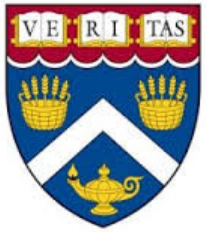
Final Project

Object Classification of Photo Instance Segmentation

Abderrazzaq, Saad



CSCI E-89 Deep Learning
Harvard University Extension School
Prof. Zoran B. Djordjević

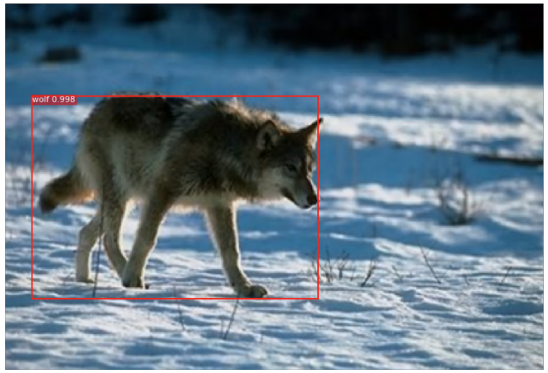
Problem Statement

- To identify instances in an image using instance segmentation by using an open source library that implements a method called Mask R-CNN on top of Keras/Tensorflow.
- Demonstrate single and multi-class instance segmentation as well as the process of annotating data. I use data from ImageNet (wolfs, signs) and my own photos to train and test image segmentation. The data is annotated using VGG Image Annotator (VIA).

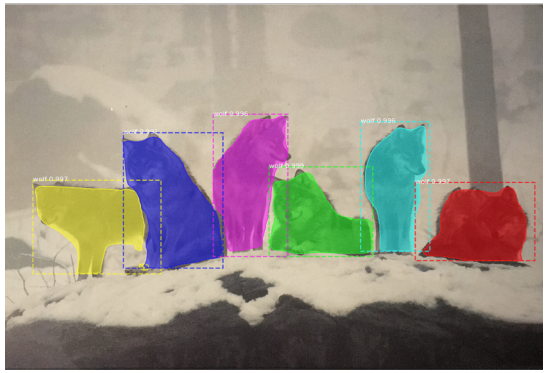
Instance Segmentation

Image Classification

Object Detection



Instance Segmentation



- Detection of one or more objects of different classes
- Differentiate instances
- Label each pixel with a category label
- Generate segment mask for each instance

Technology

- Anaconda
  - Python (Version 3.6.4)
  - Jupyter Notebook
  - Packages: NumPy, SciPy, Matplotlib, etc
- Keras (v2.1.5) / Tensoflow (v1.6.0)
- OpenCV 3.4 (i.e. cv2)
- Mask_RCNN (https://github.com/matterport/Mask_RCNN.git)
- pycocotools
- Imaug (https://github.com/aleju/imgaug)
- VGG Image Annotator (v1.0.5)

Data Set

- Dataset consists of images download from ImageNet
- The images came in a variety of sizes and quality and were all RGB.

- The first data batch consistent of variety of wolf species.
  - Out of 1391 images only 171 images annotated.
  - The total size of this data was 15M.

- The second batch consistent of traffic signs
  - Used to train the network to detect multiple traffic signs.
  - This batch consisted of 85 annotated images
  - Total size was 26M

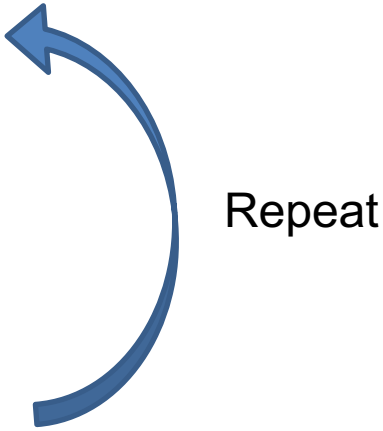
- The VGG Image Annotator (VIA) was used to annotate the images.

