

به نام خدا



درس: یادگیری عمیق

استاد: دکتر فاطمیزاده

گزارش تمرین عملی شماره ۵

سیدمحمدامین منصوری طهرانی

۹۴۱۰۵۱۷۴

۱. Batch Normalization

۱،۱ همان‌طور که در مقاله اصلی به آن اشاره شده است و در درس نیز مورد بررسی قرار گرفت، مسأله مهمی در آموزش شبکه‌های عمیق وجود دارد و آن تغییر توزیع داده‌های رسیده به هر لایه ناشی از وزن‌ها و تغییرات در لایه‌های قبلی است و این باعث می‌شود لایه‌ها نیاز داشته باشند تا به طور پیوسته با توزیع‌های جدید تطبیق پیدا کنند که مشخصاً سرعت و دقت آموزش را پایین می‌آورد. این مسأله به عنوان Covariate Shift شناخته می‌شود.

همچنین اگر توزیع ورودی لایه‌های مختلف به هم وابسته باشد و به عنوان مثال از از تابع فعالیت سیگموید استفاده کنیم، اگر در لایه‌ای به ناحیه اشباع برویم، احتمال اشباع شدن لایه‌های بعدی نیز بالا بوده و با توجه به نزدیکی گرادیان به صفر در آن نواحی، با آموزش کند موافق می‌شویم. این مسأله البته با تعویض سیگموید با ReLU قابل حل می‌باشد (به همراه نرخ یادگیری کم).

در روش Batch Normalization، از مزایای زیر بهره می‌بریم:

- میانگین و واریانس لایه‌های متوالی ثابت شده و به پارامترهای قبلی و مقدار اولیه آن‌ها وابسته نخواهیم بود.
- مشکل فلوی گرادیان که در بالا اشاره شد با کاهش این وابستگی‌ها بهبود می‌یابد. (که خود اجازه استفاده از نرخ یادگیری بالاتر را می‌دهد که منجر به آموزش سریع‌تر شبکه می‌شود.)
- به این طریق، به نوعی عملیات regularization نیز انجام شده و در موارد زیادی می‌توان از dropout هم استفاده نکرد.
- با توجه به ثابت بودن توزیع، می‌توان در قسمت‌های اشباع بعضی لایه‌ها بود، ولی به طور کلی مدل در مدهای اشباع گیر نیفتند.

۲،۱ در این روش میانگین هر بج و واریانس آن محاسبه شده و داده‌ها با این دو مقدار نرمال می‌شوند و بعد از آن در دو ضریب قابل آموزش ضرب می‌شوند:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}.$$

اندیس در رابطه فوق بیانگر شماره بعد ورودی است، در واقع عملیات گفته شده برای هر بعد انجام می‌شود. در لایه‌های کالولوشن، به ازای هر بج، میانگین و واریانس کل پیکسل‌های ناشی از اعمال هر کرنل روی هر بج محاسبه می‌شود. به عبارت دیگر برای هر کرنل به اندازه batch size تصویر داریم، روی همه این تصویرها میانگین و واریانس گرفته، به یک تصویر میانگین و واریانس می‌رسیم، روی تمام پیکسل‌های این عکس میانگین و واریانس گرفته و به دو عدد میانگین و واریانس کل برای بج می‌رسیم. با این دو عدد بجهای خروجی کرنل را نرمال می‌کنیم

و در پارامتر گاما ضرب و با پارامتر بتا جمع می‌کنیم، پس به ازای یک لایه کانولوشن با C کانال، $2C$ پارامتر داریم. برای لایه خروجی آن را نرمال کرده و به ازای هر بعد (نورون) این لایه دو پارامتر داریم.

بنابراین مجموعاً به این شبکه تعداد ۷۶۸ پارامتر اضافه شده است.

$$64 \times 2 + 64 \times 2 + 256 \times 2 = 768$$

۳،۱ کد این تابع در زیر آورده شده است و به طور خلاصه به توضیح آن می‌پردازم.

```
def batch_normalization(x, gamma, beta):
    if len(x.shape) >= 3:
        batch_mean, batch_var = tf.nn.moments(x, axes=[0, 1, 2], keep_dims=True)
    else:
        batch_mean, batch_var = tf.nn.moments(x, axes=[0], keep_dims=True)
    batch_normal = (x - batch_mean) / tf.sqrt(epsilon + batch_var)
    bn_output = tf.add(tf.multiply(batch_normal, gamma), beta)
    return bn_output
```

شرط اول برای میانگین‌گیری روی داده است وقتی بعد آن از ۳ بیشتر باشد. در این حالت همان‌طور که گفته شد بر روی همه داده‌های بج و سپس روی همه پیکسل‌های آن میانگین گرفته شده (به همراه واریانس) و در غیر این صورت فقط بر روی بعد اول داده‌ها این عملیات انجام می‌شود.(روی همه داده‌ها هر کدام فقط یک متغیر دارند). در ادامه داده با توجه به ϵ داده شده نرمال می‌شود و در ضرایب مناسب و قابل یادگیری γ, β ضرب و سپس جمع می‌شود. این ضرایب بردار هستند و ابعاد آن‌ها بالاتر شرح داده شد.

Dropout .۲

۱،۲ در این روش هر نورون در مدد آموزش به احتمال p در یادگیری حضور داشته و به احتمال $1-p$ خروجی آن صفر شده و در آموزش بی‌تأثیر خواهد بود. بنابراین شبکه‌ای با n نورون را می‌توان به عنوان یک شبکه با 2^n زیرساختار بدست آمده از **dropout** تصور نمود که بعضی از این ساختارها در طول آموزش به ندرت استفاده شده و تعدادی هم اصلاً استفاده نمی‌شوند. نکته مهم و جالب در مفهوم این روش آن است که در واقع با مجبور کردن شبکه به یادگیری و تولید خروجی‌ای که loss را کمینه کند، نوعی تطبیق به نورون‌ها تحمیل می‌شود و باعث می‌شود آن‌ها برای تولید خروجی مناسب به نورون‌های نزدیک خود تکیه نکنند و feature های مهم را به دست آورند. این حالت باعث می‌شود شبکه اصلی نویز را یاد نگرفته و overfit نشود و فقط feature های تکرار شونده و مهم را آموزش ببیند. پس هدف اصلی این روش regularization و جلوگیری از overfit شدن است.

۲،۲ همان طور که در شکل زیر مشاهده می‌شود، در مرحله آموزش، خروجی نورونی که حذف شده در صفر ضرب می‌شود و عملیات backpropagation با لحاظ کردن این موضوع انجام می‌شود. اما در حالت تست، همه نورون‌های شبکه حضور دارند و ولی وزن نورون‌هایی که در مد آموزش با احتمال p حضور داشتند، در همین عدد احتمال p ضرب می‌شود. مفهوم و علت این ضریب، آن است که مقدار Expected برای این وزن که توسط تعدادی subnet ناشی از dropout شدن نورون مربوطه‌اش بدست آمده در حالت تست در نظر گرفته شود. در واقع در زیر شبکه‌های مختلف این نورون یا حضور داشته یا نداشته و وزن مربوطه در هر زیرساختار آموزش دیده شده است پس منطقی به نظر می‌رسد که مقدار Expected این کمیت با مقدار بدست آمده نهایی در پایان آموزش ضرب در احتمال حضور نورون آن برابر باشد. در مقاله نتیجه میانگین گیری روی نتیجه زیر شبکه‌های ناشی از dropout (که محاسبات بسیاری را نسبت به یک ضرب کردن ساده می‌طلبد) با اعمال تکنیک بالا مقایسه شده و نزدیکی جواب دو روش تأیید شده است.

۳،۲ در زیر قطعه کد تابع نوشته شده آورده شده است.

```
def dropout(x, prob, mode_select):

    prob_tensor_uniform = tf.random.uniform(shape=tf.shape(x), dtype=tf.float32)
    bernoulli_tensor = prob_tensor_uniform > prob
    bernoulli_tensor = tf.to_float(bernoulli_tensor)
    dropout_output = tf.multiply(x, bernoulli_tensor)
    return_1 = dropout_output

    dropout_output = tf.multiply(x, prob)
    return_2 = dropout_output

    dropout_output_o = tf.cond(mode_select > 0, lambda: return_1, lambda: return_2)

    return dropout_output_o
```

همان‌طور که مشاهده می‌شود، از آن‌جایی که داده خروجی با توزیع برنولی نمی‌توان تولید کرد، با گذاشتن یک حد threshold (بر روی خروجی داده‌های uniform، خروجی از جنس مورد نظر در dropout تولید می‌کنم. سپس خروجی لایه‌ای که به این تابع داده شده است در این بردار خروجی برنولی (که صفر یا یک است) ضرب می‌شود. دقت کنید که دو خروجی (return value) محاسبه می‌شود. موردی که توضیح داده شد به حالت train مربوط است و در مد test باید شبکه به صورت کامل حضور داشته باشد و خروجی هر لایه فقط در احتمال حضور نورون مربوطه ضرب شود. (بردار خروجی برنولی) در مقدار بازگشتی دوم این عملیات پیاده‌سازی dropout شده است و در قسمت اصلی کد که به train و test مربوط است با feed کردن متغیری که ورودی condition بر روی این تنسور بین دو مقدار است، تعیین می‌کنم که در مد train هستم یا test و خروجی با

بازگشتی که توضیح داده شد تعیین شده و به تابع فراخواننده بازگردانده می‌شود.

Google Colab .۲

۱. نتیجه آموزش با کد تمرین قبل

```
Optimization Finished!
67% [██████] | 28/42 [00:25<00:12, 1.09it/s]Iter 9, Loss= 0.101022, Training Accuracy= 0.96875
Optimization Finished!
69% [██████] | 29/42 [00:26<00:11, 1.10it/s]Iter 9, Loss= 0.130224, Training Accuracy= 0.96094
Optimization Finished!
71% [██████] | 30/42 [00:27<00:10, 1.10it/s]Iter 9, Loss= 0.038973, Training Accuracy= 0.99219
Optimization Finished!
74% [██████] | 31/42 [00:28<00:10, 1.10it/s]Iter 9, Loss= 0.071836, Training Accuracy= 0.99219
Optimization Finished!
76% [██████] | 32/42 [00:29<00:09, 1.10it/s]Iter 9, Loss= 0.073137, Training Accuracy= 0.98438
Optimization Finished!
79% [██████] | 33/42 [00:30<00:08, 1.10it/s]Iter 9, Loss= 0.059244, Training Accuracy= 0.99219
Optimization Finished!
81% [██████] | 34/42 [00:31<00:07, 1.09it/s]Iter 9, Loss= 0.052868, Training Accuracy= 0.99219
Optimization Finished!
83% [██████] | 35/42 [00:32<00:06, 1.09it/s]Iter 9, Loss= 0.123542, Training Accuracy= 0.97656
Optimization Finished!
86% [██████] | 36/42 [00:32<00:05, 1.09it/s]Iter 9, Loss= 0.018171, Training Accuracy= 1.00000
Optimization Finished!
88% [██████] | 37/42 [00:33<00:04, 1.08it/s]Iter 9, Loss= 0.021342, Training Accuracy= 1.00000
Optimization Finished!
90% [██████] | 38/42 [00:34<00:03, 1.09it/s]Iter 9, Loss= 0.020740, Training Accuracy= 1.00000
Optimization Finished!
93% [██████] | 39/42 [00:35<00:02, 1.09it/s]Iter 9, Loss= 0.237811, Training Accuracy= 0.90625
Optimization Finished!
95% [██████] | 40/42 [00:36<00:01, 1.09it/s]Iter 9, Loss= 0.090246, Training Accuracy= 0.96875
Optimization Finished!
98% [██████] | 41/42 [00:37<00:00, 1.09it/s]Iter 9, Loss= 0.062041, Training Accuracy= 0.98438
Optimization Finished!
100% [██████] | 42/42 [00:38<00:00, 1.09it/s]Iter 9, Loss= 0.389882, Training Accuracy= 0.91406
Optimization Finished!

Testing Accuracy: 0.93150
```

۲. نتیجه آموزش با Batch Normalization

```
Optimization Finished!
64% [██████] | 27/42 [00:42<00:23, 1.57s/it]Iter 9, Loss= 1.779099, Training Accuracy= 0.66406
Optimization Finished!
67% [██████] | 28/42 [00:43<00:21, 1.57s/it]Iter 9, Loss= 1.817007, Training Accuracy= 0.64062
Optimization Finished!
69% [██████] | 29/42 [00:45<00:20, 1.56s/it]Iter 9, Loss= 1.790683, Training Accuracy= 0.64062
Optimization Finished!
71% [██████] | 30/42 [00:46<00:18, 1.56s/it]Iter 9, Loss= 1.713884, Training Accuracy= 0.74219
Optimization Finished!
74% [██████] | 31/42 [00:48<00:17, 1.55s/it]Iter 9, Loss= 1.804503, Training Accuracy= 0.64062
Optimization Finished!
76% [██████] | 32/42 [00:49<00:15, 1.55s/it]Iter 9, Loss= 1.757872, Training Accuracy= 0.63281
Optimization Finished!
79% [██████] | 33/42 [00:51<00:13, 1.54s/it]Iter 9, Loss= 1.735591, Training Accuracy= 0.74219
Optimization Finished!
81% [██████] | 34/42 [00:53<00:12, 1.55s/it]Iter 9, Loss= 1.761333, Training Accuracy= 0.66406
Optimization Finished!
83% [██████] | 35/42 [00:54<00:10, 1.54s/it]Iter 9, Loss= 1.795688, Training Accuracy= 0.67188
Optimization Finished!
86% [██████] | 36/42 [00:56<00:09, 1.54s/it]Iter 9, Loss= 1.699608, Training Accuracy= 0.78125
Optimization Finished!
88% [██████] | 37/42 [00:57<00:07, 1.54s/it]Iter 9, Loss= 1.685818, Training Accuracy= 0.82031
Optimization Finished!
90% [██████] | 38/42 [00:59<00:06, 1.54s/it]Iter 9, Loss= 1.727389, Training Accuracy= 0.72656
Optimization Finished!
93% [██████] | 39/42 [01:00<00:04, 1.54s/it]Iter 9, Loss= 1.893129, Training Accuracy= 0.59375
Optimization Finished!
95% [██████] | 40/42 [01:02<00:03, 1.54s/it]Iter 9, Loss= 1.742618, Training Accuracy= 0.75781
Optimization Finished!
98% [██████] | 41/42 [01:03<00:01, 1.55s/it]Iter 9, Loss= 1.770382, Training Accuracy= 0.74219
Optimization Finished!
100% [██████] | 42/42 [01:05<00:00, 1.56s/it]Iter 9, Loss= 1.816466, Training Accuracy= 0.63281
Optimization Finished!

Iter 9, Loss= 1.781039, Testing Accuracy= 0.65066
```

همان طور که مشاهده می شود دقت این روش خوب نشده و احتمالاً باید به جای انجام شدن قبل از pooling، بعد از آن انجام شود. البته ممکن است در تفسیر CNN برای batch normalization نیز اشتباهی رخ داده باشد. در هر صورت محل انجام syntax دقيق تابع آن با ایده و کمک گرفتن از اينترنت پياده سازی شد و علت دقت نامطلوب اين روش را هنوز نمی دانم.

