Applied Parallel Computing

parallel-computing.pro

# Object Detection

# Whale recognition

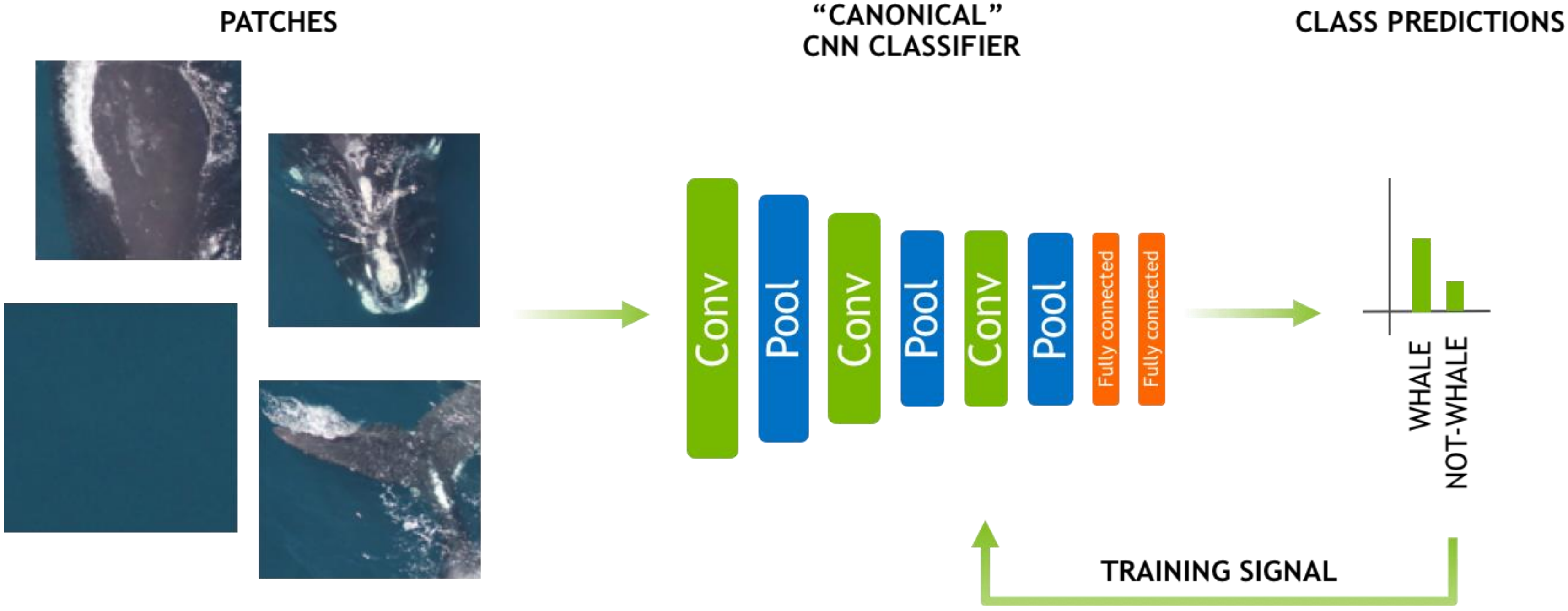
- NOAA Right Whale Recognition
  - (https://www.kaggle.com/c/noaa-right-whale-recognition)
  - Contestants were asked to identify the specific whale present in aerial images of the ocean.
  - We are going to train a convolutional neural network (CNN) to localize the whale within the image.
  - Many successful competitors in the original competition found it improved their scores to first detect and localize the whales in the image before trying to identify them using a cropped and normalized image.