Installation Overview

1. Download Anaconda
2. Download and install Ubuntu pre-requisites for OpenCV
3. Download and build OpenCV
4. Install opencv-python
5. Install Cython
6. Git clone pycocotools
7. Git clone imgaug
8. Git clone mask_rcnn
9. Download and unzip VGG Image Annotator

Process

1. Create Configuration and DataSet code for:
    1. Loading images. (mrcnn has good examples)
    2. Loading image masks
    3. Splitting images and annotations between training and validation
2. Annotate Images
3. Validate data
4. Run training
5. Run detection tests
6. Debug/analyze



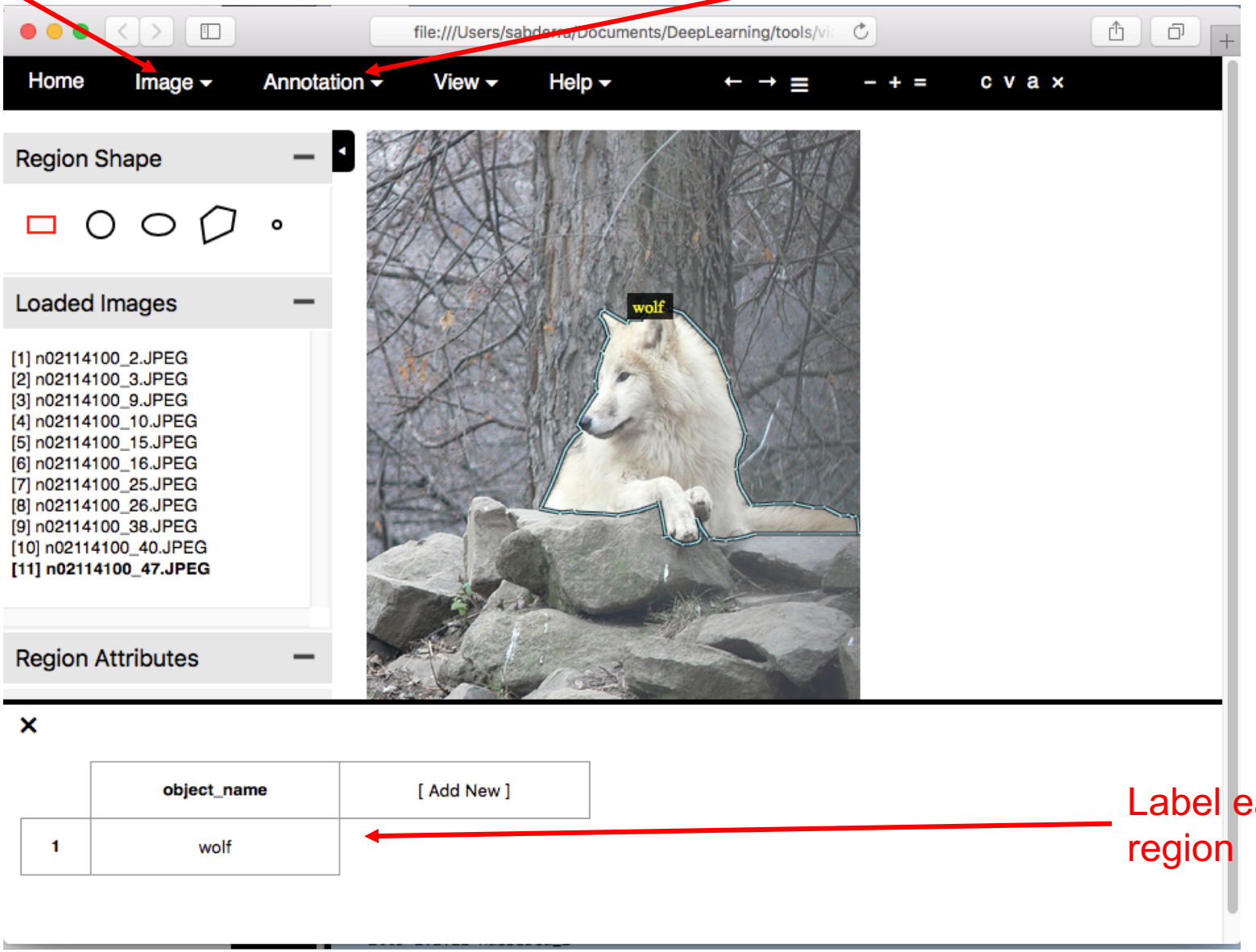
Concepts

- Region Proposals
  - An algorithm that identifies those rectangular regions giving them an ‘objectness’ score. This can be up to 1000 regions. The region of interest (ROI) is further operated on and filtered to remove all by the highest confidence of containing an object matching one of the classes.
- Intersection Over Union (IoU)
  - A measure used when evaluating object localization. This helps to narrow down the bounding boxes to the one best fitting the object. Uses the ground truth bounding box and the predicted bounding box. IoU is area of overlap over area of union.
- Non Max Suppression
- Feature Pyramids

VGG Image Annotator (VIA)

Load Images

Import/Save Annotations



Label each region

VIA JSON Format

- VIA exports annotations in both csv and json formats. I used json


```
{
  "n02114100_10.JPEG81442": {
    "fileref": "",
    "size": 81442,
    "filename": "n02114100_10.JPEG",
    "base64_img_data": "",
    "file_attributes": {},
    "regions": {
      "0": {
        "shape_attributes": {
          "name": "polygon",
          "all_points_x": [ ... ],
          "all_points_y": [ ... ]
        },
        "region_attributes": {
          "object_name": "wolf"
        }
      }
    }
  },
  "n02114100_120.JPEG174337": {
    ...
  },
  "n02114100_123.JPEG4410": {
    ...
  },
  ...
```


- The key attributes are:
  - filename is the location of the image on disk
  - all_points_x and all_points_y define the polygons for the masks and deriving the bounding boxes.
