

به نام خداوند بخشنده و مهربان

یادگیری عمیق

تمرین پنجم

محسن نقی پورفر ۹۴۱۰۶۷۵۷

Regularization ۱

BatchNormalization ۱.۱

۱.۱.۱ تاثیر اضافه کردن

در صورت اضافه کردن لایه BatchNormalization تاثیرات زیر به شبکه اعمال می شود:

- سرعت آموزش شبکه با توجه به حل شدن مسئله Internal Covariate Shift زیاد می شود.
- مسئله وابستگی ورودی های هر لایه و لایه بعدی حل می شود و این باعث ایجاد امکان یادگیری و بهینه سازی شبکه با استفاده از learning rate های بالا می شود.
- با حل مسئله ICS، می توان از شبکه های عمیق تر نیز برای مسئله خاص استفاده کرد و به نتایج معمولاً بهتری نسبت به شبکه با عمق کمتر دست یافته.
- این لایه، باعث نرمالیزه کردن توزیع خروجی هر لایه در هر نورون می شود و این نیز باعث سرعت بخشیدن به فرآیند یادگیری می شود.
- با استفاده از آماره های Batch، باعث می شود که هر شبکه منظم شود و Regularize شود و این منظم شدن را نسبت به Overfitting می گوییم. پس این لایه باعث منظم سازی شبکه و در نتیجه جلوگیری از Overfitting می شود.
- افزودن این لایه به شبکه اجازه مقداردهی اولیه ضعیف (Poor Initialization) به وزن ها و پارامتر های یادگیری شبکه را می دهد بدون آنکه در فرآیند یادگیری مشکلی ایجاد شود.
- مسئله explode vanish و یا گرادیان را به حد قابل قبولی حل می کند.
- این لایه با توجه به محدود کردن ورودی تابع فعال ساز در ناحیه خاصی که برای تابع فعال ساز relevant می باشد، تا حدی از اشباع شدن خروجی تابع جلوگیری می کند و به نوعی شبکه را stablize می کند.
- اختلافات خطی بین Batch های مختلف را از بین می برد.
- ارزش و توانایی مدل را با اضافه کردن دو پارامتر قابل یادگیری γ و β افزایش می دهد.
- در زمان تست با توجه به اینکه ممکن است Batch برای داده های تست خیلی کوچک باشد، با استفاده EMA تخمین خوبی از میانگین و واریانس می زند.

۲.۱.۱ تعداد پارامتر های افزوده شده

در اثر اضافه کردن لایه BatchNormalization به هر لایه، به ازای هر نورون آن لایه، دو پارامتر γ و β اضافه می شود که قابل یادگیری هستند.

۳.۱.۱ پیاده سازی تابع این لایه

عکس های زیر، پیاده سازی لایه BatchNormalization می باشند. همانطور که از اسم آنها نیز پیداست، عکس اول مربوط به پیاده سازی این لایه برای بعد از Convolution است و عکس دوم مربوط به پیاده سازی این لایه بعد از یک لایه Fully Connected می باشد.

```

def batch_normalization_from_scratch_conv(x, n_out, training_phase, name_scope="BatchNormalization"):
    epsilon = 1e-7
    with tf.name_scope(name_scope):
        # n_out is number of filters for conv
        gamma = tf.Variable(tf.constant(1.0, shape=[n_out]), trainable=True, name="Gamma")
        beta = tf.Variable(tf.constant(0.0, shape=[n_out]), trainable=True, name="Beta")

        ema = tf.train.ExponentialMovingAverage(decay=0.5)

        def calculate_moments_for_conv(): # calculate mean, var for conv layer
            batch_mean = tf.reduce_mean(x, axis=[0, 1, 2], keep_dims=True)
            batch_mean = array_ops.squeeze(batch_mean, [0, 1, 2])
            m = tf.reduce_mean(x, axis=[0, 1, 2], keep_dims=True)
            squared_diffs = tf.square(x - m)
            batch_var = tf.reduce_mean(squared_diffs, axis=[0, 1, 2], keep_dims=True)
            batch_var = array_ops.squeeze(batch_var, [0, 1, 2])
            return batch_mean, batch_var

        batch_mean, batch_var = calculate_moments_for_conv()

        def mean_var_with_update_at_train_time(): # Track & save the mean and variance for test time
            ema_apply_op = ema.apply([batch_mean, batch_var])
            with tf.control_dependencies([ema_apply_op]):
                return tf.identity(batch_mean), tf.identity(batch_var)

        mean, var = tf.cond(training_phase, # calculate mean, var for train/test time
                            mean_var_with_update_at_train_time,
                            lambda: (ema.average(batch_mean), ema.average(batch_var)))

        denominator = tf.rsqrt(var + epsilon) # NOTE: rsqrt -> 1 / sqrt()
        normed = gamma * (x - mean) * denominator + beta
    return normed

```

شکل ۱: پیاده سازی لایه BN برای لایه کانولوشنی

```

def batch_normalization_from_scratch_dense(x, n_out, training_phase, name_scope="BatchNormalization"):
    epsilon = 1e-7
    with tf.name_scope(name_scope):
        # n_out is number of nodes in fully connected layer
        gamma = tf.Variable(tf.constant(1.0, shape=[n_out]), trainable=True, name="Gamma")
        beta = tf.Variable(tf.constant(0.0, shape=[n_out]), trainable=True, name="Beta")

        ema = tf.train.ExponentialMovingAverage(decay=0.5)

        def calculate_moments_for_dense(): # calculate mean, var for fully connected layer
            batch_mean = tf.reduce_mean(x, axis=[0], keep_dims=True)
            m = tf.reduce_mean(x, axis=[0], keep_dims=True)
            devs_squared = tf.square(x - m)
            batch_var = tf.reduce_mean(devs_squared, axis=[0], keep_dims=True)
            return batch_mean, batch_var

        batch_mean, batch_var = calculate_moments_for_dense()

        def mean_var_with_update_at_train_time(): # Track & save the mean and variance for test time
            ema_apply_op = ema.apply([batch_mean, batch_var])
            with tf.control_dependencies([ema_apply_op]):
                return tf.identity(batch_mean), tf.identity(batch_var)

        mean, var = tf.cond(training_phase, # calculate mean, var for train/test time
                            mean_var_with_update_at_train_time,
                            lambda: (ema.average(batch_mean), ema.average(batch_var)))

        denominator = tf.rsqrt(var + epsilon) # NOTE: rsqrt -> 1 / sqrt()
        normed = gamma * (x - mean) * denominator + beta
    return normed

```

شکل ۲: پیاده سازی لایه BN برای لایه FullyConnected

Dropout ۲.۱

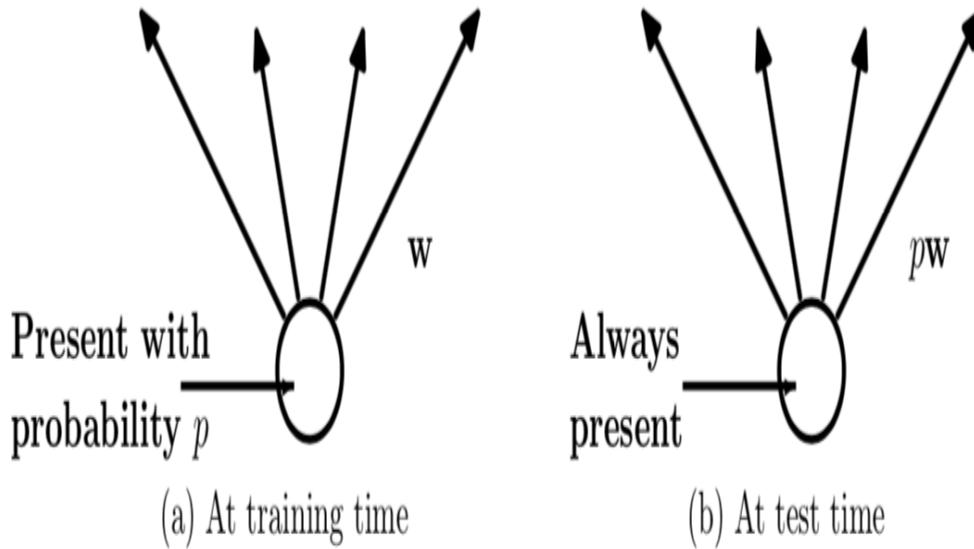
۱. تأثیر اضافه کردن Dropout

اضافه کردن این منظم ساز به هر لایه، تاثیرات زیر را همراه خود می آورد:

- تاثیر مشخص اول، همان منظم ساز بودن آن است، به این معنی که از overfitting جلوگیری می‌کند.
- با توجه به منظم ساز بودن آن، به feature co-adaptation کمک می‌شود.
- با توجه به اینکه منظم ساز می‌باشد، به فرآیند یادگیری، نویز اضافه می‌کند که این یکی از تاثیرات اضافه کردن منظم ساز است.
- باعث بیشتر شدن Sparsity در لایه‌های پنهان شبکه می‌شود که این نیز از تاثیرات اضافه کردن این منظم ساز است.
- در واقع شبیه میانگین گیری از تمام H^2 حالت ممکن برای مدل‌ها می‌باشد (در صورتی که به یک لایه پنهان با H نورون اعمال شود). در این حالت وزن‌ها به نوعی Shared هستند و این باعث منظم شدن مدل شده و از overfitting جلوگیری می‌کند.

۲.۲.۱ فرق در آموزش و تست

در هنگام آموزش، ما از یک توزیع Binomial با احتمال موقتی P استفاده می‌کیم و نورون‌های لایه‌ای که این منظم ساز روی آن اعمال شده است را به طور کامل حذف می‌کنیم. در واقع هر نورون با احتمال P در شبکه حضور دارد و یک سری از نورون‌ها در فرآیند یادگیری حذف می‌شوند. اما در فرآیند تست، همه نورون‌ها در شبکه به طور قطع حاضر مستند، اما وزن Connection آنها در عدد ضرب می‌شود و به نوعی Expected گیری انجام می‌دهیم برای خروجی هر نورون و تاثیرات آن روی نورون‌های لایه بعدی. یعنی، انتظار داریم که با احتمال P از شبکه در فرآیند یادگیری حذف می‌شود، خروجی آن به ازای وزن‌های Pw باشد. شکل زیر که از مقاله خود جناب آقای Hinton از مقاله Dropout آمده است، فرق بین اعمال این منظم ساز در فرآیند تست و آموزش را به خوبی توضیح می‌دهد.



