# Deep Learning (CS324)

# Visualizing and Understanding Convolutional Networks

—— Matthew D. Zeiler and Rob Fergus

Nie Yuhe

11911839

# Content

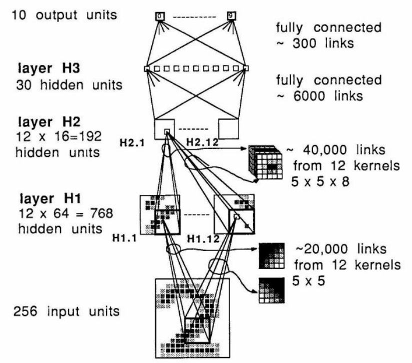
- Background
- Motivation
- Methodology
- Convnet Visualization
- Experiments
- Q&A

# Part I Background

# Previous Research — CNN

- Introduction of CNN LeCun *et al.*

  LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.,Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. NeuralComput. 1(4), 541–551 (1989)




- Good Performance on NORB, CIFAAR-10 Data Sets Ciresan *et al.*

  Ciresan, D.C., Meier, J., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: CVPR (2012)

- Good Performance on ImageNet2012 Krizhevsky *et al.*

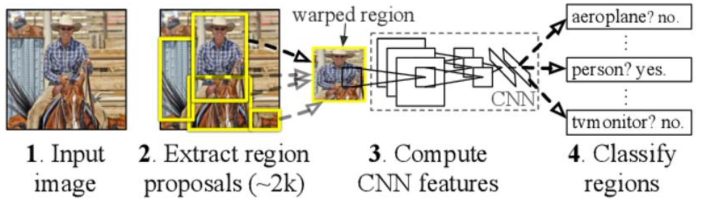
  Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)

IMAGENET

Error Rate 16.4%

- Good Performance on PASCAL VOC Data Sets Girshick *et al.*

  Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 (2014)



# Related Work — Visualization

- Mostly limited to the 1st Layer
- Performing gradient descent in image space to minimize the unit's activation Erhan *et a*l.

  Erhan, D., Bengio, Y., Courville, A., Vincent, P.: Visualizing higher-layer features of a deep network. Technical report, University of Montreal (2009)
- Projecting back from the fully connected layers  Simonyan *et al.*

  Le, Q.V., Ngiam, J., Chen, Z., Chia, D., Koh, P., Ng, A.Y.: Tiled convolutional neural networks. In: NIPS (2010)
- Visualization that identify patches that are responsible for strong activations at higher layer Girshick *et al.*

  Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 (2014)

# Related Work — Feature Generalization

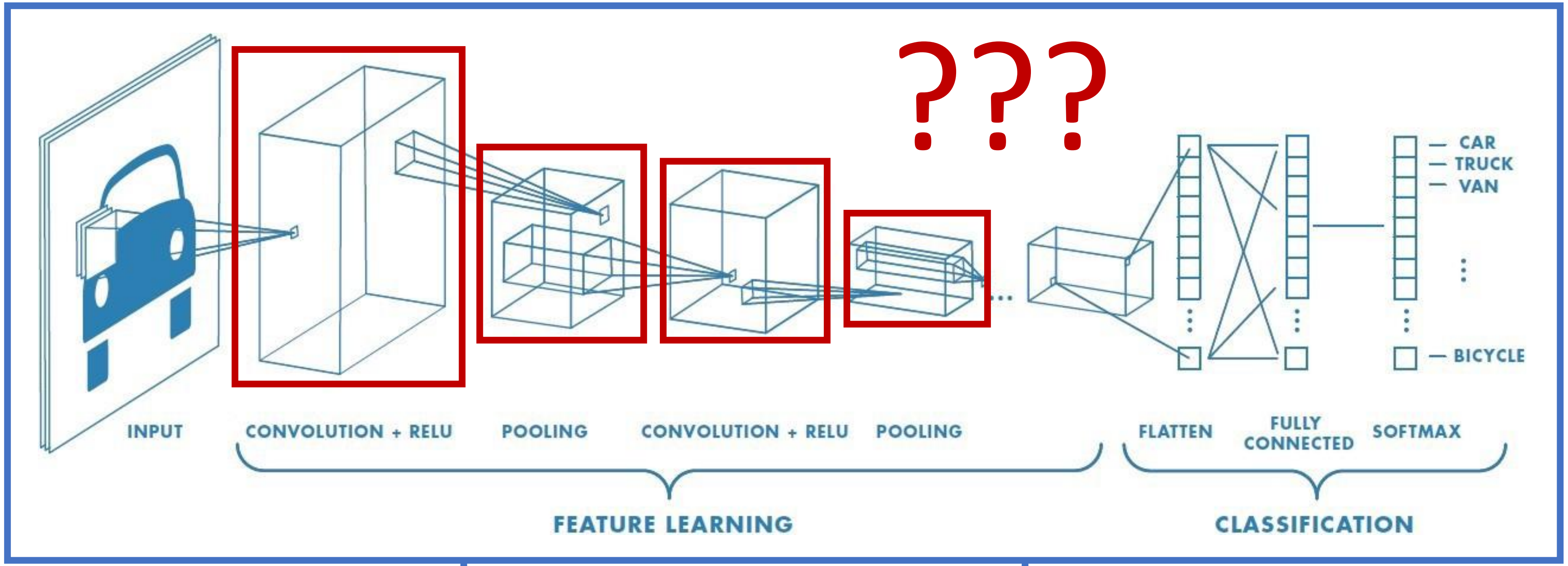
- Generalization Ability of CNN features Donahue & Girshick *et al.*

