Lab: CudaVision – Learning Vision Systems on Graphics Cards (MA-INF 4308)

Assignment 5

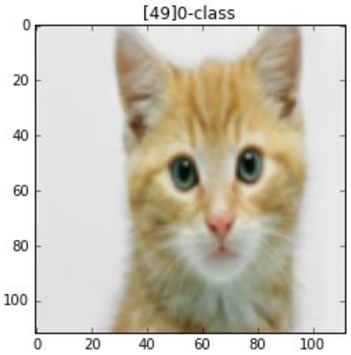
27.6.2016

Prof. Sven Behnke
Dr. Seongyong Koo

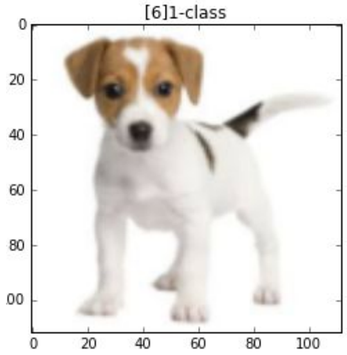
Fine-tuning CNN with Pre-trained VGG model

- **Goal: How to utilize a pre-trained CNN model for your custom dataset**

- **Generating custom dataset for TensorFlow**
  - Example, Cat of Dog?



[49]0-class



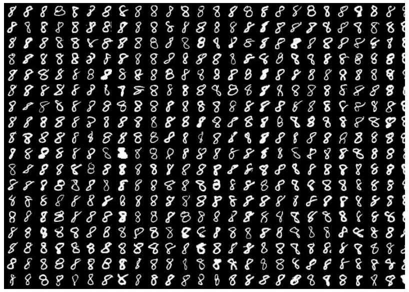
[6]1-class

- **Pre-trained VGG model**
  - 19 CNN layers with 144 millions trained weights from ILSVRC2012 dataset

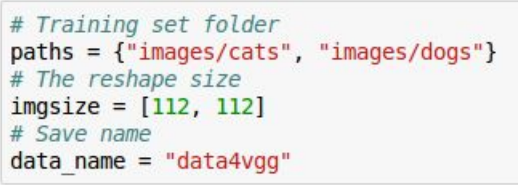
- **Theoretical reference**
  - Simonyan and Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition [http://arxiv.org/abs/1409.1556]
  - Donahue et al., DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [https://arxiv.org/abs/1310.1531]

Generating custom image dataset

- Sick and tired of MNIST?
  - Make your own custom image dataset

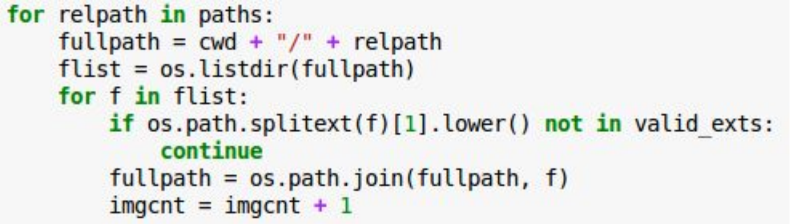


- Example code in `generate_dataset.ipynb`
  - Define where images are and reshape size
    - Dog or Cat?



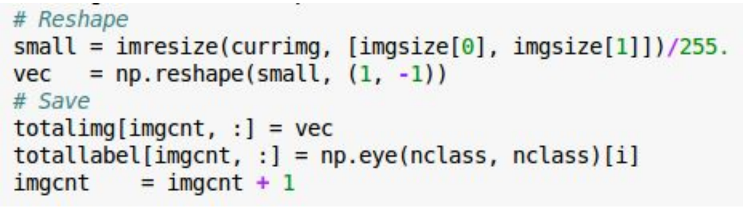
```
# Training set folder
paths = {"images/cats", "images/dogs"}
# The reshape size
imgsize = [112, 112]
# Save name
data_name = "data4vgg"
```

  - Count the number of images



```
for relpath in paths:
    fullpath = cwd + "/" + relpath
    flist = os.listdir(fullpath)
    for f in flist:
        if os.path.splitext(f)[1].lower() not in valid_exts:
            continue
        fullpath = os.path.join(fullpath, f)
        imgcnt = imgcnt + 1
```

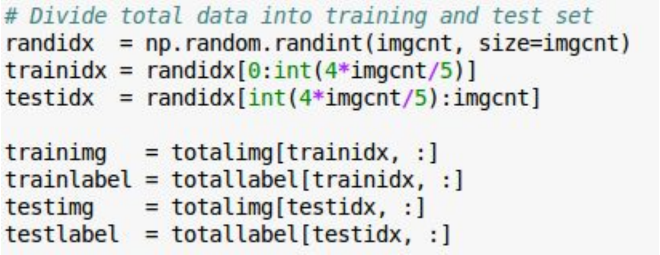
  - Reshaping



```
# Reshape
small = imresize(currimg, [imgsize[0], imgsize[1]])/255.
vec   = np.reshape(small, (1, -1))
# Save
totalimg[imgcnt, :] = vec
totallabel[imgcnt, :] = np.eye(nclass, nclass)[i]
imgcnt     = imgcnt + 1
```

