Deep Learning
Introduction to Deep Learning Models

Dr. – Ing. Gabriele Cavallaro
Postdoctoral Researcher
High Productivity Data Processing Group
Juelich Supercomputing Centre, Germany

Deep Learning in Remote Sensing: Applications

June 6th, 2018
Juelich Supercomputing Centre, Germany, Germany
Outline of the Course

1. Introduction to Deep Learning
2. Fundamentals of Convolutional Neural Networks (CNNs)
5. Model Selection and Regularization
6. Fundamentals of Long Short-Term Memory (LSTM)
7. LSTM Applications and Challenges
8. Deep Reinforcement Learning

Lecture 4 - Deep Learning in Remote Sensing: Applications
Outline

- DL Architectures for RS Applications
  - Semantic Segmentation
  - Image Classification with CNNs
  - Fully Convolutional Networks (FCNs)
  - Vanishing Gradient Problem
  - Residual Networks (ResNet)
  - Vaihingen Classification Results
Progression from Coarse to Fine Inference

- **Classification**: make a prediction for a whole input
  - What are the classes and ranked list

- **Localization or detection**: towards fine-grained inference
  - Classification and spatial location (e.g., bounding boxes)

- **Semantic segmentation**: fine-grained inference
  - Make dense predictions inferring labels for every pixel

- **Further improvements**: Provide different instances of the same class
  - Decomposition of already segmented classes into their components

- Many applications nourish from inferring knowledge from imagery
  - Autonomous driving
  - Human-machine interaction
  - Computational photography
  - Image search engines
  - Remote sensing
Image Classification CNNs

- **Convolutional layers**: convolution operation to the input
  - Emulate the response of an individual neuron to visual stimuli
  - Each convolutional neuron processes data only for its receptive field

- **Polling layers**: progressively reduce the spatial size of the representation
  - Reduce the amount of parameters and computation and control overfitting

- **Fully connected layers** connect every neuron in one layer to every neuron in another layer
  - Same principle as the traditional multi-layer perceptron (MLP) network

Semantic Segmentation Approach: Sliding Window

- Break the images into many small crops and classify the central pixel
- Redundant and computational expensive
  - Store not only every pixel but also the surrounding pixels
  - Increases the data size by a factor determined by the number of neighbouring pixels
- Very inefficient and not reusing shared features between overlapping patches

More visible with Hyperspectral images
Semantic Segmentation Approach: Fully Convolutional

- Semantic segmentation tasks have input images with different sizes
- Fully convolutional networks can take input of any arbitrary size
- Produce correspondingly-sized dense output with efficient inference and learning

"Transforming a classification-purposed CNN to produce spatial heatmaps by replacing fully connected layers with convolutional ones"

Serie of convolution and pooling layers

Fully connected layers: Inputs and feature maps of fixed size

Convolution can have inputs of any size

Semantic Segmentation Approach: Fully Convolutional

- Network as a bunch of convolutional layers to make predictions for pixels all at once
- Each layer preserve the size of the input
- The final convolutional layer output a tensor (C: number of classes)

- Problem: convolutions at original image resolution will be very expensive
  - E.g., Hyperspectral input images with D>100 bands
Semantic Segmentation Approach: Fully Convolutional

- Design networks as a bunch of convolutional layers
- With **downsampling** and **upsampling** inside the network

Rather than transitioning to a fully connected layer Increase the spatial resolution
- Computationally very efficient
- Networks can be deep and work on lower spatial resolution in many of the layers

**Input**
\[ D \times H \times W \]

**High-res:**
\[ D_1 \times H/2 \times W \]

**Low-res:**
\[ D_1 \times H/4 \times W/4 \]

**Med-res:**
\[ D_2 \times H/4 \times W/4 \]

**Med-res:**
\[ D_2 \times H/4 \times W/4 \]

**High-res:**
\[ D_1 \times H/2 \times W \]

**Predictions**
\[ H \times W \]
Semantic Segmentation Approach: Fully Convolutional

- The predictions of FCNs can be too coarse because of the upsampling steps.
- Low level of details in the upsampled output.
- FCN can be improved by making direct use of shallower and more local features.

Combining Feature Hierarchies: Skip Connections

- Combine layers of the feature hierarchy for refining the spatial precision of the output
- Fuse features across layers to define a nonlinear local-to-global representation

Incorporate lower level features in the inference

- Merge layers to fuse features

The network can adjust the coarse prediction to edges and specific part of the image.