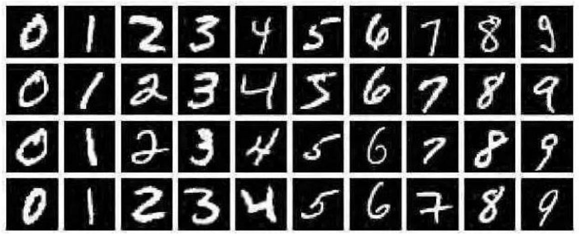
  Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.:DeCAF: A deep convolutional activation feature for generic visual recognition.arXiv:1310.1531 (2013)

  Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524 (2014)

# Part II Motivation

# What's Inside CNN?



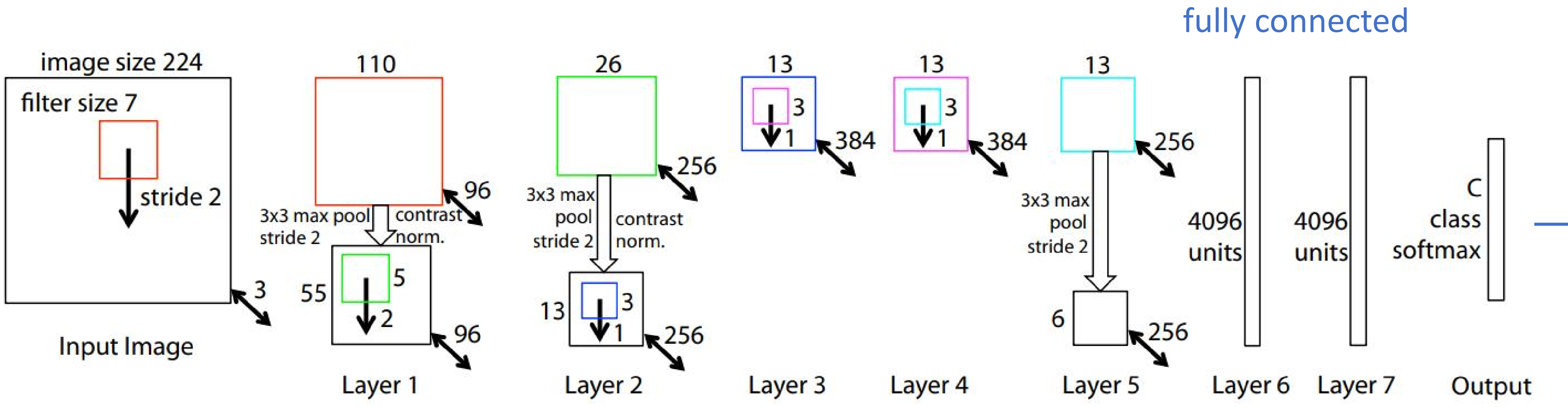



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

# Part III Methodology

# Architecture of Convnet

Similar to Krizhevsky ImageNet




Loss Function

Cross-Entropy

Input: 224 x 224 x 3(RGB) → Filter: 7 x 7 x 96(channel) Stride: 2 both x and y → Passed through a rectified a liner function (ReLU) → Max Pool: 3 x 3 Stride: 2 for both x and y → contrast normalized across feature maps

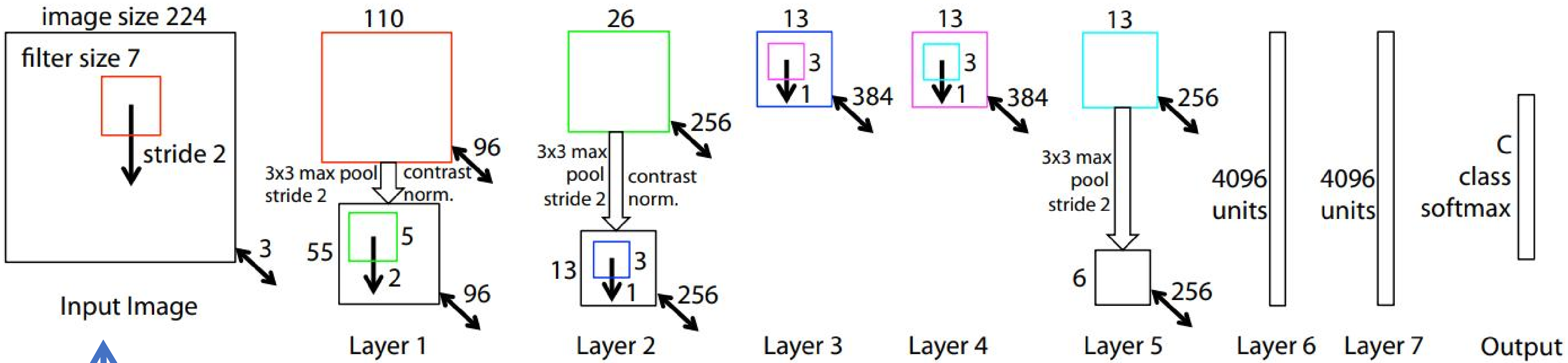
← Update Parameters: Back Propogation + SGD ←

Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Training Details

SGD batch size: 128
Learning Rate: $10^{-2}$
Momentum Term: 0.9
Dropout rate: 0.5