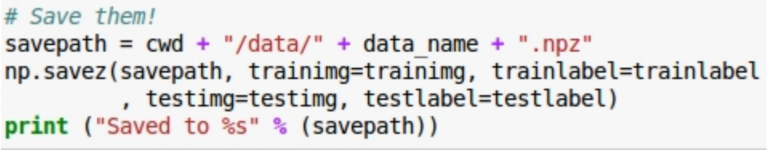
Generating custom image dataset

- Example code in `generate_dataset.ipynb`
  - Divide total data into training and test set

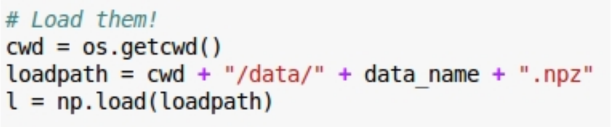


```
# Divide total data into training and test set
randidx  = np.random.randint(imgcnt, size=imgcnt)
trainidx = randidx[0:int(4*imgcnt/5)]
testidx  = randidx[int(4*imgcnt/5):imgcnt]

trainimg   = totalimg[trainidx, :]
trainlabel = totallabel[trainidx, :]
testimg    = totalimg[testidx, :]
testlabel  = totallabel[testidx, :]
```

  - Save and load data file

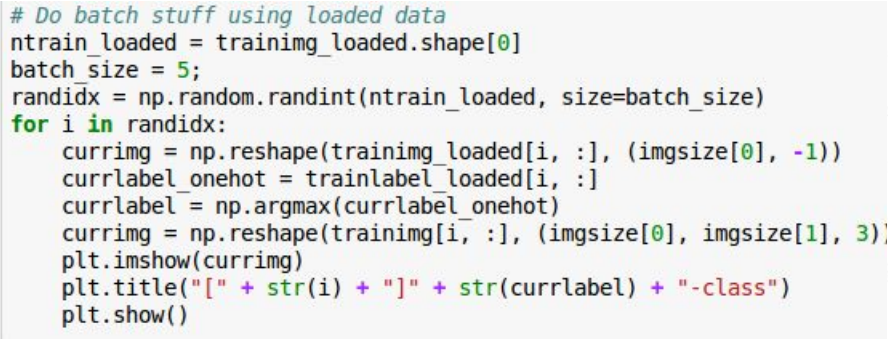


```
# Save them!
savepath = cwd + "/data/" + data_name + ".npz"
np.savez(savepath, trainimg=trainimg, trainlabel=trainlabel
         , testimg=testimg, testlabel=testlabel)
print ("Saved to %s" % (savepath))
```

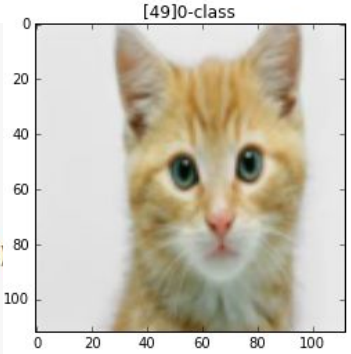


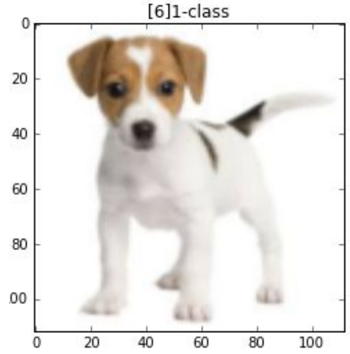
```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/" + data_name + ".npz"
l = np.load(loadpath)
```

  - Plot imgs



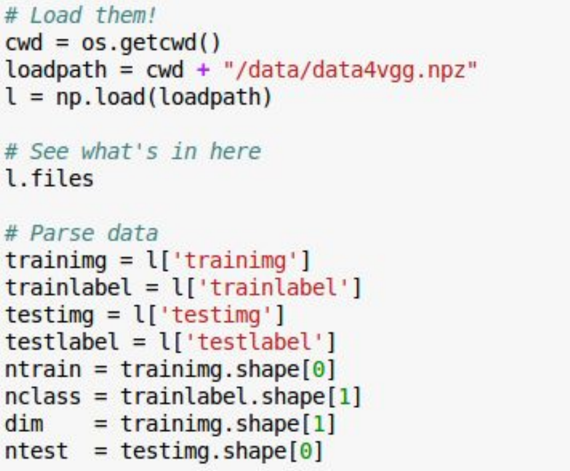
```
# Do batch stuff using loaded data
ntrain_loaded = trainimg_loaded.shape[0]
batch_size = 5;
randidx = np.random.randint(ntrain_loaded, size=batch_size)
for i in randidx:
    currimg = np.reshape(trainimg_loaded[i, :], (imgsize[0], -1))
    currlabel_onehot = trainlabel_loaded[i, :]
    currlabel = np.argmax(currlabel_onehot)
    currimg = np.reshape(trainimg[i, :], (imgsize[0], imgsize[1], 3))
    plt.imshow(currimg)
    plt.title("[" + str(i) + "]" + str(currlabel) + "-class")
    plt.show()
```