# Whale recognition

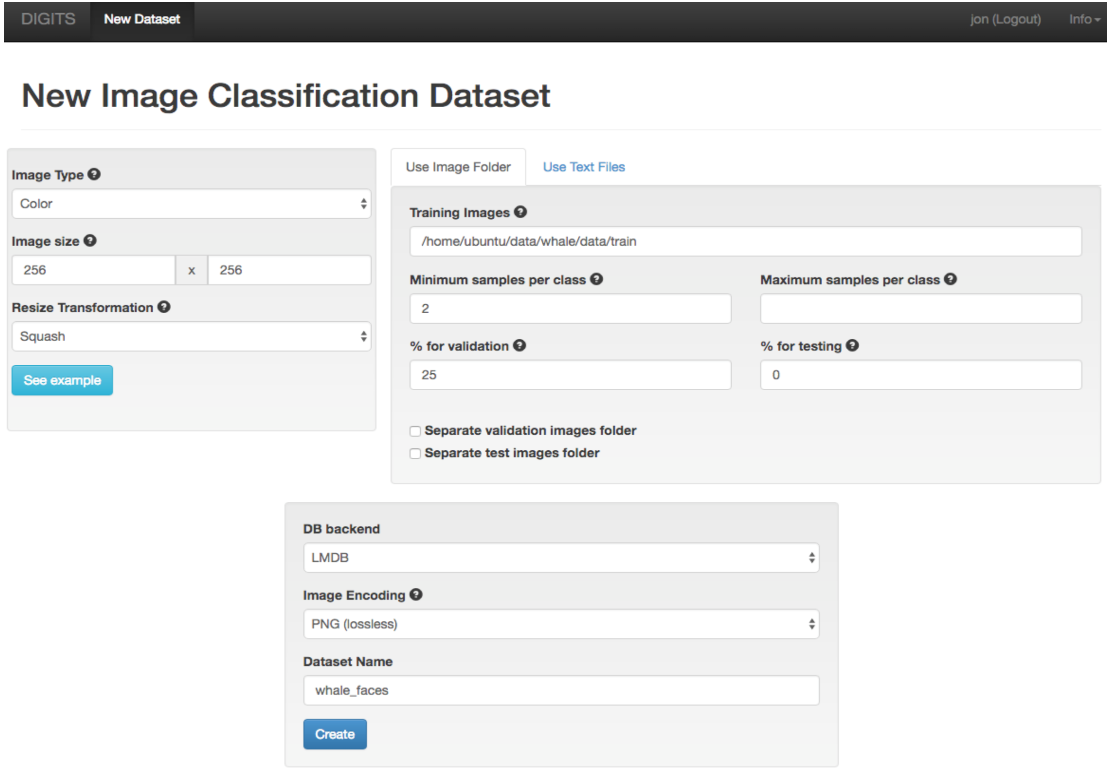
## Object detection approaches:

- Sliding window
  - ✓ The simplest approach is to first train a CNN classifier on image patches that can differentiate the object from non-object examples.
  - ✓ We can inspect each patch in a larger image, and make a determination whether there is a whale present.
- Candidate generation and classification
- Fully-convolutional network (FCN)
- DetectNet