All weights are initialized to $10^{-2}$ and biases are set to 0




IMAGENET
2012 training set
1.3 million images
1000 different classes

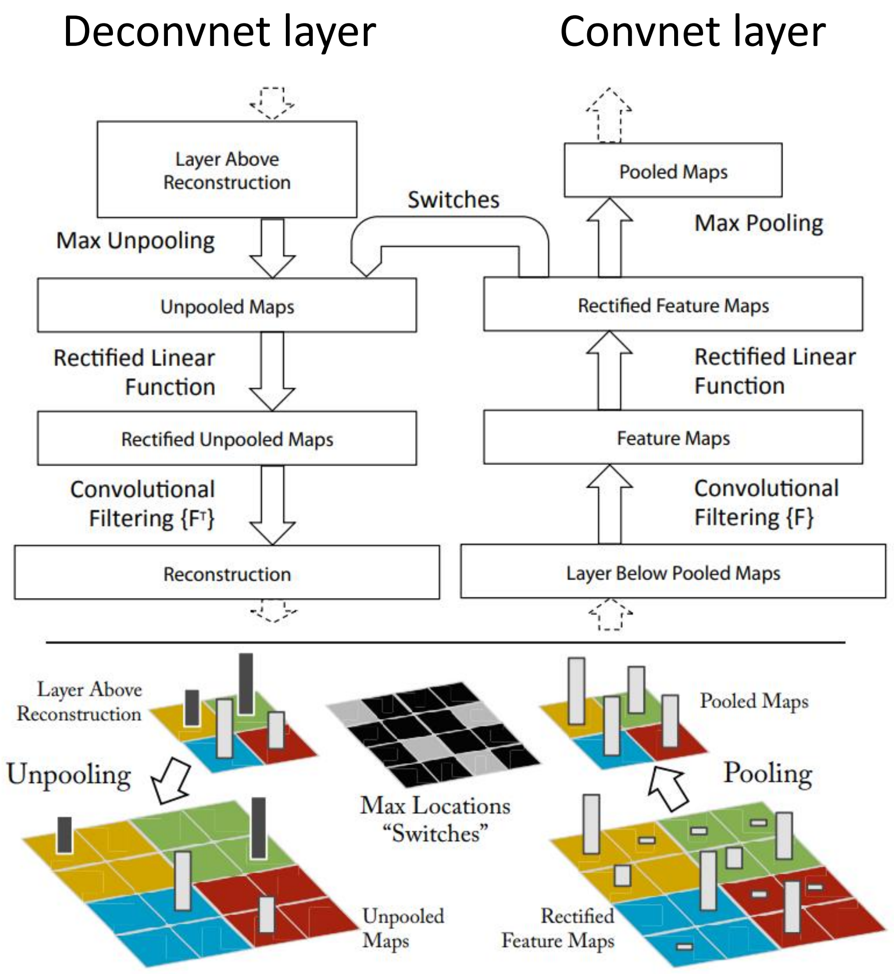
Renormalize each filter in the convolutional layers whose RMS value exceeds a fixed radius of $10^{-1}$ to this fixed radius

70 epochs
12 days
single GTX580 GPU

# Architecture of Deconvnet

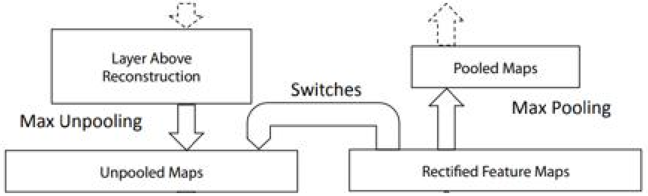
The deconvnet will reconstruct an approximate version of the convnet features from the layer beneath.

- Unpooling
  recording the locations of the maxima within each pooling region in a set of switch variables
- Rectification
  pass the reconstructed signal through ReLU
- Filtering
  transposed version of the same filters
  filpping each filter vertically and horizontally

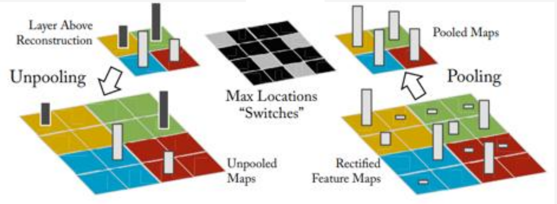



Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Deconvnet: Unpooling



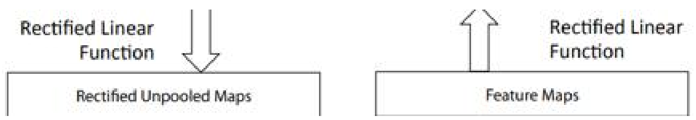

```python
class DPooling(object):

    def __init__(self, layer):

        self.layer = layer
        self.poolsize = layer.pool_size


    def up(self, data, learning_phase = 0):

        [self.up_data, self.switch] = \
                self.__max_pooling_with_switch(data, self.poolsize)
        return self.up_data

    def down(self, data, learning_phase = 0):

        self.down_data = self.__max_unpooling_with_switch(data, self.switch)
        return self.down_data
  # Compute unpooled output using pooled data and switch
  def __max_unpooling_with_switch(self, input, switch):

      print('switch '+str(switch.shape))
      print('input  '+str(input.shape))
      tile = np.ones((math.floor(switch.shape[1] / input.shape[1]),
          math.floor( switch.shape[2] / input.shape[2])))
      print('tile '+str(tile.shape))
      tile=numpy.expand_dims(tile,axis=3)
      input=numpy.squeeze(input,axis=0)
      out = np.kron(input, tile)
      print('out '+str(out.shape))
      unpooled = out * switch
      unpooled=numpy.expand_dims(unpooled,axis=0)
      return unpooled
```




```python
def __max_pooling_with_switch(self, input, poolsize):

    switch = np.zeros(input.shape)
    out_shape = list(input.shape)
    row_poolsize = int(poolsize[0])
    col_poolsize = int(poolsize[1])
    print(row_poolsize)
    print(col_poolsize)
    out_shape[1] = math.floor(out_shape[1] / poolsize[0])
    out_shape[2] = math.floor(out_shape[2] / poolsize[1])
    print(out_shape)
    pooled = np.zeros(out_shape)

    for sample in range(input.shape[0]):
        for dim in range(input.shape[3]):
            for row in range(out_shape[1]):
                for col in range(out_shape[2]):
                    patch = input[sample,
                            row * row_poolsize : (row + 1) * row_poolsize,
                            col * col_poolsize : (col + 1) * col_poolsize,dim]
                    max_value = patch.max()
                    pooled[sample, row, col,dim] = max_value
                    max_col_index = patch.argmax(axis = 1)
                    max_cols = patch.max(axis = 1)
                    max_row = max_cols.argmax()
                    max_col = max_col_index[max_row]
                    switch[sample,
                            row * row_poolsize + max_row,
                            col * col_poolsize + max_col,
                          dim]  = 1
    return [pooled, switch]
```