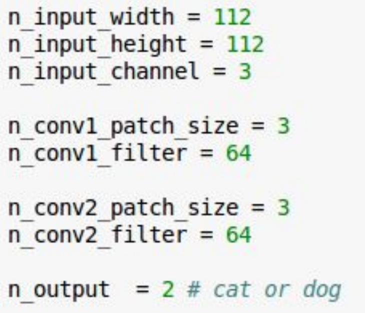


CNN with custom dataset

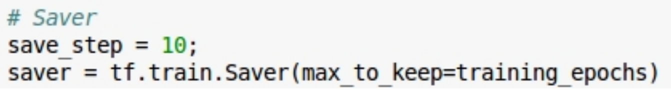
- In `cnn_custom_dataset.ipynb`
  - Load custum dataset
  - CNN definition
  - Training
  - Save net



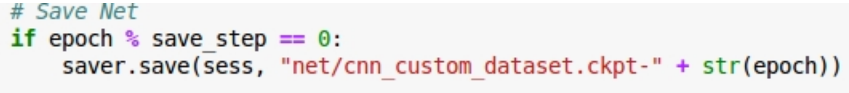
```
# Load them!
cwd = os.getcwd()
loadpath = cwd + "/data/data4vgg.npz"
l = np.load(loadpath)

# See what's in here
l.files

# Parse data
trainimg = l['trainimg']
trainlabel = l['trainlabel']
testimg = l['testimg']
testlabel = l['testlabel']
ntrain = trainimg.shape[0]
nclass = trainlabel.shape[1]
dim    = trainimg.shape[1]
ntest  = testimg.shape[0]
```



```
n_input_width = 112
n_input_height = 112
n_input_channel = 3

n_conv1_patch_size = 3
n_conv1_filter = 64

n_conv2_patch_size = 3
n_conv2_filter = 64

n_output  = 2 # cat or dog
```

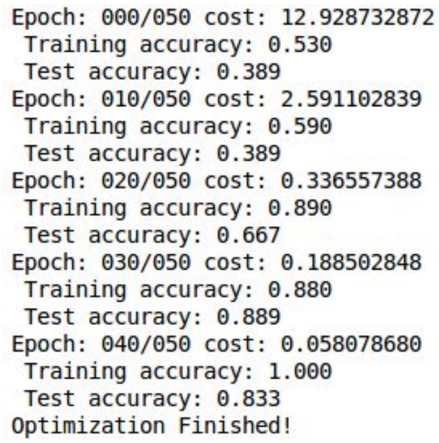


```
# Saver
save_step = 10;
saver = tf.train.Saver(max_to_keep=training_epochs)
```



```
# Save Net
if epoch % save_step == 0:
    saver.save(sess, "net/cnn_custom_dataset.ckpt-" + str(epoch))
```

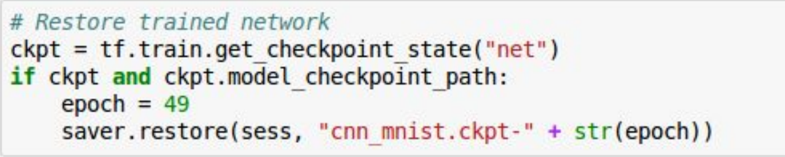
- Converged!
  - Test accuracy 0.833



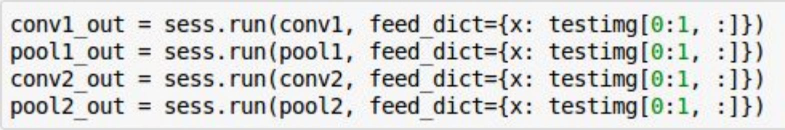
```
Epoch: 000/050 cost: 12.928732872
 Training accuracy: 0.530
 Test accuracy: 0.389
Epoch: 010/050 cost: 2.591102839
 Training accuracy: 0.590
 Test accuracy: 0.389
Epoch: 020/050 cost: 0.336557388
 Training accuracy: 0.890
 Test accuracy: 0.667
Epoch: 030/050 cost: 0.188502848
 Training accuracy: 0.880
 Test accuracy: 0.889
Epoch: 040/050 cost: 0.058078680
 Training accuracy: 1.000
 Test accuracy: 0.833
Optimization Finished!
```