نتیجه آموزش با Dropout

```
Optimization Finished!
69% [██████] | 29/42 [00:44<00:19,  1.52s/it]Iter 9, Loss= 0.838212, Training Accuracy= 0.74219
Optimization Finished!
71% [██████] | 30/42 [00:45<00:18,  1.51s/it]Iter 9, Loss= 0.588556, Training Accuracy= 0.78906
Optimization Finished!
74% [██████] | 31/42 [00:47<00:16,  1.52s/it]Iter 9, Loss= 0.653573, Training Accuracy= 0.81250
Optimization Finished!
76% [██████] | 32/42 [00:48<00:15,  1.53s/it]Iter 9, Loss= 0.668198, Training Accuracy= 0.77344
Optimization Finished!
79% [██████] | 33/42 [00:50<00:13,  1.54s/it]Iter 9, Loss= 0.618865, Training Accuracy= 0.82031
Optimization Finished!
81% [██████] | 34/42 [00:51<00:12,  1.53s/it]Iter 9, Loss= 0.711292, Training Accuracy= 0.78125
Optimization Finished!
83% [██████] | 35/42 [00:53<00:10,  1.53s/it]Iter 9, Loss= 0.771906, Training Accuracy= 0.75781
Optimization Finished!
86% [██████] | 36/42 [00:54<00:09,  1.52s/it]Iter 9, Loss= 0.488619, Training Accuracy= 0.81250
Optimization Finished!
88% [██████] | 37/42 [00:56<00:07,  1.53s/it]Iter 9, Loss= 0.548571, Training Accuracy= 0.80469
Optimization Finished!
90% [██████] | 38/42 [00:57<00:06,  1.53s/it]Iter 9, Loss= 0.624010, Training Accuracy= 0.85156
Optimization Finished!
93% [██████] | 39/42 [00:59<00:04,  1.54s/it]Iter 9, Loss= 0.173417, Training Accuracy= 0.64062
Optimization Finished!
95% [██████] | 40/42 [01:01<00:03,  1.57s/it]Iter 9, Loss= 0.644918, Training Accuracy= 0.78906
Optimization Finished!
98% [██████] | 41/42 [01:02<00:01,  1.55s/it]Iter 9, Loss= 0.647642, Training Accuracy= 0.81250
Optimization Finished!
100% [██████] | 42/42 [01:04<00:00,  1.55s/it]Iter 9, Loss= 0.871374, Training Accuracy= 0.73438
Optimization Finished!

Iter 9, Loss= 1.065827, Testing Accuracy= 0.89150
```

کاملاً قابل مشاهده است که این روش به خوبی عمل کرده و به دقت بالایی رسیده است. (در همه شبیه‌سازی‌ها داریم $mini - batch size = 64$)

۳. نتیجه استفاده همزمان از هر دو منظم‌ساز

نتیجه برای وقتی که بلافاصله بعد از کانولوشن، ReLU و سپس batch normalization انجام شود (پایین بودن دقت آموزش به دلیل آموزش subnets در هر مرحله سپس dropout است که کاملاً مورد انتظار است چون شبکه بسیار ناقصی است):

```
▶ 81% [██████] | 34/42 [01:18<00:18,  2.31s/it]Iter 9, Loss= 1.344785, Training Accuracy= 0.63281
Optimization Finished!
83% [██████] | 35/42 [01:21<00:16,  2.31s/it]Iter 9, Loss= 1.513081, Training Accuracy= 0.50000
Optimization Finished!
86% [██████] | 36/42 [01:23<00:13,  2.31s/it]Iter 9, Loss= 1.332425, Training Accuracy= 0.59375
Optimization Finished!
88% [██████] | 37/42 [01:25<00:11,  2.31s/it]Iter 9, Loss= 1.321240, Training Accuracy= 0.69531
Optimization Finished!
90% [██████] | 38/42 [01:28<00:09,  2.31s/it]Iter 9, Loss= 1.455424, Training Accuracy= 0.56250
Optimization Finished!
93% [██████] | 39/42 [01:30<00:07,  2.33s/it]Iter 9, Loss= 1.668297, Training Accuracy= 0.46094
Optimization Finished!
95% [██████] | 40/42 [01:32<00:04,  2.32s/it]Iter 9, Loss= 1.476156, Training Accuracy= 0.55469
Optimization Finished!
98% [██████] | 41/42 [01:35<00:02,  2.31s/it]Iter 9, Loss= 1.403039, Training Accuracy= 0.57812
Optimization Finished!
100% [██████] | 42/42 [01:37<00:00,  2.31s/it]Iter 9, Loss= 1.584723, Training Accuracy= 0.56250
Optimization Finished!

Iter 9, Loss= 1.471206, Testing Accuracy= 0.82570
```

دقت استفاده همزمان دو روش کمتر از dropout تنها است و احتمالاً batch normalization مشکل دارد و باید عیب آن یافت شود. متأسفانه کد خود را با کدی که دقت بالا گرفته بود چک کردم و روش یکی بود و مشکلی در تابع یافت نشد.

نتیجه برای وقتی که در لایه‌های کانولوشنی، پس از maxpooling ابتدا dropout کنیم و قبل از آن batch normalization (که دقیق آن نشان می‌دهد این کار مناسب نیست):

```

    76% | 32/42 [01:08<00:21,  2.14s/it]Iter 9, Loss= 2.175354, Training Accuracy= 0.24219
Optimization Finished!
    79% | 33/42 [01:10<00:19,  2.12s/it]Iter 9, Loss= 2.107434, Training Accuracy= 0.28125
Optimization Finished!
    81% | 34/42 [01:12<00:17,  2.13s/it]Iter 9, Loss= 2.206264, Training Accuracy= 0.20312
Optimization Finished!
    83% | 35/42 [01:14<00:14,  2.13s/it]Iter 9, Loss= 2.201075, Training Accuracy= 0.25000
Optimization Finished!
    86% | 36/42 [01:16<00:12,  2.15s/it]Iter 9, Loss= 2.186883, Training Accuracy= 0.25781
Optimization Finished!
    88% | 37/42 [01:19<00:10,  2.15s/it]Iter 9, Loss= 2.151793, Training Accuracy= 0.24219
Optimization Finished!
    90% | 38/42 [01:21<00:08,  2.14s/it]Iter 9, Loss= 2.156369, Training Accuracy= 0.23438
Optimization Finished!
    93% | 39/42 [01:23<00:06,  2.14s/it]Iter 9, Loss= 2.203928, Training Accuracy= 0.22656
Optimization Finished!
    95% | 40/42 [01:25<00:04,  2.14s/it]Iter 9, Loss= 2.133796, Training Accuracy= 0.25000
Optimization Finished!
    98% | 41/42 [01:27<00:02,  2.15s/it]Iter 9, Loss= 2.256237, Training Accuracy= 0.18750
Optimization Finished!
100% | 42/42 [01:29<00:00,  2.15s/it]Iter 9, Loss= 2.132067, Training Accuracy= 0.28125
Optimization Finished!

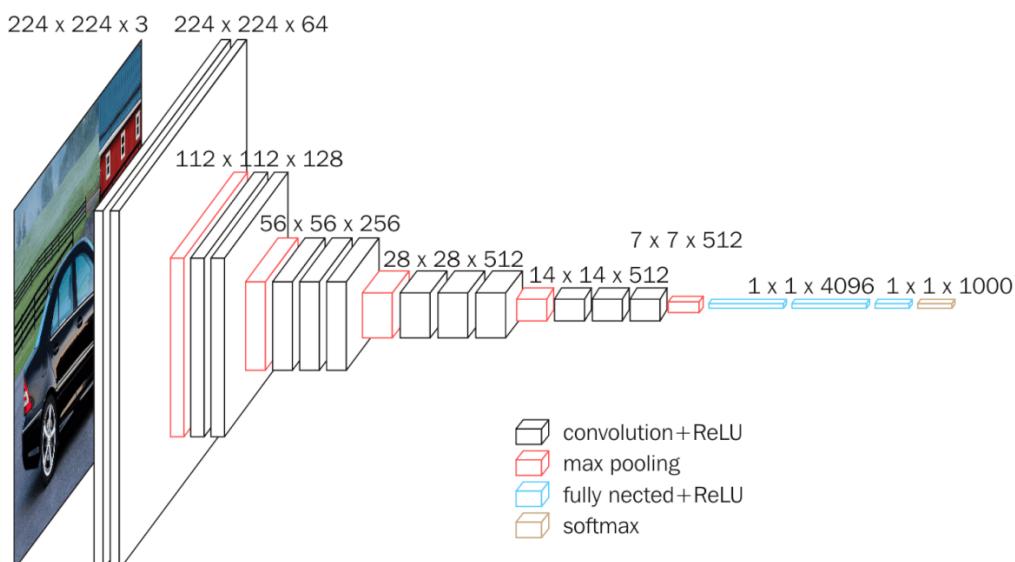
Iter 9, Loss= 2.036235, Testing Accuracy= 0.55480

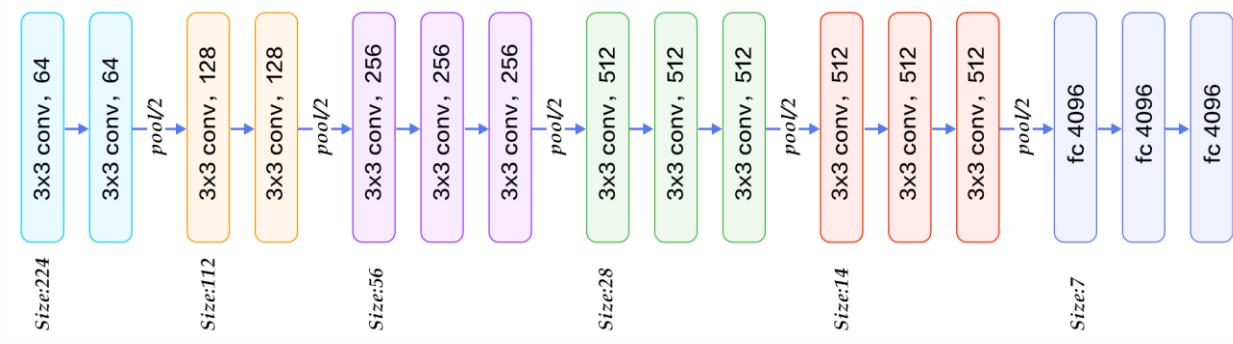
```

Visualization .۳

۱.۳ با توجه به توضیحات [۱]، این شبکه یک شبکه کانولوشنی خیلی عمیق می‌باشد که دقیق آن بر روی داده‌های ImageNet که دیتابستی مشتمل بر ۱۴ میلیون تصویر از ۱۰۰۰ دسته است، ۹۷,۲ درصد می‌باشد. تصویر معماري آن را در زیر می‌توانید ببینید:

در این ساختار از لایه‌های کانولوشنی با کرنل‌های 3×3 و لایه‌های max pooling با اندازه 2×2 استفاده شده است. کل ساختار ۱۶ لایه دارد. در لایه آخر یک لایه softmax برای تبدیل خروجی‌ها به احتمال هر کدام از ۱۰۰۰ لایه قرار داده می‌شود.





دیتاست ImageNet یک دیتاست عظیم است. ImageNet یک مجموعه داده تصویری است که براساس سلسله مراتب WordNet سازماندهی شده است. هر مفهوم معنی دار در WordNet، که احتمالاً توسط چندین عبارت یا عبارات کلمه توصیف شده است، به عنوان "متراffد مجموعه" یا "synset" نامیده می شود. در WordNet بیش از 100,000 synsets وجود دارد، اکثریت آنها اسم ها (+ 80,000) هستند. در ImageNet به طور متوسط 1000 تصویر برای نشان دادن هر synset ارائه شده است. تصاویر هر یک از مفاهیم، کنترل کیفیت شده و توسط انسان تفسیر شده‌اند.

در مورد کاربرد آن می‌توان به Transfer Learning اشاره کرد. در واقع می‌توان کل مدل را به جز قسمت آخر که لایه‌های fully connected قرار دارند نگهداشت و از وزن‌های آن‌ها که یک‌بار آموزش داده شده و محاسبه شده‌اند استفاده کرد تا در عکس‌های جدید با تفاوت اندک نسبت به ImageNet، طبقه‌بندی انجام داد. در واقع وزن‌ها و کرنل‌های آموزش دیده شده ویژگی‌های ۱۰۰۰ دسته داده ImageNet را در خود دارند و می‌توانیم با استفاده از آن‌ها لایه آخر را آموزش دهیم و عکس‌های با اندازه متفاوت و یا کلاس‌های دارای شباهت با این عکس‌ها را آموزش بدھیم و با سرعت بالا از شبکه استفاده کنیم بدون این‌که مدت طولانی آن را آموزش داده باشیم.