https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks

# Deconvnet: Rectification



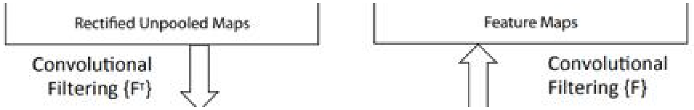

```python
class DActivation(object):
    def __init__(self, layer, linear = False):

        self.layer = layer
        self.linear = linear
        self.activation = layer.activation
        input = K.placeholder(shape = layer.output_shape)

        output = self.activation(input)
        # According to the original paper,
        # In forward pass and backward pass, do the same activation(relu)
        # Up method
        self.up_func = K.function(
                [input, K.learning_phase()], [output])
        # Down method
        self.down_func = K.function(
                [input, K.learning_phase()], [output])
```

```python
def up(self, data, learning_phase = 0):

    self.up_data = self.up_func([data, learning_phase])
    self.up_data=np.squeeze(self.up_data,axis=0)
    self.up_data=numpy.expand_dims(self.up_data,axis=0)
    print(self.up_data.shape)
    return self.up_data

def down(self, data, learning_phase = 0):

    self.down_data = self.down_func([data, learning_phase])
    self.down_data=np.squeeze(self.down_data,axis=0)
    self.down_data=numpy.expand_dims(self.down_data,axis=0)
    print(self.down_data.shape)
    return self.down_data
```

https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks

# Deconvnet: Filtering

```python
class DConvolution2D(object):

    def __init__(self, layer):
        self.layer = layer

        weights = layer.get_weights()
        W = weights[0]
        b = weights[1]

        filters = W.shape[3]
        up_row = W.shape[0]
        up_col = W.shape[1]
        input_img = keras.layers.Input(shape = layer.input_shape[1:])

        output=keras.layers.Conv2D(filters,(up_row,up_col),kernel_initializer=tf.constant_initializer(W),
                                   bias_initializer=tf.constant_initializer(b),padding='same')(input_img)
        self.up_func = K.function([input_img, K.learning_phase()], [output])
        # Deconv filter (exchange no of filters and depth of each filter)
        W = np.transpose(W, (0,1,3,2))
        # Reverse columns and rows
        W = W[::-1, ::-1,:,:]
        down_filters = W.shape[3]
        down_row = W.shape[0]
        down_col = W.shape[1]
        b = np.zeros(down_filters)
        input_d = keras.layers.Input(shape = layer.output_shape[1:])

        output=keras.layers.Conv2D(down_filters,(down_row,down_col),kernel_initializer=tf.constant_initializer(W),
                                   bias_initializer=tf.constant_initializer(b),padding='same')(input_d)
        self.down_func = K.function([input_d, K.learning_phase()], [output])
```