CNN with custom dataset

- In `cnn_custom_dataset.ipynb`
  - Restore trained network

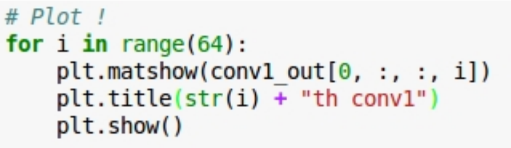


```
# Restore trained network
ckpt = tf.train.get_checkpoint_state("net")
if ckpt and ckpt.model_checkpoint_path:
    epoch = 49
    saver.restore(sess, "cnn_mnist.ckpt-" + str(epoch))
```

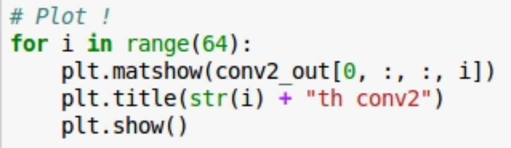


```
conv1_out = sess.run(conv1, feed_dict={x: testimg[0:1, :]})
pool1_out = sess.run(pool1, feed_dict={x: testimg[0:1, :]})
conv2_out = sess.run(conv2, feed_dict={x: testimg[0:1, :]})
pool2_out = sess.run(pool2, feed_dict={x: testimg[0:1, :]})
```

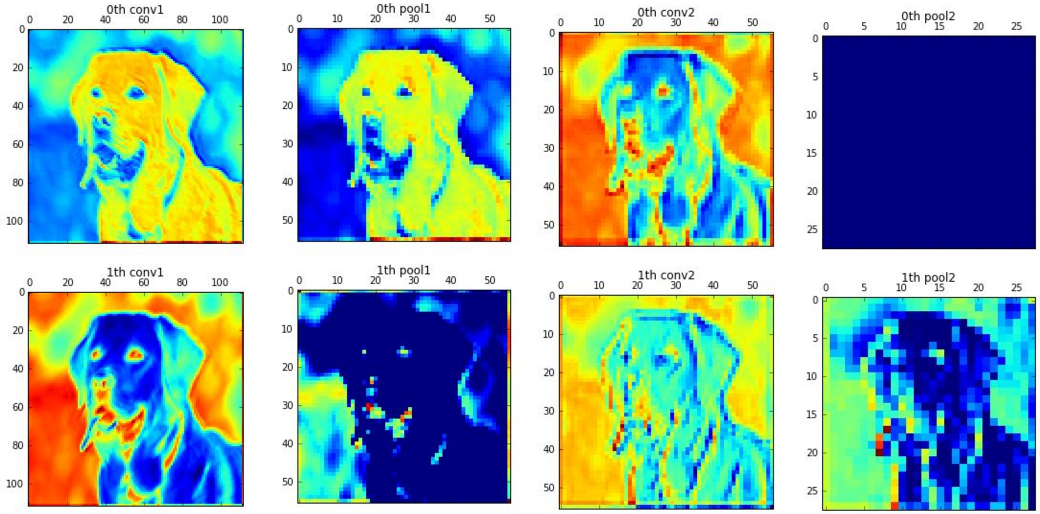
  - Plot



```
# Plot !
for i in range(64):
    plt.matshow(conv1_out[0, :, :, i])
    plt.title(str(i) + "th conv1")
    plt.show()
```