# Sliding window



# Sliding window



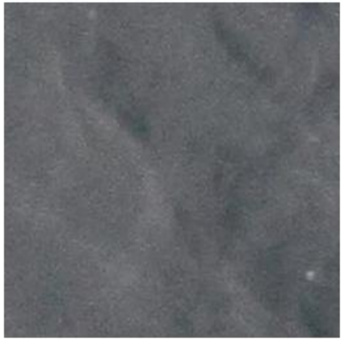
# Sliding window


not face


face


face


face

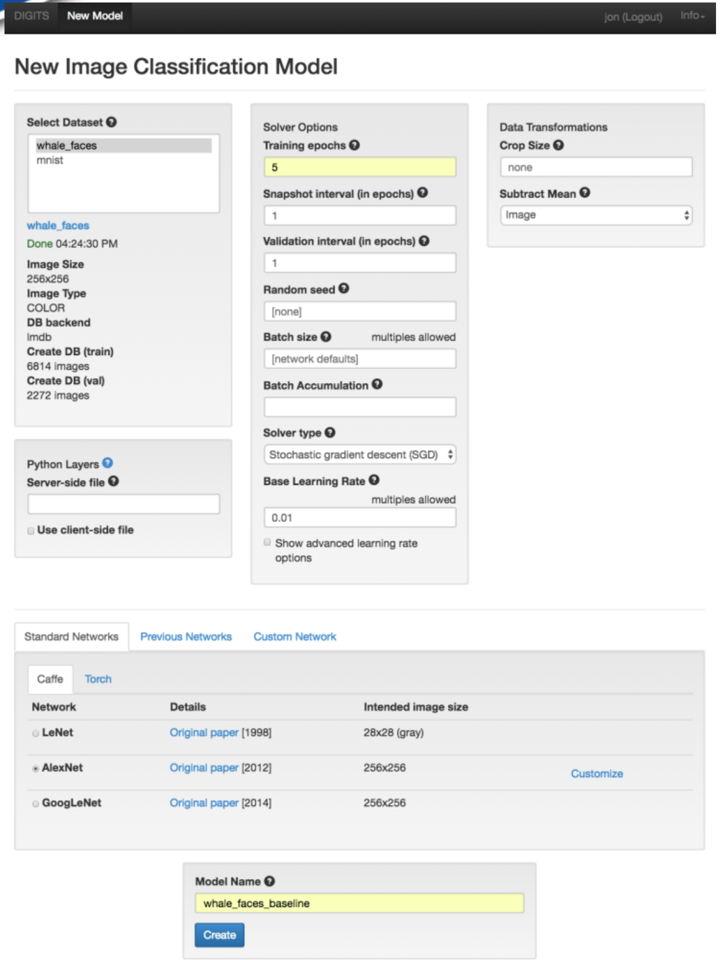

not face


face


not face


face

# Sliding window



# Sliding window

```
import numpy as np
import matplotlib.pyplot as plt
import caffe
import time

MODEL_JOB_NUM = '20160920-092148-8c17'  ## Remember to set this to be the job number for your model
DATASET_JOB_NUM = '20160920-090913-a43d'  ## Remember to set this to be the job number for your dataset

MODEL_FILE = '/home/ubuntu/digits/digits/jobs/' + MODEL_JOB_NUM + '/deploy.prototxt'
PRETRAINED = '/home/ubuntu/digits/digits/jobs/' + MODEL_JOB_NUM + '/snapshot_iter_270.caffemodel‘
MEAN_IMAGE = '/home/ubuntu/digits/digits/jobs/' + DATASET_JOB_NUM + '/mean.jpg'

# load the mean image
mean_image = caffe.io.load_image(MEAN_IMAGE)

# Choose a random image to test against
RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = 'data/samples/w_' + RANDOM_IMAGE + '.jpg'
```

# Sliding window

```
# Tell Caffe to use the GPU
caffe.set_mode_gpu()
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       channel_swap=(2,1,0),
                       raw_scale=255,
                       image_dims=(256, 256))

# Load the input image into a numpy array and display it
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()

# Calculate how many 256x256 grid squares are in the image
rows = input_image.shape[0]/256
cols = input_image.shape[1]/256

# Initialize an empty array for the detections
detections = np.zeros((rows,cols))
```

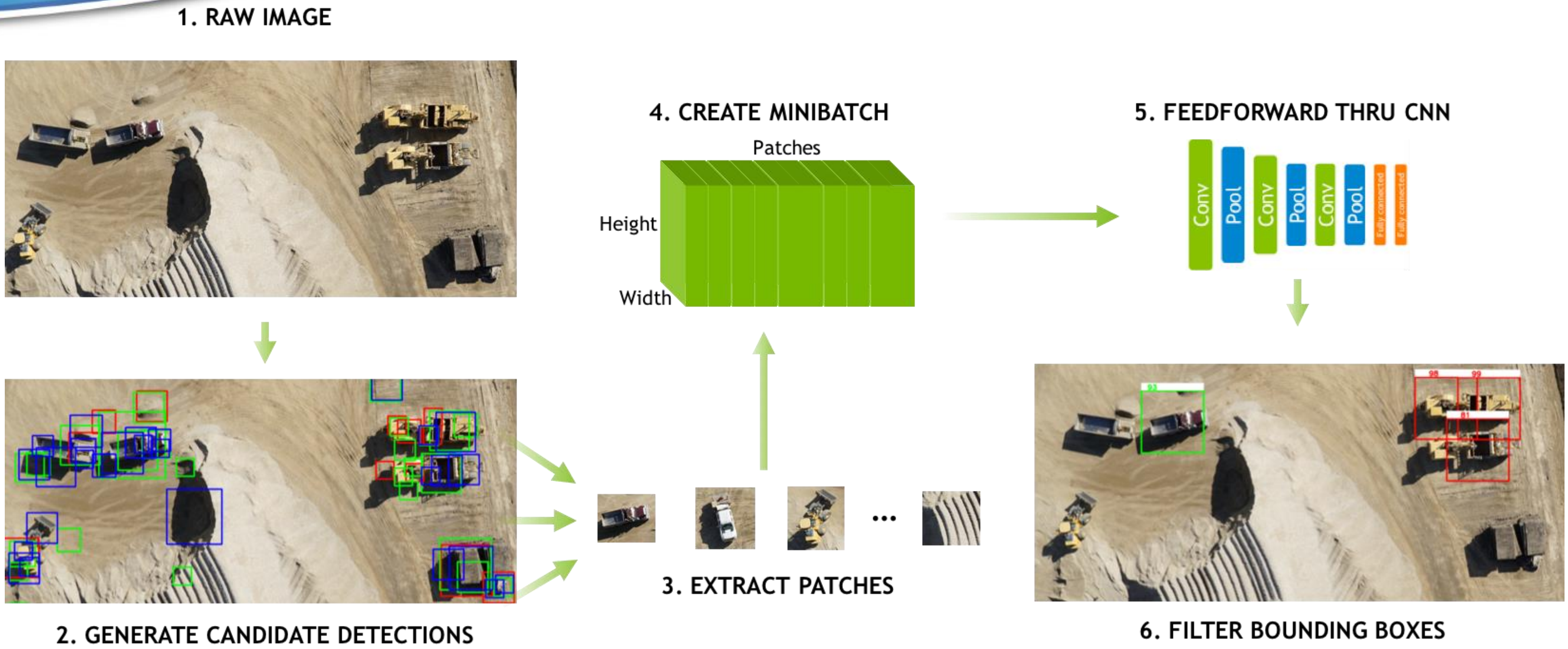
# Sliding window

```
# Iterate over each grid square using the model to make a class prediction
start = time.time()
for i in range(0,rows):
    for j in range(0,cols):
        grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256]
        # subtract the mean image
        grid_square -= mean_image
        # make prediction
        prediction = net.predict([grid_square])
        detections[i,j] = prediction[0].argmax()
end = time.time()

# Display the predicted class for each grid square
plt.imshow(detections)

# Display total time to perform inference
print 'Total inference time: ' + str(end-start) + ' seconds'
```