۲.۳. در مرجع [2] و تصویر سوال قبل ساختار شبکه (نوع لایه و ابعاد آن) نشان داده شده است. به طور خلاصه به آن اشاره می‌کنیم.

۱. لایه اول: لایه کانولوشن با ۶۴ کانال (کرنل) 3×3

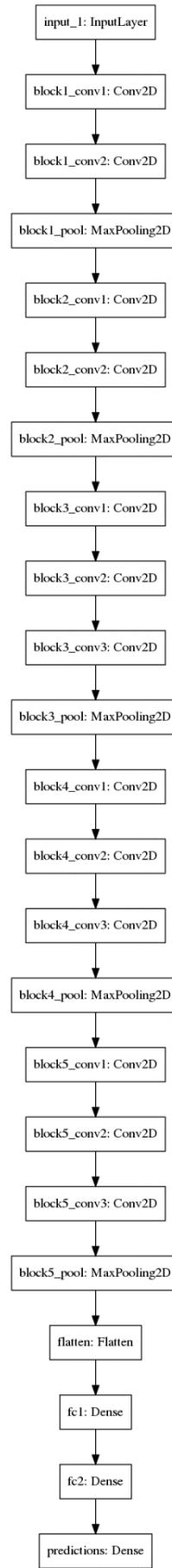
۲. لایه دوم: لایه کانولوشن با ۶۴ کانال (کرنل) 3×3 و بعد از آن maxpooling (2×2)

۳. لایه سوم: لایه کانولوشن با ۱۲۸ کانال (کرنل) 3×3

۴. لایه چهارم: لایه کانولوشن با $128 \times 3 \times 3$ کanal (کرنل) و بعد از آن maxpooling (2×2)
۵. لایه پنجم: لایه کانولوشن با $256 \times 3 \times 3$ کanal (کرنل)
۶. لایه ششم: لایه کانولوشن با $256 \times 3 \times 3$ کanal (کرنل)
۷. لایه هفتم: لایه کانولوشن با $256 \times 3 \times 3$ کanal (کرنل) و بعد از آن maxpooling (2×2)
۸. لایه هشتم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل)
۹. لایه نهم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل)
۱۰. لایه دهم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل) و بعد از آن maxpooling (2×2)
۱۱. لایه یازدهم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل)
۱۲. لایه دوازدهم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل)
۱۳. لایه سیزدهم: لایه کانولوشن با $512 \times 3 \times 3$ کanal (کرنل) و بعد از آن maxpooling (2×2)
۱۴. لایه تمام متصل با 4096 نورون
۱۵. لایه تمام متصل با 4096 نورون
۱۶. لایه خروجی با 1000 نورون با تابع فعالیت softmax برای تبدیل خروجی به احتمال پیش‌بینی هر کلاس از 1000 کلاس طبقه‌بند.

تصویر زیر با استفاده از Keras در Console رسم شده است.

تصویر کنار آن هم summary مدل با استفاده از Keras است.



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

None

مرجع تصویر نیز مقاله اصلی شبکه VGG16 است. [3]

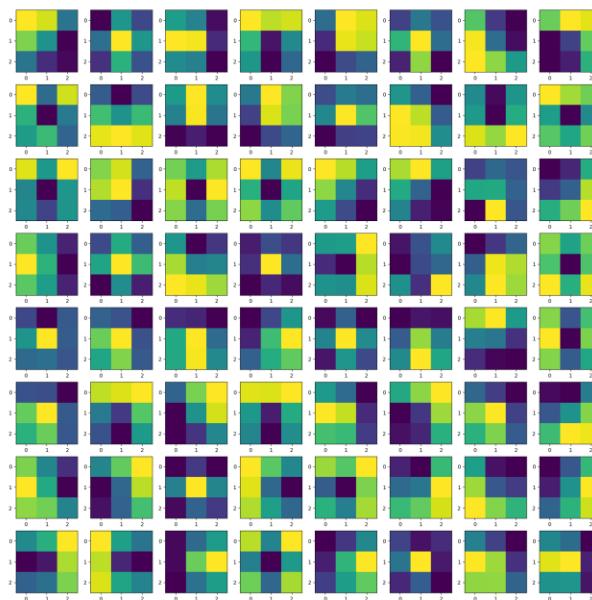
Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv<receptive field size>-<number of channels>”. The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

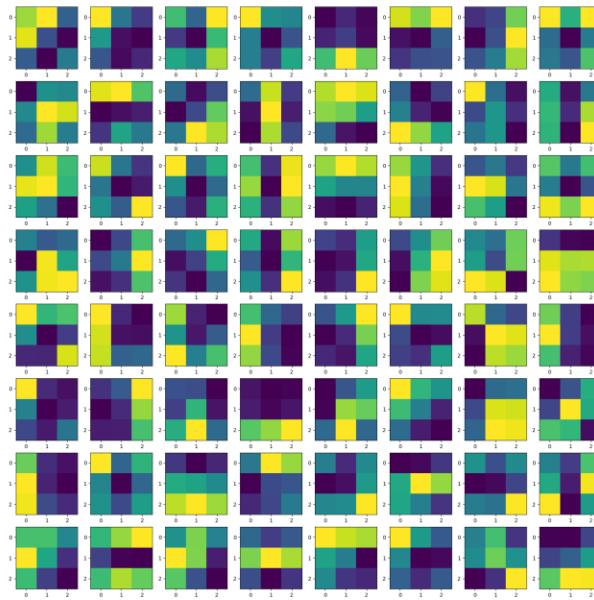
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

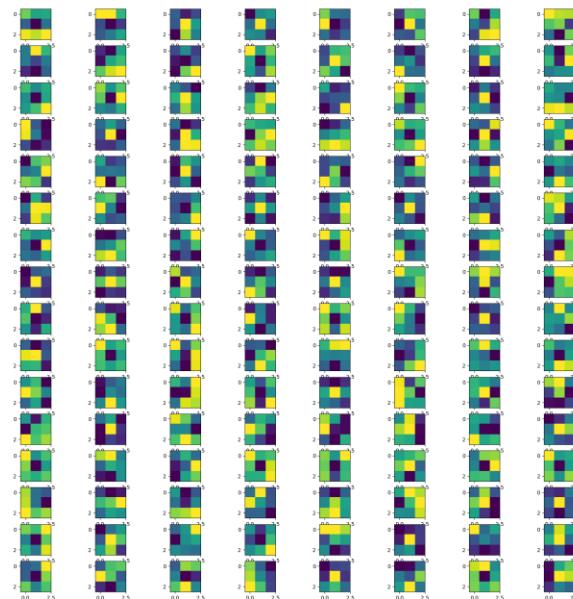
۳.۳. کرنل ها ابعاد کوچکی دارند و از روی آن ها به ویژگی کرنل های لایه های مختلف نمی توان پی برد. در واقع هر کرنل فقط می تواند ویژگی های کوچکی مثل جهت و لبه و smoothing را پیاده سازی کند. در تصاویر لایه های اولیه کرنل هایی از این دست دیده می شود که حول یک مرکز میانگین می گیرند یا در یک جهت مشتق می گیرند و خواص اولیه تصویر را بدست می آورند. در لایه های آخر ویژگی های بدست آمده از چند لایه گذر کرده اند و این کرنل ها به کمک لایه های قبلی ویژگی های عمیق تر تصویر را بدست می آورند و به صورت چشمی اطلاعاتی در اختیار نمی گذارند.