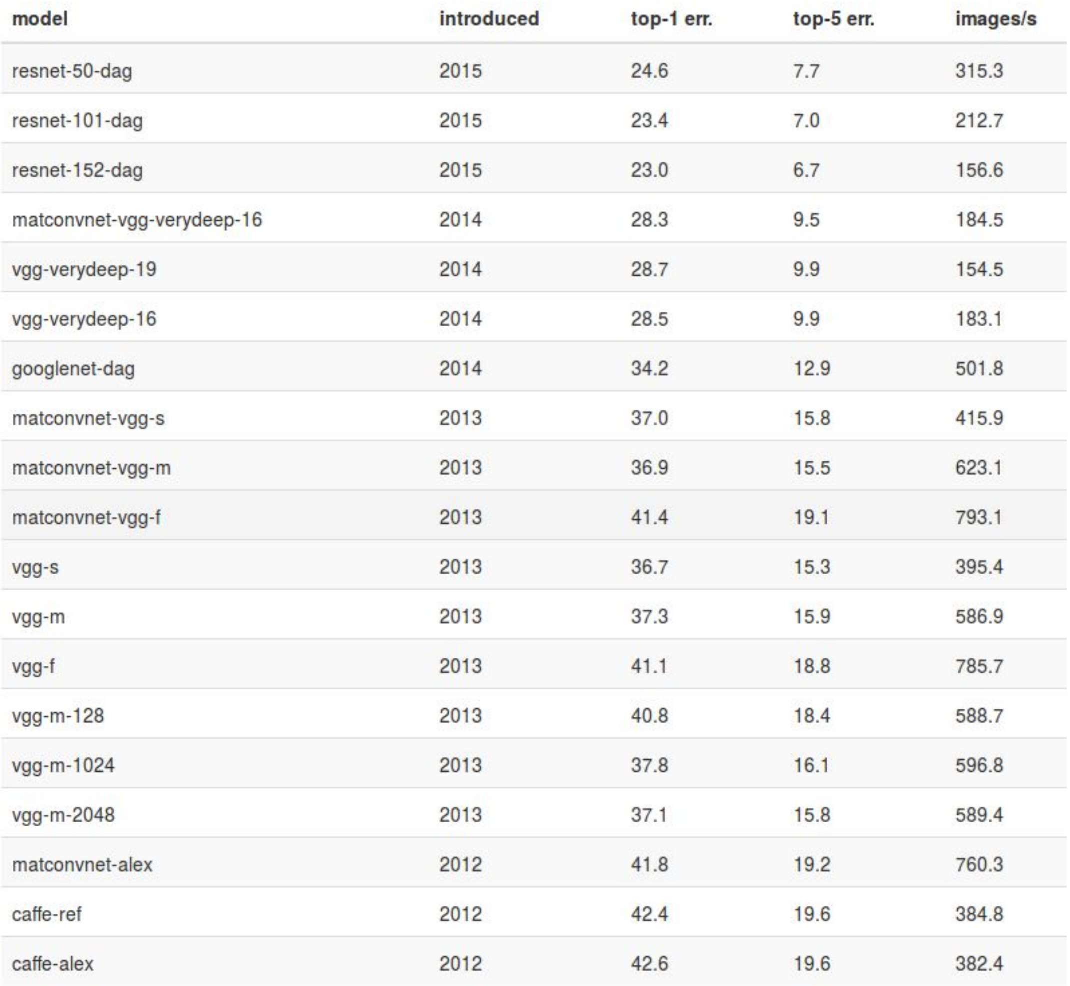


```
# Plot !
for i in range(64):
    plt.matshow(conv2_out[0, :, :, i])
    plt.title(str(i) + "th conv2")
    plt.show()
```



VGG model

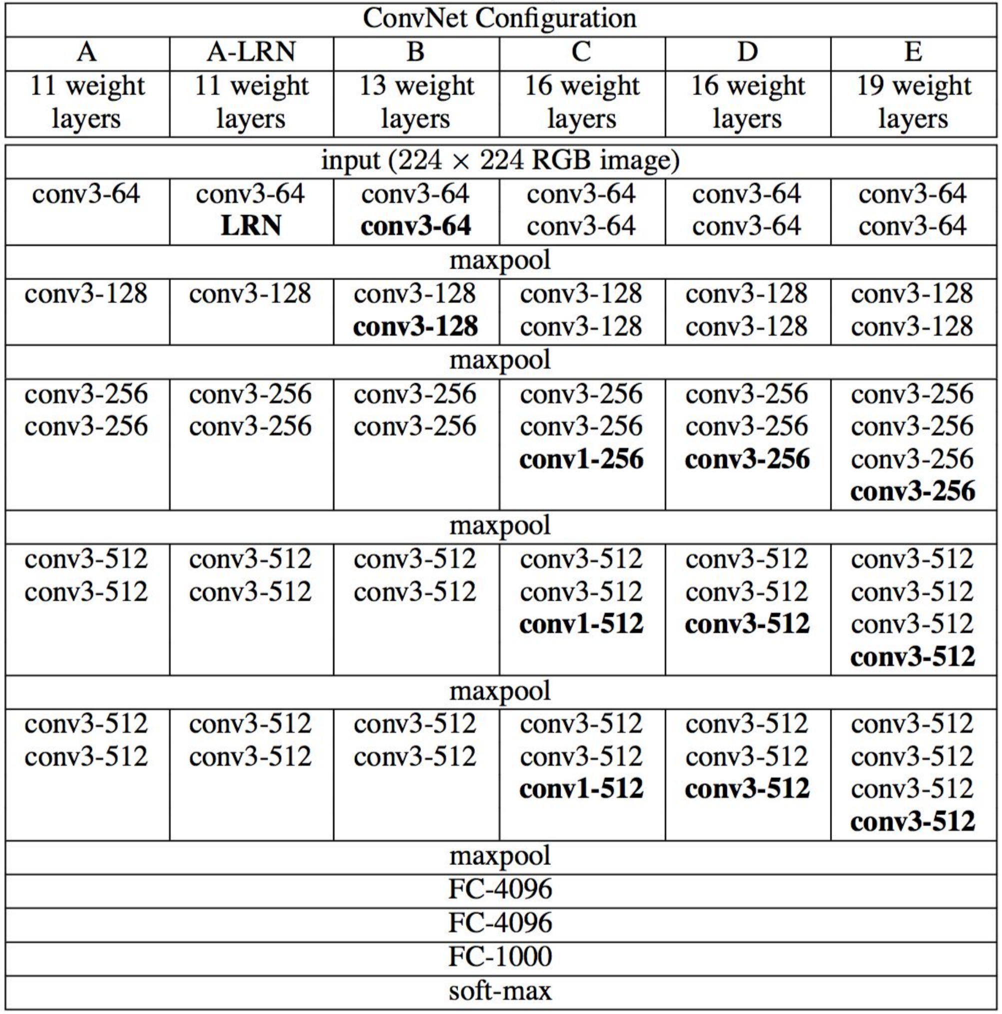
- ImageNet Large Scale Visual Recognition Challenge [http://image-net.org]
  - Summary of models on the ILSVRC2012 validation data
  - 1000 classes
    1.3M training
    50K validation
    100K testing imgs.
  - We will use
    vgg-verydeep-19
    pre-trained model
    [http://www.robots.ox.ac.uk/~vgg/research/very_deep/]
  - You can get
    pre-trained models
    in MatConvNet
    [http://www.vlfeat.org/matconvnet/pretrained/]