شکل ۳: فرق dropout در تست و آموزش

۳.۲.۱ پیاده سازیتابع این لایه

در عکس زیر پیاده سازی مربوط به این منظم ساز روی ورودی خود آمده است.

```

def dropout(x, keep_prob, training_phase):
    with tf.name_scope("Dropout"):
        input_shape = x.get_shape().as_list()
        m = tf.cond(training_phase,
                    lambda: np.random.binomial(1, keep_prob, size=input_shape),
                    lambda: tf.constant(keep_prob, shape=input_shape))
        m = tf.multiply(x, m)
    return m

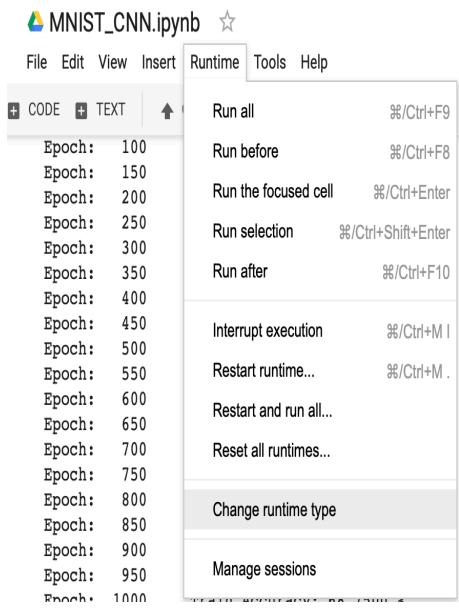
```

شکل ۴: پیاده سازی لایه Dropout

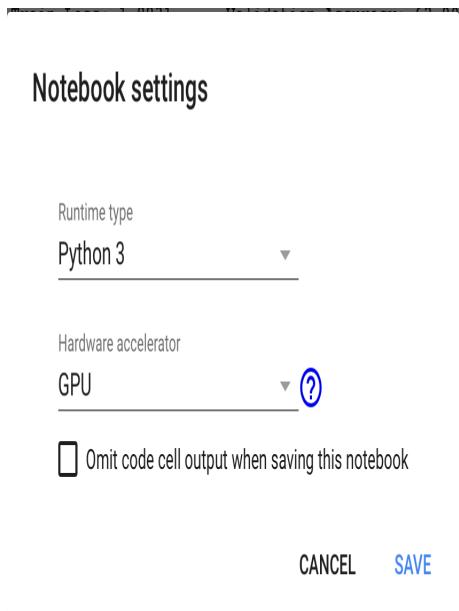
Google Colab ۲

۱.۲ گزارش نتیجه و مراحل اجرا در این محیط

ابتدا در گوگل sign in می کنیم. ابتدا در این محیط فایل نوتبوکی با پایتون ۳ ایجاد می کنیم. سپس در گزینه Runtime Change Runtime Type را انتخاب نموده و در پنجره باز شده Hardware Accelerator GPU را انتخاب می کنیم. حال شروع به کد زنی می کنیم. عکس مربوط به مرحله Run شده در بالا، در زیر آمده است. در آخر بعد از اتمام کد زنی برای اجرای کل نوتبوک از تب Tensorboard در بالا گزینه all Runtime را انتخاب کرده تا همه سلوک های کد اجرا شوند و فایل های Tensorboard ایجاد و دانلود شوند. همچنین نتایج و عکس های مربوط به نتایج حاصل از اجرای کد در این نوتبوک، که در فایل های Tensorboard ذخیره شده بود دانلود شده و در کنار گزارش آمده است. همچنین عکس مربوط به نتایج آنها نیز در گزارش آمده است. همچنین فایل نوتبوک نوشته شده در محیط Google Colab در ضمیمه کار این گزارش به نام MNIST_CNN آمده است. همانطور که مشخص است، استفاده از یکی از منظم ساز های BN و یا Dropout نتیجه دقت شبکه را به طرز قابل توجهی نسبت به عدم استفاده از آنها افزایش می دهد.



شكل ٥: مراحل کار با Google Colab

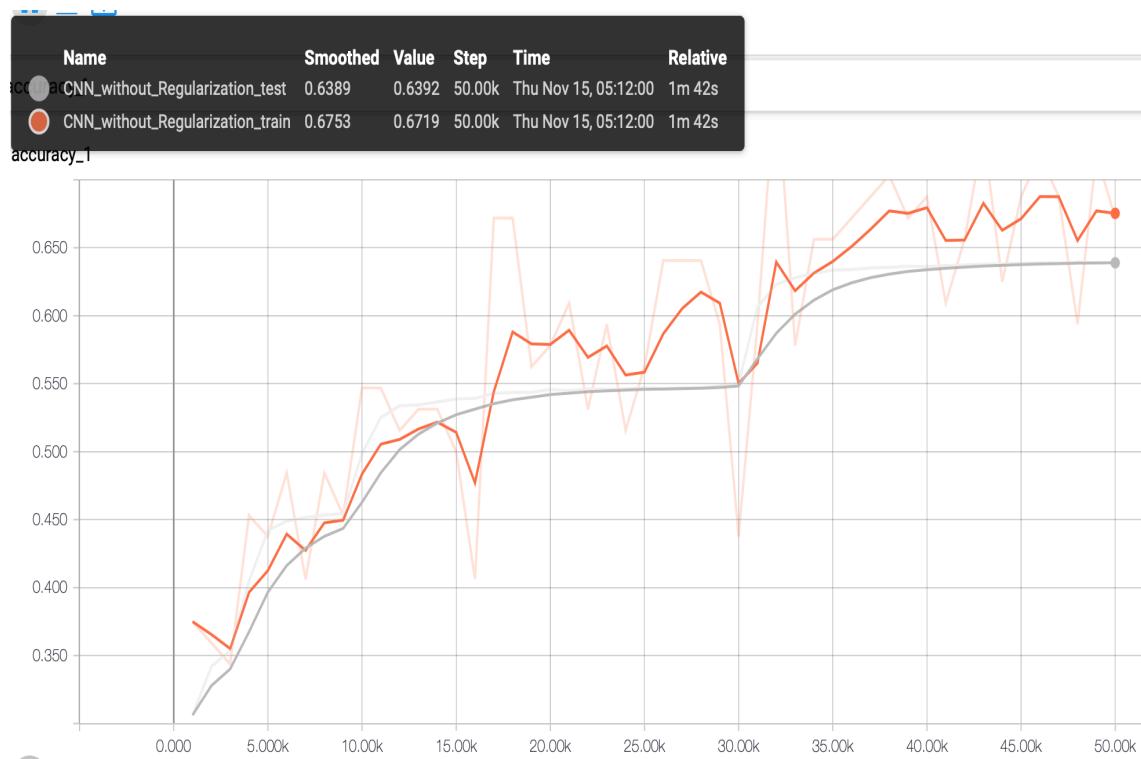


شكل ٦: مراحل کار با Google Colab

Cross_Entropy_1



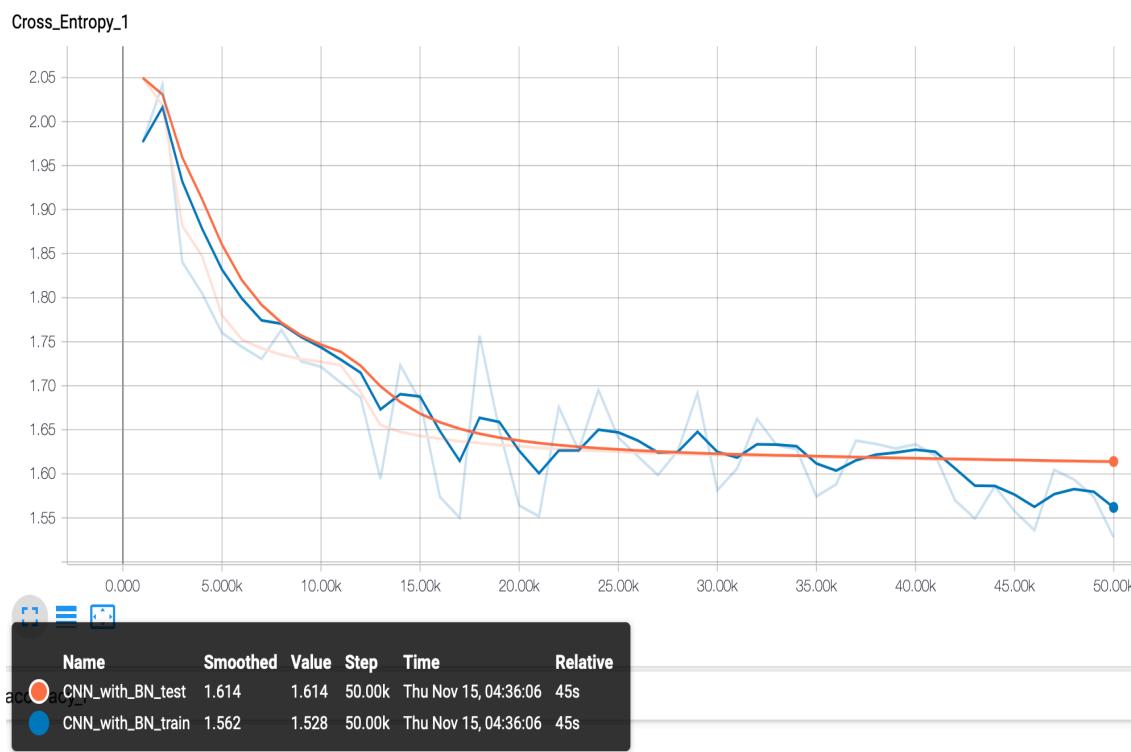
شکل ۷: نتیجه تابع هزینه بدون استفاده از منظم ساز