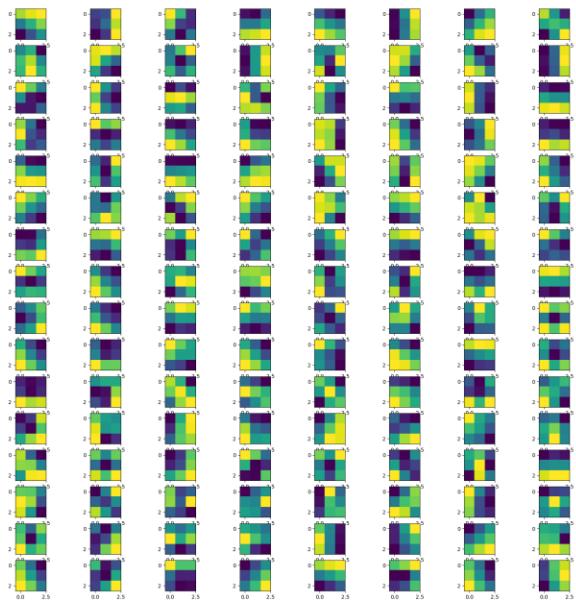
لایه ۱



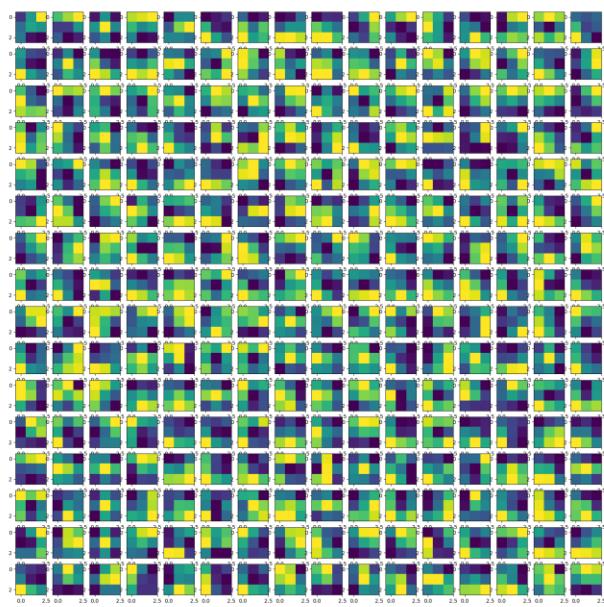
$r_{42}y$



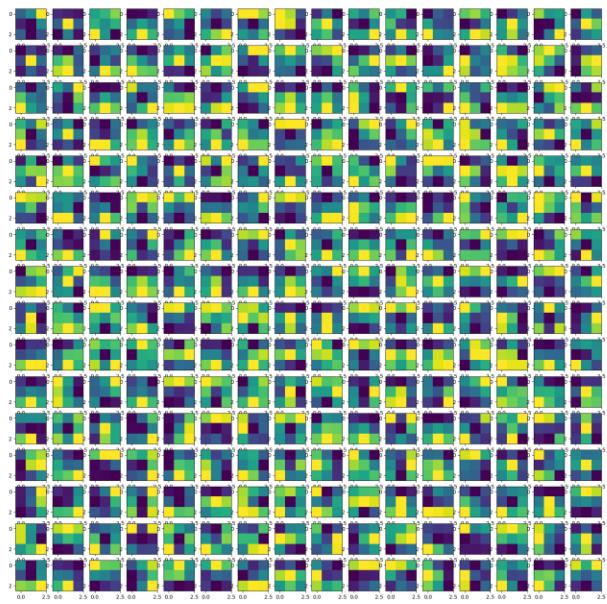
$r_{42}y$



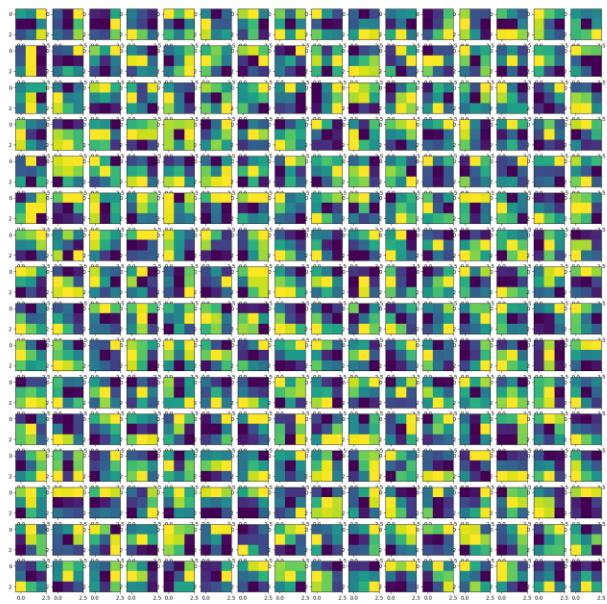
$f_{\theta}(\gamma)$



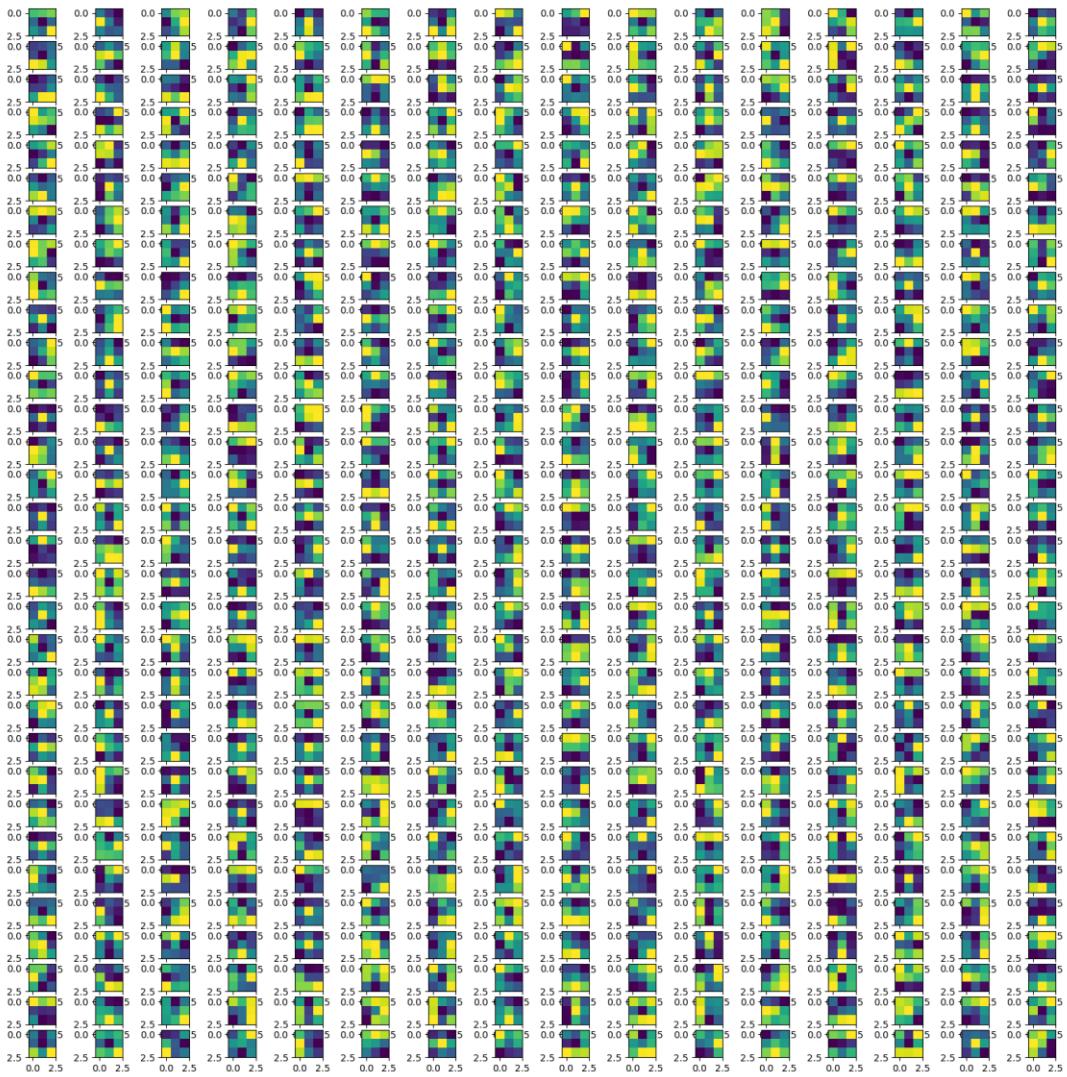
$\delta_{\theta}(\gamma)$



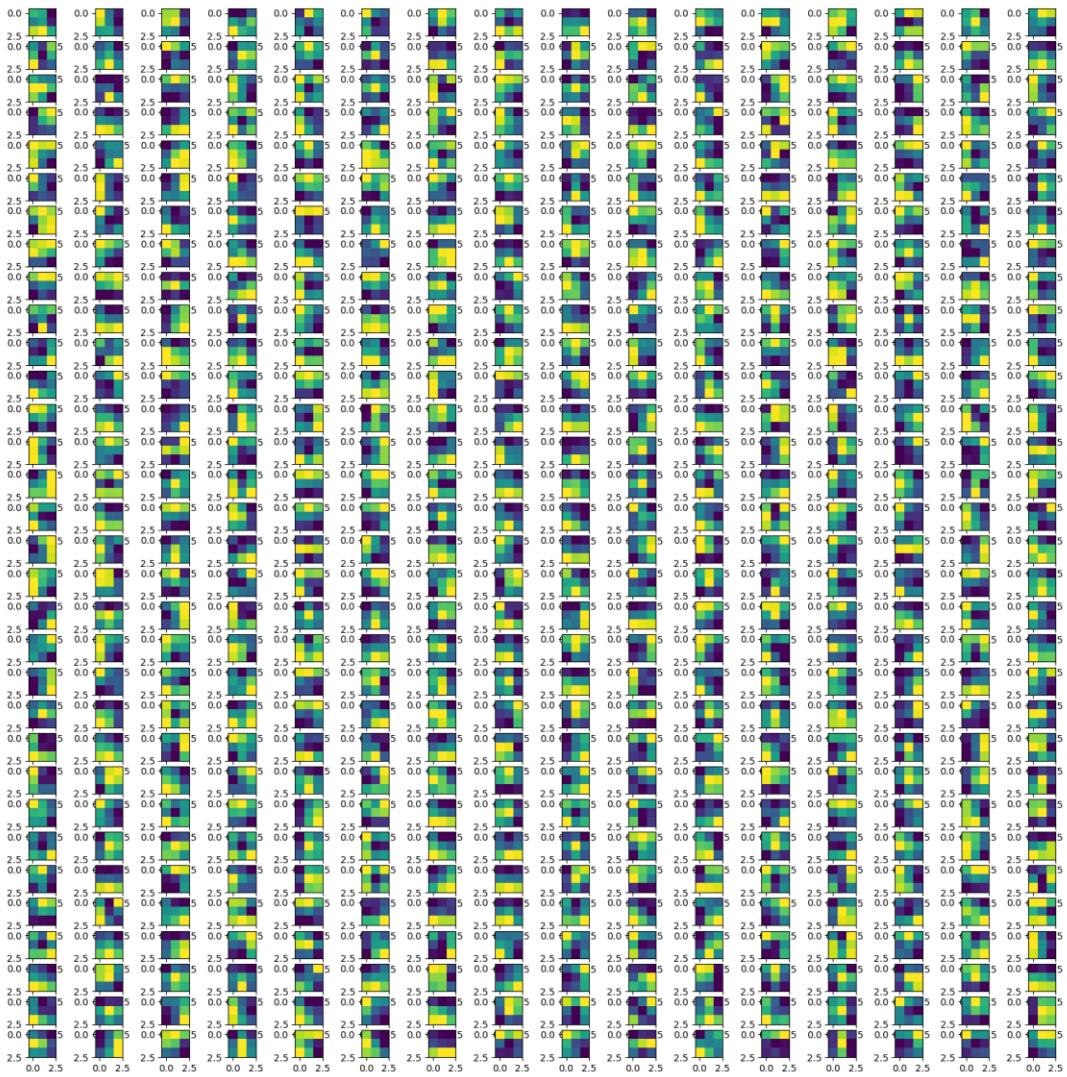
$\gamma_{4,2}$



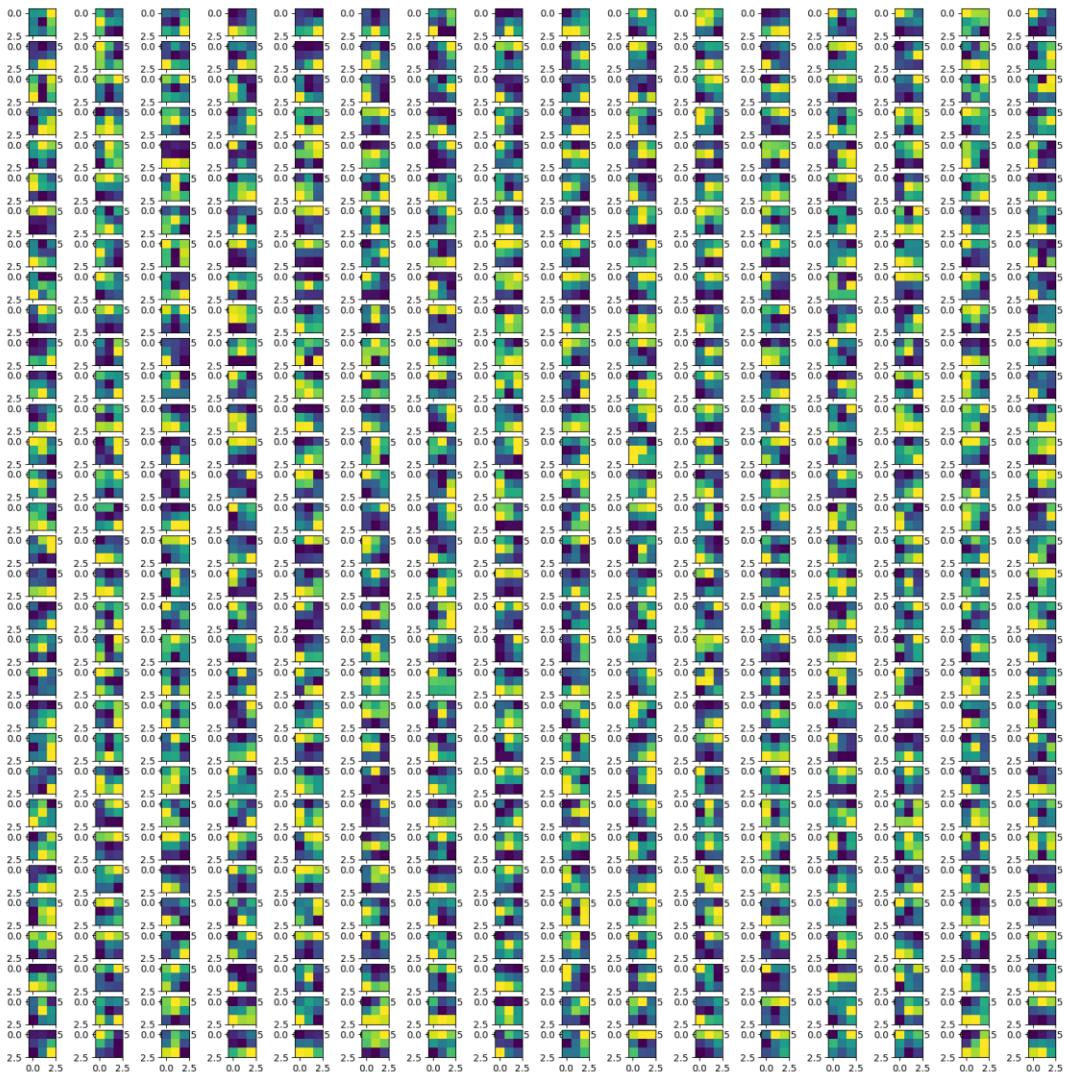
$\gamma_{4,2}$



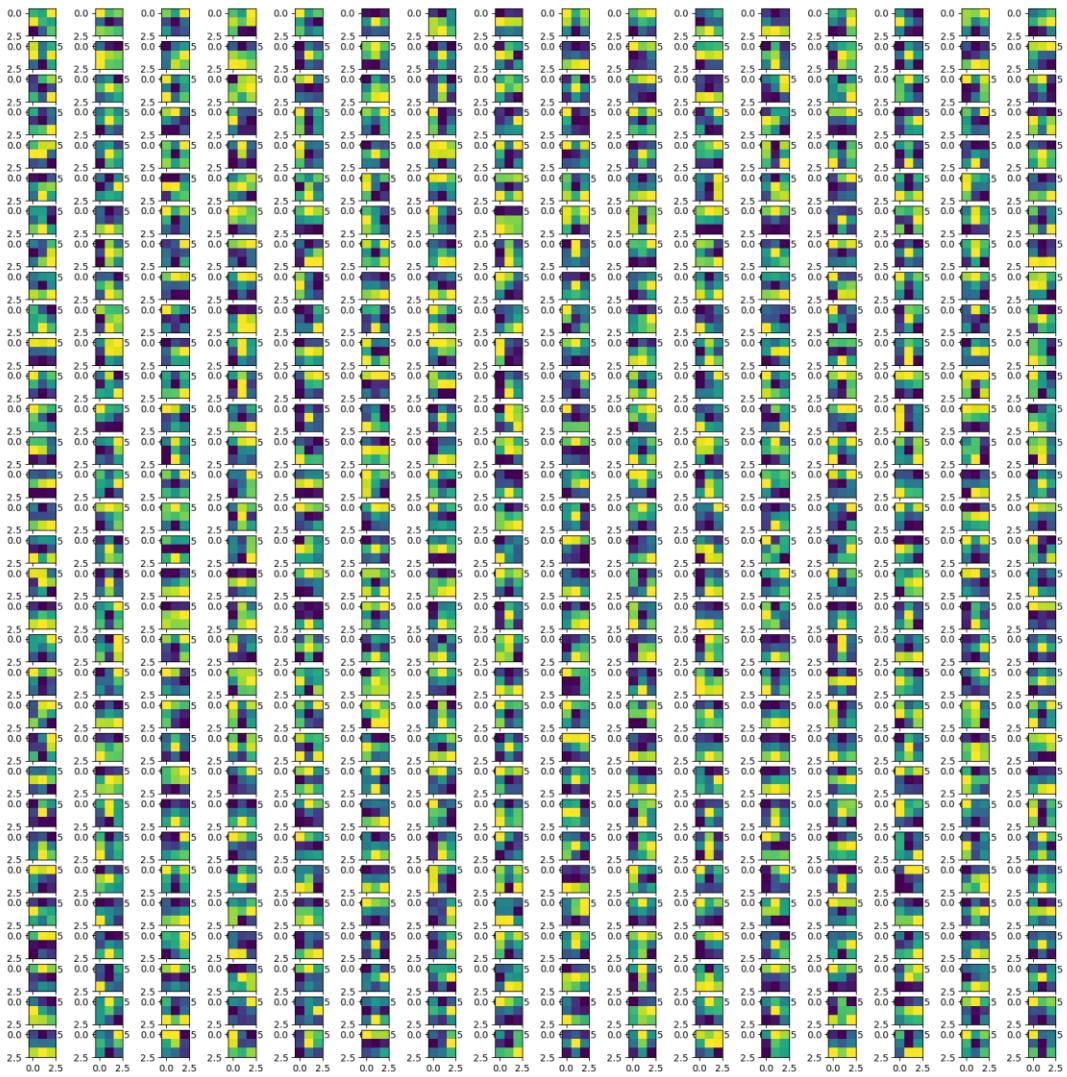
$\lambda \triangleleft \gamma$



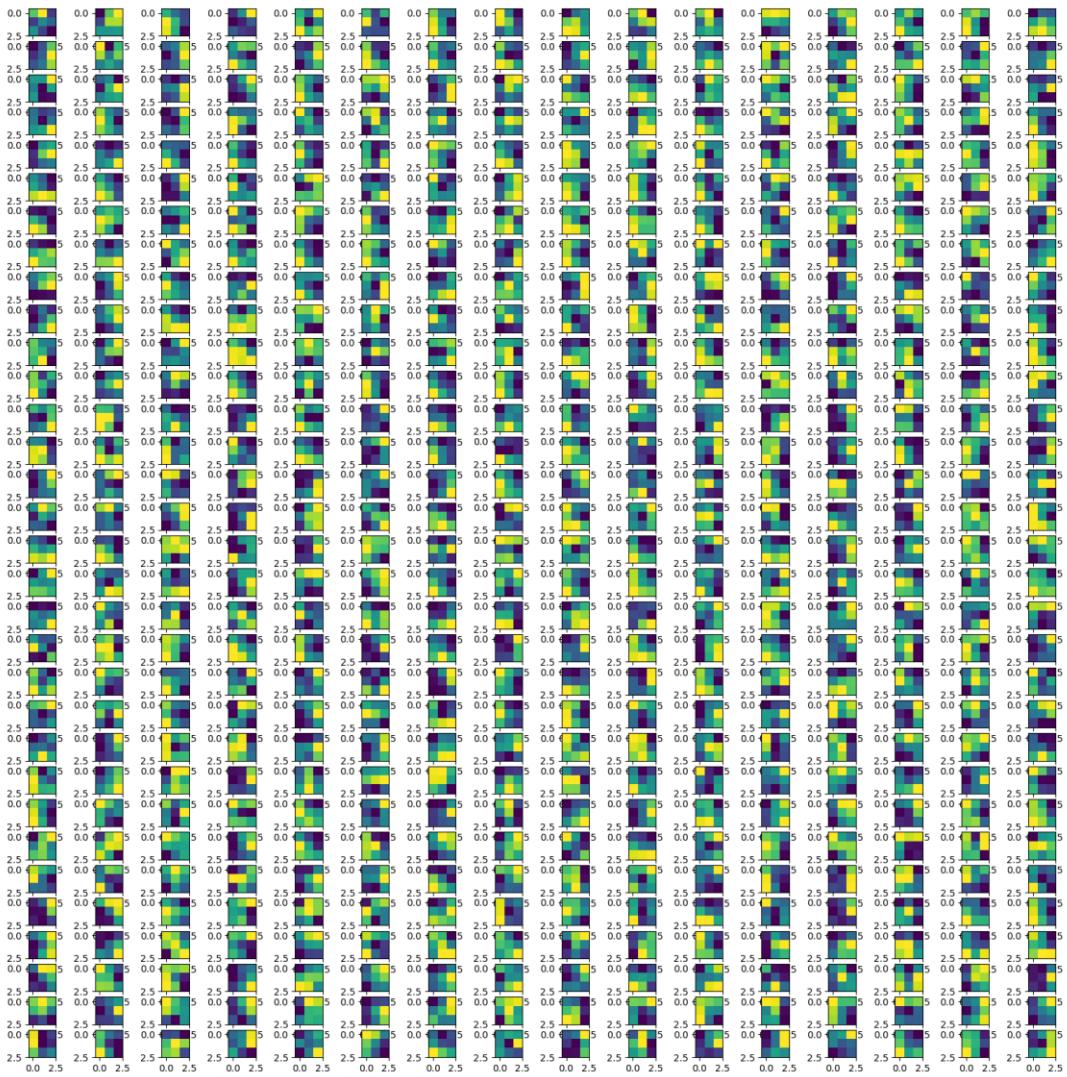