model	introduced	top-1 err.	top-5 err.	images/s
resnet-50-dag	2015	24.6	7.7	315.3
resnet-101-dag	2015	23.4	7.0	212.7
resnet-152-dag	2015	23.0	6.7	156.6
matconvnet-vgg-verydeep-16	2014	28.3	9.5	184.5
vgg-verydeep-19	2014	28.7	9.9	154.5
vgg-verydeep-16	2014	28.5	9.9	183.1
googlenet-dag	2014	34.2	12.9	501.8
matconvnet-vgg-s	2013	37.0	15.8	415.9
matconvnet-vgg-m	2013	36.9	15.5	623.1
matconvnet-vgg-f	2013	41.4	19.1	793.1
vgg-s	2013	36.7	15.3	395.4
vgg-m	2013	37.3	15.9	586.9
vgg-f	2013	41.1	18.8	785.7
vgg-m-128	2013	40.8	18.4	588.7
vgg-m-1024	2013	37.8	16.1	596.8
vgg-m-2048	2013	37.1	15.8	589.4
matconvnet-alex	2012	41.8	19.2	760.3
caffe-ref	2012	42.4	19.6	384.8
caffe-alex	2012	42.6	19.6	382.4

VGG model

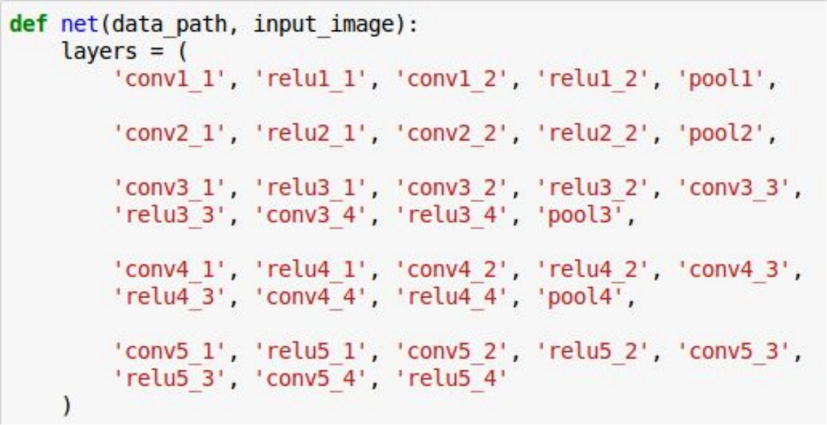
- Simonyan and Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition [http://arxiv.org/abs/1409.1556]
  - Tested very deep CNN (from 11 to 19)
  - with small 3x3 filters
  - # of parameters of 19 weight layers: 144 millions
  - Winner of ImageNet 2014
  - 25.5% top-1 val. error 8% top-5 val. error



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					

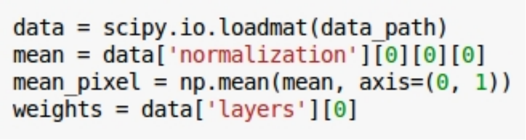
VGG model

- How to load VGG pretrained network
  - Download `imagenet-vgg-verydeep-19.mat`
    - It is located in /home/local/labvision
  - Definition of VGG-19 in `vgg.ipynb`

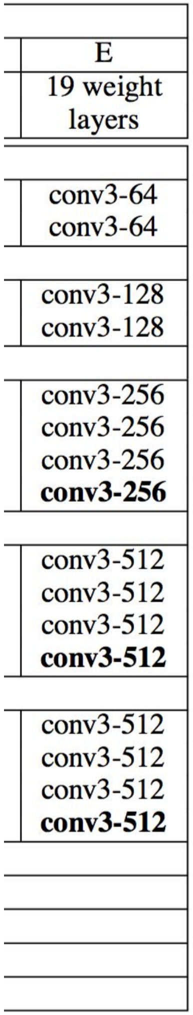


```
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )
```