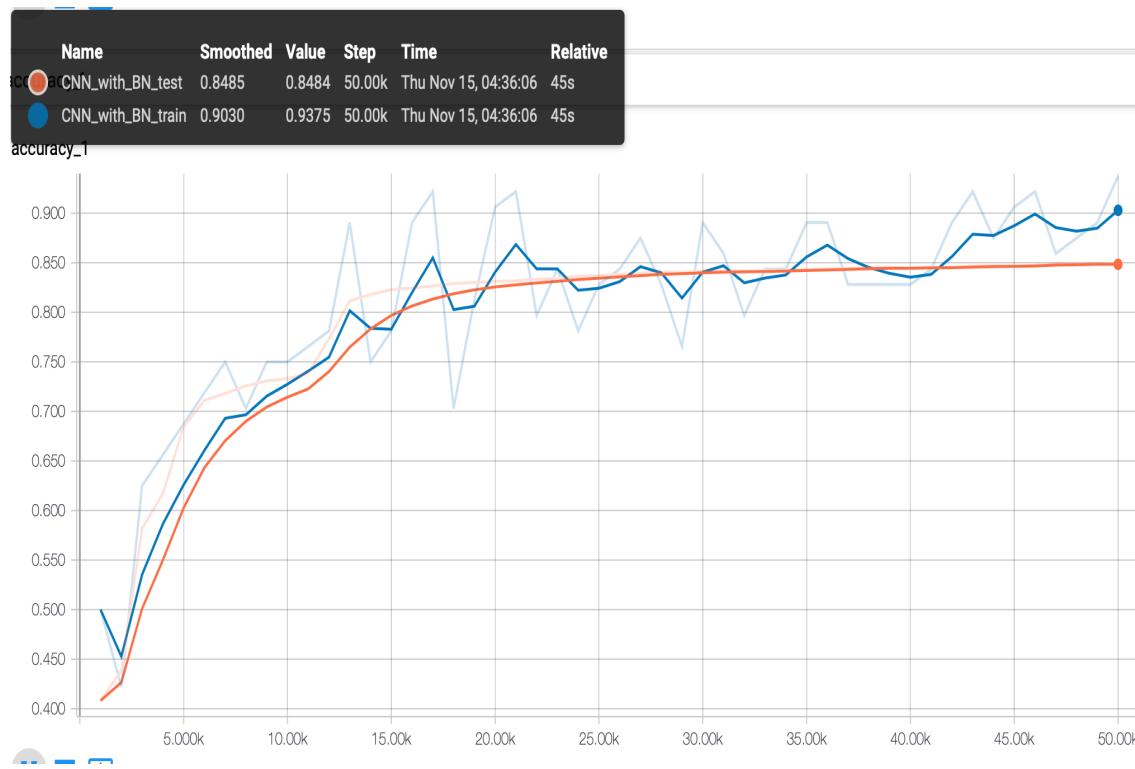
شکل ۸: نتیجه دقت بدون استفاده از منظم ساز

۲.۲ مقایسه در حالت وجود یا عدم وجود منظم سازها

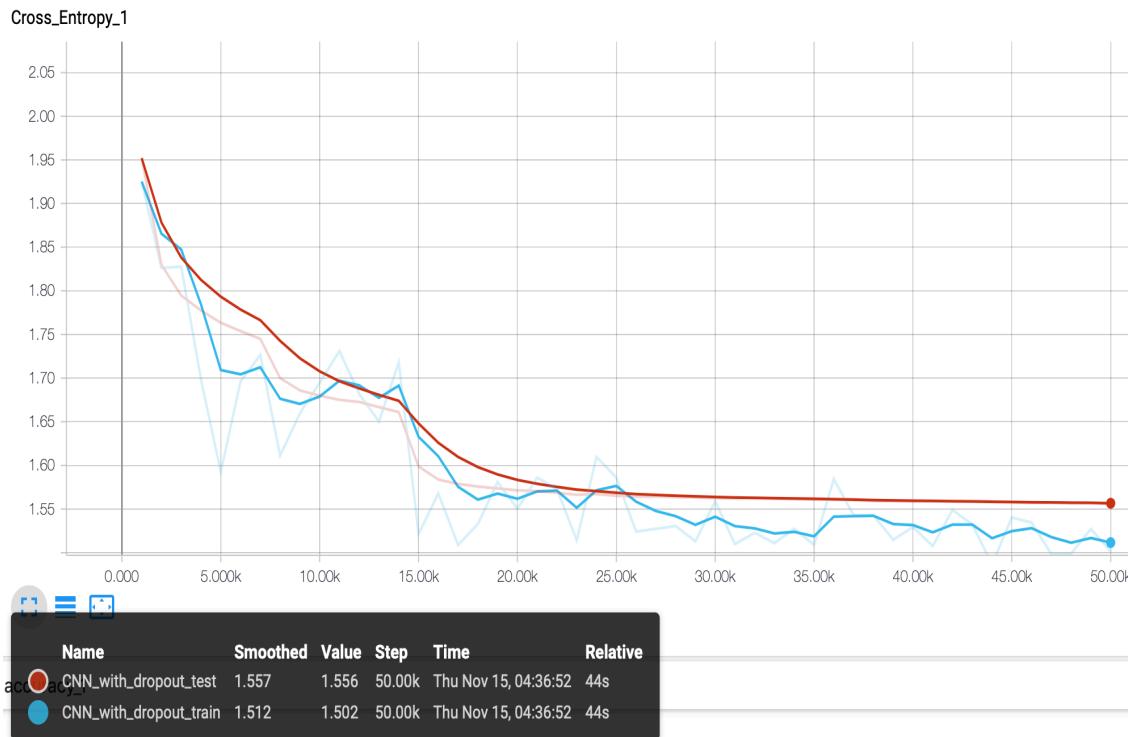
عکس مربوط به مقایسه‌ها و نتایج در زیر آمده است.



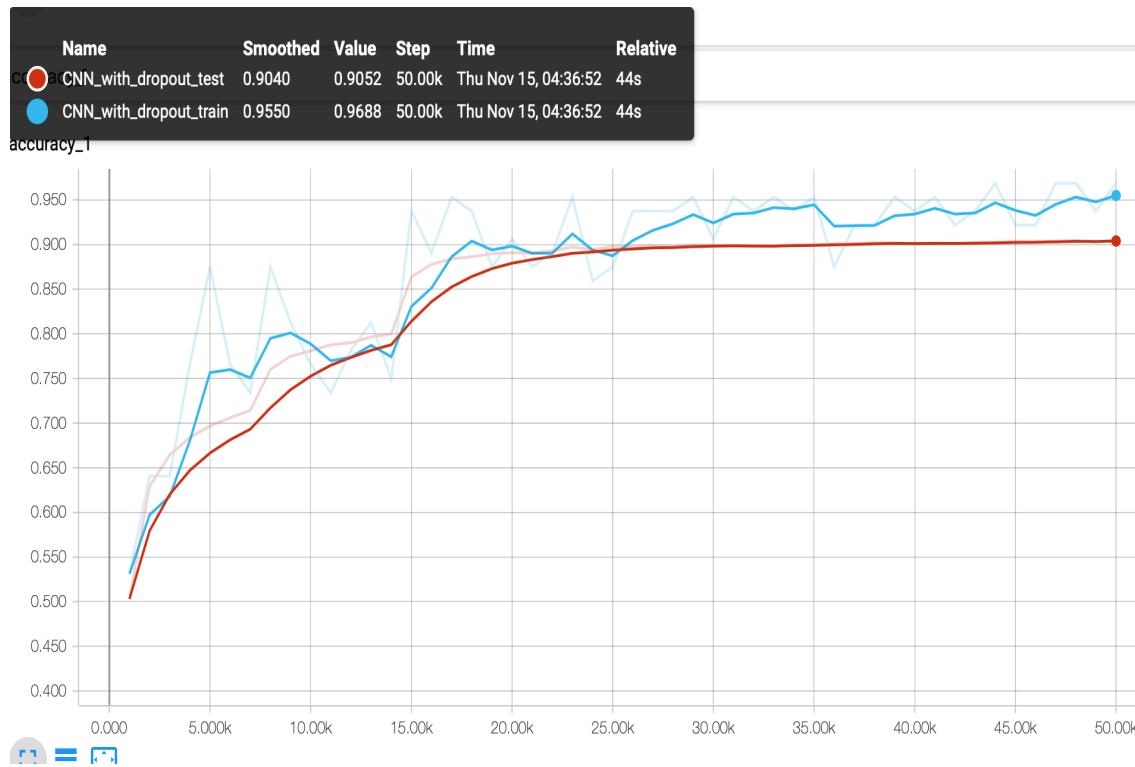
شکل ۹: نتیجه تابع هزینه با استفاده از BN



شکل ۱۰: نتیجه دقت با استفاده از BN



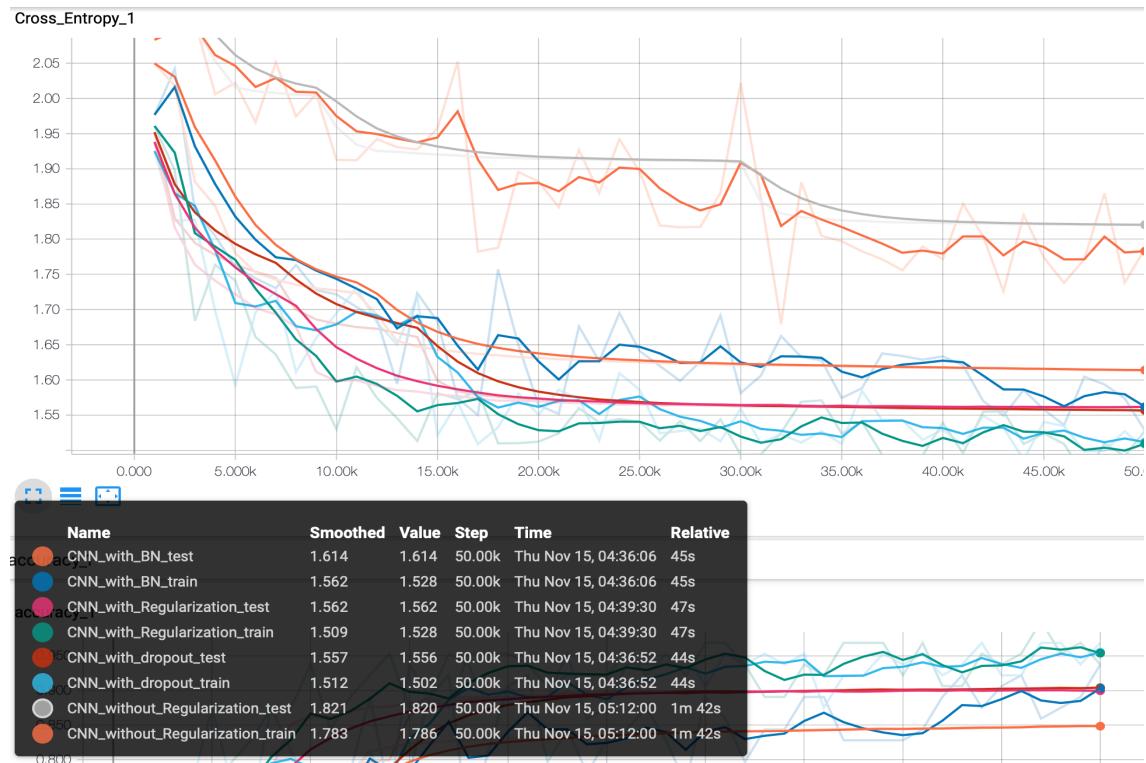
شکل ۱۱: نتیجه تابع هزینه با استفاده از Dropout