$\vartheta_4 \gamma$

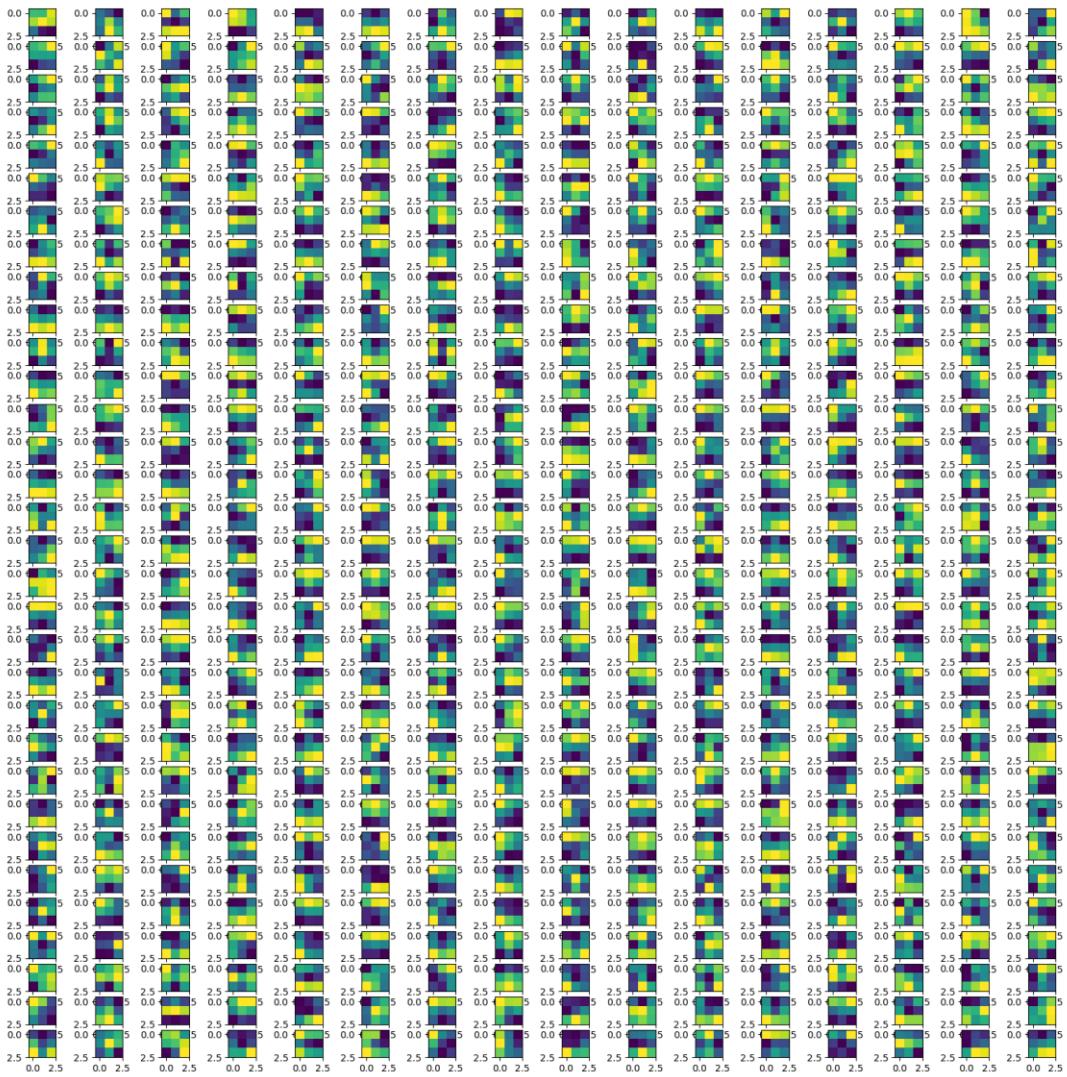


$\mathcal{I} + \mathcal{A}_2 \mathcal{Y}$



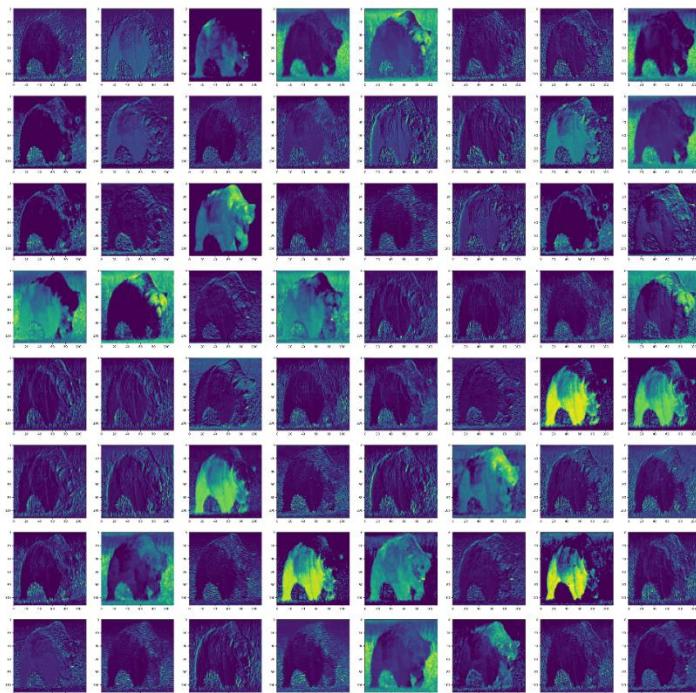
11 42

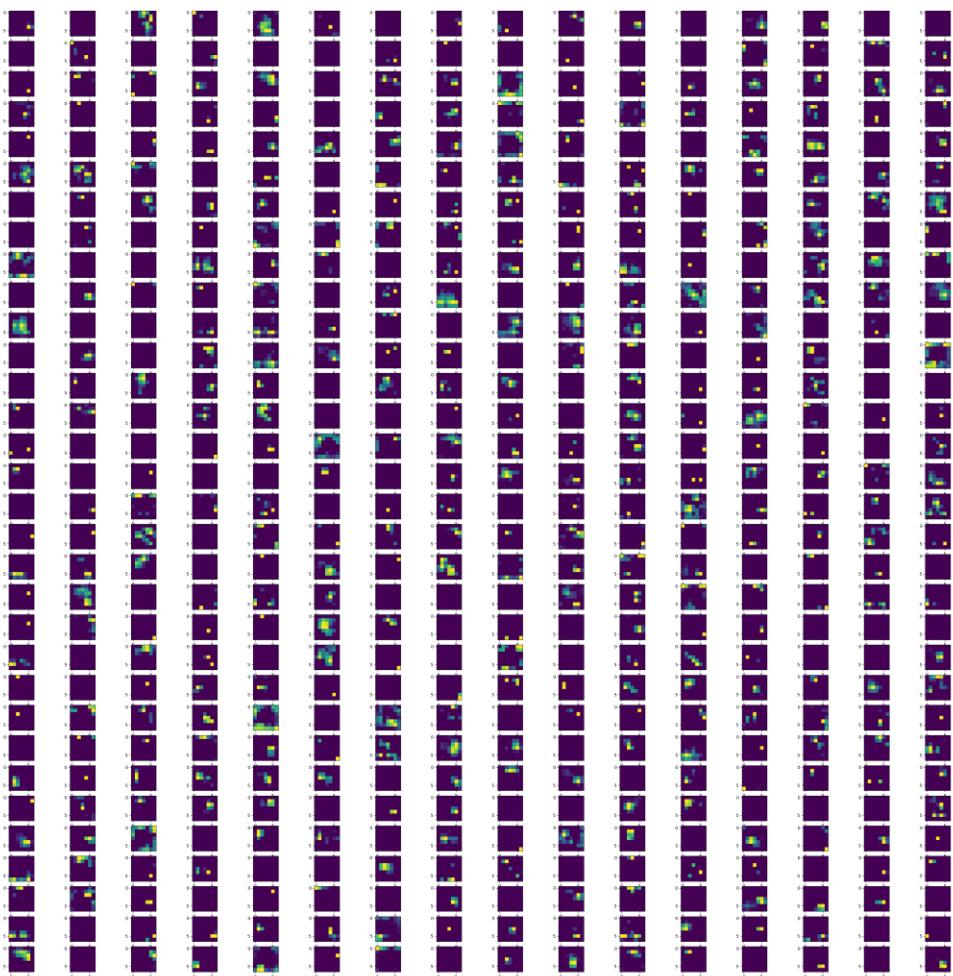




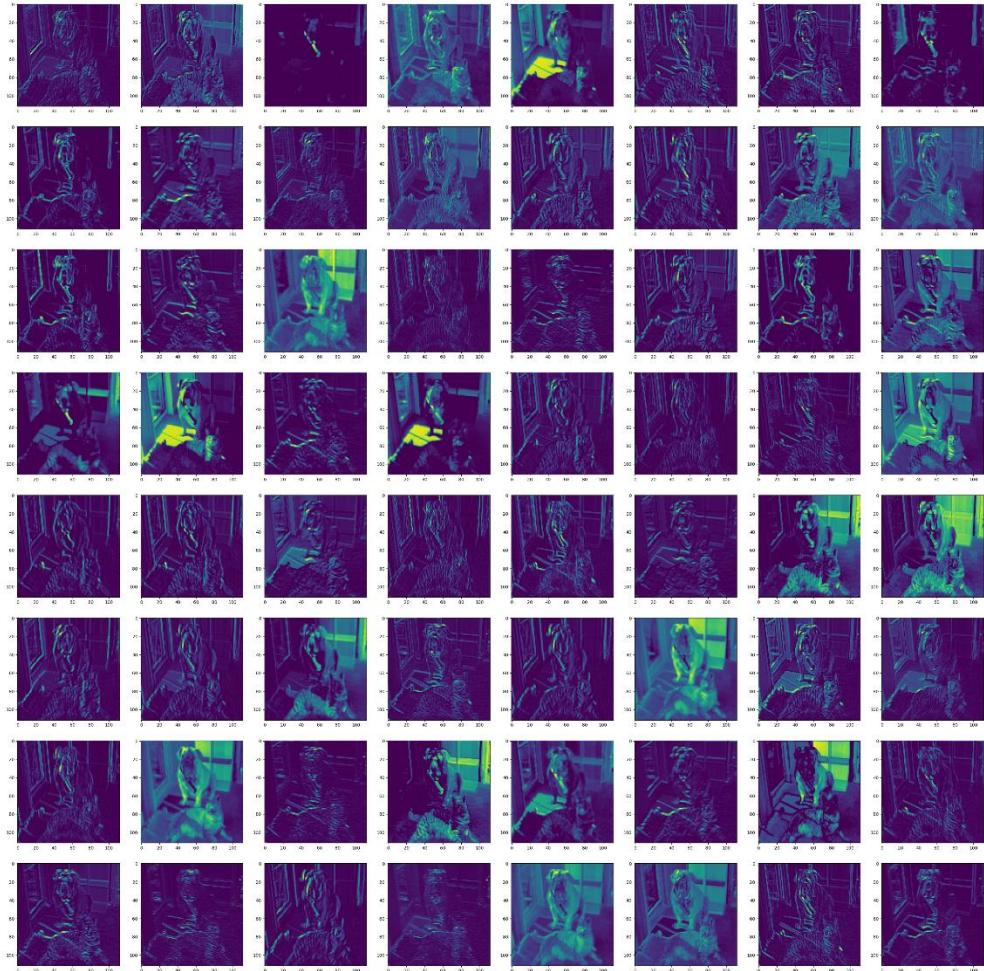
۴.۳ کد این قسمت و دو قسمت قبل با نام `VGG.py` پیوست شده است. از آوردن توضیح کد آن خودداری می‌کنیم چون نکته خاصی ندارد. خروجی دو لایه خروجی `maxpooling` سوم و سیزدهم را مطابق خواسته سوال به علاوه نتیجه پیش‌بینی مدل برای عکس‌های داده شده در زیر می‌آوریم.