  - Load mean_pixel and trained weights of each layer



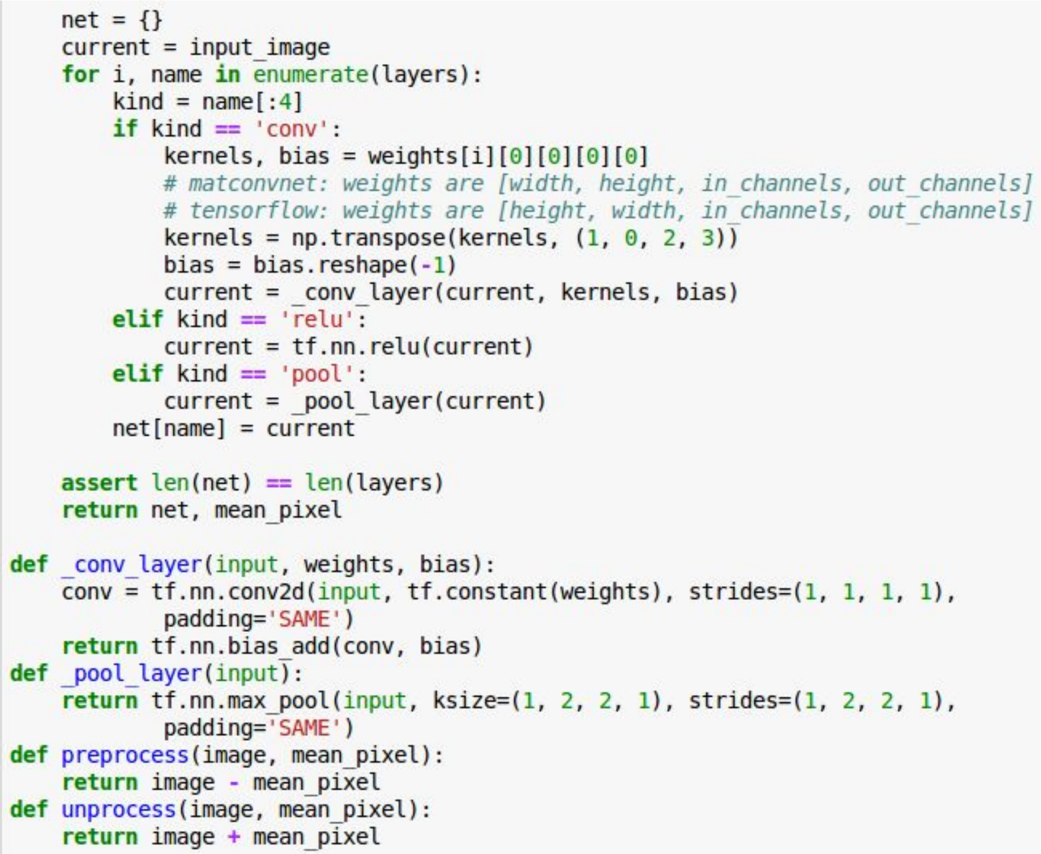
```
data = scipy.io.loadmat(data_path)
mean = data['normalization'][0][0][0]
mean_pixel = np.mean(mean, axis=(0, 1))
weights = data['layers'][0]
```



E 19 weight layers
conv3-64 conv3-64
conv3-128 conv3-128
conv3-256 conv3-256 conv3-256 **conv3-256**
conv3-512 conv3-512 conv3-512 **conv3-512**
conv3-512 conv3-512 conv3-512 **conv3-512**

VGG model

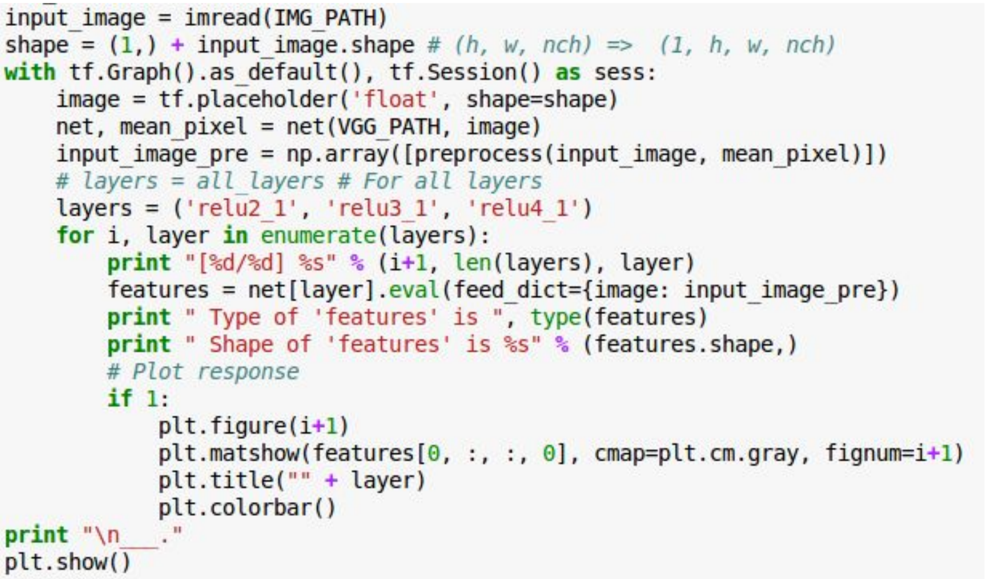
- Constructing VGG network with pretrained parameters