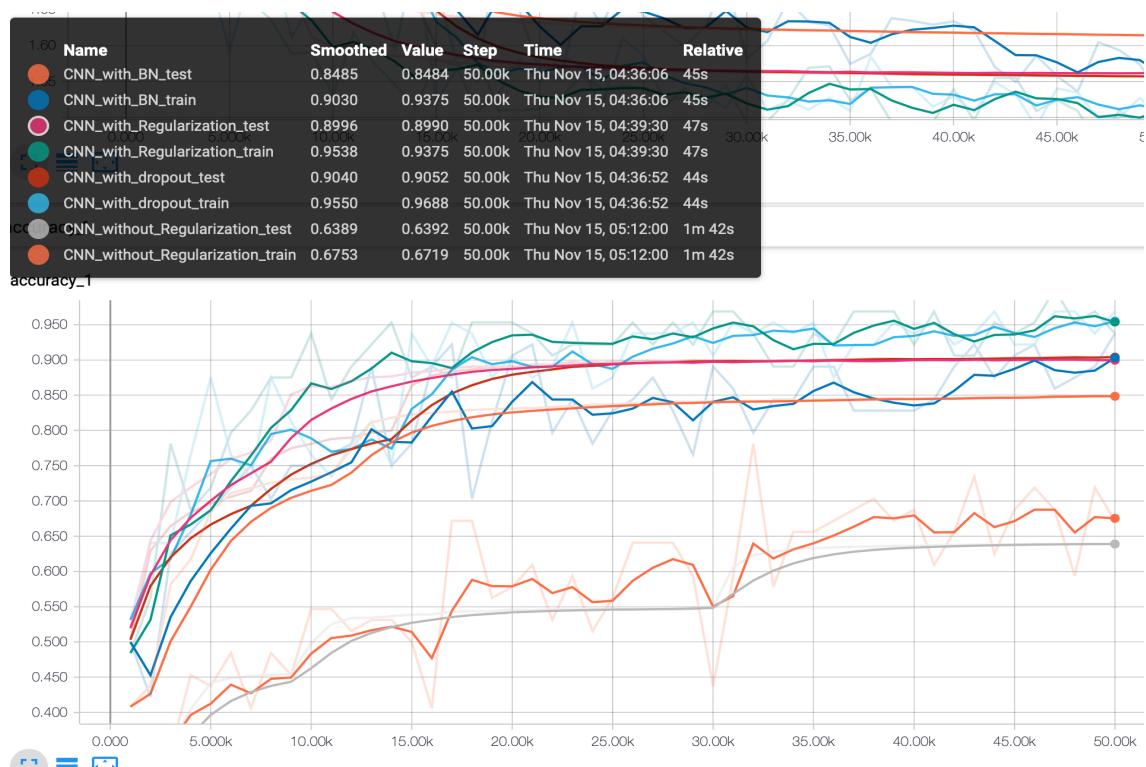
شکل ۱۲: نتیجه دقت با استفاده از Dropout

۳.۲ گزارش نتیجه در اثر وجود دو منظم ساز

عکس مربوط به مقایسه‌ها و نتایج در زیر آمده است. همانطور که از شکل مشخص است، نتیجه دقت شبکه در صورت استفاده از هر دو منظم‌ساز، نسبت به استفاده از یکی از آنها و یا عدم استفاده از هیچ یک، بهتر شده و دقت شبکه کمی بالاتر رفته است.



شکل ۱۳: نتیجه تابع هزینه با استفاده دو روش منظم سازی



شکل ۱۴: نتیجه دقت با استفاده دو روش منظم سازی

۱.۳ توضیح در مورد شبکه VGG

این شبکه در سال ۲۰۱۴ در کنفرانس ICLR معرفی شد. این شبکه دو نوع VGG16 و VGG19 دارد که به ترتیب از ۱۶ و ۱۹ لایه تشکیل شده‌اند. در این شبکه‌ها فیلترهای وزن در لایه‌های کانولوشنی بسیار کوچک می‌باشند و در سایز ۳ در ۳ می‌باشند که طبق گفته مقاله این شبکه، این فیلترها باعث عمق‌تر کردن شبکه و عین حال نتیجه بهتر نسبت به مدل‌های مشابه گرفتند، می‌باشند. در واقع وجود این فیلترها باعث شده تا تعداد لایه‌های شبکه تا ۱۶ یا ۱۹ لایه پیش‌برود و دقت آن نیز نسبت به مدل‌های مشابه بهتر باشد. این شبکه برندۀ مسابقه Localization IMAGENET Challenge در تسکن در سال ۲۰۱۴ مقام دومی در تسکن Classification در همان سال می‌باشد. این مسابقه که هرساله برگزار می‌شود دارای دیتای بسیار معروفی به نام ILSVRC باشد. این شبکه با پیشران عمق خود با استفاده از فیلترهای سایز فیلترهای کانولوشن و بسیار کوچک بودن سایز آن سعی در بهتر کردن دقت خود داشته است و در این زمینه نیز موفق بوده است. طبق گفته نویسندهای این مقاله، این شبکه نه تنها برای دیتای مسابقه ILSVRC بسیار خوب عمل می‌کند بلکه روی دیتای مسابقه‌های دیگر نیز بسیار خود عمل کرده است. این شبکه علاوه بر دیتای ILSVRC در مقاله خود بر روی دیتاهای VOC-2007, VOC-2012, Caltech-101 و Caltech-256 تست شده است. نتایج مربوط به دقت این شبکه در جداول زیر آمده است. این شبکه برای ارزیابی روی دیتای ILSVRC از دو معیار Top-1 Error و Top-5 Error استفاده کرده است که اولی نسبت تعداد عکس‌های به اشتباه طبقه‌بندی شده در داده تست است ولی دومی نسبت تعداد عکس‌هایی به کل است که کلاس درست برای این عکس‌ها در ۵ کلاس اول محتمل که شبکه پیش‌بینی کرده است می‌باشد. بنابراین مشخص است که خطای Top-5 نسبت به خطای Top-1 همیشه مقدار کمتری برای شبکه‌های مختلف خواهد داشت. علت وجود این نوع جدید از خطای نیز وجود ۱۰۰۰ کلاس در دیتای ILSVRC می‌باشد که تعداد خیلی زیادی است و شهود آن به این معنی است که اگر شبکه برای عکس ورودی از این ۱۰۰۰ کلاس، کلاس درست را در ۵ کلاس محتمل ترین برای یک عکس بیند، قابل قبول است و خطای نیست. نتایج مربوط به این خطای نیز برای این دیتا در جدول زیر آمده است. همانطور که از جداول نیز مشخص است، دقت این شبکه روی دیتای اصلی، در حدود ۷۶ درصد برای Top-1 و در حدود ۹۳ درصد برای Top-5 می‌باشد.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	23.7	6.8	6.8
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-		7.9
GoogLeNet (Szegedy et al., 2014) (7 nets)	-		6.7
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

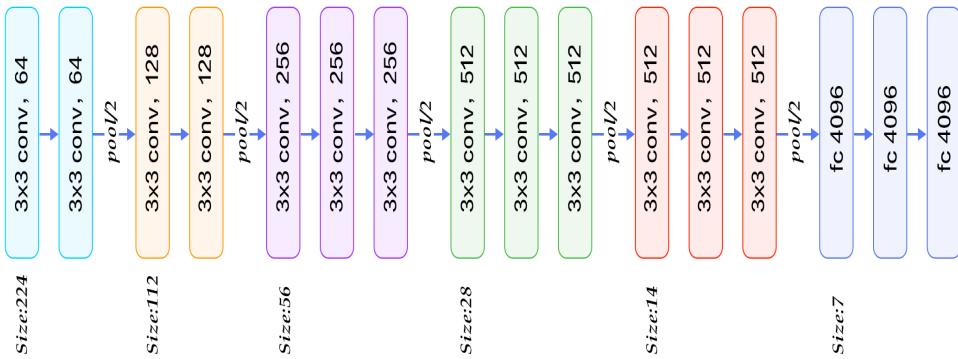
شکل ۱۵: جدول مربوط نتایج شبکه VGG روی دیتای ILSVRC

Method	VOC-2007 (mean AP)	VOC-2012 (mean AP)	Caltech-101 (mean class recall)	Caltech-256 (mean class recall)
Zeiler & Fergus (Zeiler & Fergus, 2013)	-	79.0	86.5 ± 0.5	74.2 ± 0.3
Chatfield et al. (Chatfield et al., 2014)	82.4	83.2	88.4 ± 0.6	77.6 ± 0.1
He et al. (He et al., 2014)	82.4	-	93.4 ± 0.5	-
Wei et al. (Wei et al., 2014)	81.5 (85.2*)	81.7 (90.3*)	-	-
VGG Net-D (16 layers)	89.3	89.0	91.8 ± 1.0	85.0 ± 0.2
VGG Net-E (19 layers)	89.3	89.0	92.3 ± 0.5	85.1 ± 0.3
VGG Net-D & Net-E	89.7	89.3	92.7 ± 0.5	86.2 ± 0.3

شکل ۱۶: جدول مربوط نتایج شبکه VGG روی دیتاهای دیگر

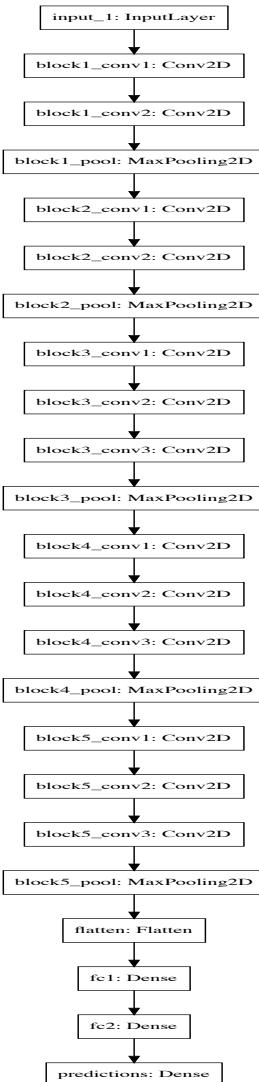
۲.۳ توضیح معماری شبکه VGG

این شبکه از ۱۶ لایه تشکیل شده است. عکس مربوط به معماری شبکه و پارامترهای هر لایه و ابعاد هر لایه در عکس دوم قابل مشاهده است. این شبکه در کل حدود ۱۳۸ میلیون پارامتر قابل یادگیری دارد. همچنین نوع لایه‌ها در عکس دوم و اول قابل مشاهده است.