# Sliding window

- Advantages:
  - We can train a detector using only patch based training data (which is more widely available).
- Disadvantages:
  - Slow to make predictions, especially if there is large overlap between grid squares which leads to a great deal of redundant computation
  - challenging to produce a balanced training dataset that is robust to false alarm causing clutter
  - difficult to achieve scale invariance for object detection

# Candidates generation



# Candidates generation

- Use some computationally cheaper, sensitive, but false alarm prone algorithm to generate candidate detections.
  - Cascade classifiers
  - Selective search.
- Advantages:
  - The speedup due to a smaller number of candidate detections to test
  - Depending on the candidate generation algorithm we may get more accurate localization of the object
- Disadvantages
  - A more complex multi-stage processing pipeline
  - An additional model to build or train for candidate generation
  - A non-trivial false alarm rate
  - Variable inference time dependent on the number of candidates generated

# FCN

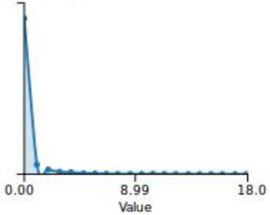
- The commonly used fully-connected layers can be replaced with convolutional layers.
  - Convolutional filters are the same size as the feature map outputs for the previous layer
  - Number of filters is equal to the number of neurons in the fully-connected layer it replaces.
  - Images of varying size can be input in to the network for classification.
    - ✓ If the input image is smaller than the expected image size for the network (called the receptive field of the network) then we will still just obtain a single classification for the image.
    - ✓ However, if the image is larger than the receptive field then we will obtain a heatmap of classifications, much like we obtained from the sliding window approach.

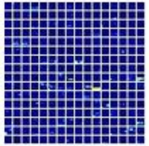
# FCN

pool5

Activation

**Data shape:** [256 6 6]
**Mean:** 0.365569
**Std deviation:** 1.15494



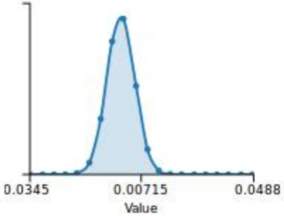
fc6

Weights

*(InnerProduct layer)*

37,752,832 learned parameters

**Data shape:** [4096 9216]
**Mean:** -0.000265418
**Std deviation:** 0.00484409

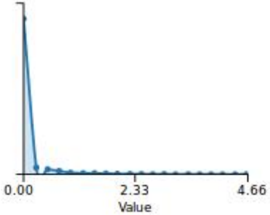


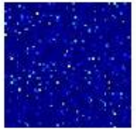

*Not shown*

fc6

Activation

**Data shape:** [4096]
**Mean:** 0.109813
**Std deviation:** 0.356604



# FCN

- Fc6 receives its input from pool5.
  - The shape of the activations at pool5 is 256*6*6.
  - The shape of the activations at fc6 is 4096
    - ✓ fc6 has 4096 output neurons.
    - ✓ To turn fc6 into an equivalent convolutional layer, create a convolutional layer with 6*6 kernel size and 4096 output feature maps.

