

سوال ۱: Regularization

بخش اول: Batch Normalization

: ۱.۱ تاثیر Batch Normalization

این کار، باعث می‌شود میانگین Batch ها صفر شود و نیز واریانس داده‌های آن کم (حدود ۱) باشد؛ و چون عملکرد مناسب توابع فعالیت (سیگموید) در بازه $-1 \text{--} 1$ است (یعنی مشتقشان در خارج این بازه تقریباً صفر است)، Batch Normalization داده‌ها را به این ناحیه فعالیت می‌برد و باعث می‌شود شبکه آموزش‌پذیرتر باشد و کمتر در نواحی flat گیر کند. همچنین عملی مشابه regularization انجام می‌دهد؛ زیرا تنها روی یک داده fit نمی‌شود و داده‌های بیشتری را در یک mini-batch در نظر می‌گیرد. همچنین بعد از learning-rate بزرگتری استفاده کرد؛ و نیز حساسیت شبکه به شرایط اولیه کمتر است.

: ۱.۲ پارامترهای Batch Normalization

یک نکته قابل ذکر است:

$$z = g(Wu + b)$$

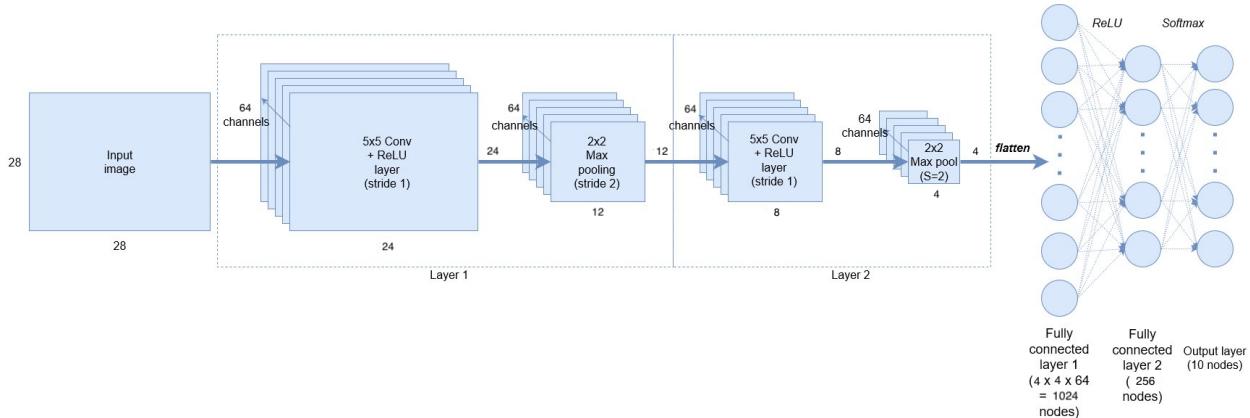
آن که طبق توضیح مقاله اصلی، BN معمولاً بعد از ضرب کردن وزن‌ها و قبل از تابع فعال‌ساز استفاده می‌شود اما در سوال خواسته شده بعد از فعال‌ساز و قبل از لایه Pooling استفاده شود.

where W and b are learned parameters of the model, and $g(\cdot)$ is the nonlinearity such as sigmoid or ReLU. This formulation covers both fully-connected and convolutional layers. We add the BN transform immediately before the nonlinearity, by normalizing $x = Wu + b$. We could have also normalized the layer inputs u , but since u is likely the output of another nonlinearity, the shape of its distribution is likely to change during training, and constraining its first and second moments would not eliminate the covariate shift. In contrast, $Wu + b$ is more likely to have a symmetric, non-sparse distribution, that is “more Gaussian” (Hyvärinen & Oja, 2000); normalizing it is likely to produce activations with a stable distribution.

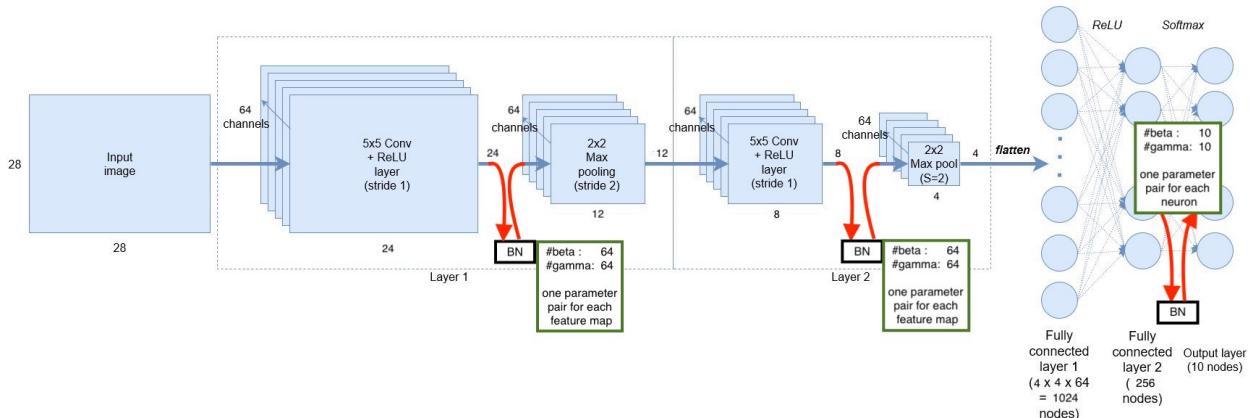
Note that, since we normalize $Wu + b$, the bias b can be ignored since its effect will be canceled by the subsequent mean subtraction (the role of the bias is subsumed by β in Alg. 1). Thus, $z = g(Wu + b)$ is replaced with

$$z = g(\text{BN}(Wu))$$

شبکه تمرین قبل ساختار زیر را داشت:



ساختار شبکه با وجود لایه‌های Batch Normalization به صورت زیر می‌شود:



که تمام اطلاعات لازم در شکل مشخص شده است. در نتیجه مجموعاً ۲۷۶ ضریب به شبکه اضافه می‌کند.