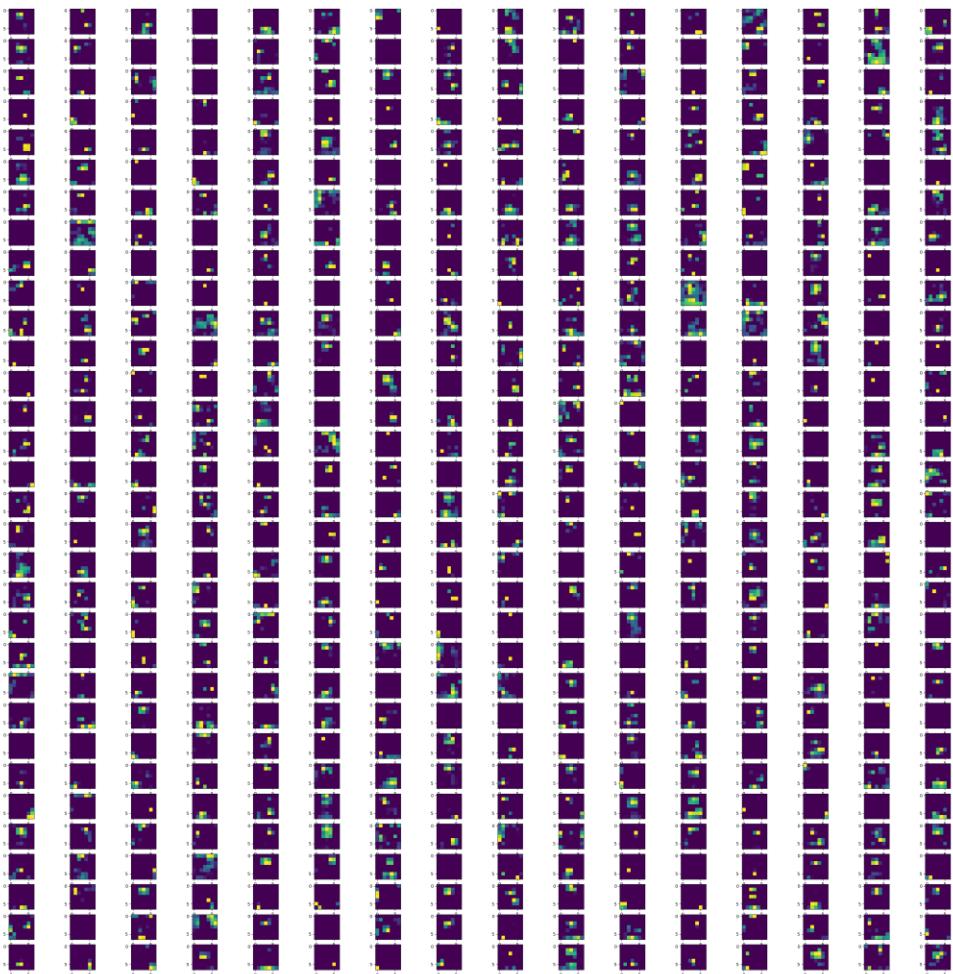
```
Using TensorFlow backend.  
2018-12-19 22:40:09.807284: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow  
brown_bear (97.07%)  
brown_bear (97.07%)  
boxer (42.01%)  
boxer (42.01%)  
seashore (63.60%)  
seashore (63.60%)  
beagle (97.03%)  
beagle (97.03%)  
castle (63.30%)  
castle (63.30%)  
space_shuttle (99.79%)  
  
Process finished with exit code 0
```