# FCN

```
import numpy as np
import matplotlib.pyplot as plt
import caffe
import copy
from scipy.misc import imresize
import time

JOB_NUM = '20160920-110807-298d'  ## Remember to set this to be the job number for your model

MODEL_FILE = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/deploy.prototxt'
PRETRAINED = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/snapshot_iter_270.caffemodel‘

# Choose a random image to test against
RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = 'data/samples/w_' + RANDOM_IMAGE + '.jpg'

# Tell Caffe to use the GPU
caffe.set_mode_gpu()
```

# FCN

```
# Load the input image into a numpy array and display it
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()

# Initialize the Caffe model using the model trained in DIGITS
# This time the model input size is reshaped based on the randomly selected input image
net = caffe.Net(MODEL_FILE,PRETRAINED,caffe.TEST)
net.blobs['data'].reshape(1, 3, input_image.shape[0], input_image.shape[1])
net.reshape()
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_transpose('data', (2,0,1))
transformer.set_channel_swap('data', (2,1,0))
transformer.set_raw_scale('data', 255.0)

# This is just a colormap for displaying the results
my_cmap = copy.copy(plt.cm.get_cmap('jet')) # get a copy of the jet color map
my_cmap.set_bad(alpha=0) # set how the colormap handles 'bad' values
```

# FCN

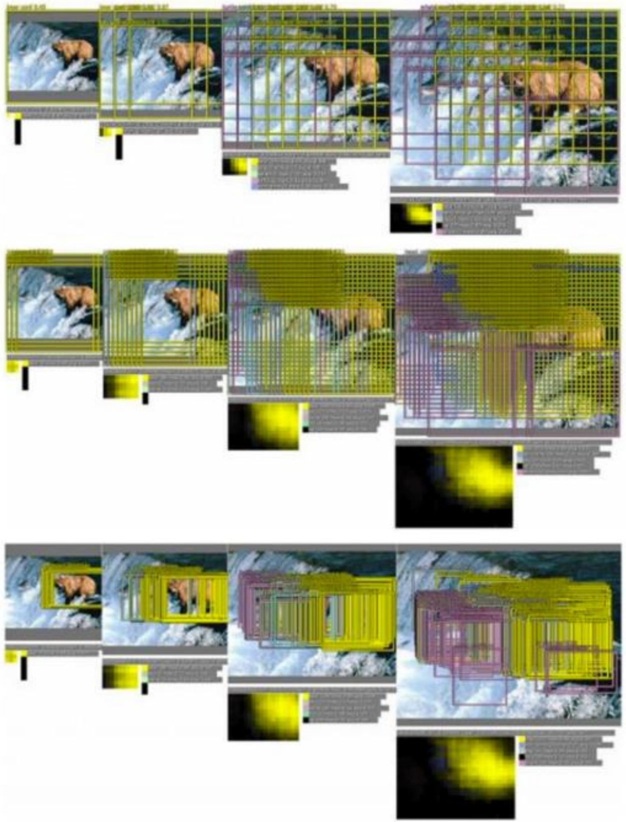
```
# Feed the whole input image into the model for classification
start = time.time()
out = net.forward(data=np.asarray([transformer.preprocess('data', input_image)]))
end = time.time()

# Create an overlay visualization of the classification result
im = transformer.deprocess('data', net.blobs['data'].data[0])
classifications = out['softmax'][0]
classifications =
imresize(classifications.argmax(axis=0),input_image.shape,interp='bilinear').astype('flo
at')
classifications[classifications==0] = np.nan
plt.imshow(im)
plt.imshow(classifications,alpha=.5,cmap=my_cmap)
plt.show()

# Display total time to perform inference
print 'Total inference time: ' + str(end-start) + ' seconds'
```

