار overfiting نشویم