۱.۳: پیاده سازی Batch Normalization

این کد با نام BN.py پیاده شده است. برای مثال، قیمت نرمال کردن لایه اول کانولوشن آن به شکل زیر است:

```
with tf.name_scope("batch_normalization_1"):
    eps1 = 0.0000001
    mean_b_1, var_b_1 = tf.nn.moments(conv1_out, axes=[0, 1, 2], keep_dims=True)
    sigma_1 = tf.sqrt(eps1 + var_b_1)
    bn1_normalized = tf.divide((conv1_out - mean_b_1), sigma_1)
    gamma1 = tf.Variable(
        tf.random_normal(shape=(1, 1, 1, 64), mean=0, stddev=0.01),
        name='gamma1')
    beta1 = tf.Variable(
        tf.zeros(shape=(1, 1, 1, 64)),
        name='beta1')
    bn1_out = bn1_normalized * gamma1 + beta1
```

و حاصل خروجی آن با $\text{Batch Size} = 64$ این گونه است:

```
step    0 accuracy is 0.19000 and cross_entropy(loss) is 2.29892
step    200 accuracy is 0.95000 and cross_entropy(loss) is 1.38952
step    400 accuracy is 0.97000 and cross_entropy(loss) is 0.73858
step    600 accuracy is 0.99000 and cross_entropy(loss) is 0.46366
step    800 accuracy is 1.00000 and cross_entropy(loss) is 0.32062
step   1000 accuracy is 0.99000 and cross_entropy(loss) is 0.22469
step   1200 accuracy is 0.99000 and cross_entropy(loss) is 0.18928
step   1400 accuracy is 0.98000 and cross_entropy(loss) is 0.21119
step   1600 accuracy is 0.99000 and cross_entropy(loss) is 0.14620
step   1800 accuracy is 0.99000 and cross_entropy(loss) is 0.15151
Test Accuracy: 1.0
```

که نتیجه فوق العاده‌ای دارد! دو نکته در این پیاده‌سازی قابل ذکر است:

1. در این پیاده‌سازی، روش بهبود یافته استفاده نشده است:

Input: Network N with trainable parameters Θ ;
subset of activations $\{x^{(k)}\}_{k=1}^K$

$$\begin{aligned}\mathbb{E}[x] &\leftarrow \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}] \\ \text{Var}[x] &\leftarrow \frac{m}{m-1} \mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]\end{aligned}$$

و یعنی پارامترهای میانگین و واریانس در زمان آموزش Learn نمی‌شوند.

2. در زمان تست، چون از روش بهبود یافته استفاده نمی‌شود، طبق گفته مقاله برای تک داده‌ها باید لایه BN را حذف کرد:

(Duchi et al., 2011). The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference; we want the output to depend only on the input, deterministically.

اما در اینجا، چون تست را روی تک داده انجام نداده‌ایم، لایه‌های BN را نگه داشته‌ایم و حذفی صورت نگرفته است.

(نکته: در Loss این قسمت از regularization عادی (L2) استفاده نشده است.)

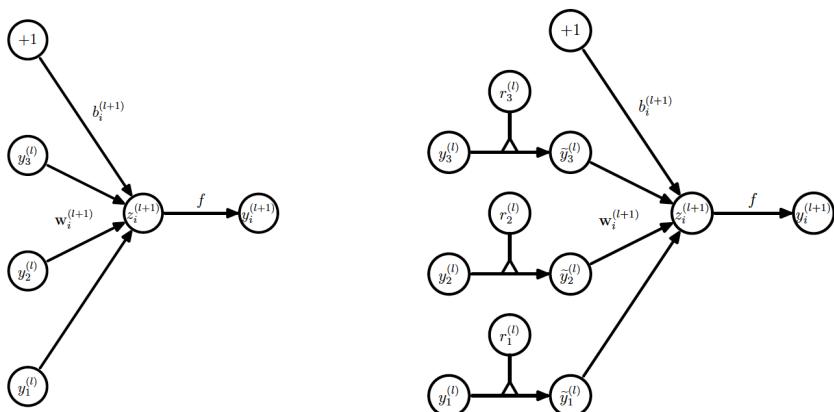
بخش دوم: Dropout

۲.۱: تاثیر Dropout

روش Dropout نیز عملی از جنس regularization انجام می‌دهد، و مانع از overfitting می‌شود؛ زیرا در هر اجرا، تعدادی از نورون‌های شبکه وجود ندارند؛ پس شبکه به استفاده از یک سری نورون مشخص وابسته نشده یا اصطلاحاً تنبل نمی‌شود، و جوری آموزش داده می‌شود که بدون حضور همه نورون‌هاییش نیز عملکرد مناسبی داشته باشد و robust می‌شود. همچنین طی این فرآیند، نورون‌هایی که اثر کمتری دارند کمتر آموزش داده می‌شوند (مشتقشان در هنگام حضورشان زیاد نخواهد بود) و فعالیت‌های اصلی شبکه با نورون‌های تاثیر گذار ترش انجام می‌شود.

۲.۲: به کار بردن Dropout در مرحله آموزش و تست:

در مرحله آموزش، هر نورون با احتمال داده شده p در لایه بعد خود اثر دارد و با احتمال $1-p$ حذف می‌گردد. این کار را با ضرب کردن 0 یا 1 یا به عبارتی دیگر متغیری با توزیع بولی p (در شکل زیر \mathbf{r} ها) مدل می‌کنیم:



اما در مرحله تست، همه نورون‌ها حضور دارند. برای آن که تاثیر Dropout در آموزش را وارد مرحله تست کنیم، خروجی نورون‌ها را در احتمال حضورشان در مرحله آموزش (یا نسبت تعداد بارهای حضورشان در زمان آموزش) ضرب می‌کنیم (برای مثال در اینجا خروجی نورون‌ها در p ضرب می‌شود).

این کار، باعث می‌شود که خروجی Expected Value تمام حالت‌های آموزش شده باشد و به نحوی مرحله تست، میانگین حالات اتفاق افتاده در مرحله آموزش می‌شود.

۲.۳: پیاده سازی Dropout

این کد با نام Dropout.py پیاده شده است.