End to end, joint learning of semantics and location
Residual Networks (ResNet)
Vanishing Gradient Problem

“No matter how deep a network is, it should not be any worse than the shallower network”


- With more parameters to learn, the train data should be fit at least as well as before

- **Vanishing gradient** problem: difficulty in learning the parameters of the earlier layers
  - Networks with gradient based methods (e.g., Backpropagation).

- Possible Solutions:
  - Multi-level hierarchy
  - Long short-term memory
  - Faster hardware
  - Residual networks
  - Other activation functions (e.g., ReLU)

Degradation Problem

- Vanishing gradient becomes worse as the number of layers increases
- Accuracy gets saturated and then degrades rapidly
- Unexpectedly, such degradation is not caused by overfitting
  - adding more layers to a suitably deep model leads to higher training error
- Example with 2 “plain” networks with different depths on CIFAR-10 dataset

The deeper network has higher training error and this test error

Residual Networks (ResNet)

- Architecture which solves vanishing gradient problem in a simple way
- Residual networks can be much deeper than their ‘plain’ counterparts,
  - Yet they require a similar number of parameters (weights)

“If there is trouble sending the gradient signal backwards, why not provide the network with a shortcut at each layer to make things happen more smoothly”?


The gradient could “skip” all the layers and reach the bottom without being diminished
The ResNet50 FCN Model
ResNet50 FCN

- `/semseg/resnet50-fcn/resnet50_edit.py`

```
x = ZeroPadding2D((3, 3))(img_input)
x = Conv2D(64, (7, 7), strides=(2, 2), name='conv1')(x)
x = BatchNormalization(axis='bn_axis', name='bn_conv1')(x)
x = Activation('relu')(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv_block(x, 3, [64, 64, 256], stage=2, block='a')
x = identity_block(x, 3, [64, 64, 256], stage=2, block='b')

x = conv_block(x, 3, [128, 128, 512], stage=3, block='a')
x = identity_block(x, 3, [128, 128, 512], stage=3, block='b')

x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a')
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='b')

x = conv_block(x, 3, [512, 512, 2048], stage=5, block='a')
x = identity_block(x, 3, [512, 512, 2048], stage=5, block='b')

x = AveragePooling2D((7, 7), name='avg_pool')(x)
```

Diagram of ResNet50 FCN with layers and operations.
The ResNet is adapted into an FCN
Residual Network 50 FCN

- Python file `~/semseg/resnet50-fcn/model_generator.py`

```python
# the function to generate a FCN version of the ResNet50 model:
def generate_resnet50_fcn(use_pretraining):
    num_labels = 6
    input_dim_row = 256
    input_dim_col = 256
    input_shape = (input_dim_row, input_dim_col, 3)
    input_tensor = Input(shape=input_shape)
    weights = 'imagenet' if use_pretraining else None
    standard_model = ResNet50(include_top=False, weights=weights, input_tensor=input_tensor)

    # get the activations after different network parts by name:
x32 = standard_model.get_layer('act3d').output
x16 = standard_model.get_layer('act4f').output
x8 = standard_model.get_layer('act5g').output

    # apply 1x1 convolution to compress the depth of the output tensors to the number of classes:
c32 = Convolution2D(filters=num_labels, kernel_size=(1, 1), name='conv_labels_32')(x32)
c16 = Convolution2D(filters=num_labels, kernel_size=(1, 1), name='conv_labels_16')(x16)
c8 = Convolution2D(filters=num_labels, kernel_size=(1, 1), name='conv_labels_8')(x8)

    # resize the spatial dimensions to fit the spatial input size:
r32 = Lambda(resize_bilinear, name='resize_labels_32')(c32)
r16 = Lambda(resize_bilinear, name='resize_labels_16')(c16)
r8 = Lambda(resize_bilinear, name='resize_labels_8')(c8)

    # sum up the activations of different stages to get information of different solution
m = Add(name='merge_labels')([r32, r16, r8])

    # apply a softmax activation function to get the probability of each class for each pixel
x = Reshape((input_dim_row * input_dim_col, num_labels))(m)
x = Activation('softmax')(x)
x = Reshape((input_dim_row, input_dim_col, num_labels))(x)

    # return the FCN version of the ResNet50 model:
return Model(inputs=input_tensor, outputs=x)
```
The ResNet50 FCN Training Function
Python Function for ResNet50 FCN with Augmentation