# FCN

- In many cases the FCN is able to locate the whale's face with greater precision than the sliding window approach.
  - It will still find a larger amount of the whale.
  - It is sometimes confused by breaking waves or sunlight reflecting from the ocean surface.
    - ✓ Caused by background clutter and the whale's body could be mitigated using appropriate data augmentation.
- Inference time for the FCN is about **1.5** seconds
  - For the sliding window approach it took **10** seconds.
- Ways to improve the classification accuracy and localization precision:
  - Pass the input image through the network multiple times at varying scales.
    - ✓ This improves the models tolerance to scale variation in the appearance of the object of interest.
  - Modify the network layer strides to provide finer or coarser grained classification heatmap outputs.
    - ✓ Multiple versions of the input image can improve the final classification and detection result drastically.
    - ✓ A well known example of this approach was presented in the paper OverFeat.

# OverFeat

***OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,***
*Sermanet et al., 2014*



greedy merging procedure

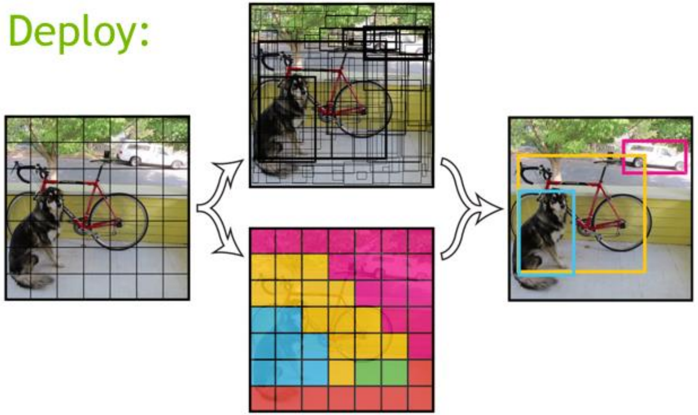
# DetectNet

## Train:


Pascal VOC 2012 images

For each grid square predict:
- Class confidence
- Bounding box relative to square

## Deploy:




**Citation:**

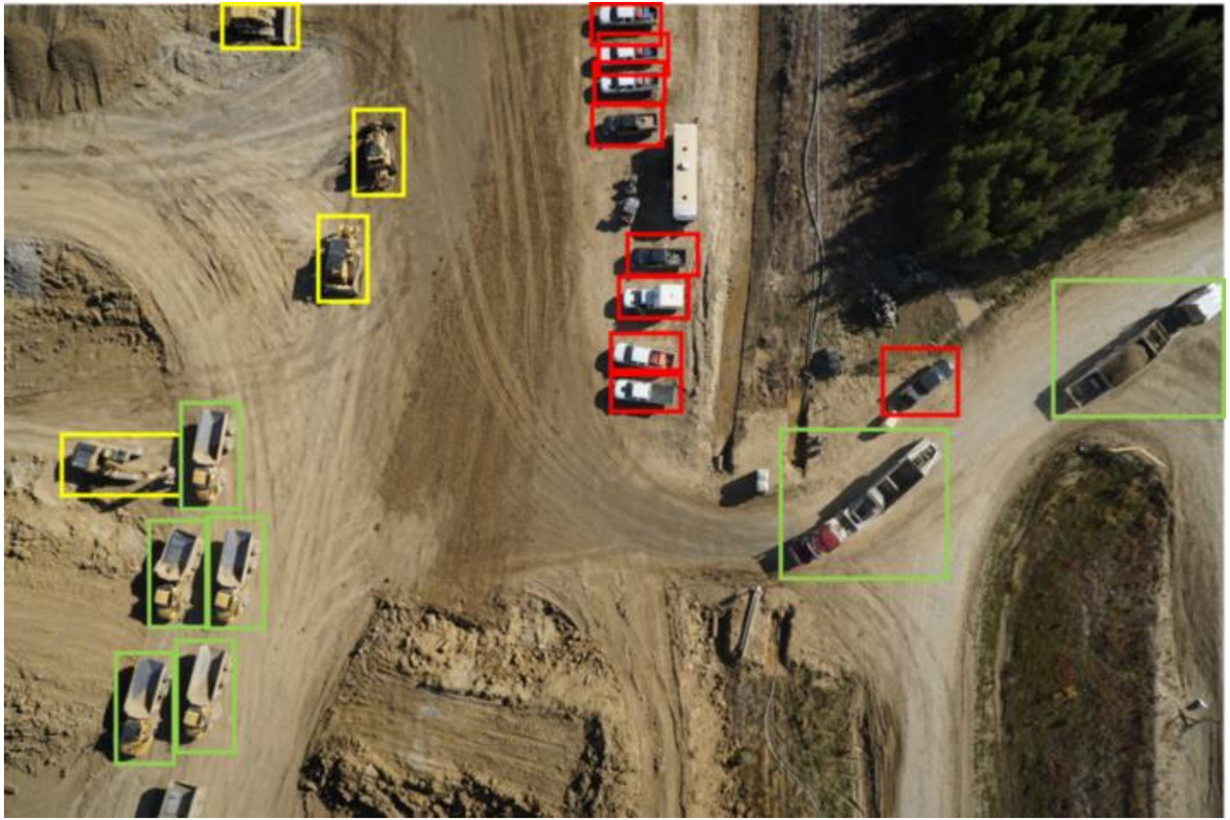
Redmon, Divvala, Girshick, Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, arXiv: 1506.02640