  - object_name was used to label the specific polygon. This needs to be mapped to the class index.
  - Note, shape of the image is missing and not written by via v1.0.5.

Dataset - load_by_annotations

```
    def load_by_annotations(self, dataset_dir, annotations_list, class_ids):
        """Load a specfic set of annotations and from them images.
        dataset_dir: Root directory of the dataset.
        annotations_list: The annotations (and images) to be loaded.
        class_ids: List of classes to use.
        """

        ann_class_names = set()
        for a in annotations_list:
            regions = a['regions']
            for r, v in regions.items():
                ann_class_names.add(v['region_attributes']['object_name'])

        # Add classes.
        for i, name in enumerate(ann_class_names):
            # Skip over background if it appears in the class name list
            index = i + 1
            if name != 'BG':
                print('Adding class {:3}:{}'.format(index, name))
                self.add_class('sign', index, name)
                self.map_name_to_id[name] = index

        # Add images
        for a in annotations_list:
            # Get the x, y coordinaets of points of the polygons that make up
            # the outline of each object instance. There are stores in the
            # shape_attributes (see json format above)
            polygons = [r['shape_attributes'] for r in a['regions'].values()]

            r_object_name = [r['region_attributes']['object_name'] for r in a['regior

            assert len(polygons) == len(r_object_name)

            # load_mask() needs the image shape. This is only managable since
            # the dataset is small. Consider pre-processing and/or updating via.
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "sign",
                image_id=a['filename'],  # use file name as a unique image id
                path=image_path,
                width=width, height=height,
                polygons=polygons,
                r_object_name=r_object_name)

```

- Given a list of annotations, this function loads the classes, polygons, and labels.
- Annotations are collected by the split_annotations() fcn