تابع مربوط به آن به صورت زیر است:

```
def dropout(neuron_in, prob):
    shape = tf.shape(neuron_in)
    sto_var = tf.random_uniform(shape=shape, minval=0, maxval=1)
    r = tf.cast((sto_var <= prob), tf.float32)
    neuron_out = tf.multiply(neuron_in, r)
    return neuron_out
```

و یک مدل feed کردن آن اینگونه است:

```
feed_dict={x: x_sample, y: y_sample,
           keep_prob_input: input_prob, keep_prob: neuron_prob, in_existence: 1.0, existence: 1.0}
```

که احتمال بودن نوروں ورودی ۰.۸ و احتمال بودن نوروں مخفی ۰.۵ است. دو پارامتر existence نیز برای راحتی کار تست وارد شده اند که هنگام تست، ضریب حضور نوروں را در خروجی Dropout ها ضرب می کند (در اینجا تعداد حضور را نمی شماریم و از همان احتمالی که با آن در آموزش نوروں ها را نگه داشتیم استفاده می کنیم).

خروجی و دقت آن نیز به صورت زیر است:

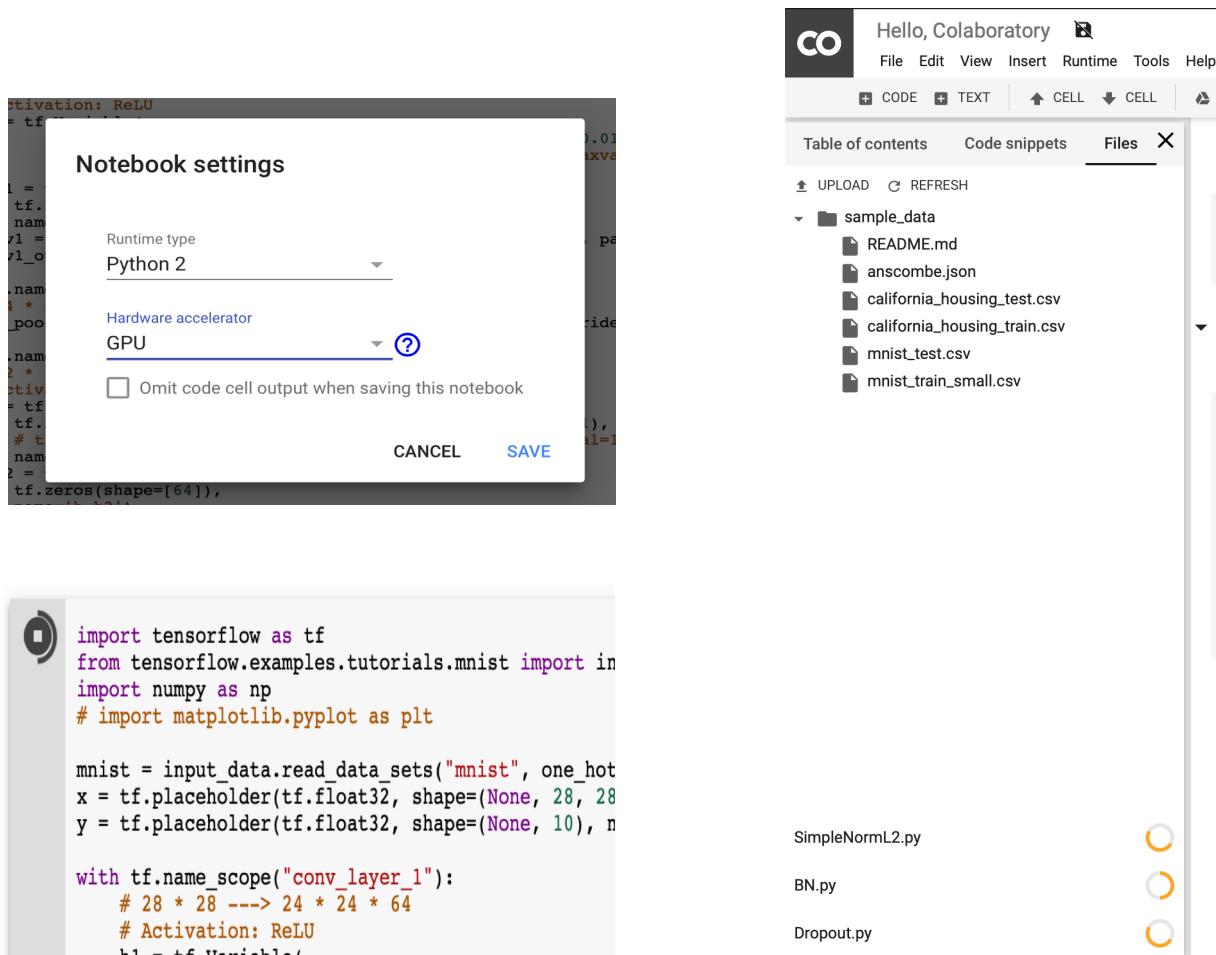
```
step      0 accuracy is 0.11000 and cross_entropy(loss) is 2.30257
step    200 accuracy is 0.53000 and cross_entropy(loss) is 2.18249
step    400 accuracy is 0.72000 and cross_entropy(loss) is 1.61622
step    600 accuracy is 0.89000 and cross_entropy(loss) is 1.04115
step    800 accuracy is 0.94000 and cross_entropy(loss) is 0.67838
step   1000 accuracy is 0.93000 and cross_entropy(loss) is 0.54833
step   1200 accuracy is 0.94000 and cross_entropy(loss) is 0.49174
step   1400 accuracy is 0.97000 and cross_entropy(loss) is 0.38025
step   1600 accuracy is 0.97000 and cross_entropy(loss) is 0.31230
step   1800 accuracy is 0.94000 and cross_entropy(loss) is 0.32646
Test Accuracy:  0.97
```

که بهبود آن از BN کمتر است اما مشابه یک regularization عادی (L2) عمل می کند.