http://pjreddie.com/darknet/yolo/

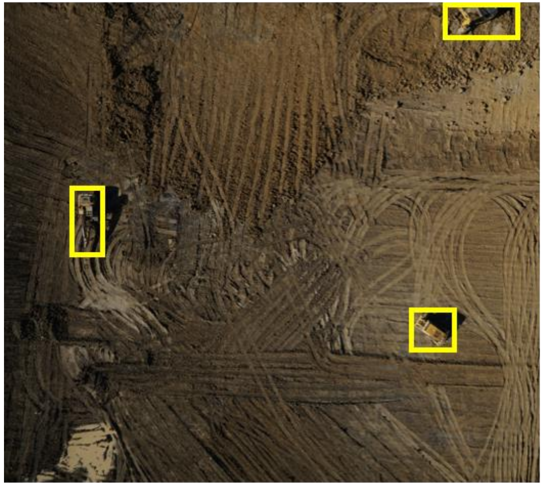
# DetectNet

- Advantages:
  - Simple one-shot detection, classification and bounding box regression pipeline.
  - Very low latency.
  - Very low false alarm rates due to strong, voluminous background training data.
- Disadvantages:
  - In order to train this type of network specialized training data is required where all objects of interest are labelled with accurate bounding boxes.
  - This type of training data is much rarer and costly to produce.