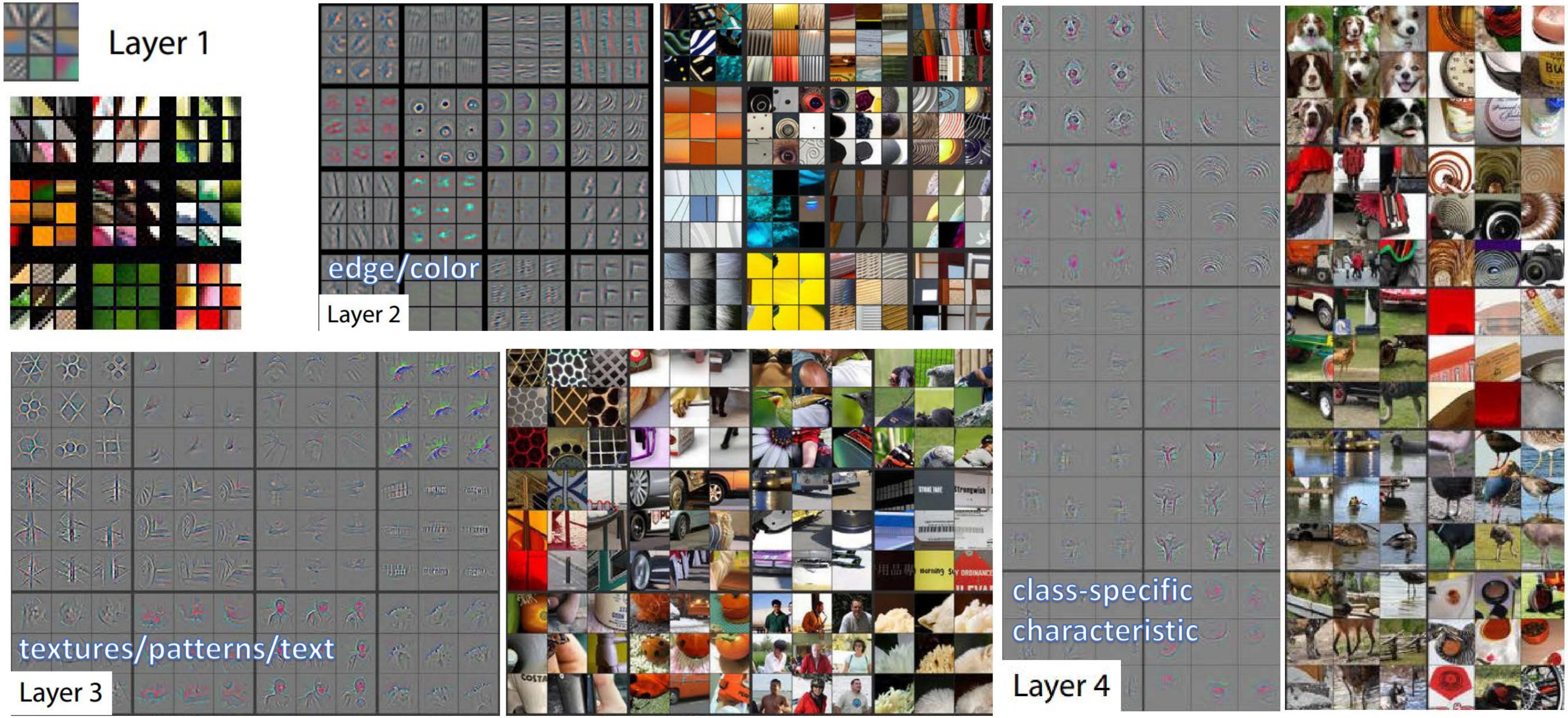



```python
def up(self, data, learning_phase = 0):
    #Forward pass
    self.up_data = self.up_func([data, learning_phase])
    self.up_data=np.squeeze(self.up_data,axis=0)
    self.up_data=numpy.expand_dims(self.up_data,axis=0)
    #print(self.up_data.shape)
    return self.up_data

def down(self, data, learning_phase = 0):
    # Backward pass
    self.down_data= self.down_func([data, learning_phase])
    self.down_data=np.squeeze(self.down_data,axis=0)
    self.down_data=numpy.expand_dims(self.down_data,axis=0)
    #print(self.down_data.shape)
    return self.down_data
```