سوال ۲: Google Colab

بخش اول: اجرای شبکه ساده

تصاویری از کار دو محیط در زیر آمده است (اول سعی کردم کدها را در Cloud آپلود کنم (شکل راست) اما به دلیل آن که طول می کشید، خود کدها را در محل اجرا کمی و پیست کردم):



نتیجه شیکه ساده(L2) به صورت زیر است (دقیق شود ۲۰۰۰۰ مرحله شیکه آموزش دیده شده است):

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
step      0 accuracy is 0.16000 and cross_entropy(loss) is 2.48680
step    2000 accuracy is 0.09000 and cross_entropy(loss) is 2.44219
step    4000 accuracy is 0.09000 and cross_entropy(loss) is 2.38578
step    6000 accuracy is 0.88000 and cross_entropy(loss) is 0.51592
step    8000 accuracy is 0.97000 and cross_entropy(loss) is 0.24574
step   10000 accuracy is 0.97000 and cross_entropy(loss) is 0.29242
step   12000 accuracy is 0.97000 and cross_entropy(loss) is 0.25652
step   14000 accuracy is 0.98000 and cross_entropy(loss) is 0.25789
step   16000 accuracy is 0.95000 and cross_entropy(loss) is 0.30450
step   18000 accuracy is 0.98000 and cross_entropy(loss) is 0.23935
Test Accuracy:  0.95
```

بخش دوم: اجرای مستقل Dropout و Batch Normalization

نتیجه اجرای بسیار سریع کد BN روی GPU در Google Colab به صورت زیر است:

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
step    0 accuracy is 0.09000 and cross_entropy(loss) is 2.30200
step   200 accuracy is 0.91000 and cross_entropy(loss) is 1.44900
step   400 accuracy is 0.97000 and cross_entropy(loss) is 0.73542
step   600 accuracy is 0.98000 and cross_entropy(loss) is 0.49802
step   800 accuracy is 0.98000 and cross_entropy(loss) is 0.32469
step  1000 accuracy is 0.99000 and cross_entropy(loss) is 0.28260
step  1200 accuracy is 0.98000 and cross_entropy(loss) is 0.20291
step  1400 accuracy is 0.99000 and cross_entropy(loss) is 0.16999
step  1600 accuracy is 0.99000 and cross_entropy(loss) is 0.14706
step  1800 accuracy is 0.98000 and cross_entropy(loss) is 0.11608
Test Accuracy: 1.0
```

دقت شود شبکه تنها ۲۰۰۰ بار آموزش دیده شده که یک دهم حالت قبل است، و نتیجه بسیار بهتری داده است.

همچنین نتیجه Dropout نیز به شکل زیر است:

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
step    0 accuracy is 0.06000 and cross_entropy(loss) is 2.30259
step   200 accuracy is 0.34000 and cross_entropy(loss) is 2.22580
step   400 accuracy is 0.70000 and cross_entropy(loss) is 1.70794
step   600 accuracy is 0.87000 and cross_entropy(loss) is 1.13866
step   800 accuracy is 0.90000 and cross_entropy(loss) is 0.81970
step  1000 accuracy is 0.97000 and cross_entropy(loss) is 0.58160
step  1200 accuracy is 0.96000 and cross_entropy(loss) is 0.47306
step  1400 accuracy is 0.95000 and cross_entropy(loss) is 0.41622
step  1600 accuracy is 0.98000 and cross_entropy(loss) is 0.33201
step  1800 accuracy is 0.94000 and cross_entropy(loss) is 0.31403
Test Accuracy: 0.98
```

این کد میز زیر ۱۵ ثانیه اجرا شده و مجددا تنها ۲۰۰۰ بار آموزش دیده شده که یک دهم حالت عادی است، و نتیجه بسیار بهتری دارد.

بخش سوم: اجرای همزمان Dropout و Batch Normalization

استفاده همزمان از BN و Dropout نتیجه زیر را دارد:

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
step    0 accuracy is 0.03000 and cross_entropy(loss) is 2.30300
step   200 accuracy is 0.63000 and cross_entropy(loss) is 2.05391
step   400 accuracy is 0.91000 and cross_entropy(loss) is 1.38756
step   600 accuracy is 0.96000 and cross_entropy(loss) is 0.81707
step   800 accuracy is 0.98000 and cross_entropy(loss) is 0.54211
step  1000 accuracy is 0.95000 and cross_entropy(loss) is 0.41627
step  1200 accuracy is 0.97000 and cross_entropy(loss) is 0.32127
step  1400 accuracy is 0.98000 and cross_entropy(loss) is 0.25645
step  1600 accuracy is 0.99000 and cross_entropy(loss) is 0.19094
step  1800 accuracy is 0.98000 and cross_entropy(loss) is 0.18664
Test Accuracy:  0.99
```

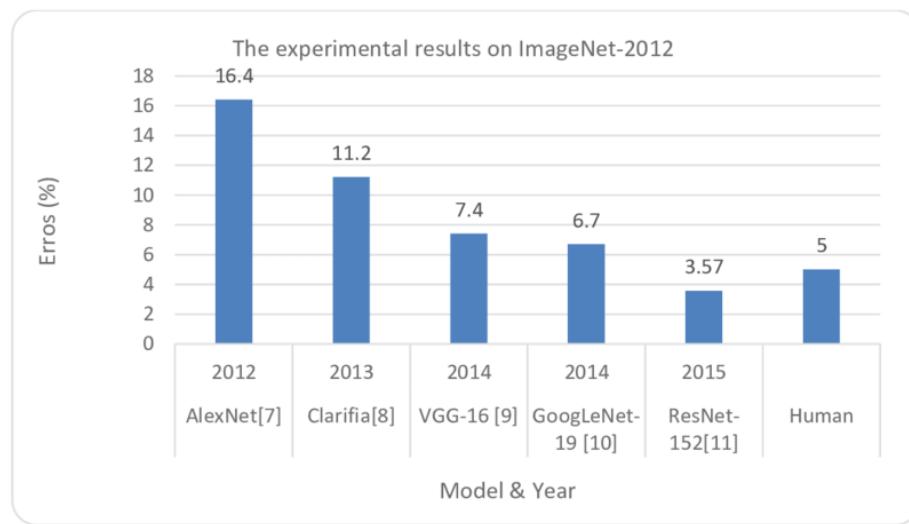
مقایسه

روش	تعداد مرحله آموزش برای رسیدن به صحت بیشتر از ۹۵ درصد	سرعت آموزش دیدن (عکس زمان طول کشیدن هر مرحله)
عادی (L2)	حدود ۸۰۰۰	بسیار کند
BN	حدود ۳۰۰	سریع
Dropout	حدود ۹۰۰	بسیار سریع
(Dropout و BN) ترکیبی	حدود ۶۰۰	سریع