Dataset - load_mask

```
    def load_mask(self, image_id):
        """Generate instance masks for an image.
        Returns:
        masks: A bool array of shape [height, width, instance count] with
            one mask per instance.
        class_ids: a 1D array of class IDs of the instance masks.
        """

        # If not an object in our dataset image, delegate to parent class.
        image_info = self.image_info[image_id]
        if image_info["source"] not in self.class_names:
            print("warning: source {} not part of our classes, delegating to parer
            return super(self.__class__, self).load_mask(image_id)

        # Convert polygons to a bitmap mask of shape
        # [height, width, instance_count]
        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                        dtype=np.uint8)
        class_ids = []
        for i, p in enumerate(info["polygons"]):
            # Get indexes of pixels inside the polygon and set them to 1
            try:
                rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x']
                mask[rr, cc, i] = 1
                class_id = self.map_name_to_id[info['r_object_name'][i]]
                class_ids.append(class_id)
            except:
                print(image_info)
                raise

        # Return mask, and array of class IDs of each instance. Since we have
        # one class ID only, we return an array of 1s
        class_ids = np.array(class_ids, dtype=np.int32)
        return mask.astype(np.bool), class_ids
```

- Overridden from Mask RCNN *load_mask* to define how to load from the via format.
- *map_name_to_id* is used for associating the label string with the image id int.

Image Augmentation

```
import imgaug as ia
from imgaug import augmenters as iaa

ia.seed(1)

# http://imgaug.readthedocs.io/en/latest/source/augmenters.html#sequential
seq_of_aug = iaa.Sequential([
    iaa.Crop(percent=(0, 0.1)), # random crops

    # horizontally flip 50% of the images
    # iaa.Fliplr(0.5), # Does not make sense for signs

    # Gaussian blur to 50% of the images
    # with random sigma between 0 and 0.5.
    iaa.Sometimes(0.5,
        iaa.GaussianBlur(sigma=(0, 0.5))
    ),

    # Strengthen or weaken the contrast in each image.
    iaa.ContrastNormalization((0.75, 1.5)),

    # Add gaussian noise.
    # For 50% of all images, we sample the noise once per pixel.
    # For the other 50% of all images, we sample the noise per pixel AND
    # channel. This can change the color (not only brightness) of the
    # pixels.
    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5),

    # Make some images brighter and some darker.
    # In 20% of all cases, we sample the multiplier once per channel,
    # which can end up changing the color of the images.
    iaa.Multiply((0.8, 1.2), per_channel=0.2),

    # Apply affine transformations to each image.
    # Scale/zoom them from 90% 5o 110%
    # Translate/move them, rotate them
    # Shear them slightly -2 to 2 degrees.
    iaa.Affine(
        scale={"x": (0.9, 1.1), "y": (0.9, 1.1)},
        translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
        rotate=(-5, 5),
        shear=(-2, 2)
    )
], random_order=True) # apply augmenters in random order
```

- The imgaug library is used
- Supports deterministic setting for image/mask
- Need to test augmenter results to ensure safe

Training

```
model.load_weights(COCO_MODEL_PATH, by_name=True, exclude=[
    "mrcnn_class_logits", "mrcnn_bbox_fc",
    "mrcnn_bbox", "mrcnn_mask"])
```

Use a pre trained model for training

```
def train(model, dataset_path, epochs=30):
    """Train the model."""

    # Create the train and val dataset.
    dataset_train, dataset_val = create_datasets(dataset_path+'/train', config.CLASS_NAMES)

    # Prepare them
    dataset_train.prepare()
    dataset_val.prepare()

    # Experiment with training options.
    # Since we're using a very small dataset, and starting from
    # COCO trained weights, we don't need to train too long. Also,
    # no need to train all layers, just the heads should do it.
    print("Training network heads")
    history = model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=epochs,
                layers='heads',
                augmentation=seq_of_aug
                )

    return history
```

Split the data between training and val
Default is .8 for training.