https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks

# Part IV Convnet Visualization

# Convnet Visualization

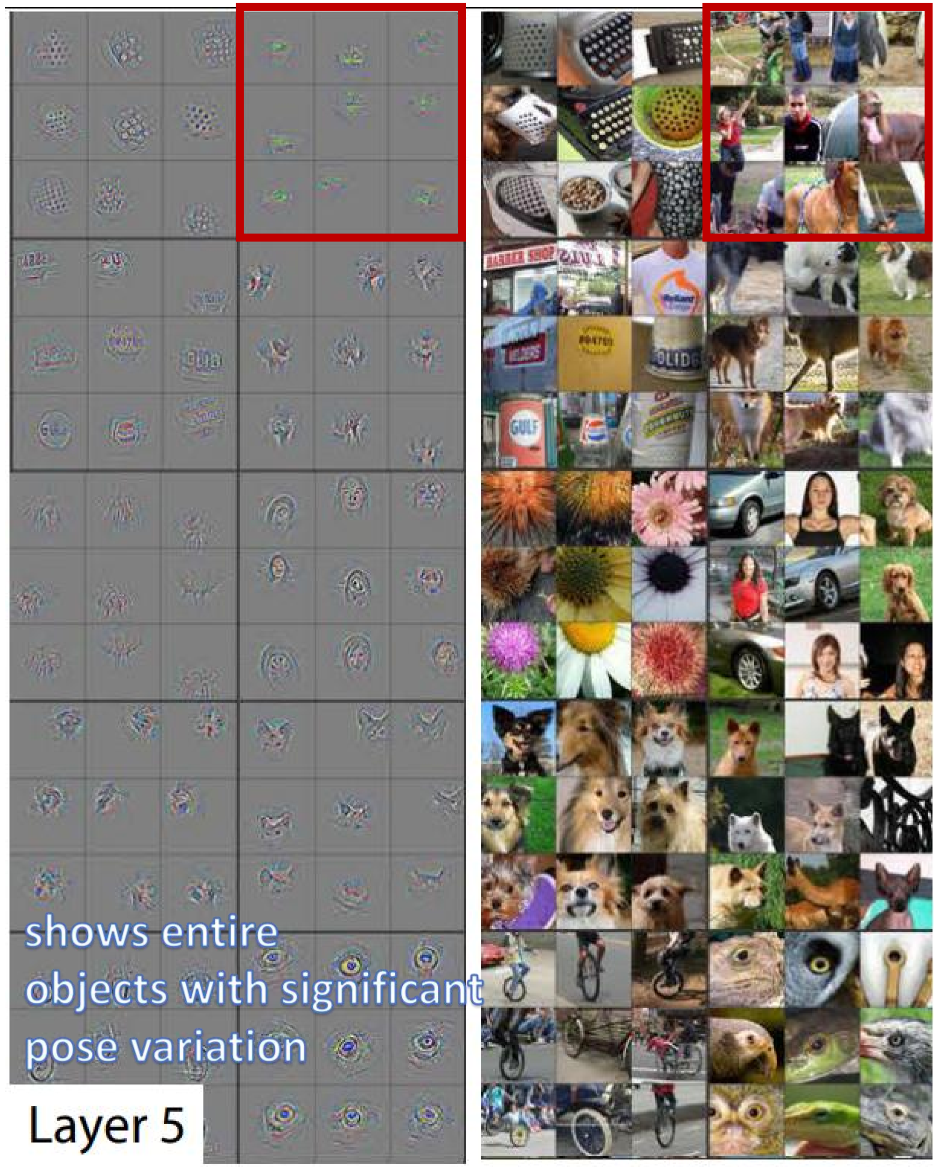



Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Convnet Visualization

Original image

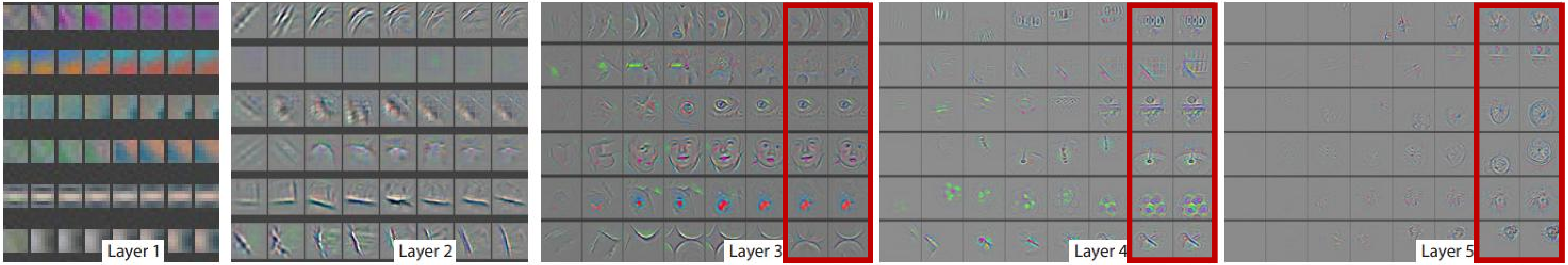
Features detected in feature map 127 of convolutional layer 3 of block 3 of VGG16 model




Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus
https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks

# Feature Evolution

The strongest activation (across all training examples) for a given feature map, projected down to pixel space using the deconvnet approach

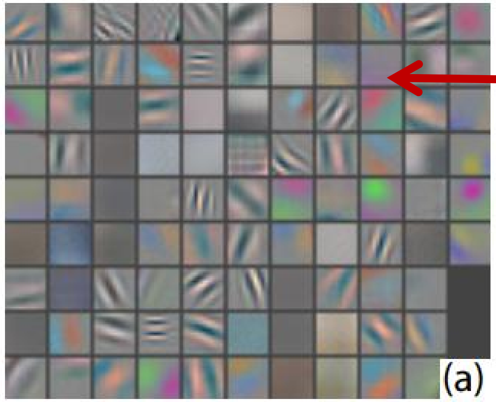


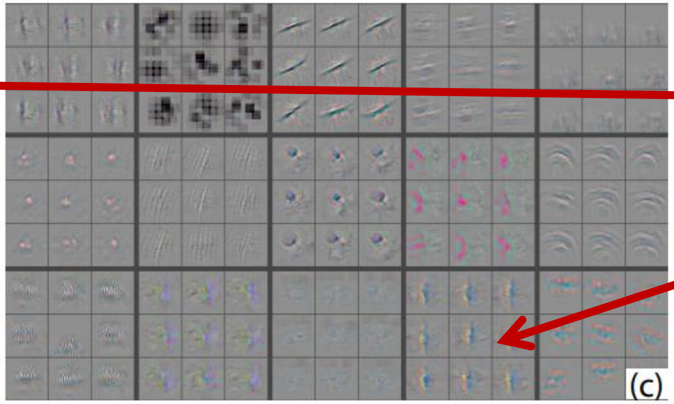

A randomly chosen subset of features at epochs [1,2,5,10,20,30,40,64]

Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Architecture Selection

Krizhevsky *et al.*



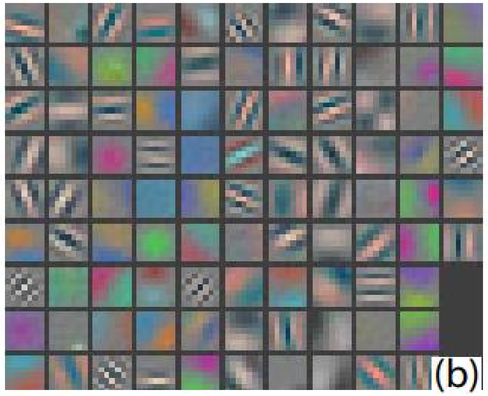



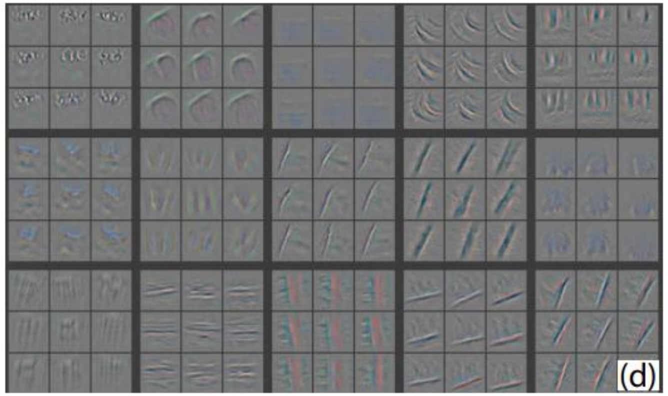

little coverage of the mid frequencies

aliasing artifacts

(i) reduced the 1st layer filter size from 11x11 to 7x7
(ii) made the stride of the convolution 2, rather than 4