سوال ۳: VGG16 Visualization

بخش اول: توضیح کلیات VGG16

شبکه VGG16، یک شبکه عمیق با ۱۶ لایه مخفی است (VGG19 شبکه مشابهی با ۱۹ لایه مخفی است). این شبکه، برای دسته بندی روی دیتاست ImageNet طراحی شده است که در واقع قرار است ۱۰۰۰ کلاس موجود در این دیتاست را دسته بندی کند. تصاویر ورودی این شبکه، تصاویر رنگی مربعی با ابعاد ۲۲۴ در ۲۲۴ است. نام آن، بر گرفته از گروه سازنده آن در دانشگاه آکسفورد است که Visual Geometry Group نام دارند؛ و این شبکه را برای مسابقه ImageNet Large Scale Visual Recognition Challenge، یا ILSVRC، در سال ۲۰۱۴، طراحی کردند و رتبه دوم را در دسته بندی در این مسابقات کسب کردند. البته ای شبکه برای Image Localization نیز استفاده شد و در آن مسابقه رتبه اول را کسب کرد (اما کاربری اصلی شبکه به صورت عادی این کار نیست و فقط برای پیدا کردن ویژگی های مناسب در آن حوزه از آن استفاده می شود). از ۱۶ لایه این شبکه، ۱۳ لایه لایه های کانولوشنی، همه با کرنل های ۳ در ۳، و ۳ لایه آن Fully Connected هستند (در بخش بعد دقیق تر توضیح داده می شود). سایز مدل آموزش دیده آن (وزن ها) ۵۲۸ MB است. صحت آن نیز طبق تصویر زیر، ۹۲.۶ درصد است (دقیق تر بگوییم، برای Top-1 دقت ۷۰.۶ درصد و برای Top-5 دقت ۸۹.۹ درصد دارد).

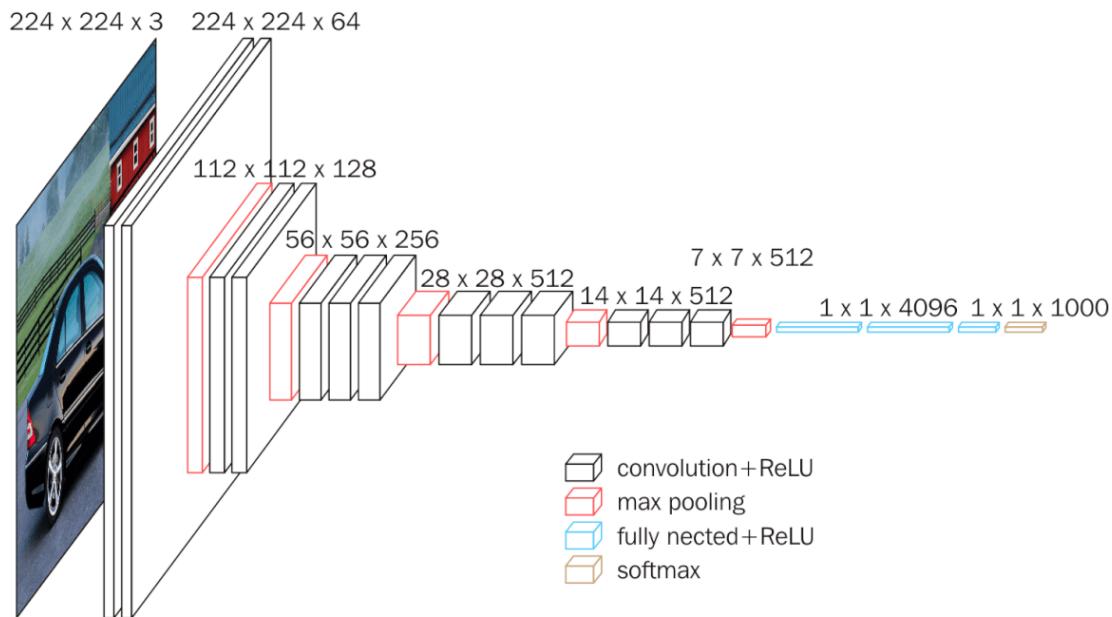


بخش دوم: توضیح لایه ها و ابعاد VGG16

تصویر زیر به طور خلاصه گویای مطلب است:



1. Convolution using 64 filters
2. Convolution using 64 filters + Max pooling
3. Convolution using 128 filters
4. Convolution using 128 filters + Max pooling
5. Convolution using 256 filters
6. Convolution using 256 filters
7. Convolution using 256 filters + Max pooling
8. Convolution using 512 filters
9. Convolution using 512 filters
10. Convolution using 512 filters + Max pooling
11. Convolution using 512 filters
12. Convolution using 512 filters
13. Convolution using 512 filters + Max pooling
14. Fully connected with 4096 nodes
15. Fully connected with 4096 nodes
16. Output layer with Softmax activation with 1000 nodes



لود شدن VGG16 روی کامپیوتر:

```

VGG x
2018-12-15 02:17:43.696550: I tensorflow/core/platform/cpu_feature
Layer (type)          Output Shape         Param #
=====
input_1 (InputLayer)   (None, 224, 224, 3)    0
block1_conv1 (Conv2D)  (None, 224, 224, 64)   1792
block1_conv2 (Conv2D)  (None, 224, 224, 64)   36928
block1_pool (MaxPooling2D) (None, 112, 112, 64) 0
block2_conv1 (Conv2D)  (None, 112, 112, 128)  73856

```

بخش سوم: نمایش فیلترهای VGG16

مدل را بصورت زیر تعریف می‌کنیم:

```
model = tf.keras.applications.vgg16.VGG16(include_top=True, weights='imagenet',
                                         input_tensor=None, input_shape=None,
                                         pooling='max', classes=1000)
```

ضرایب، یک لیست ۳۲ تایی است که به ترتیب ۲ تا ۲ تا، ضرایب و بایاس‌های ۱۶ لایه در آن قرار دارد.