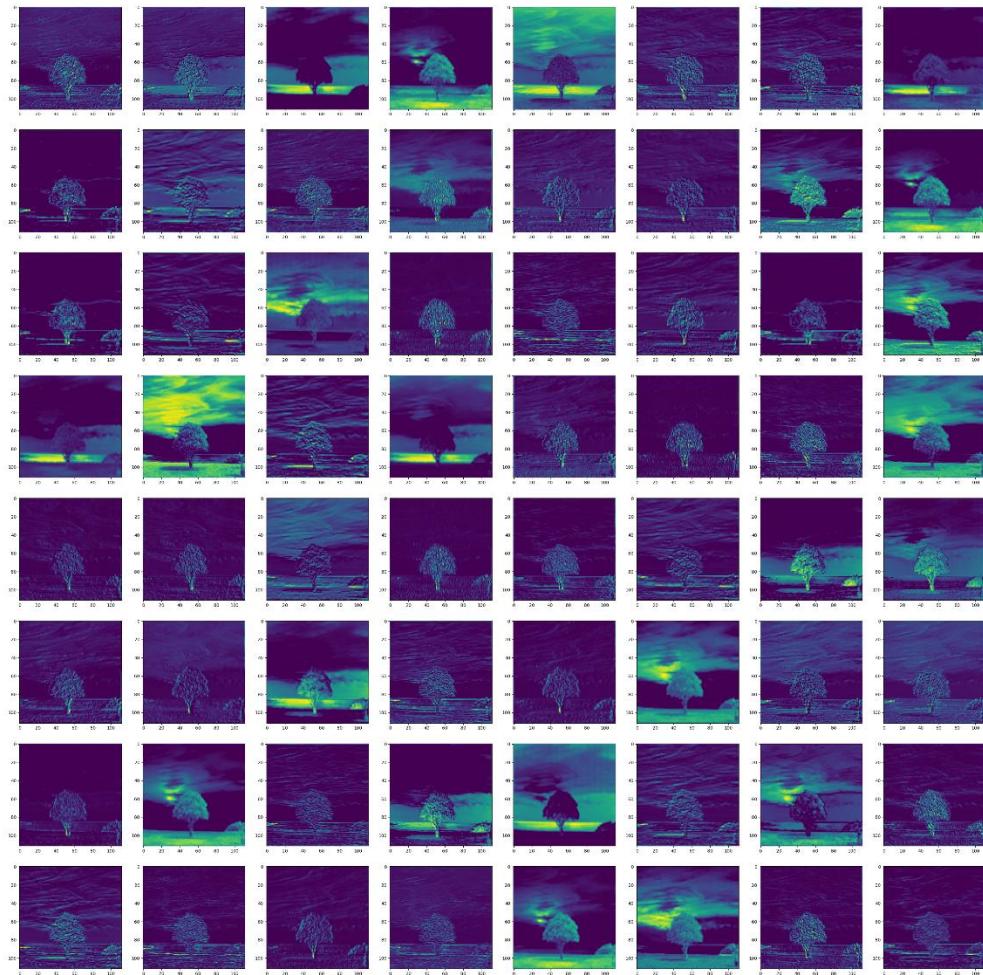


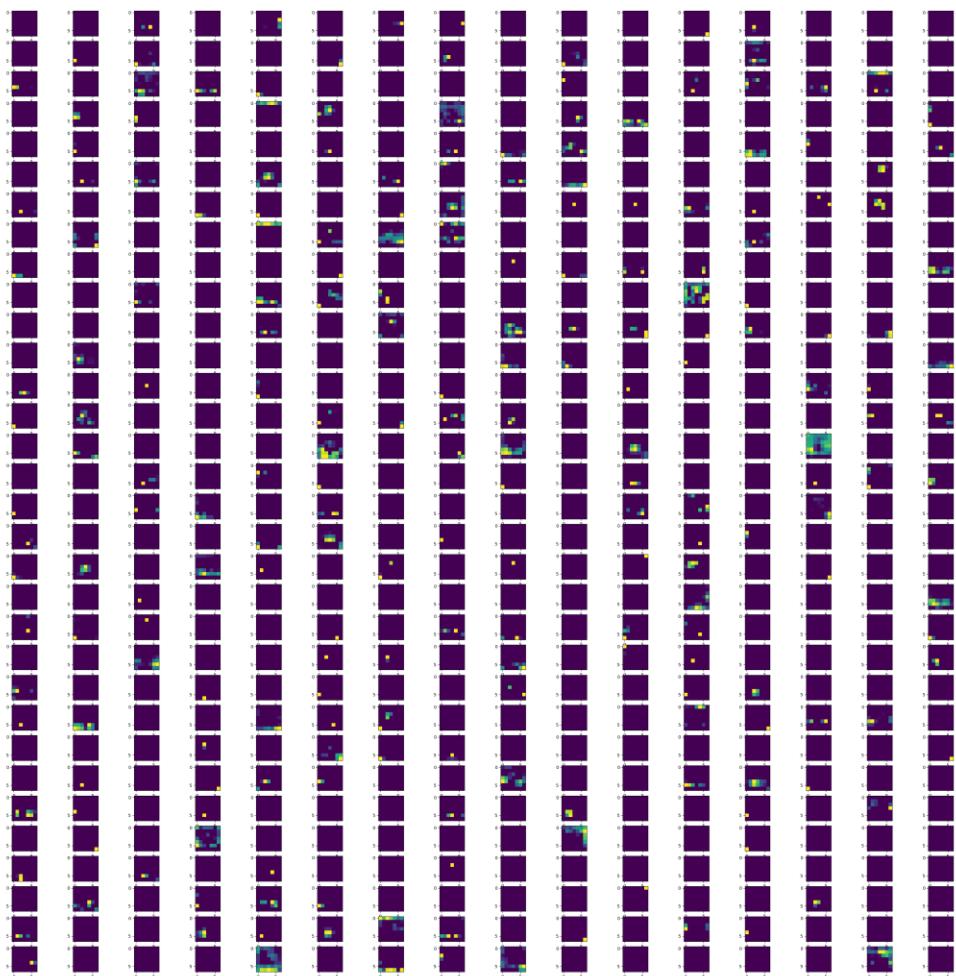
.Cat dog



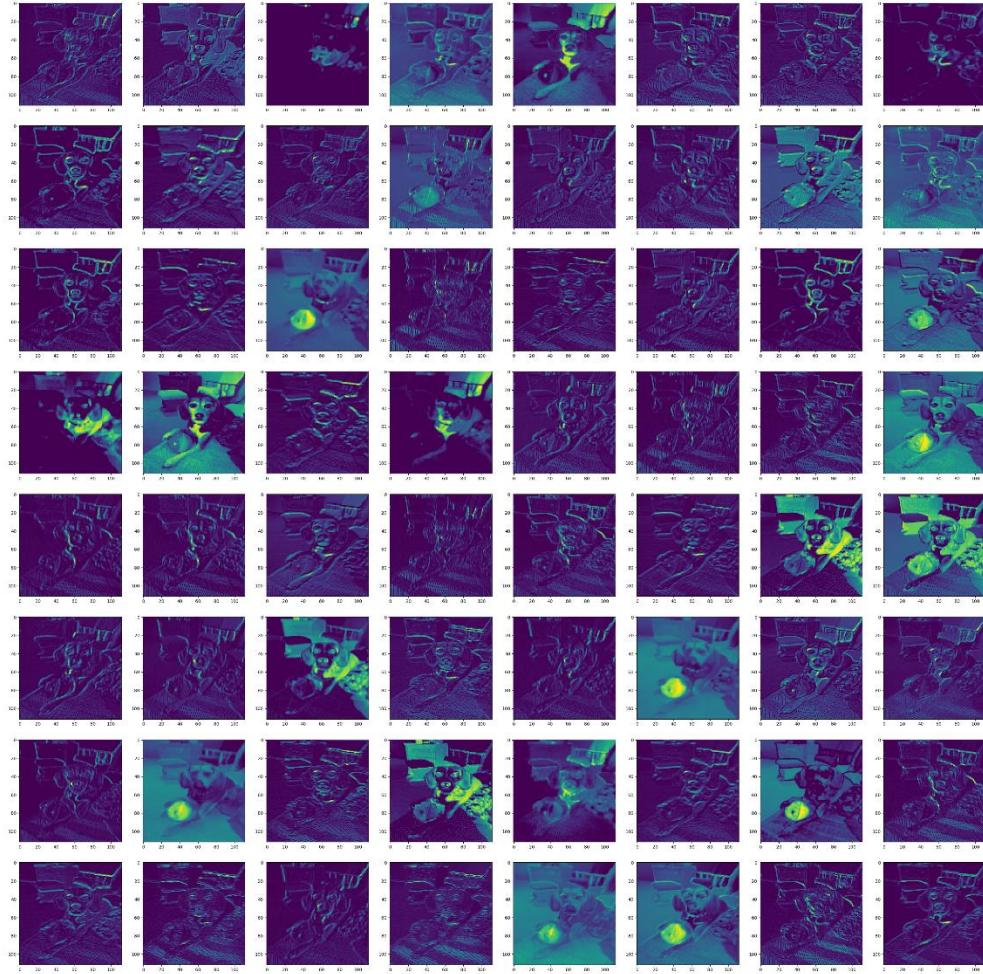


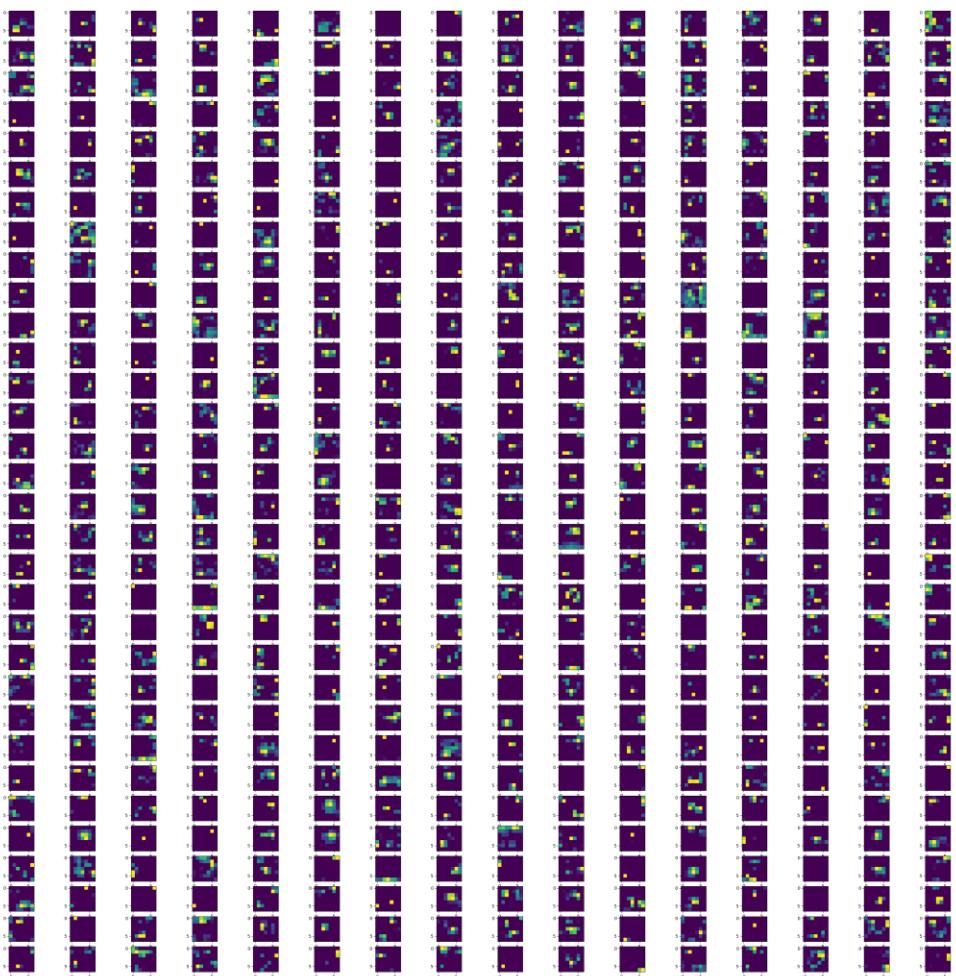
:dd tree



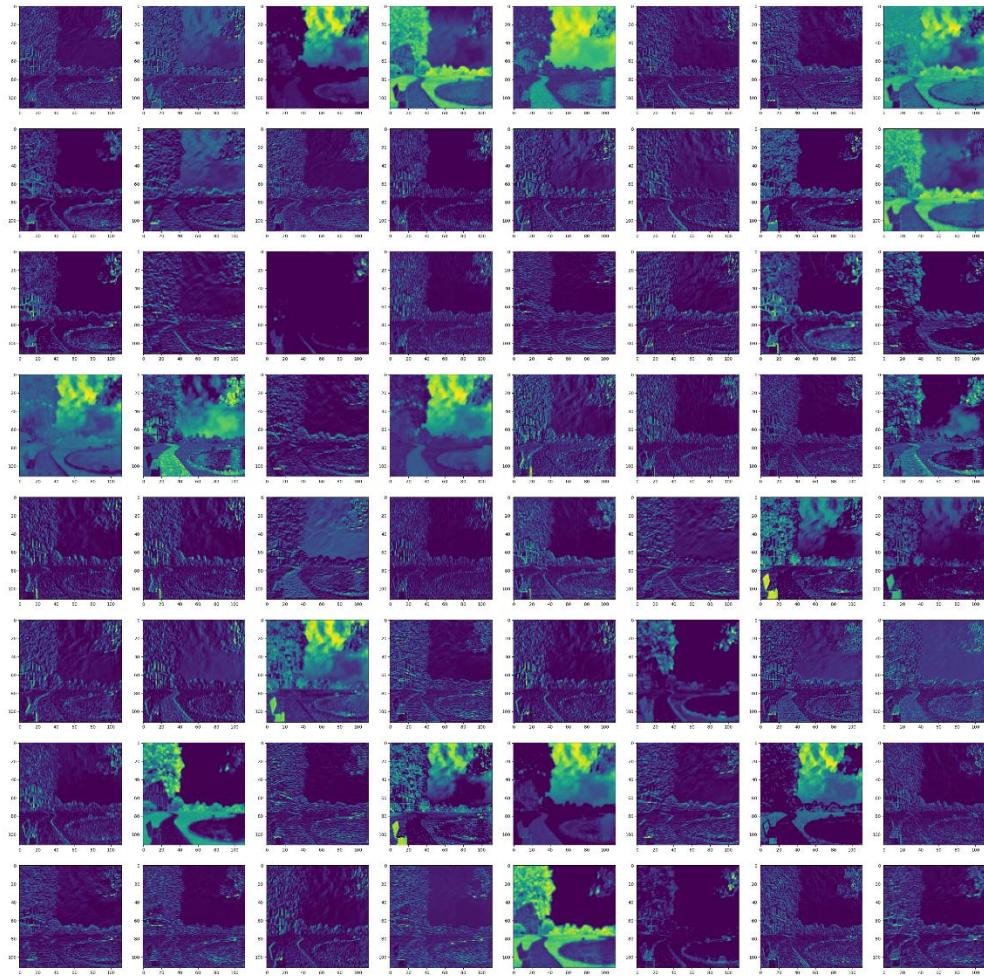


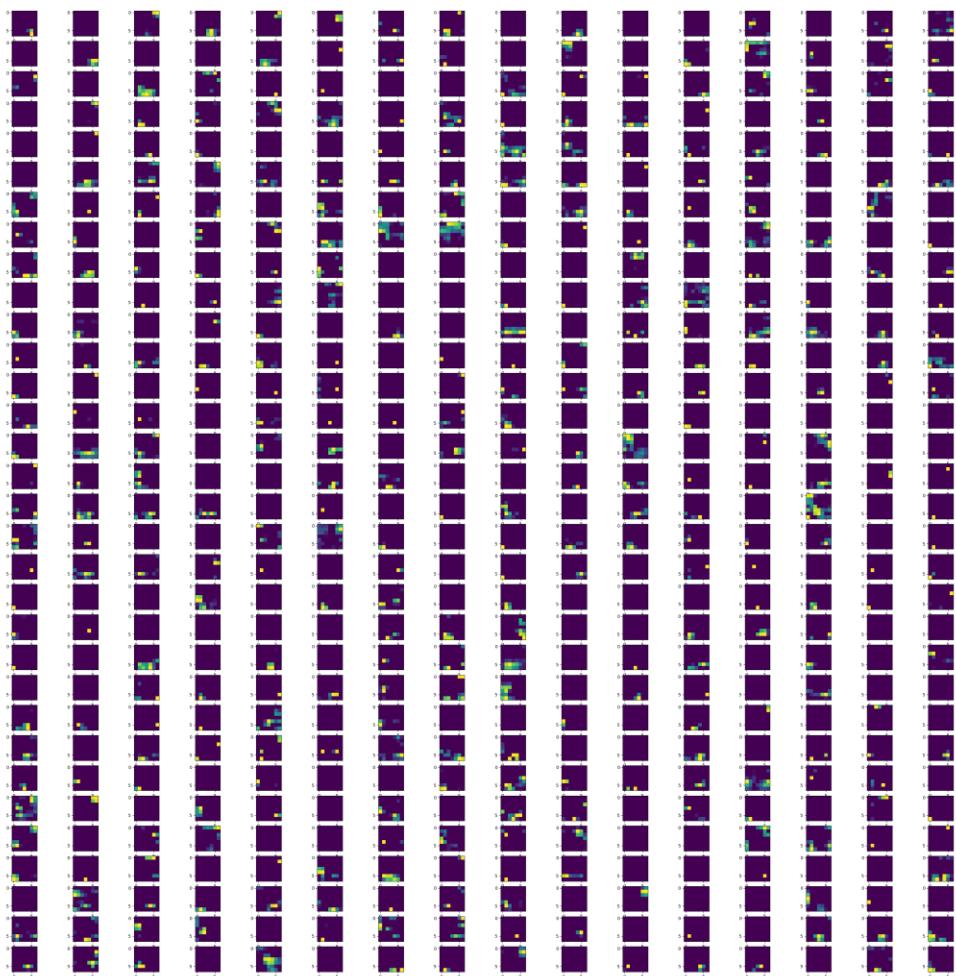
.dog beagle



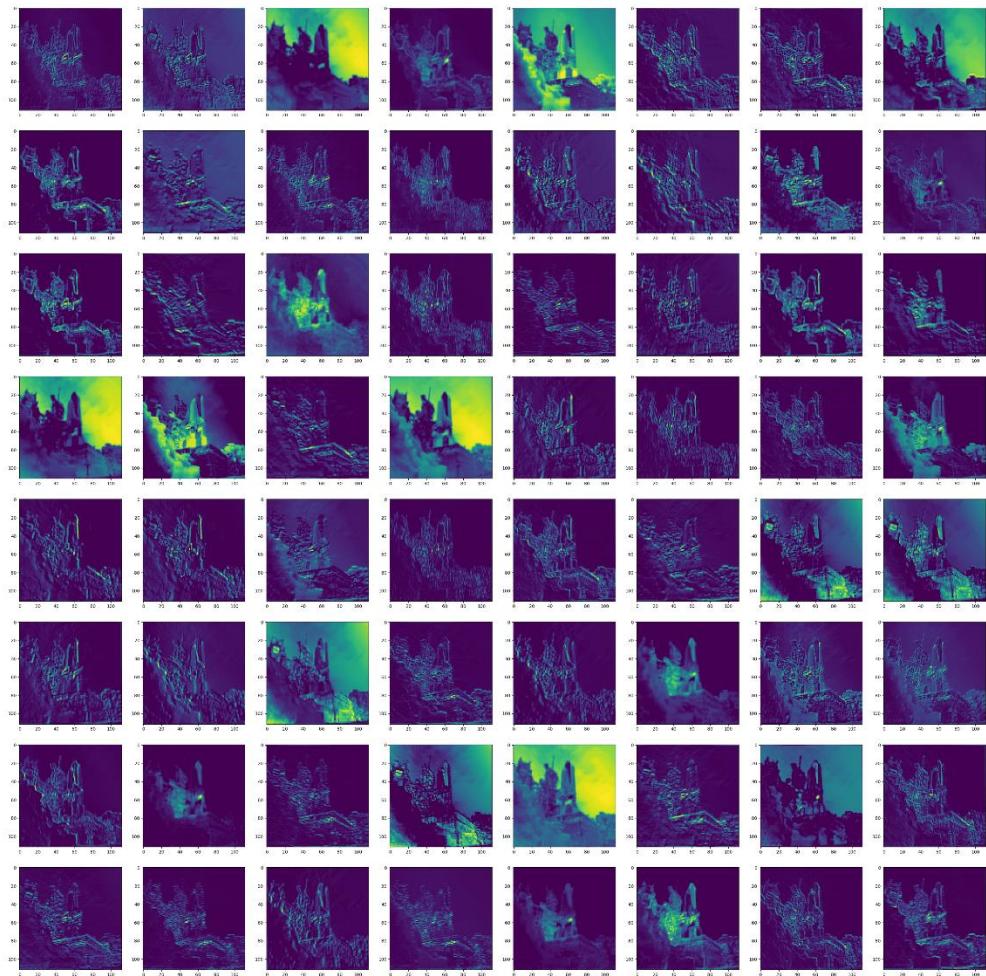


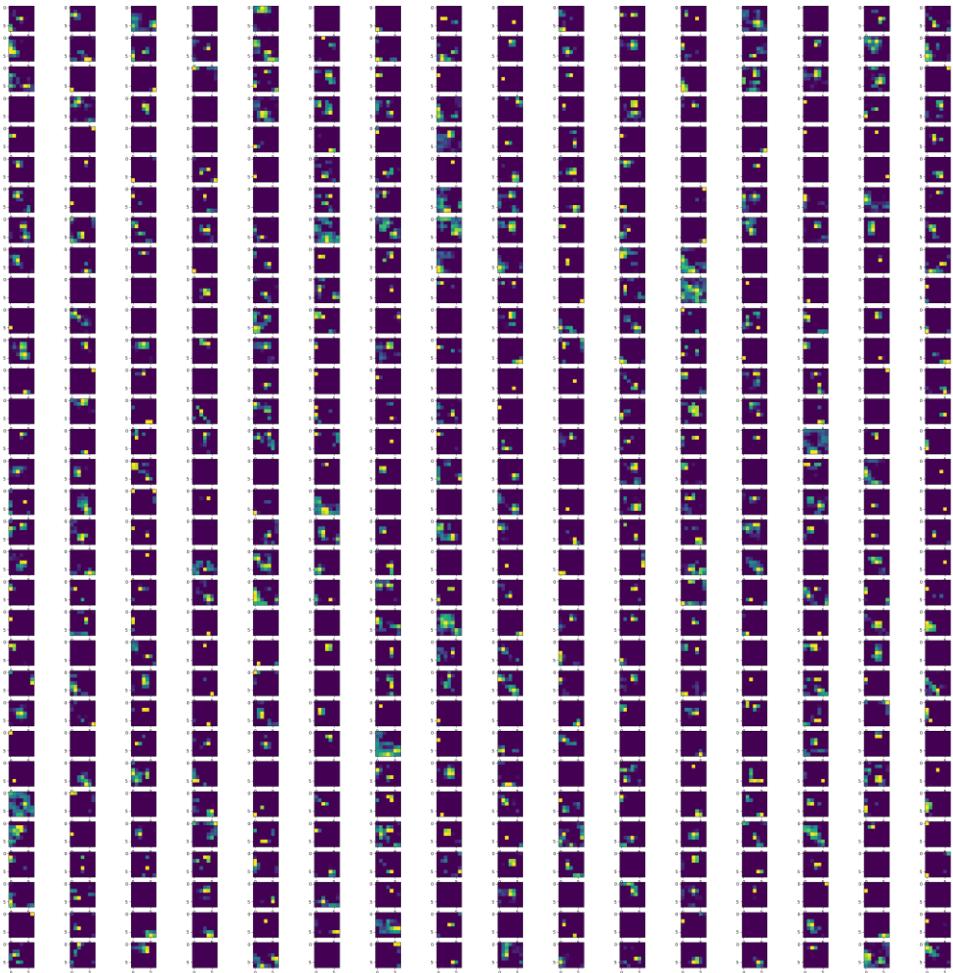
scenery





space shuttle



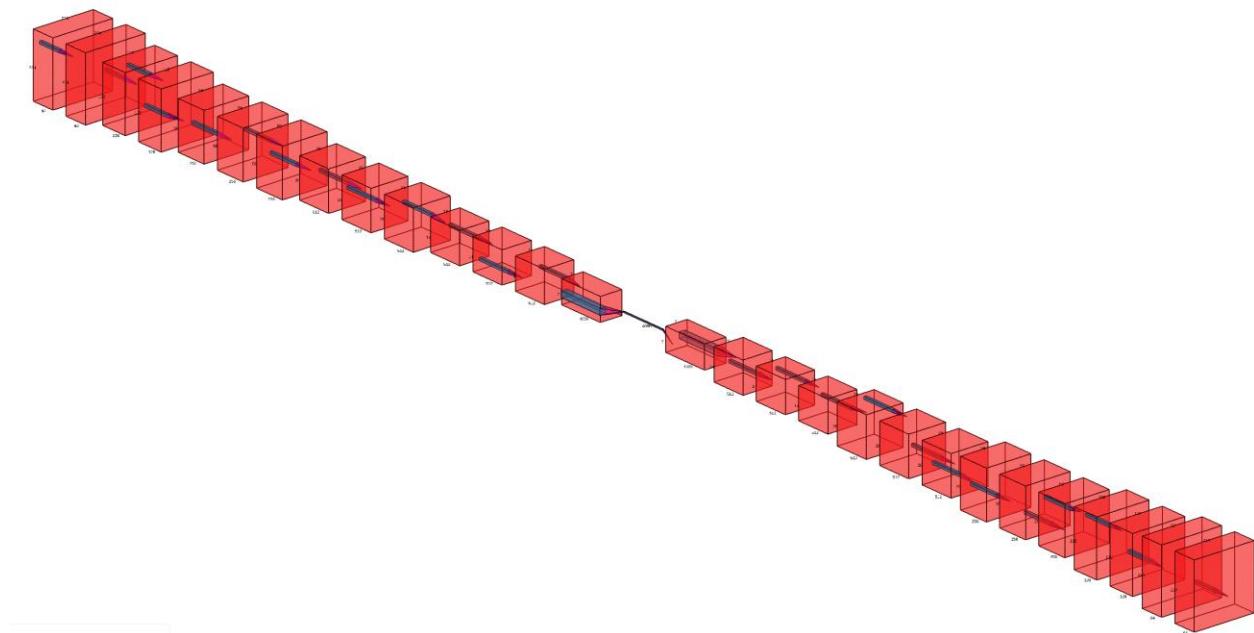


مشخص است که لایه‌های اولیه شکل تصویر را حفظ می‌کنند و ویژگی‌های اندکی از آن را مثل لبه و روشنایی‌های موضعی یا اطلاعات از این دست را کد می‌کنند اما ویژگی‌هایی که لایه‌های عمیق‌تر شبکه کد می‌کنند هیچ شباهتی به چیزی که ما درک می‌کنیم ندارد و احتمالاً مانند لایه‌های عمیق‌تر بینایی انسان ویژگی‌های پیچیده‌ای که ترکیب ویژگی‌های ساده‌تر است را کد می‌کنند. در هر صورت آن‌چه شبکه عصبی از واقعیت می‌بیند، در لایه‌های اولیه به درک انسان نزدیک‌تر است و

این از تصاویر بالا به سادگی استنباط می‌شود. تصاویر با معنی به لایه‌های کم‌عمق‌تر (لایه ۳) و تصاویر کوچک‌تر و بی‌مفهوم (برای ما) به لایه ۱۳ مربوط‌اند.

Deconvolution .۴

۱.۴. رسم شبکه و مشخصات آن: تصویر زیر به کمک مرجع [4] رسم شده‌است و ابعاد آن را در ادامه توضیح می‌دهیم.

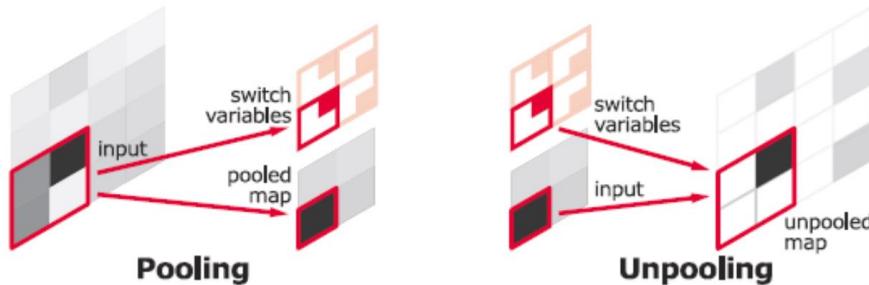


با توجه به کد نوشته شده در تابع `net.py`, ابتدا تصویری ۲۲۴ در ۲۲۴ به ورودی لایه کانولوشنی اول داده می‌شود. این لایه ۶۴ کanal با کرنل‌های ۳ در ۳ دارد. (تمامی کرنل‌ها تا انتهای به جز ۳ کرنل که به آن‌ها جداگانه اشاره می‌شود ۳ در ۳ هستند). این لایه کانولوشن دوبار اجرا می‌شود پس لایه دوم هم ۶۴ کanal با همان کرنل‌ها دارد. پس از آن لایه **max pooling** قرار دارد و ابعاد تصویر را نصف می‌کند و ۳ لایه کانولوشنی در مقابل این تصویر قرار می‌گیرد که هر کدام ۱۲۸ کرنل داشته و بر روی تصویر رسیده به این لایه که ۱۱۲ در ۱۱۲ است اعمال می‌شود. پس از آن هم یک لایه **max pooling** دیگر ابعاد تصاویر را دوباره نصف می‌کند و سه لایه کانولوشنی به عمق (تعداد کرنل) ۲۵۶ در مسیر تصویر ۵۶ در ۵۶ قرار داده می‌شود. در ادامه این روند که عمق شبکه زیاد شده و ابعاد تصاویر کوچک می‌شوند، با عبور از لایه **max pooling** بعدی، ابعاد تصویر به ۲۸ در ۲۸ رسیده و سه لایه کانولوشنی هر کدام با ۵۱۲ کرنل ۳ در ۳ (مثل قبل) بر روی تصویر عمل می‌کنند و با اعمال **max pooling** بعدی، ابعاد تصویر ۷ در ۷ می‌شود. اما لایه بعدی یک لایه کانولوشنی با ۴۰۹۶ کرنل ۷ در ۷ بوده که باعث می‌شود خروجی هر اعمال هر کرنل یک عدد بشود (یکی از دو بردار ۴۰۹۶ تایی قرار گرفته در وسط در