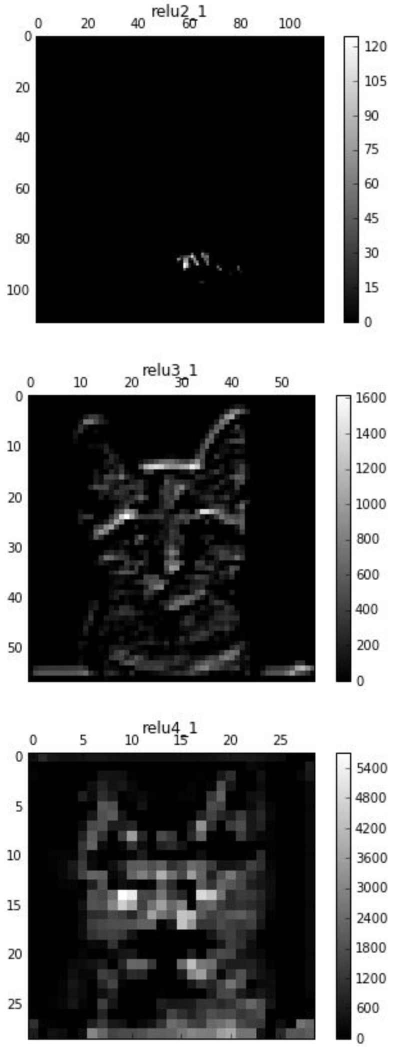
```
    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channels, out_channels]
            # tensorflow: weights are [height, width, in_channels, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current)
        net[name] = current

    assert len(net) == len(layers)
    return net, mean_pixel

def _conv_layer(input, weights, bias):
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1, 1, 1, 1),
            padding='SAME')
    return tf.nn.bias_add(conv, bias)
def _pool_layer(input):
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),
            padding='SAME')
def preprocess(image, mean_pixel):
    return image - mean_pixel
def unprocess(image, mean_pixel):
    return image + mean_pixel
```

VGG model

- Visualization of pre-trained features on cat image



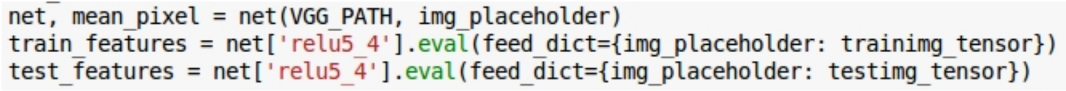
```
input_image = imread(IMG_PATH)
shape = (1,) + input_image.shape # (h, w, nch) =>  (1, h, w, nch)
with tf.Graph().as_default(), tf.Session() as sess:
    image = tf.placeholder('float', shape=shape)
    net, mean_pixel = net(VGG_PATH, image)
    input_image_pre = np.array([preprocess(input_image, mean_pixel)])
    # layers = all_layers # For all layers
    layers = ('relu2_1', 'relu3_1', 'relu4_1')
    for i, layer in enumerate(layers):
        print "[%d/%d] %s" % (i+1, len(layers), layer)
        features = net[layer].eval(feed_dict={image: input_image_pre})
        print " Type of 'features' is ", type(features)
        print " Shape of 'features' is %s" % (features.shape,)
        # Plot response
        if 1:
            plt.figure(i+1)
            plt.matshow(features[0, :, :, 0], cmap=plt.cm.gray, fignum=i+1)
            plt.title("" + layer)
            plt.colorbar()
print "\n___."
plt.show()
```



Assignment 5

- **Using the pre-trained VGG model, fine tuning VGG model for your custum dataset**
  - Load costum dataset and build training and test data tensor
    - Load cat/dog image data generated by `generate_dataset.ipynb`
    - Make a training_tensor and test_tensor each of which has (n, 112, 112, 3) dimension. `np.ndarray, np.reshape`

  - Load pre-trained VGG model as same way in `vgg.ipynb`
  - Define VGG features as the output of `relu5_4`



```
net, mean_pixel = net(VGG_PATH, img_placeholder)
train_features = net['relu5_4'].eval(feed_dict={img_placeholder: trainimg_tensor})
test_features = net['relu5_4'].eval(feed_dict={img_placeholder: testimg_tensor})
```

  - Construct your fully connected layers on top of the VGG features
    - You would need to vectorize VGG features to be input of the fc layer
    - Try to build multiple layers between 7x7x512 dimensional input to 2-dim output
  - Training fc layers using your training images. (Fine-tuning)
  - How much can you improve your test accuracy? It should outperform our CNN model, 0.833.