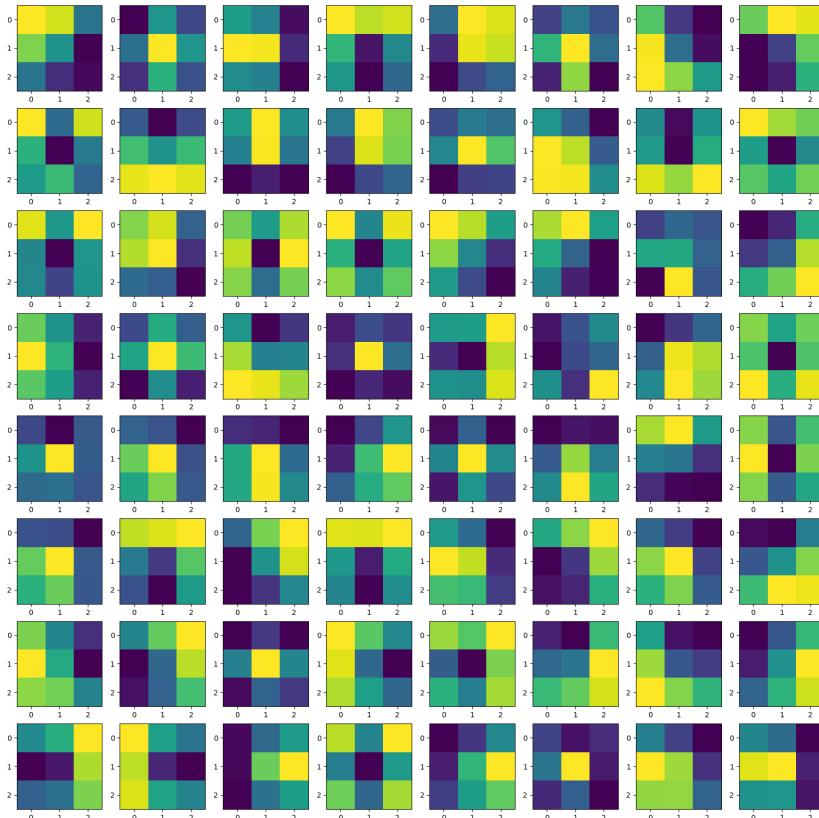
شکل شبکه را، باز دیگر، با دستور plot_model موجود در کتابخانه tensorflow.keras.utils ذخیره می‌کنیم. این تصویر با نام model_part_2.png در فolder موجود است.

برای رسم کرنل‌ها، از دستور زیر استفاده کردہ‌ایم (کد مخصوص خود سایت vgg است):

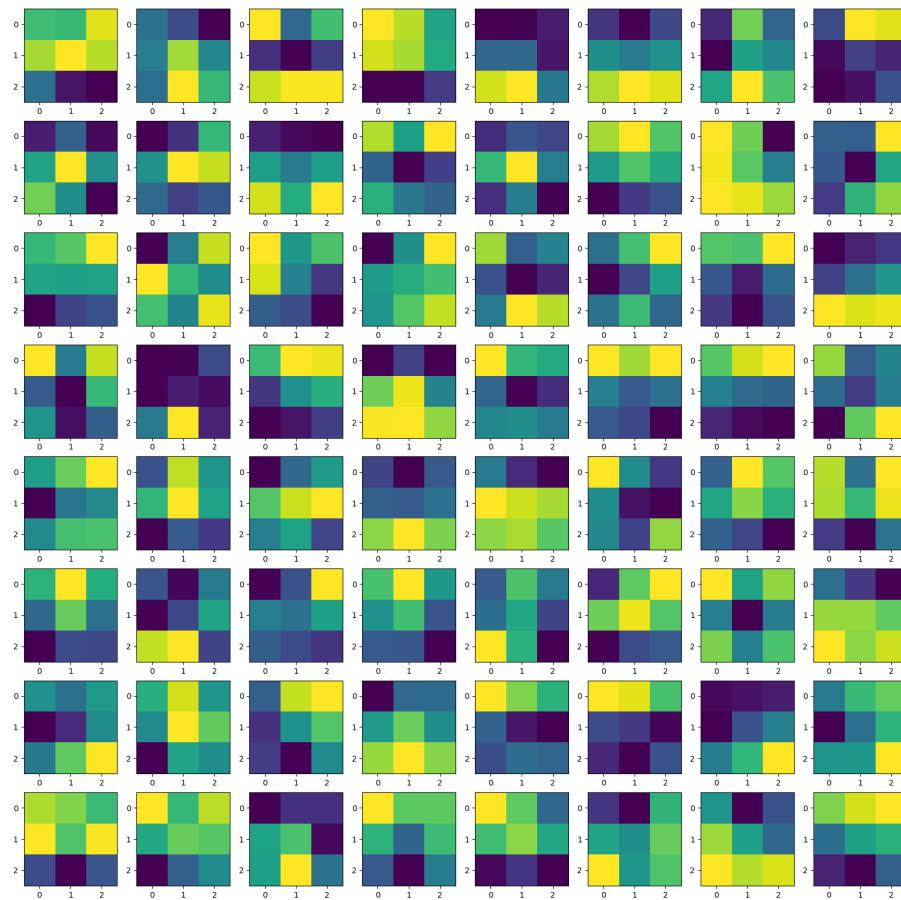
```
layer_w = layer_dict[layer_name].get_weights()[0][:, :, 0, :]
```

که نام لایه‌ها، طبق تعریف خود شبکه به صورت زیر است:
['block1_conv1', 'block1_conv2', 'block2_conv1', ..., 'block5_conv3']

ضرایب لایه اول کانولوشنی به صورت زیر است (تمام اشکال این بخش در فolder vgg_weights ذخیره شده‌اند):



و ضرایب لایه آخر کانولوشنی نیز اینگونه اند:



نکته: فیلترهای کانولوشنی که ۳ در ۳ هستند، عمقی برابر تعداد کانالهای لایه قبل خود دارند، که مثلاً ۵۱۲ کanal است و نمایش تمام عمق آن ممکن نیز. لذا تنها ورق اول از این فیلترها نمایش داده شده اند.