شکل ۱۷: معماری شبکه VGG16

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
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
Total params: 138,357,544		
Trainable params: 138,357,544		
Non-trainable params: 0		

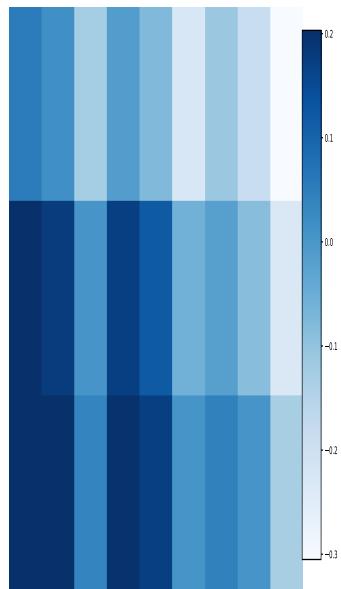


شکل ۱۸: ابعاد وزن‌ها و ورودی و تعداد پارامترهای قابل بادگیری در لایه‌های مختلف در شبکه VGG16

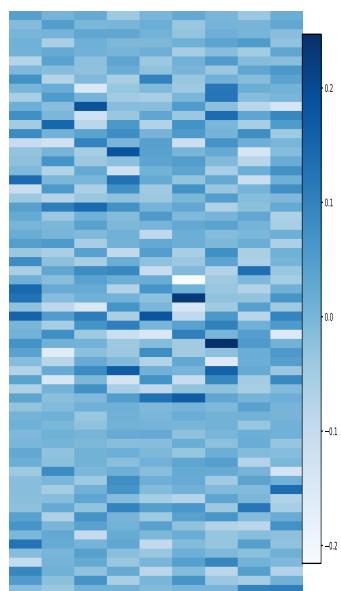
۳.۳ گزارش فیلترهای هر لایه و مقایسه اولین و آخرین فیلتر

با توجه به اینکه عکس مربوط به خروجی وزن‌های فیلتر هر لایه بسیار زیاد است، تمامی این عکس‌ها در پوشش‌هایی با نام‌های هر بلاک شبکه VGG16 ذخیره شده‌اند. در اینجا تنها چند نمونه از این خروجی‌ها را در گزارش می‌آوریم اما برای دسترسی به همه فیلترها لطفاً به پوشش‌های همراه گزارش مراجعه بفرمایید. در مقایسه فیلتر لایه اول و آخر می‌توان گفت، از آنجایی که در لایه آخر اعداد فیلترها هر کدام در بازه $0..0.03$ تا $0..0.02$ هستند اما اعداد فیلتر لایه اول از بازه $0..0.02$ هستند، پس اعداد این فیلتر ویژگی‌های با معنی تر و ساده‌تری را از ورودی خود که عکس باشد، استخراج می‌کنند در حالی که هر چه به سمت لایه‌های آخر نزدیک‌تر می‌شویم، این ویژگی‌های استخراج شده پیچیده‌تر و پیچیده‌تر می‌شوند. این مطلب از میزان حساسیت فیلترهای لایه اول و آخر معلوم است. پس بازه اعداد فیلتر لایه آخر کمتر و ویژگی استخراج شده توسط این لایه نیز بسیار پیچیده‌تر می‌باشد.

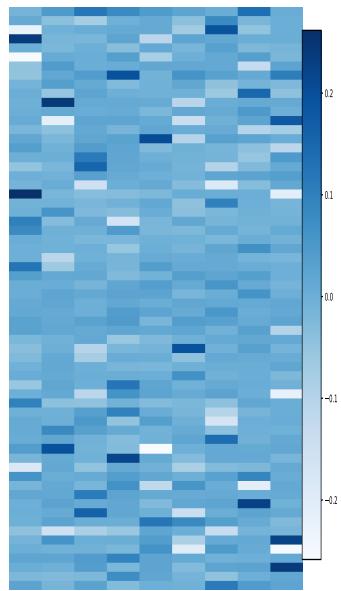
لازم به ذکر است که در عکس‌های بالا و عکس‌های ضمیمه شده، تمام فیلترهای ۳ در ۳ به صورت یک تابع نمایش داده شده‌اند، یعنی اگر یک فیلتر لایه سوم در ۳ در ۳ می‌باشد، این عکس به صورت ۲۵۶ در ۹ نمایش داده شده‌است. برای هر فیلتر نیز یک عکس جدا نمایش داده شده است.



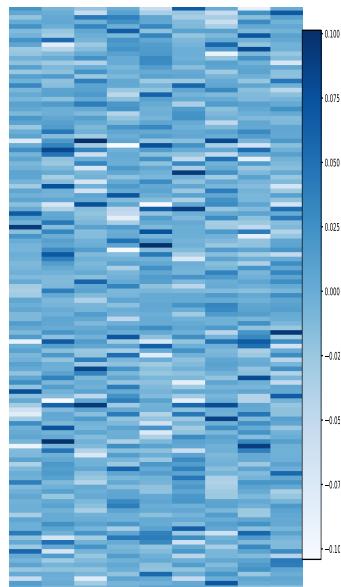
شکل ۱۹: فیلتر سیزدهم لایه block1_conv1



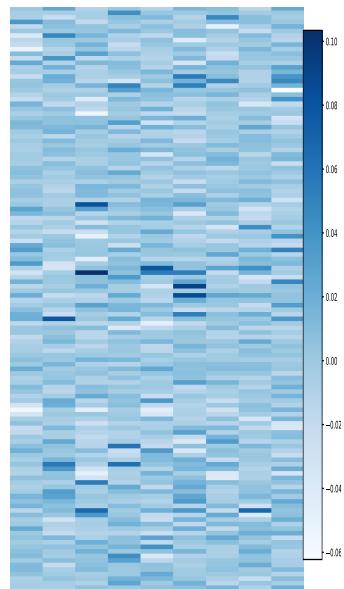
شکل ۲۰: فیلتر نوزدهم لایه block1_conv2



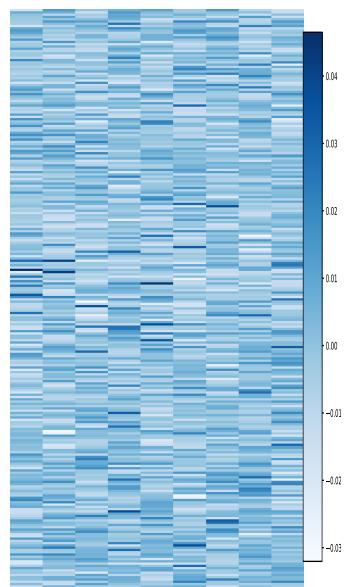
شکل ۲۱: فیلتر دوازدهم لایه block2_conv1



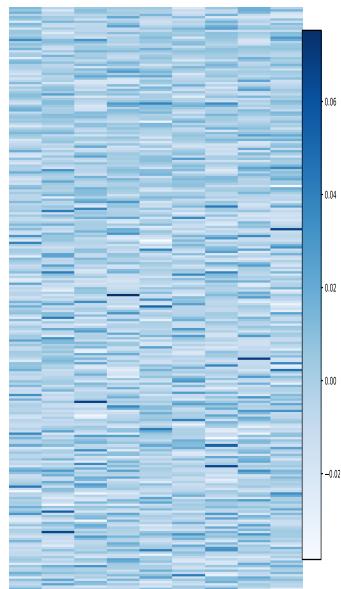
شکل ۲۲: فیلتر هشتم لایه block2_conv2



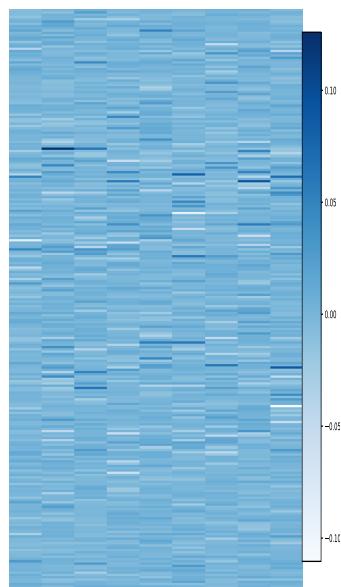
شكل ٢٣: فیلتر پنجه و یکم لایه block3_conv1



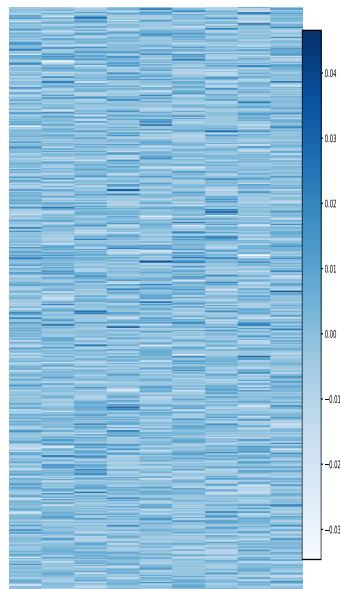
شكل ٢٤: فیلتر پنجه و یکم لایه block3_conv2



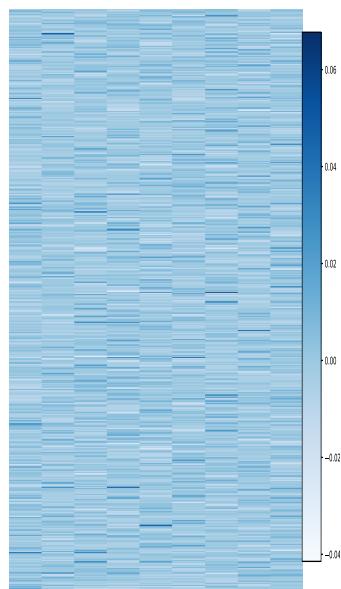
شكل ٢٥: فیلتر پنجاه و یکم لایه block3_conv3



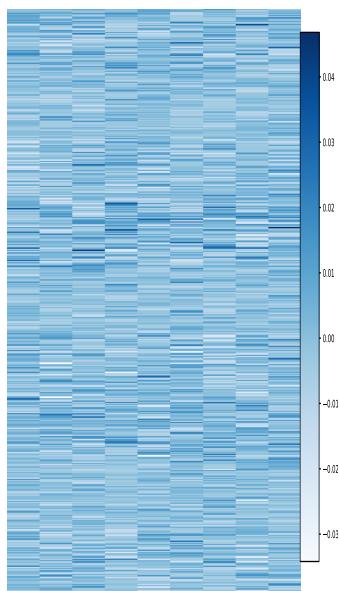
شكل ٢٦: فیلتر پنجاه و یکم لایه block4_conv1