Adjust



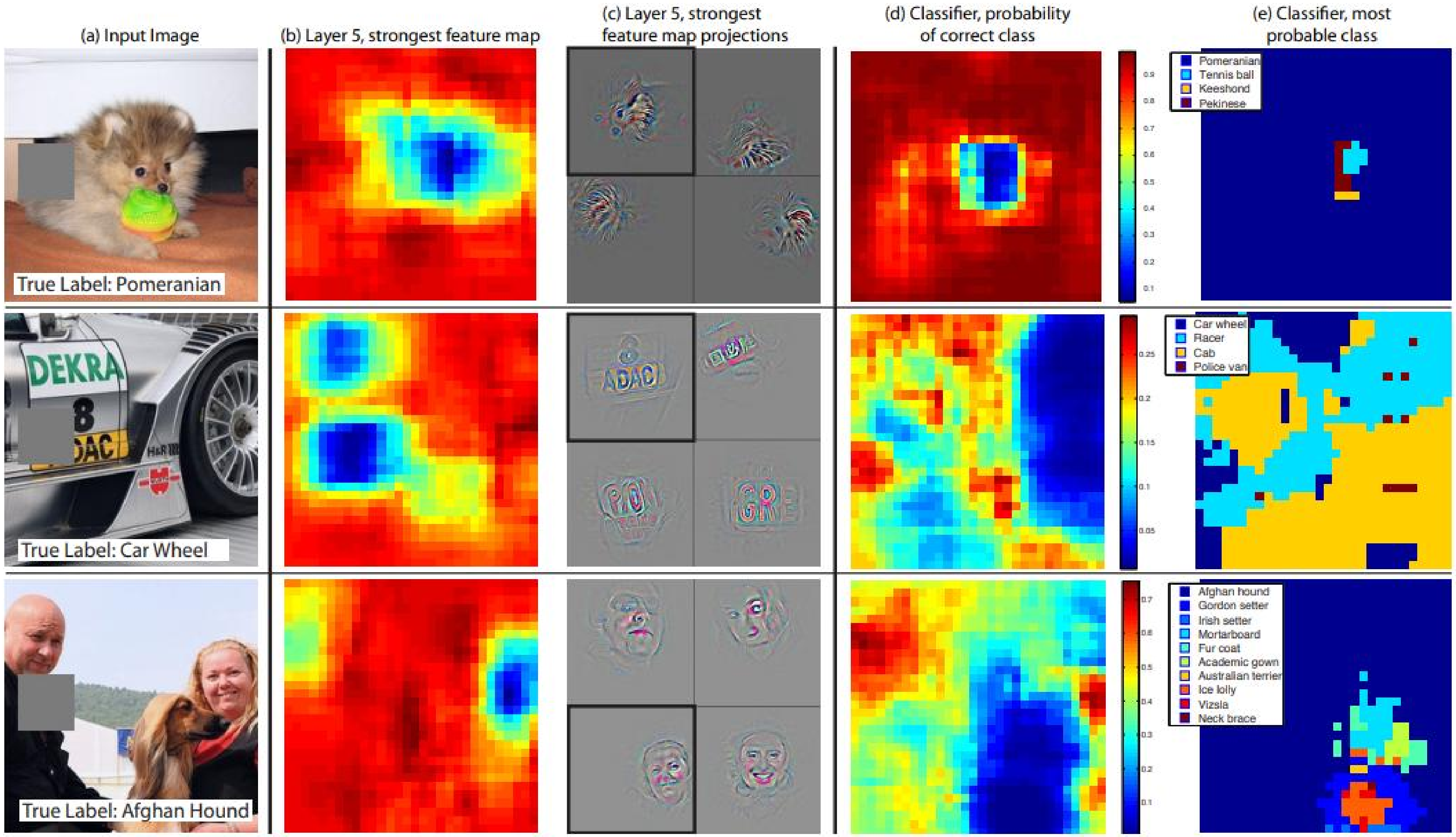




improves the classification performance

# Occlusion Sensitivity

the probability of the correct class drops significantly when the object is occluded

the visualization genuinely corresponds to the image structure that stimulates that feature map




Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Part V Experiments

# Experiement Result - ImageNet2012

Change the Srchitecture

(i) reduced the 1st layer filter size from 11x11 to 7x7

(ii) made the stride of the convolution 2, rather than 4

| Error % | Val Top-1 | Val Top-5 | Test Top-5 |
|---|---|---|---|
| Gunji *et al.* [12] | - | - | 26.2 |
| DeCAF [7] | - | - | 19.2 |
| Krizhevsky *et al.* [18], 1 convnet | 40.7 | 18.2 | −− |
| Krizhevsky *et al.* [18], 5 convnets | 38.1 | 16.4 | 16.4 |
| Krizhevsky *et al.* *[18], 1 convnets | 39.0 | 16.6 | −− |
| Krizhevsky *et al.* *[18], 7 convnets | 36.7 | 15.4 | 15.3 |
| Our replication of Krizhevsky *et al.* , 1 convnet | 40.5 | 18.1 | −− |
| 1 convnet as per Fig. 3 | 38.4 | 16.5 | −− |
| 5 convnets as per Fig. 3 – (a) | 36.7 | 15.3 | 15.3 |
| 1 convnet as per Fig. 3 but with layers 3,4,5: 512,1024,512 maps – (b) | 37.5 | 16.0 | 16.1 |
| 6 convnets, (a) & (b) combined | 36.0 | 14.7 | 14.8 |
| Howard [15] | - | - | 13.5 |
| Clarifai [28] | - | - | 11.7 |