چون فیلترها ۳ در ۳ و بسیار کوچک اند، مقایسه دقیقی نمی‌توان میان آنها انجام داد. اما به صورت خیلی تقریبی، می‌توان گفت فیلتر لایه اول اعمال قابل فهمی از جمله مشتق در راستاهای مختلف، میانگین گیری گاووسی، مشتق مرتبه ۲ و این قبیل کارها را انجام می‌دهد (که همانطور که مشخص است مثلاً برای مشتق جهتی، یک نیمه وزن تاریک و یک نیمه آن روشن است). اما در ضرایب فیلتر لایه آخر این مفاهیم قابل فهم دیده نمی‌شوند و فیلتر به صورت رندومتر و پراکنده‌تری در آمده است.

بخش چهارم: مشاهده عملکرد VGG16 روی تصاویر داده شده

اشکال این بخش در فolder vgg_images ذخیره شده اند. برای هر تصویر، ۱۶ فیلتر از لایه ۳ (بعد از max pooling) ۶۴ کanal دارد و ما این ۶۴ تا را ۴ تا در میان برداشته ایم که ۱۶ تا شود و هر کدام ابعاد ۱۱۲ در ۱۱۲ دارد) و ۱۶ فیلتر از لایه ۱۳ (بعد از max pooling) ۵۱۲ کanal دارد و ما ۳۲ تا در میان از آن برداشته ایم که ۱۶ عدد شود و هر کدام ابعاد ۷ در ۷ دارد)، ذخیره شده اند.

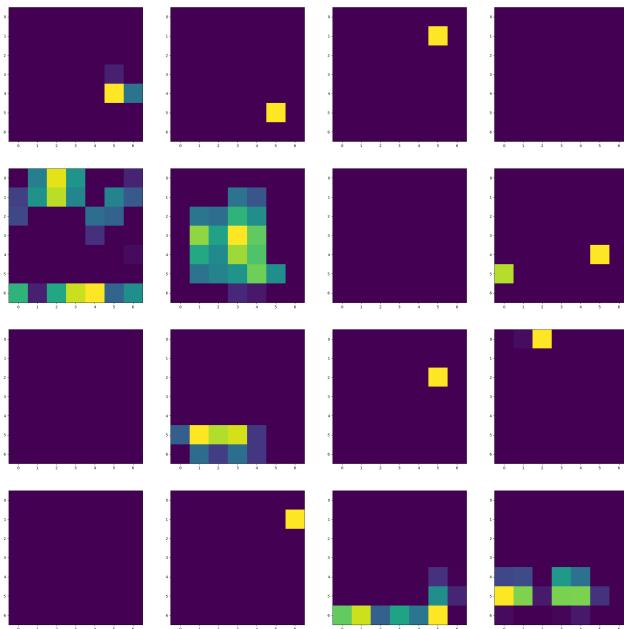
نتیجه تشخیص شبکه vgg روی این تصاویر به صورت زیر است:

```
brown_bear is predicted as: brown_bear with belief: 97.06575274467468 %
cat_dog is predicted as: boxer with belief: 42.014119029045105 %
dd_tree is predicted as: seashore with belief: 63.60258460044861 %
dog_beagle is predicted as: beagle with belief: 97.02553153038025 %
scenery is predicted as: castle with belief: 63.299620151519775 %
space_shuttle is predicted as: space_shuttle with belief: 99.79477524757385 %

Process finished with exit code 0
```

و یک نمونه از تصاویر در زیر آمده است:





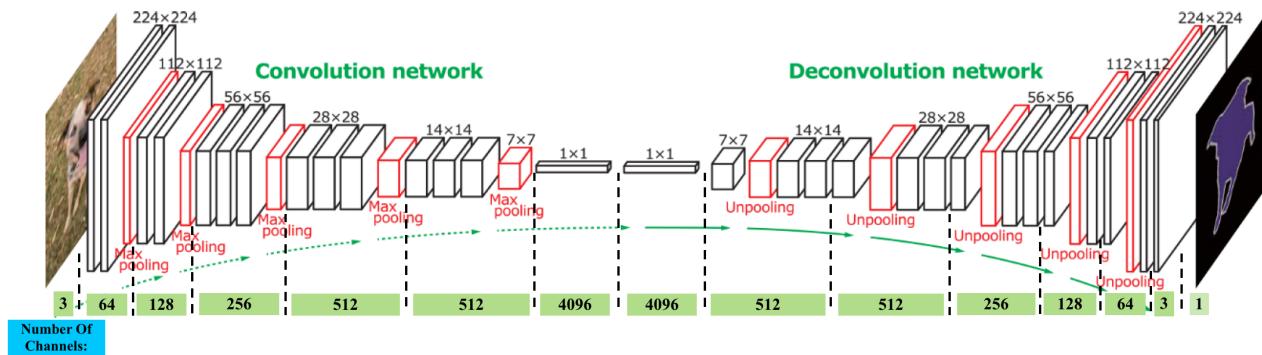
تحلیل و مقایسه:

طبق توضیح بخش قبل، در نورون‌های لایه‌های ابتدایی بیشتر شبکه در حال استخراج ویژگی‌های جزئی از تصویر است، مانند مشتق جهت دار در هر نقطه، مشتق دوم(گرادیان)، ویژگی‌های تصویر blur و به همین دلیل تصاویر در لایه‌های ابتدایی، فرم کلی عکس ورودی را دارند و فقط ویژگی‌های نقاطهای آنها متمایز شده است. اما در لایه‌های انتهایی‌تر، نورون‌ها ویژگی‌هایی مربوط به تمام تصویر را استخراج می‌کنند، مانند میانگین روش‌نایابی روی تمام نقاط؛ و برای همین در تص، یعنی خروجی این لایه‌ها شکل کلی عکس معلوم نیست و تنها نقاط روش‌نایابی روی آن معلوم است که کد گذاری شده پارامترهایی از عکس است.