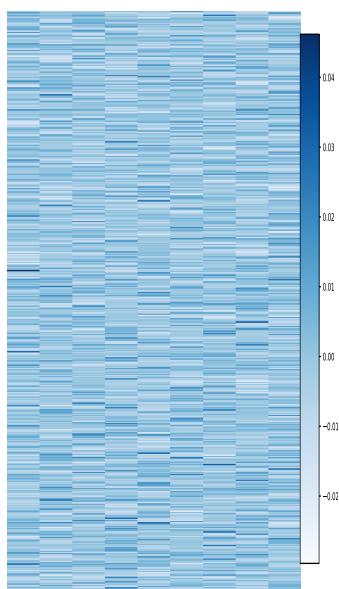
شكل ٢٧: فیلتر پنجاه و یکم لاپه block4_conv2



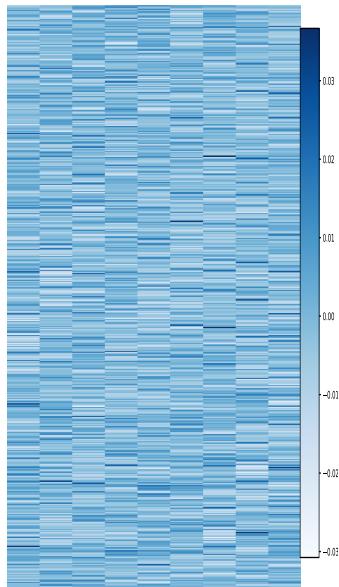
شكل ٢٨: فیلتر پنجاه و یکم لاپه block4_conv3



شكل ٢٩: فیلتر پنجاه و یکم لایه block5_conv1



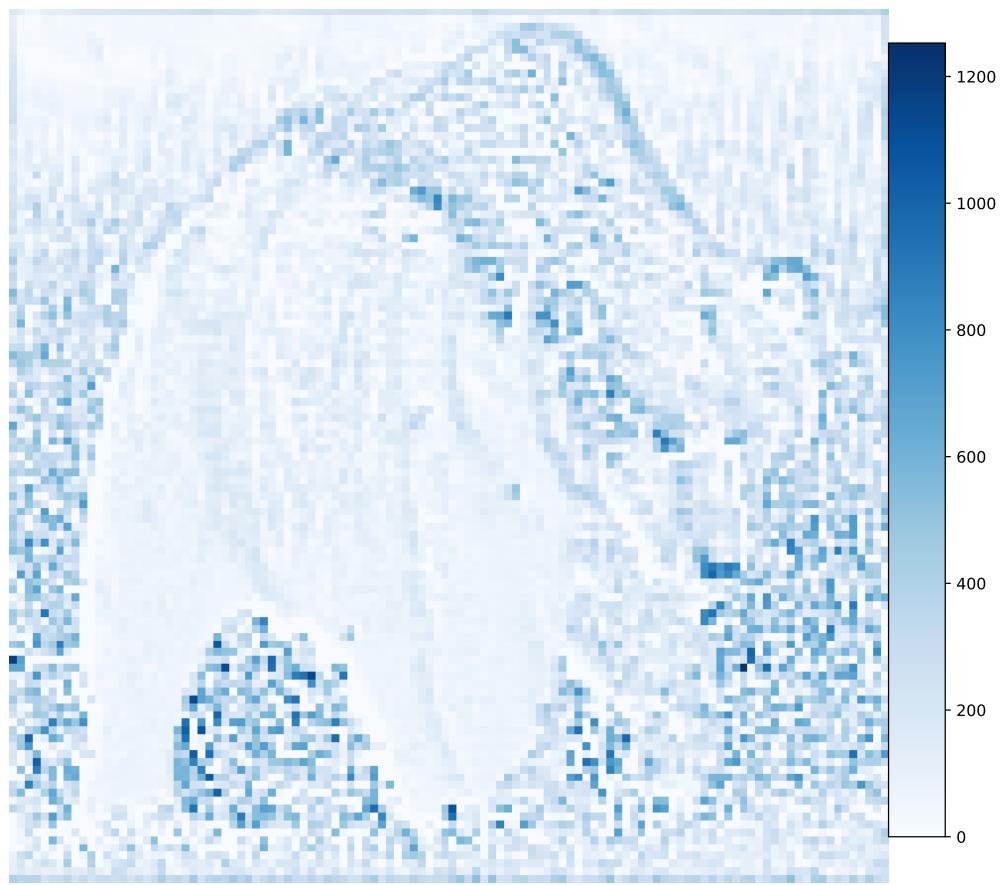
شكل ٣٠: فیلتر پنجاه و یکم لایه block5_conv2



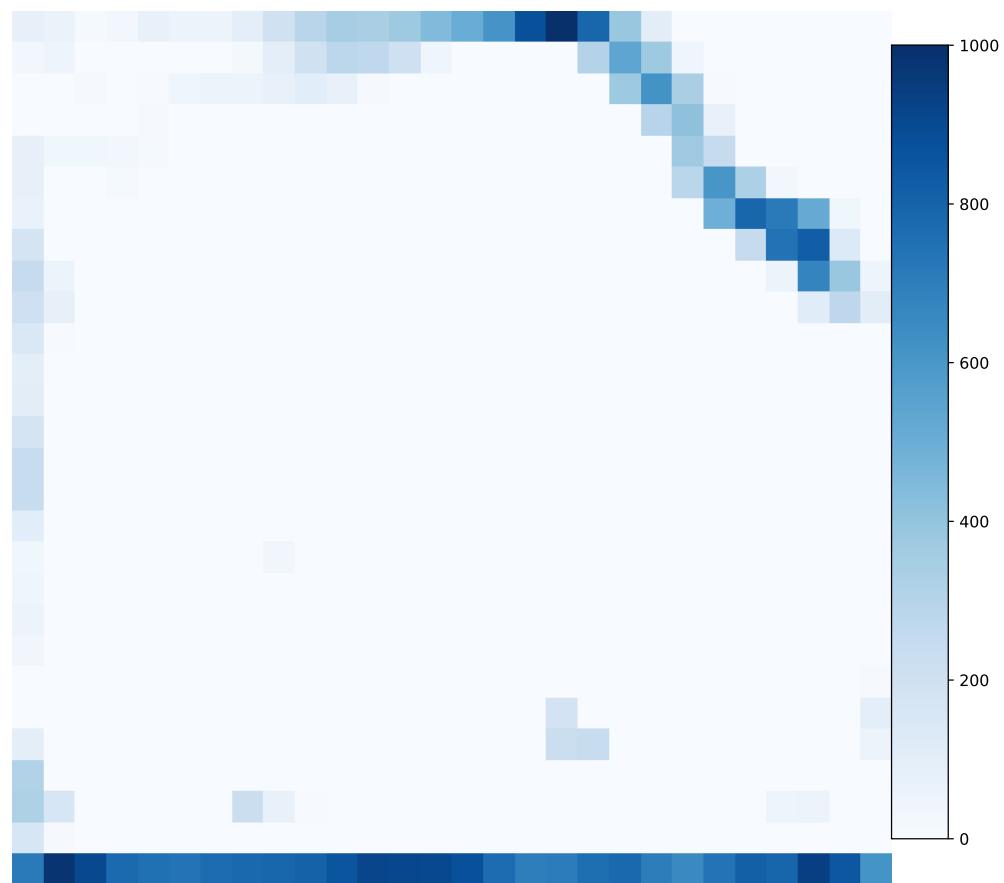
شکل ۳۱: فیلتر دوم لایه 3 block5_conv3

۴.۳ تحلیل نتایج لایه های ۳ و ۱۳ حاصل از ورودی های جدید شبکه

تصویر مربوط به یکی از فیلترهای لایه سوم و سیزدهم برای هر تصویر در زیر آمده است. همانطور که مشخص قابل مشاهده است، لایه سوم که لایه کم عمق تری نسبت به لایه سیزدهم است، ویژگی های قابل مشاهده تری نسبت به لایه سیزدهم استخراج کرده است و تک Edge Detection در این لایه به مراتب قابل مشاهده تراز لایه سیزدهم است. در واقع این لایه ویژگی های ساده و مشخص تری را ابتدا از تصویر استخراج می کند و در ورودی بعد از گلدر از چند لایه به لایه سیزدهم می دهد و لایه سیزدهم از ورودی استخراج شده از تصویر، ویژگی جدیدتری را استخراج می کند که این ویژگی به شدت Abstract تراز ویژگی های استخراج شده از لایه سوم است. این مطلب در تمام فیلترها برای ۶ عکس ورودی قابل مشاهده است. در نتیجه می توان گفت که نورون های لایه سوم، وظیفه استخراج کردن ویژگی های ساده تری نسبت به نورون های لایه های بعد از خود از جمله لایه سیزدهم را دارند. پس هر چه شکله عمیق تر می شود، ویژگی های استخراج شده دارای تجد و پیچیدگی بیشتری می باشند که توسط انسان قابل تشخیص به صورت آسان نیستند. خروجی لایه سیزدهم و سوم برای هر عکس نیز در ضمیمه در پوشه imagesFilters آمده است.