Data Augmentation

Weak

# Experiement Result - ImageNet2012

## Varying ImageNet Model Sizes

- overall depth of the model is important for obtaining good performance

A slight increase in error when only remove the fully connected layers/convolutionallayers

dramatically worse

- increase the size of CNN layer will give a good performance

Changing the size of the fully connected layers makes little difference to performance

Increasing the size of the middle convolution layers goes give a useful gain in performance

| Error % | Train Top-1 | Val Top-1 | Val Top-5 |
|---|---|---|---|
| Our replication of Krizhevsky *et al.* [18], 1 convnet | 35.1 | 40.5 | 18.1 |
| Removed layers 3,4 | 41.8 | 45.4 | 22.1 |
| Removed layer 7 | 27.4 | 40.0 | 18.4 |
| Removed layers 6,7 | 27.4 | 44.8 | 22.4 |
| Removed layer 3,4,6,7 | 71.1 | 71.3 | 50.1 |
| Adjust layers 6,7: 2048 units | 40.3 | 41.7 | 18.8 |
| Adjust layers 6,7: 8192 units | 26.8 | 40.0 | 18.1 |
| Our Model (as per Fig. 3) | 33.1 | 38.4 | 16.5 |
| Adjust layers 6,7: 2048 units | 38.2 | 40.2 | 17.6 |
| Adjust layers 6,7: 8192 units | 22.0 | 38.8 | 17.0 |
| Adjust layers 3,4,5: 512,1024,512 maps | 18.8 | **37.5** | **16.0** |
| Adjust layers 6,7: 8192 units and Layers 3,4,5: 512,1024,512 maps | **10.0** | 38.3 | 16.9 |

# Feature Generalization

| # Train | Acc % 15/class | Acc % 30/class |
|---|---|---|
| Bo *et al.* [3] | – | $81.4 \pm 0.33$ |
| Yang *et al.* [17] | 73.2 | 84.3 |
| Non-pretrained convnet | $22.8 \pm 1.5$ | $46.5 \pm 1.7$ |
| ImageNet-pretrained convnet | $\mathbf{83.8 \pm 0.5}$ | $\mathbf{86.5 \pm 0.5}$ |

Caltech-101

| # Train | Acc % 15/class | Acc % 30/class | Acc % 45/class | Acc % 60/class |
|---|---|---|---|---|
| Sohn *et al.* [24] | 35.1 | 42.1 | 45.7 | 47.9 |
| Bo *et al.* [3] | $40.5 \pm 0.4$ | $48.0 \pm 0.2$ | $51.9 \pm 0.2$ | $55.2 \pm 0.3$ |
| Non-pretr. | $9.0 \pm 1.4$ | $22.5 \pm 0.7$ | $31.2 \pm 0.5$ | $38.8 \pm 1.4$ |
| ImageNet-pretr. | $\mathbf{65.7 \pm 0.2}$ | $\mathbf{70.6 \pm 0.2}$ | $\mathbf{72.7 \pm 0.4}$ | $\mathbf{74.2 \pm 0.3}$ |

Caltech-256

| Acc % | [22] | [27] | [21] | Ours | Acc % | [22] | [27] | [21] | Ours |
|---|---|---|---|---|---|---|---|---|---|
| Airplane | 92.0 | **97.3** | 94.6 | 96.0 | Dining table | 63.2 | **77.8** | 69.0 | 67.7 |
| Bicycle | 74.2 | **84.2** | 82.9 | 77.1 | Dog | 68.9 | 83.0 | **92.1** | 87.8 |
| Bird | 73.0 | 80.8 | 88.2 | **88.4** | Horse | 78.2 | 87.5 | **93.4** | 86.0 |
| Boat | 77.5 | 85.3 | 60.3 | **85.5** | Motorbike | 81.0 | **90.1** | 88.6 | 85.1 |
| Bottle | 54.3 | **60.8** | 60.3 | 55.8 | Person | 91.6 | 95.0 | **96.1** | 90.9 |
| Bus | 85.2 | **89.9** | 89.0 | 85.8 | Potted plant | 55.9 | 57.8 | **64.3** | 52.2 |
| Car | 81.9 | **86.8** | 84.4 | 78.6 | Sheep | 69.4 | 79.2 | **86.6** | 83.6 |
| Cat | 76.4 | 89.3 | 90.7 | **91.2** | Sofa | 65.4 | **73.4** | 62.3 | 61.1 |
| Chair | 65.2 | **75.4** | 72.1 | 65.0 | Train | 86.7 | **94.5** | 91.1 | 91.8 |
| Cow | 63.2 | 77.8 | **86.8** | 74.4 | Tv | 77.4 | **80.7** | 79.8 | 76.1 |
| Mean | 74.3 | 82.2 | **82.8** | 79.0 | # won | 0 | **11** | 6 | 3 |

PASCAL 2012

Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus

# Part VI Q&A

# Thank You!