سوال ۳: Deconvolution

بخش اول: بررسی شبکه داده شده

این بر اساس کد نوشته شده و سایز کرnel های convolution و deconvolution ، شکل زیر بیانگر شبکه خواهد بود:

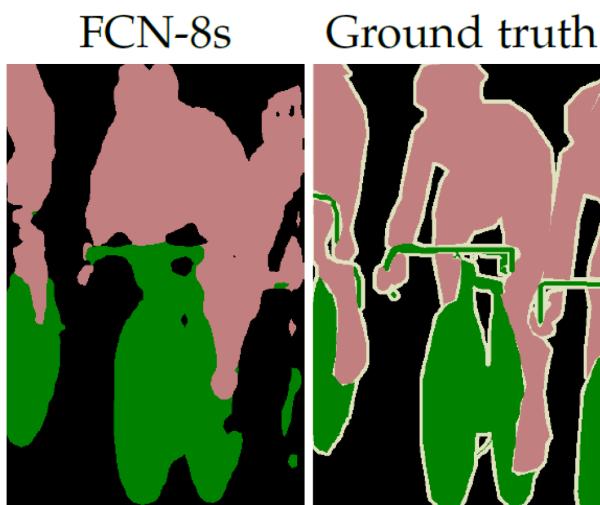


که ترتیب لایه‌ها نیز در بالا مشخص شده است.

بخش دوم: توضیح کاربرد شبکه داده شده

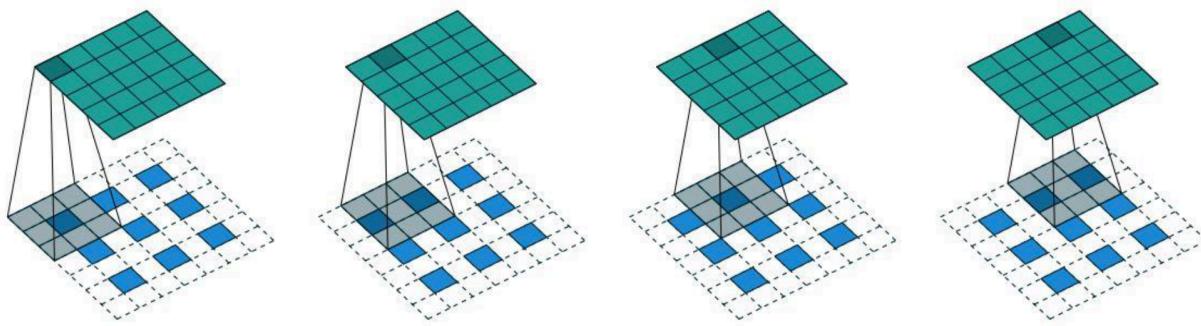
این شبکه، که نیمه دوم آن Deconvolutional Neural Network یا DNN نیز نام دارد، یک نوع شبکه برای بازیابی تصویر از ویژگی‌هایی محدود است (مجموع نیمه چپ و راست شبکه، Fully Convolutional Network یا FCN نام دارد). بعضی موقع‌ها لفظ Reverse Engineering نیز به آن نسبت می‌دهند و کاربردهای مختلفی می‌توانند داشته باشد:

1. چون تصویر اصلی را از تعداد محدودی ویژگی می‌سازد، و نیز چون لایه‌های Max Pool ویژگی‌های ضعیف را حذف می‌کنند، نویزهای تصویر در نیمه دوم شبکه حذف می‌شود و یا صدمات تصویر اولیه طی این فرآیند جبران می‌شود.
2. کاربرد وسیعی در Semantic Segmentation دارد، که در تصویر بالا نیز معلوم است و کد نیز برای این منظور نوشته شده است؛ زیرا loss آن با یک grand truth به ابعاد ۲۲۴ در ۱ مقایسه می‌شود؛ مانند تصویر زیر:



بخش سوم: توضیح عملکرد لایه Deconvolution

این لایه، به نحوی برای خنثی کردن اثر لایه convolution به کار می‌رود (خنثی کردن مفهومی و یا فقط خنثی کردن اثر تغییر ابعاد). این لایه یک تصویر با ابعاد کم را گرفته و تصویری با ابعاد بزرگتر از ورودی تولید می‌کند. برای مثال در تصویر زیر، یک تصویر ۳ در ۳ را با کرنل ۳ تایی به تصویر ۵ در ۵ تبدیل می‌کند (که مثلاً تصویر ۳ در ۳، نتیجه Valid اعمال همین کرنل ۳ در ۳ روی تصویر ۵ در ۵ است):



این کار الته راههای مختلفی دارد و لزوماً مانند تصویر بالا، با صفر گذاشتن میان تصویر کوچکتر ساخته نمی‌شود. به نحوی می‌توان گفت عمل upsampling انجام می‌شود. از تفاوت‌های آن نیز می‌توان به این اشاره کرد که در این لایه، اطلاعاتی تولید می‌شود که این کار روش قطعی‌ای نیز ندارد در حالی که در لایه convolution تنها اطلاعاتی مختصر می‌شوند.

بخش چهارم: درباره پیاده سازی و آموزش شبکه داده شده

مجموعه داده مناسب برای آموزش شبکه داده شده:

- مجموعه داده‌های ImageNet: که هر ساله در مسابقات متعدد استفاده می‌شوند و grand truth های متنوعی نیز برای آنها ساخته شده است.
- مجموعه داده‌های CamVid، KITTI، DUS، CitySpaces، Mapillary: برای آموزش Segmentation خودرو های خودران
- مجموعه داده‌ی ScanNet: برای آموزش Segmentation روی تصاویر رنگی
- مجموعه داده‌ی deepGlobe Challenge: برای آموزش Segmentation در مسابقه Facebook