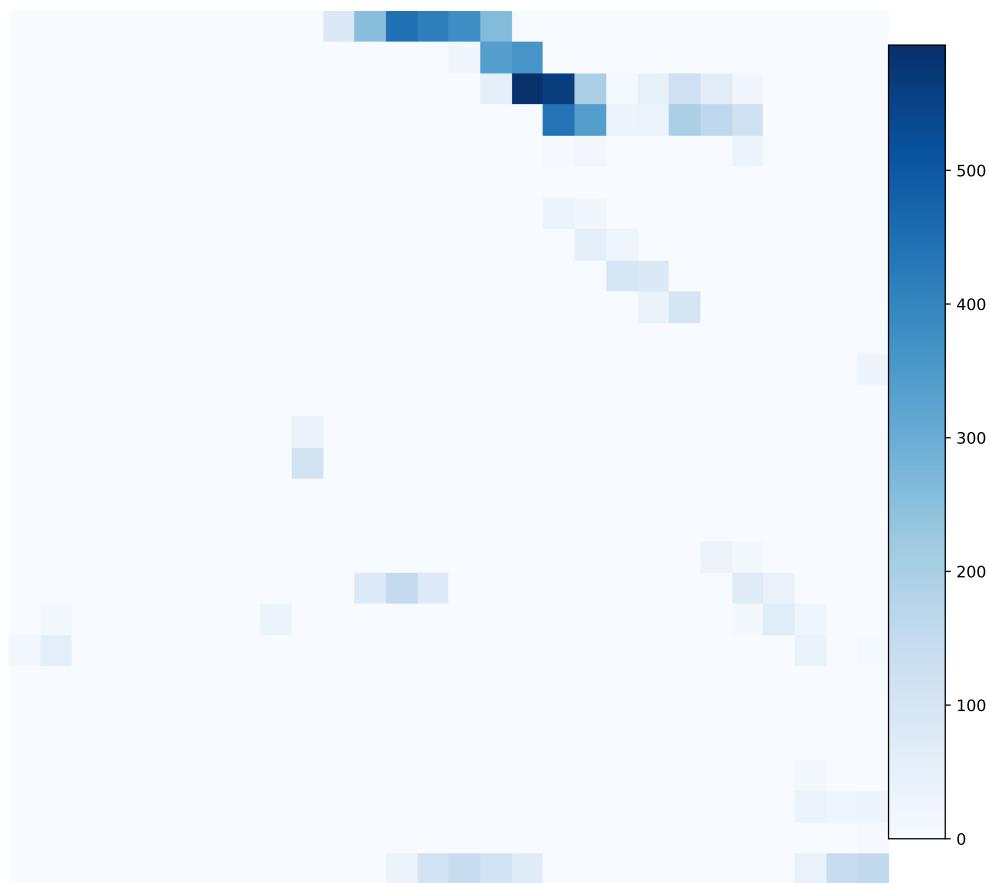
شکل ۳۲: خروجی فیلتر ششم لایه سوم عکس brown_bear



شکل ۳۳: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس brown_bear



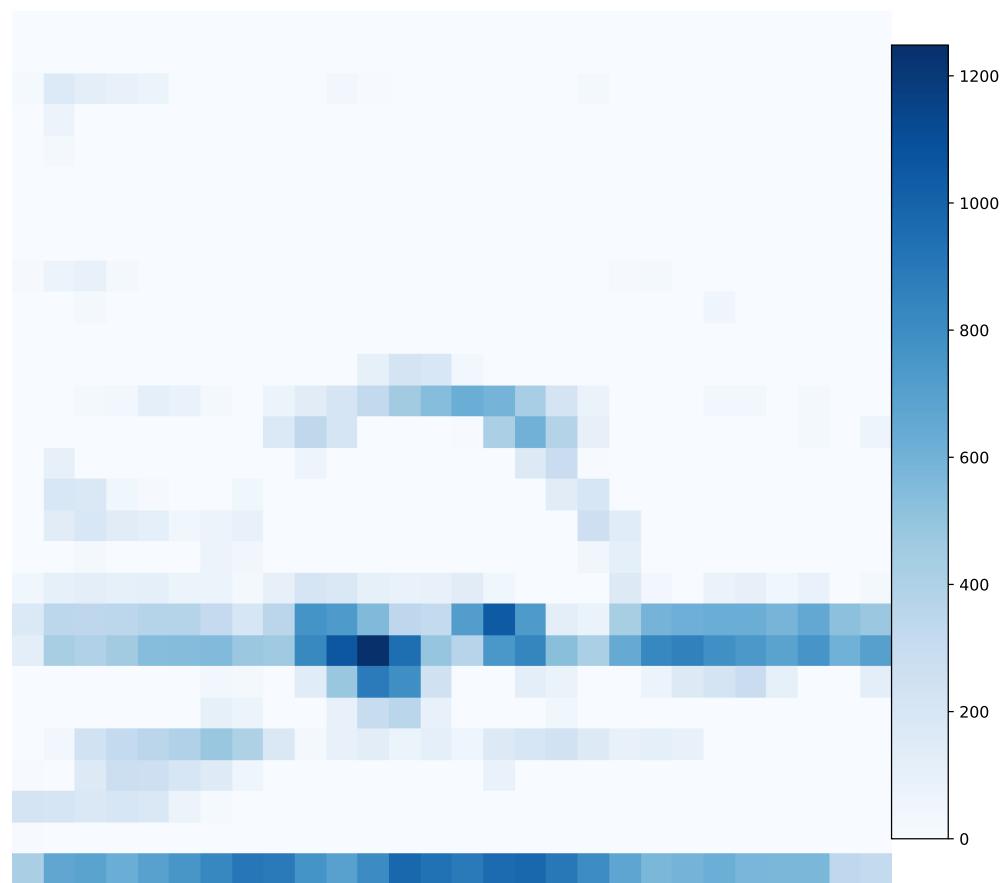
شکل ۳۴: خروجی فیلتر ششم لایه سوم عکس cat_dog



شکل ۳۵: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس cat_dog



شکل ۳۶: خروجی فیلتر ششم لایه سوم عکس dd_tree



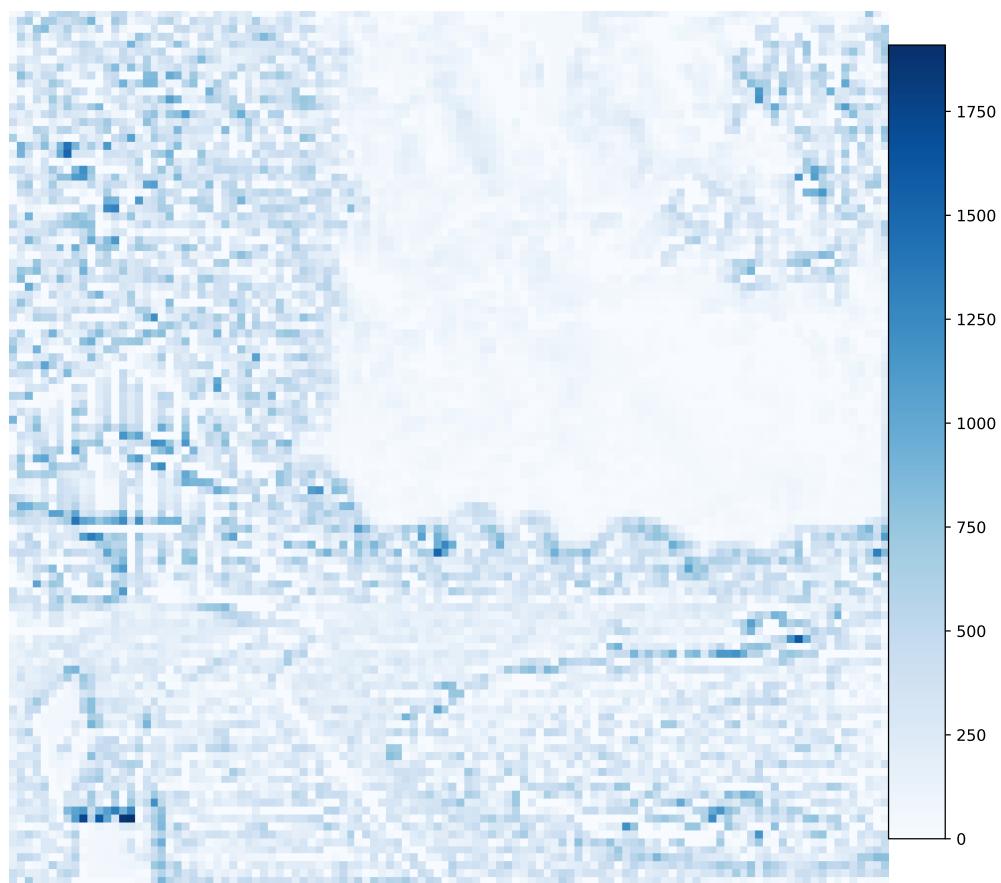
شکل ۳۷: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس dd_tree



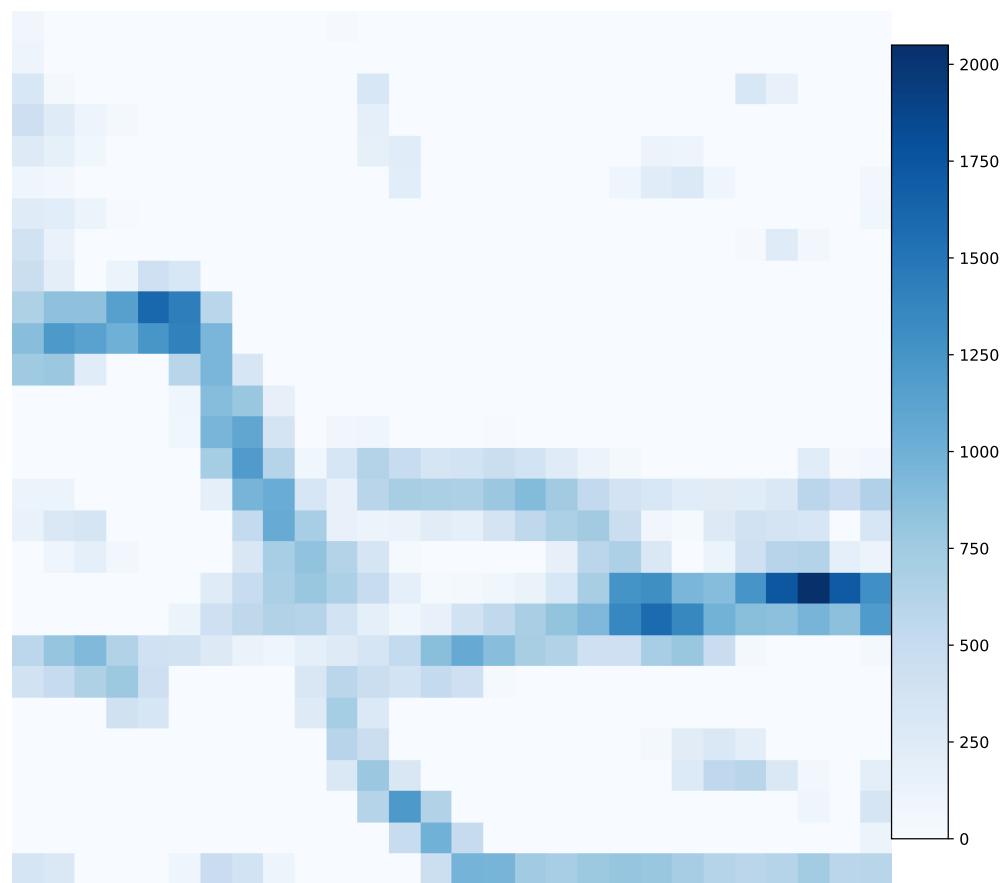
شکل ۳۸: خروجی فیلتر ششم لایه سوم عکس dog_beagle



