

Conv3d

tf.nn.conv3d(input, filter, strides, padding, name=None)

input.Shape [batch, in-depth, in-height,
in-width, in-channels] 输入图片维度
RGB : 3
代表一个 sample 输入 n 个帧每帧一个图.

filter.Shape [filter-depth, filter-height, filter-width]

strides.Shape [stride-batch, strides-depth,
strides-height, strides-width]

Pooling

tf.nn.max_pool3d(input, ksize, strides, padding)

[LogisticRegression]

0.6499 (50)

0.7482 (100)

0.832 (1000)

0.847 (5000)

Conv2D

input-shape=(3, 128, 128)

代表 128×128 的彩色RGB图像。

Conv3D

input-shape=(3, 10, 128, 128)

代表 10帧 128×128 的彩色RGB图像。

3D CNN Keras

```
model.add(Convolution3D(10,  
                         kernel_dim1=9,  
                         kernel_dim2=9,  
                         kernel_dim3=9,  
                         input_shape=~,  
                         activation='relu'))
```

```
model.add(MaxPooling3D(pool_size=(2,2,2)))
```

如何将输入数据转化为符合3D卷积
的输入形式？

1. 获取图片中所有的文件名，用 `os.listdir()`
 函数 -

`filenames = os.listdir(image_path)`

2. 获取图片文件

使用 `cv2.imread()` 函数进行读取，
得到的结果是类似于 $(128, 128, 3)$ 的数组。

`im1 = cv2.imread('C:/01.jpg')`

`im2 = cv2.imread('C:/02.jpg')`

3. 使用 `np.reshape()` 改变数组的维度，改为
 $(1, 128, 128, 3)$ 。

`im1 = im1.reshape((1, 128, 128, 3))`

`im2 = im2.reshape((1, 128, 128, 3))`

4. 使用 `np.concatenate()` 函数进行合并。
`im1, im2` 表示要合并的 array, `axis` 表示要合并
的维度。

`im10 = np.concatenate((im1, im2), axis=0)`

5. 每个方法，两张图片 都将得到类似于

(10, 128, 128, 3) 的 4D 张量，这时，只需要再进行一次 reshape 和 concatenate 即可得到 5D 张量。

$imw10_1 = \text{reshape}(1, 10, 128, 128, 3)$

$imw10_2 = \text{reshape}(1, 10, 128, 128, 3)$

$imw = \text{np}.\text{concatenate}([imw10_1, imw10_2], axis=0)$

train-set (n-samples, 1, img-rows, img-cols, img-depths)

$\text{train-set}[h][0][\underbrace{:}[:][\underbrace{:}[:][\underbrace{:}[:]] = X\text{-train}[h, \underbrace{:}[:][\underbrace{:}[:][\underbrace{:}[:]$

$h \times 28 \times 28 \times 10$

对于 $X\text{-train}$:

$[[[\underbrace{\dots [^0]}_{28}, [\underbrace{\dots [^1]}_{28}, \dots [\underbrace{\dots [^9]}_{28}]] [\underbrace{:}[:][\underbrace{:}[:]\dots [\underbrace{:}[:]\dots [\underbrace{:}[:]]]$

Label: 作为每次输入 10 帧，这 10 帧的
label 会是一样的，所以是每行图像块一个 label.

2D CNN 与 3D CNN 在 数据预处理 有本质区别。

2DCNN：先打乱图片顺序

3DCNN：要在模型实现阶段再打乱图片顺序。

在分批时

process: (从 test-labels 为例)

$$b[0] = 88$$

① $\text{label}[0 : 8] = 0$

—— start = $b[0] = 88$

② $b[1] = 90$

start = $b[0] = 88$

end = start + $b[1] = 88 + 90 = 178$

③ $\text{label}[88, 177] = 1$



TRAINING DATASET = 200,000

TESTING DATASET = 8,733

① 2D CNN on notMNIST dataset :

初始 epoch 就可达到 86.59% accuracy.