تصویر فوق) و بعد از آن این لایه ۴۰۹۶ تایی به لایه بعدی که ۴۰۹۶ تایی است به صورت تمام متصل با کرنل‌های ۱ در ۱ که همان تبدیل خطی می‌باشد وصل است. سپس خروجی با ۴۰۹۶ کرنل ۷ در ۷ مرتبط می‌شود. این روند تا رسیدن به ابعاد تصویر اصلی یا همان ۲۲۴ در ۲۲۴ درست مانند طرفی که توصیف شد ادامه می‌یابد و همه لایه‌های به جای کانولوشن، unpooling هستند.

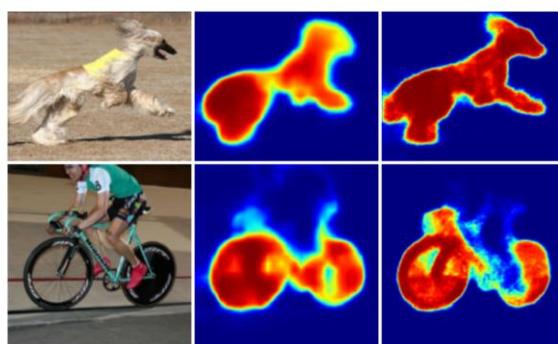
یک نکته قابل توجه این است که شبکه Deconvnet نوشته شده در کد داده شده، از **VGG16** به عنوان **backbone** استفاده می‌کند. (می‌توان با ساختار نشان داده شده در قسمت قبل مقایسه کرده و این گفته را تأیید کرد.)

۲. کاربرد شبکه‌های Deconvnet در بازسازی تصاویر با استفاده از ویژگی‌های بدست آمده در لایه fully connected است. در صورتی که از این شبکه‌ها استفاده نشود، upsampling نسبت زیادی دارد. (رسیدن از ۴۰۹۶ عدد در یک بردار به تصویر ۲۲۴ در ۲۲۴!) در حالی که به وسیله این شبکه‌ها unpooling به نحو مناسبی انجام می‌شود و اطلاعات تصویر و ویژگی‌های آن آرام‌تر و بهتر به تصویر نهایی منجر می‌شود.



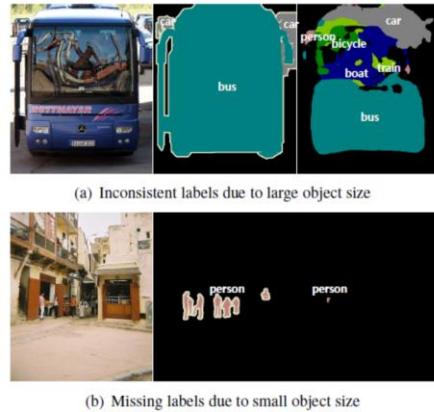
Remember positions when Pooling (Left), Reuse the position information during Unpooling (right)

در unpooling باید به خاطر بسپاریم که مقدار ماکریم در فرآیند pooling کجا بوده تا بتوانیم تصویر بزرگتر را بازسازی کنیم. همچنین با transposed convolution به نوعی عکس کانولوشن را انجام داده و با padding مناسب می‌توان interpolation انجام داد. نتیجه این ایده در مقایسه بازسازی تصاویر زیر، برتری روش Deconvnet را نسبت به بازسازی مستقیم از طریق لایه fully connected دهد:

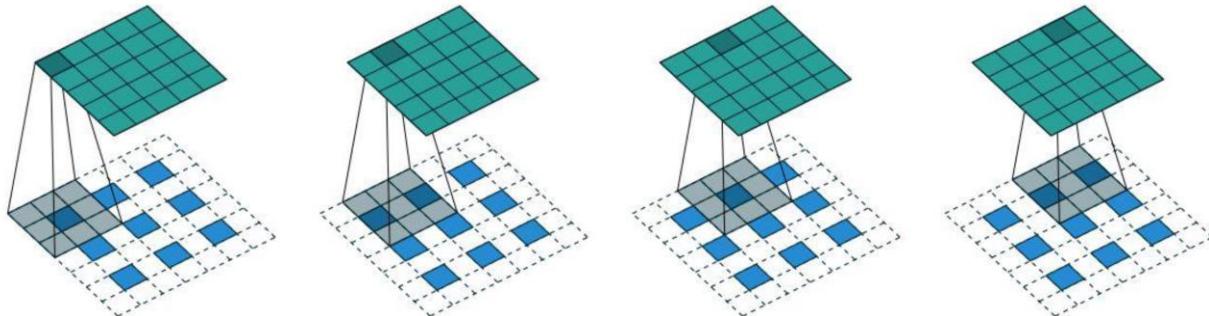


Input Image (Left), FCN-8s (Middle), DeconvNet (Right)

همچنین در تشخیص proposal هایی که در تصویر وجود دارد از Deconvnet استفاده می‌شود.



۳.۴. در شکل زیر عملیات Transpose Convolution مشاهده می‌شود. خروجی unpooling شده به وسیله کرنلی که padding لازم دارد، با عملیاتی ماتریسی شبیه کانولوشن به تصویری با ابعاد بزرگتر تبدیل می‌شود. این عملیات متناظر تبدیل وارون پاسخ ضربه گرفتن نیست و از این رو نمی‌توان آن را معکوس کانولوشن padding گفت! شباهت آن به کانولوشن فقط در همین عملیات ضرب ماتریسی ورودی در کرنل است که با همراه است تا بتواند با sweep کردن بر روی خروجی تصویر با ابعاد بزرگتر را تولید کند. لایه‌های deconvolution به این طریق به کمک unpooling ای که توضیح داده شد می‌توانند ویژگی‌های تصویر را بازسازی کرده و از بردار ویژگی ۴۰۹۶ تایی، تصویر خروجی با شباهت خوبی نسبت به تصویر ورودی بسازند.



References

- [1] [VGG in Tensorflow](#)
- [2] [VGG Net Architecture Quora](#)
- [3] [Very Deep Convolutional Networks For Large-Scale Image Recognition](#)
- [4] [Neural Net SVG Online](#)