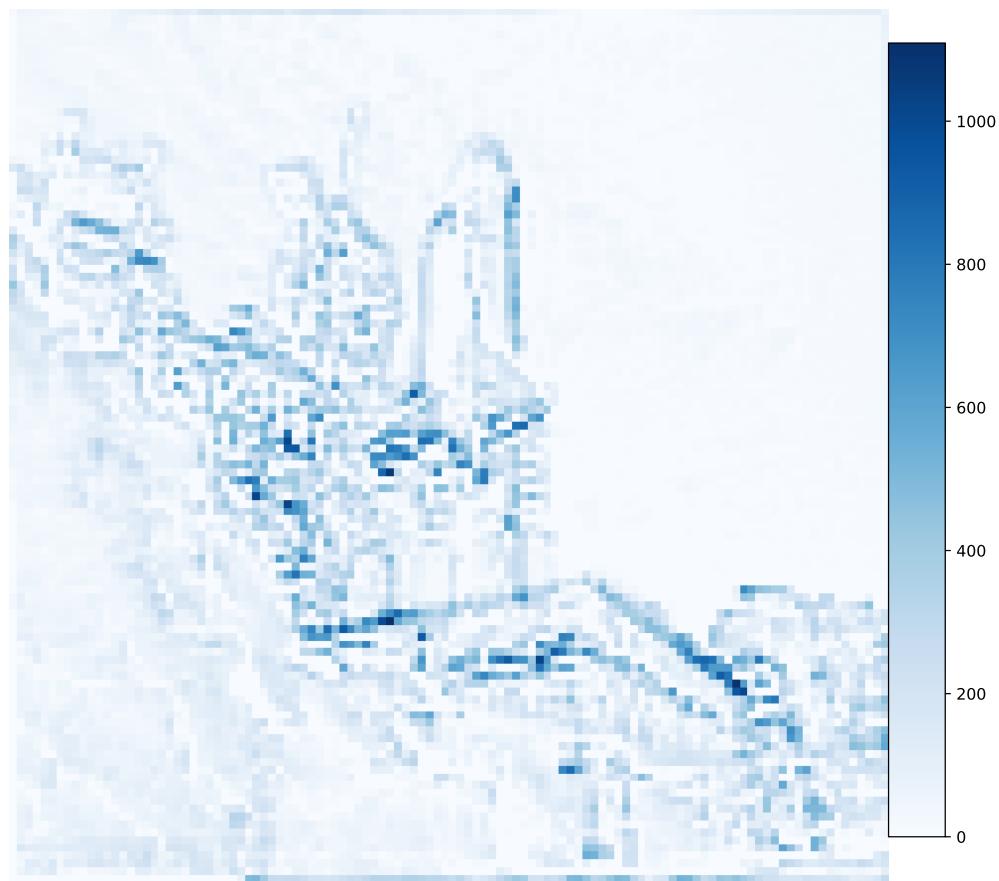
شکل ۳۹: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس dog_beagle



شکل ۴۰: خروجی فیلتر ششم لایه سوم عکس scenery



شکل ۴۱: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس scenery



شکل ٤٢: خروجی فیلتر ششم لایه سوم عکس space_shuttle

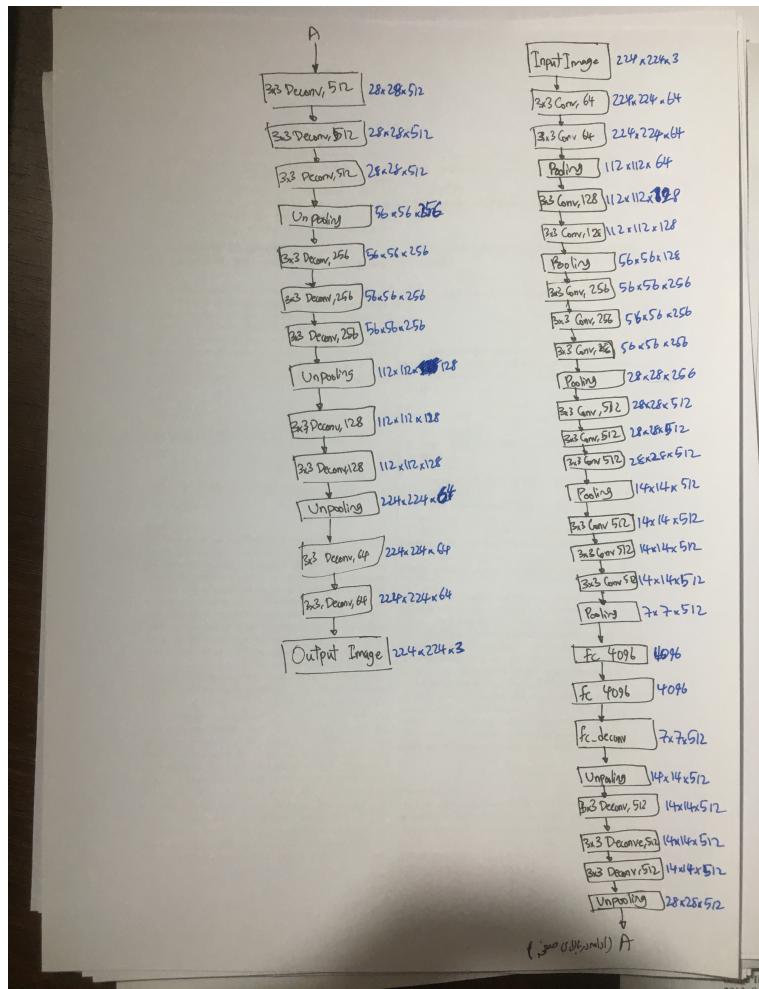


شکل ۴۳: خروجی فیلتر بیست و هفتم لایه سیزدهم عکس space_shuttle

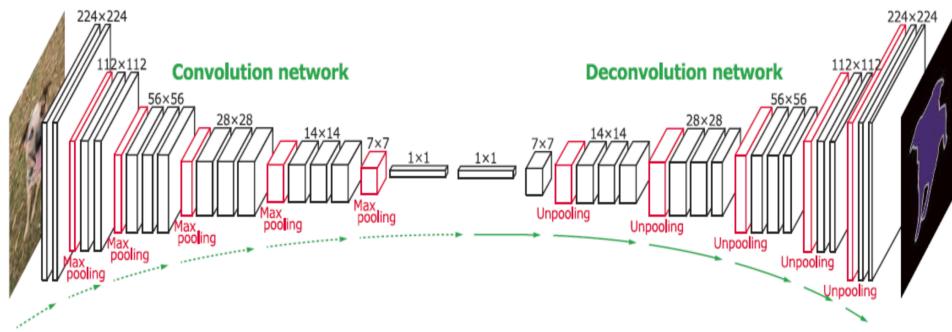
DeConvolution ۴

۱.۴ رسم شبکه عصبی و مشخصات هر لایه

همانطور که از کد مشخص است، این شبکه، در قسمت encoder همان شبکه VGG16 می‌باشد که معماری و مشخصات آن در سوال دوم کشیده و توضیح داده شد. این شبکه در قسمت خود، دقیقاً بر عکس VGG عمل می‌کند یعنی به ازای هر لایه Convolution یک لایه Deconvolution و به ازای Pooling یک لایه Unpooling گذاشته شده است و در نهایت در خروجی شبکه، سایز تصویر برابر سایز ورودی شبکه می‌باشد.



شکل ۴۴: عکس مربوط به رسم شبکه عصبی موجود در کد net.py که همان شبکه معروف Deconvnet است.



شکل ۴۵: عکس مربوط به شبکه Deconvnet در اینترنت

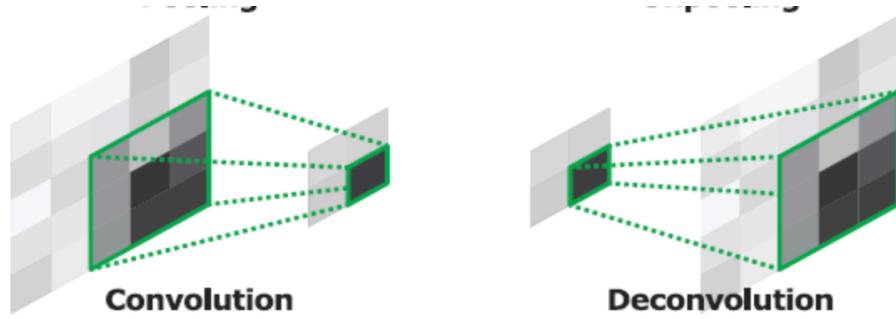
۲.۴ کاربر در شبکه های عمیق

باتوجه به اینکه یک شی در تصویر معمولاً خیلی بزرگتر یا خیلی کوچکتر از Receptive Field می‌باشد، که این باعث می‌شود که به مشکل Fragmentation و یا Mislabeled Objects بخوریم. شی‌های کوچک معمولاً در مسئله‌های Classification توسط شبکه ignore می‌شوند و به عنوان Background می‌شوند. در آنها برخورد می‌شود. به همین منظور با مسئله Instance-wise Segmentation Semantic Segmentation برخورد می‌شود. در

ابتدا ۵۰ از ۲۰۰۰ تا توسط الگوریتم EdgeBox مانند Object Detection انتخاب می‌شوند و سپس شبکه Deconvnet به هر عامل می‌شود و خروجی همه پروپوزال‌ها باهم Aggregate می‌شوند. بنابراین شبکه Deconvnet یک کاربرد بزرگ و خیلی مهم در مسئله Segmentation دارد. در واقع مسئله خیلی کوچک یا بزرگ بودن شی در تصویر نسبت به Receptive Field در FCN را که باعث Segmentation پرخطا می‌شود، شبکه Deconvnet به خوبی حل می‌کند و به دقت خیلی بهتری می‌رسد. همچنین این شبکه ویژگی‌های Generative تری را نسبت به شبکه‌های دیگر در لایه‌های خود یادمی گیرد.

۳.۴ نحوه عملکرد این لایه و تفاوت با لایه Convolution

عملکرد لایه DeConvolution درست بر عکس لایه Convolution می‌باشد. این لایه در واقع از عکس با سایز کوچکتر عملیات Conv بر عکس را انجام می‌دهد و به عکس با سایز بزرگتر می‌رسد. در واقع در این عملیات، Upsampling رخ می‌دهد که برخلاف Unpooling قابل یادگیری است. تصویر زیر عملکرد لایه DeConvolution را نشان می‌دهد. در واقع ما در Convolution، عملیات DownSampling را انجام می‌دهیم اما در DeConvolution به صورت عکس عمل می‌کنیم و Upsampling را انجام می‌دهیم.



شکل ۴۶: تفاوت لایه convolution با لایه DeConvolution