Location `~/semseg/resnet50-fcn/train_resnet50_fcn.py`

```python
def train_with_augmentation(data_path, output_model, transfer_learning_flag):
    num_labels = 6

    print(f'Load data ...
')
    x_train, y_train = dio.load_data(data_path + '/vaihingen_train.hdf5')
    print(f'Training samples: {x_train.shape}')
    x_val, y_val = dio.load_data(data_path + '/vaihingen_val.hdf5')
    print(f'Validation samples: {x_val.shape}')

    print(f'Generate augmented images ...
')
    x_train_aug, y_train_aug = augmentation.every_element_randomly_once(x_train, y_train)
    x_val_aug, y_val_aug = augmentation.every_element_randomly_once(x_val, y_val)

    print(f'{x_train.dtype}
print(f'{x_train_aug.dtype}

# put each array together with its augmented version:
print(f'{x_train = np.concatenate([x_train, x_train_aug])
y_train = np.concatenate([y_train, y_train_aug])
x_val = np.concatenate([x_val, x_val_aug])
y_val = np.concatenate([y_val, y_val_aug])

# shuffle the samples:
print(f'{x_train, y_train = augmentation.shuffle_4d_sample_wise(x_train, y_train)
x_val, y_val = augmentation.shuffle_4d_sample_wise(x_val, y_val)

print(f'Preprocess the input data (normalization, centering) ...
')
x_train = preprocess_input(x_train, mode="tf")
x_train = preprocess_input(x_train, mode="tf")

print(f'Load model ...
')
resnet50_fcn_model = model_generator.generate_resnet50_fcn(use_pretraining=transfer_learning_flag)
resnet50_fcn_model.summary()

print(f'Compile model ...
')
resnet50_fcn_model.compile(optimizer=keras.optimizers.Adam(),
loss=keras.losses.categorical_crossentropy,
metrics=["Accuracy"])
resnet50_fcn_model.fit(x_train, y_train,
batch_size=8, epochs=20, validation_data=(x_val, y_val))
resnet50_fcn_model.save_weights(output_model)
```

Load:
- `vaihingen_train.hdf5`
- `vaihingen_validation.hdf5`

Apply one random augmentation to each 256x256 patch
- Rotate(90,180,270) or
- Flip (up,down)

Concatenate original data with the augmented data

Preprocessing

Generate the model

Train the model
Python Function for ResNet50 FCN with Augmentation

- ~/semseg/resnet50-fcn/train_resnet50_fcn.py
- train_resnet50_fcn.py requires 4 parameters

```python
def main(arguments):
    data_path = arguments[1]
    output_model = arguments[2]
    augmentation_flag = arguments[3]
    transfer_learning_flag = arguments[4]

    if augmentation_flag=='True':
        train_with_augmentation(data_path, output_model, transfer_learning_flag)
    elif augmentation_flag=='False':
        train(data_path, output_model, transfer_learning_flag)
    else:
        sys.exit()

if __name__ == '__main__':
    if len(sys.argv)<4:
        print('*******************************************************************************
        print('Four parameters need to be specified:
        print('1. Location of waiblingen_train.hdf5 and waiblingen_val.hdf5 (e.g., /home/bnclab/train002/senseg/data/)
        print('2. Location + name of the output model (e.g., /home/bnclab/train002/senseg/models/resnet50_fcn_weights.hdf5)
        print('3. Augmentation: True or False!
        print('4. Transfer learning (load weights trained on ImageNet): True or False!
        print('*******************************************************************************
        sys.exit()

main(sys.argv)
```
Practical 4:
Check the Outcomes of the Job Previously Submitted
Batch Scripts Previously Submitted

#!/bin/bash -x
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --output=train_resnet50_fcn_out.%j
#SBATCH --error=train_resnet50_fcn_err.%j
#SBATCH --time=01:00:00
#SBATCH --mail-user=g.cavallaro@fz-juelich.de
#SBATCH --mail-type=ALL
#SBATCH --job-name=train_resnet50_fcn

#SBATCH --partition=gpus
#SBATCH --gres=gpu:1

#SBATCH --reservation=deep_learning

### location executable
RESNET50_FCN=/homea/hpclab/train002/semseg/resnet50-fcn/train_resnet50_fcn.py

module restore dl_tutorial

### submit
python $RESNET50_FCN /homea/hpclab/train002/semseg/data/
/homea/hpclab/train002/semseg/models/resnet50_fcn_weights.hdf5 True False

Lecture 4 - Deep Learning in Remote Sensing: Applications
ResNet50 FCN Trained Model (20 Epochs)

Check the output files: train_resnet50_fcn_out
train_resnet50_fcn_err

=================================================================
Total params: 23,609,234
Trainable params: 23,556,114
Non-trainable params: 53,120

compile model ...
Train on 4166 samples, validate on 736 samples
Epoch 1/20
4166/4166 [==============================] - 102s 24ms/step -
loss: 0.9315 - acc: 0.6992 - val_loss: 0.7566 - val_acc: 0.7103