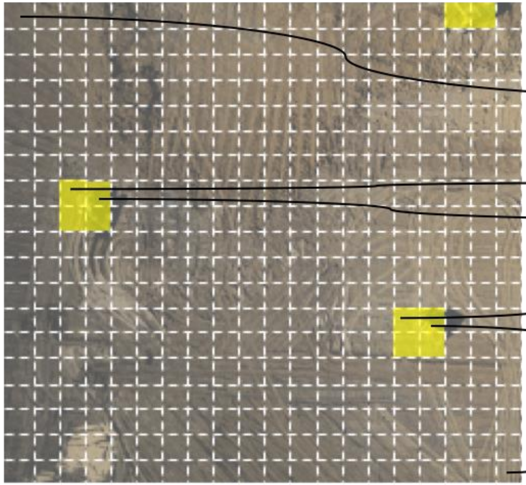
# DetectNet



# DetectNet



Training image with bounding box annotations



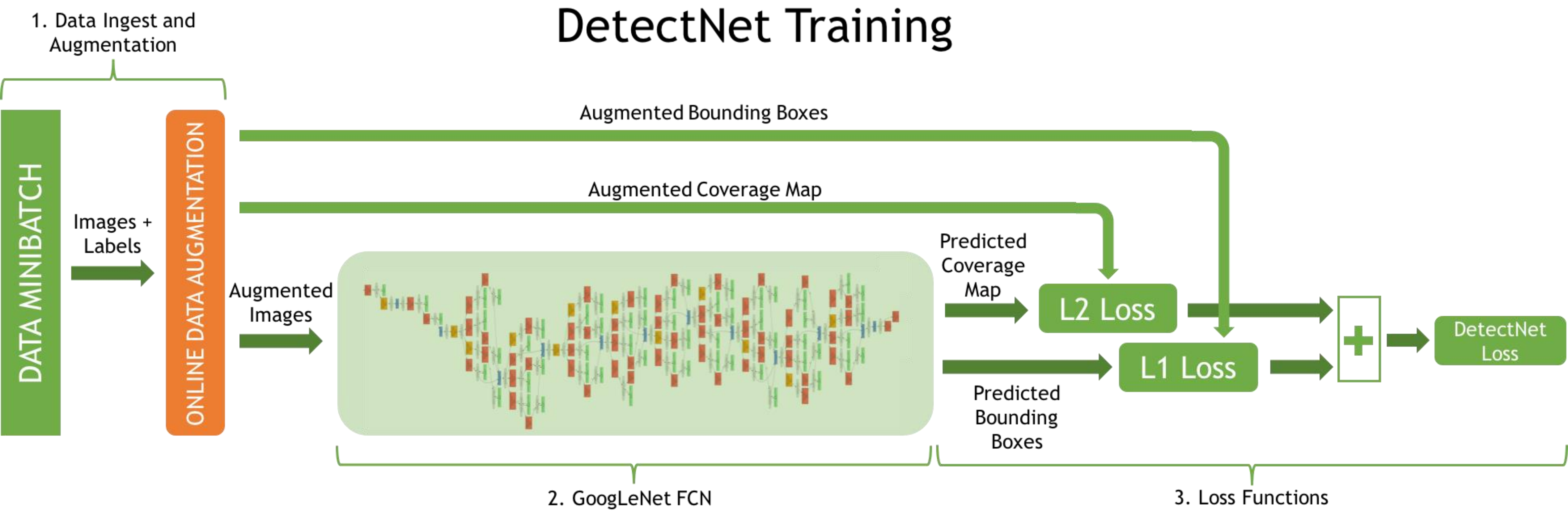
Bounding boxes mapped to grid squares

Bounding box coordinates in pixels relative to center of grid square

| class | $x_1$ | $y_1$ | $x_2$ | $y_2$ | coverage |
|---|---|---|---|---|---|
| dontcare | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... |
| digger | -2 | -8 | 18 | 24 | 1 |
| digger | -18 | -8 | 2 | 24 | 1 |
| ... | ... | ... | ... | ... | ... |
| digger | -6 | -8 | 22 | 24 | 1 |
| digger | -24 | -8 | 8 | 24 | 1 |
| ... | ... | ... | ... | ... | ... |
| dontcare | 0 | 0 | 0 | 0 | 0 |

DetectNet input data representation

# DetectNet


DetectNet Training
1. Data Ingest and Augmentation
DATA MINIBATCH
Images + Labels
ONLINE DATA AUGMENTATION
Augmented Bounding Boxes
Augmented Coverage Map
Augmented Images
Predicted Coverage Map
L2 Loss
L1 Loss
Predicted Bounding Boxes
+
DetectNet Loss
2. GoogLeNet FCN
3. Loss Functions

# DetectNet

## Object Detection Dataset Options

Images can be stored in any of the supported file formats ('.png','.jpg','.jpeg','.bmp','.ppm').

**Training image folder**

/home/ubuntu/data/whale/data_336x224/train/images

Label files are expected to have the .txt extension. For example if an image file is named foo.png the corresponding label file should be foo.txt.

**Training label folder**

/home/ubuntu/data/whale/data_336x224/train/labels

**Validation image folder**

/home/ubuntu/data/whale/data_336x224/val/images

**Validation label folder**

/home/ubuntu/data/whale/data_336x224/val/labels

**Pad image (Width x Height)**

width x height

**Resize image (Width x Height)**

width x height

**Channel conversion**

RGB

**Minimum box size (in pixels) for validation set**

25

**Custom classes**

**Feature Encoding**

PNG (lossless)

**Label Encoding**

None

**Encoder batch size**

32

**Number of encoder threads**

4

**DB backend**

LMDB

**Dataset Name**

whales_detectnet

Create

# DetectNet

## Source image

## Inference visualization

# DetectNet

- DetectNet is able to accurately detect most whale faces
  - Tightly drawn bounding box
  - Very low false alarm rate.
  - Inference is extremely fast with DetectNet. Average time taken to pass a single 336x224 pixel image forward through DetectNet is just 22ms.

# KITTI

- A huge dataset for cars detection on images
  - [Creative Commons Attribution-NonCommercial-ShareAlike 3.0]()