在第 50 个 epoch 时 达到 稳定, $\text{max acc} = 93.53\%$

训练速度十分快，每个 epoch 耗时 140S 左右。

第 6 个 epoch 后，准确率稳定在 90.06% 以上并稳步上升。

② 3D CNN on notMNIST dataset:

初始 epoch 较低，为 69.60%

训练速度缓慢，前 5 个 epoch 每个耗 1100S ~ 1700S 左右。

从第 6 个 epoch 开始，训练速度大幅提升为 360S。

准确率上升也缓慢，如第 5 个 epoch 时， $acc = 84.14\%$ 。

从 $5 \rightarrow 6$ (epoch)， $acc : 84.14\% \rightarrow 84.58\%$ (上升如此之低)

让人有些怀疑其最终结果如何。⑥

▲ 现在所期盼的是 3D CNN 的 max acc 可以超过 93.35%。

否则这种架构的 3D CNN 的效果显然要差于 2D 的。

在 TRAINING 上 acc 每次仅增 2%.....



① 3D CNN 效果好的模型构架是什么样的？

② 或者还是说像 notMNIST 这样的数据集并不适合使用 3D CNN 来建模？

【A】② 据我分析，notMNIST 数据集与视频帧数据集

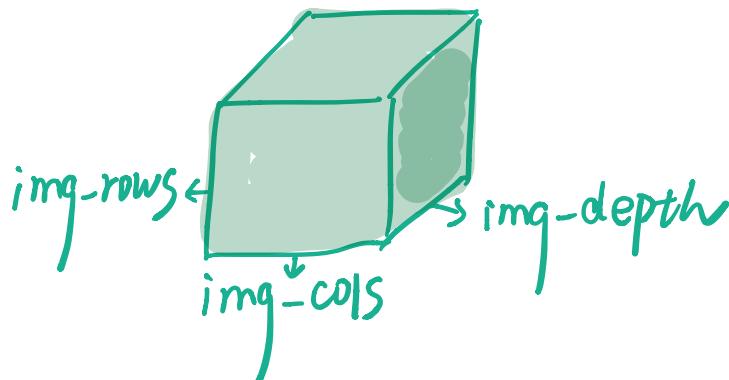
是不同的，1)前者每张图片之间并无关联，所以在此所选用的输入张量为

INPUT-SHAPE = (1, img-cols, img-rows, img-depth)

其中，代码中 $img\text{-}cols = img\text{-}rows = 28$, $img\text{-}depth \boxed{= 1}$

△ ATTENTION: $img\text{-}depth = 1$ 意味着每次输入帧数为1. 故其实从本质上讲，输入的与2D的输入是一样的。

2)后者因为同一个视频中的所有帧数都具有相同的标签，所以才可以使用几帧连续输入的形式，例如，CASME II 中三帧连输入等。



BUT 我曾试验将三帧或是十帧具有相同 label 的图片（例如，从 A 文件夹中依次读取3或10张图）然后叠成一个立方体，并赋予这个立方体一个标签0，但是效果却十分不理想，acc 仅为 10.35%，且并不能看出有丝毫上升的趋势。

ANALYSIS 因为每次使用的 image 都不同，所以提取出的特征也是不同的，于是这导致

很难使得 TEST 集中的数据无法成功匹配到 TRAIN 集中的 label.

故我选择使用 frame=1 来实验.

从第 7 个 epoch 开始，训练速率稳定在 $9 \frac{\text{min}}{\text{epoch}} = 540\text{s}$ 左右，每次在 TRAIN 集上的 acc 保持 $2\% \sim 4\%$ 的增长速率，但在 TEST 集上的 acc 却增长缓慢，每次 0.5% 的增长速率。最终准确率在 91.79%，比 2D 的要略低。

在股票数据的预处理中，

先对 train 集中的数据放缩，模拟人图像处理，将每个位置的数据放缩到 $[0, 255]$ 之间。

那就没有必要在一开始就对数值放缩了，可以在对每个 batch 处理时再放缩。

但在处理时因为是对每列的数据放缩，故需先将 $\text{data}[i]$ 转置。

a b



这样对每行放缩后，



$\rightarrow a$

再转置回来即可.