Epoch 20/20
4166/4166 [==============================] - 67s 16ms/step -
loss: 0.4269 - acc: 0.8338 - val_loss: 0.4783 - val_acc: 0.8239
Training time: 1448.95877409 seconds

Check the output files: train_resnet50_fcn_out
train_resnet50_fcn_err

Lecture 4 - Deep Learning in Remote Sensing: Applications
Pre-Trained ResNet50 Trained Model (20 Epochs)

Total params: 23,609,234
Trainable params: 23,556,114
Non-trainable params: 53,120

Compile model ...
Train on 4166 samples, validate on 736 samples
Epoch 1/20
4166/4166 [==============================] - 92s 22ms/step -
loss: 0.6261 - acc: 0.7748 - val_loss: 1.2206 - val_acc: 0.7112

Epoch 20/20
4166/4166 [==============================] - 67s 16ms/step -
loss: 0.1395 - acc: 0.9450 - val_loss: 0.5231 - val_acc: 0.8440
Training time: 1439.80376601 seconds

Lecture 4 - Deep Learning in Remote Sensing: Applications
Practical 6:
Test the Models
Test Set

Vaihingen_15

Vaihingen_23

Thematic classes:

- **Impervious surfaces**
- **Building**
- **Low vegetation**
- **Tree**
- **Car**
- **Clutter/background**
Test ResNet50 FCN

- Use the function ~/semseg/resnet50-fcn/evaluate_network.py

- Run the test on the login node (i.e., no batch script submission)

- Setup the Python environment: $ module restore dl_tutorial

- `evaluate_network.py` requires 2 parameters

```bash
***************************************************************
Two parameters need to be specified:
1. The number of the patch to test (e.g., 15, 23)
2. The network model (e.g., /homea/hpclab/train002/semseg/models/resnet50_fcn_weights.hdf5)
***************************************************************
```

- Run the test on the Vaihingen 15:
  $ python evaluate_network.py 15 ~/semseg/models/resnet50_fcn_weights.hdf5

Or

- Run the test on the Vaihingen 23:
  $ python evaluate_network.py 23 ~/semseg/models/resnet50_fcn_weights.hdf5
Experiment 1: Test ResNet50 FCN

Groundtruth | Classification map | Classification errors
---|---|---
Vaihingen_15 | | OA: 80.68%
Vaihingen_23 | | OA: 78.78%

Legend:
- Impervious surfaces
- Building
- Low vegetation
- Tree
- Car
- Clutter/background

Lecture 4 - Deep Learning in Remote Sensing: Applications
## Experiment 1: Test ResNet50 FCN

### Confusion matrix

<table>
<thead>
<tr>
<th>ACTUAL</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>678470</td>
<td>27787</td>
</tr>
<tr>
<td>77693</td>
<td>911674</td>
</tr>
<tr>
<td>50176</td>
<td>35116</td>
</tr>
<tr>
<td>15034</td>
<td>4887</td>
</tr>
<tr>
<td>18528</td>
<td>13</td>
</tr>
<tr>
<td>226</td>
<td>581</td>
</tr>
</tbody>
</table>

### Vaihingen_15

<table>
<thead>
<tr>
<th>Class</th>
<th>Pixels</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impervious surfaces</td>
<td>855112</td>
<td>79.3%</td>
</tr>
<tr>
<td>Building</td>
<td>1075170</td>
<td>84.8%</td>
</tr>
<tr>
<td>Low vegetation</td>
<td>1309897</td>
<td>80.1%</td>
</tr>
<tr>
<td>Tree</td>
<td>1643189</td>
<td>80.3%</td>
</tr>
<tr>
<td>Car</td>
<td>31684</td>
<td>39.9%</td>
</tr>
<tr>
<td>Clutter/background</td>
<td>7183</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

### Vaihingen_23

<table>
<thead>
<tr>
<th>Class</th>
<th>Pixels</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impervious surfaces</td>
<td>801652</td>
<td>72.3%</td>
</tr>
<tr>
<td>Building</td>
<td>885284</td>
<td>82.8%</td>
</tr>
<tr>
<td>Low vegetation</td>
<td>1345728</td>
<td>79.3%</td>
</tr>
<tr>
<td>Tree</td>
<td>1789171</td>
<td>79.9%</td>
</tr>
<tr>
<td>Car</td>
<td>15447</td>
<td>42.3%</td>
</tr>
<tr>
<td>Clutter/background</td>
<td>7756</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Lecture 4 - Deep Learning in Remote Sensing: Applications
Experiment 2: Test Pre-Trained ResNet50 with ImageNet

Vaihingen_15

Groundtruth
Classification map
Classification errors

OA: 82.15%

Vaihingen_23

Groundtruth
Classification map
Classification errors

OA: 82.04%

Legend:
- Impervious surfaces
- Building
- Low vegetation
- Tree
- Car
- Clutter/background
Experiment 2: Test Pre-Trained ResNet50 with ImageNet

### Confusion Matrix

<table>
<thead>
<tr>
<th>Class</th>
<th>Pixels</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impervious surfaces</td>
<td>855112</td>
<td>75.6%</td>
</tr>
<tr>
<td>Building</td>
<td>1075170</td>
<td>93.6%</td>
</tr>
<tr>
<td>Low vegetation</td>
<td>1309897</td>
<td>72.2%</td>
</tr>
<tr>
<td>Tree</td>
<td>1643189</td>
<td>86.4%</td>
</tr>
<tr>
<td>Car</td>
<td>31684</td>
<td>81.9%</td>
</tr>
<tr>
<td>Clutter/background</td>
<td>7183</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

### Confusion Matrix

<table>
<thead>
<tr>
<th>Class</th>
<th>Pixels</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impervious surfaces</td>
<td>801652</td>
<td>76.2%</td>
</tr>
<tr>
<td>Building</td>
<td>885284</td>
<td>94.6%</td>
</tr>
<tr>
<td>Low vegetation</td>
<td>1345728</td>
<td>66.6%</td>
</tr>
<tr>
<td>Tree</td>
<td>1789171</td>
<td>90.4%</td>
</tr>
<tr>
<td>Car</td>
<td>15447</td>
<td>84.9%</td>
</tr>
<tr>
<td>Clutter/background</td>
<td>7756</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Lecture 4 - Deep Learning in Remote Sensing: Applications
## Results Summary

<table>
<thead>
<tr>
<th>Location</th>
<th>Model</th>
<th>OA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vaihingen 15</td>
<td>ResNet50 FCN</td>
<td>80.68%</td>
</tr>
<tr>
<td></td>
<td>Pre-trained ResNet50</td>
<td>82.15%</td>
</tr>
<tr>
<td>Vaihingen 23</td>
<td>ResNet50 FCN</td>
<td>78.78%</td>
</tr>
<tr>
<td></td>
<td>Pre-trained ResNet50</td>
<td>82.04%</td>
</tr>
</tbody>
</table>
Tutorial Competition

Lecture 4 - Deep Learning in Remote Sensing: Applications
## Our Competition

### Vaihingen_15

<table>
<thead>
<tr>
<th>WG</th>
<th>ACCURACY</th>
<th>TRAINING TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WG</th>
<th>ACCURACY</th>
<th>TRAINING TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimizers

SGD  10 1 0.1 0.001 0.0001

RMSprop

Adagrad

Adadelta

Adam

Adamax
What can you change?

- `/homea/hpclab/train001/....../data_io.py`

```
def generate_dataset(image_numbers, overlap_factor):
    # input / output size of the FCN:
    size = 256

    # generate and save the training and validation set:
    overlap = 0.6
```

LIST TO BE DONE
- Number of epochs
- More augmented data

........

Multiple GPUS
Lecture Bibliography (1)


Lecture Bibliography (2)

  Online: [http://gsp.humboldt.edu/olm_2015/Courses/GSP_216_Online/lesson4-1/radiometric.html](http://gsp.humboldt.edu/olm_2015/Courses/GSP_216_Online/lesson4-1/radiometric.html)


- [17] Normalized Difference Vegetation Index (NDVI)
  Online: [http://www.agasyst.com/portals/NDVI.html](http://www.agasyst.com/portals/NDVI.html)

- [18] NDVI & Classification

   Online: https://link.springer.com/chapter/10.1007/978-3-642-14212-3_12

[23] Rome Image dataset
   Online: https://b2share.eudat.eu/records/daf6c389e54340b4b1416cf874251e77


   Online: https://www.jpl.nasa.gov/missions/airborne-visible-infrared-imaging-spectrometer-aviris/

[26] Indian Pines dataset: 220 Band AVIRIS Hyperspectral Image
   Online: https://purr.purdue.edu/publications/1947/1


[28] Indian Pines Raw and Processed
   Online: http://hdl.handle.net/11304/9ec5eac8-61b4-4617-ae1c-1f8c8cd3cd74

   Online: https://www.youtube.com/watch?v=nU-CjAKry5c