Wrapping from MaskRCNN that builds and compiles a model using Keras

```
history = train(model, dataset_path, 75)
```

Kick off the training

Training Parameters

- No significant tuning was performed. I attempted to tune learning rate, activation, batch normalization, but changes did not out do default config.
- Configuration:

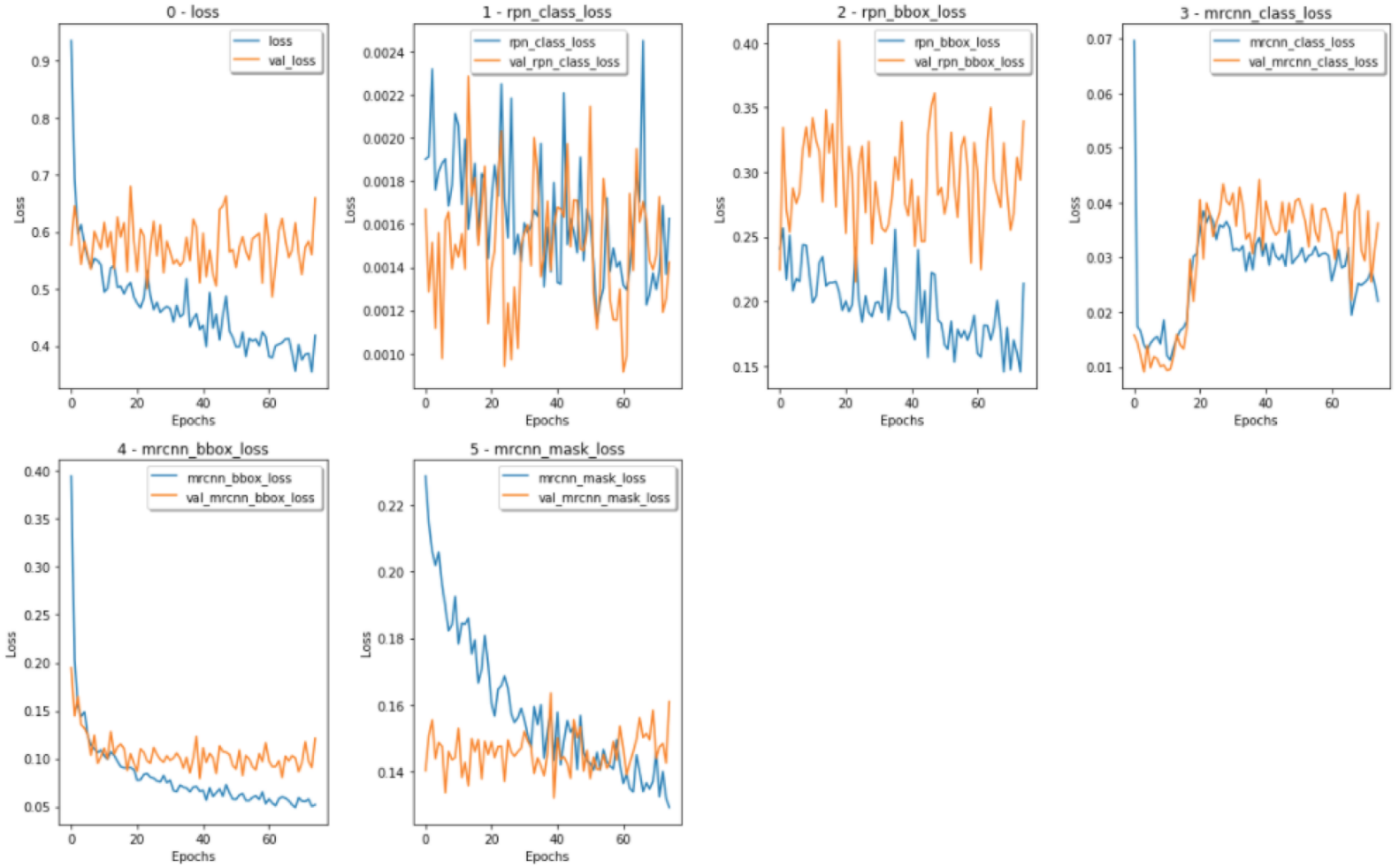
```
ACTIVATION                relu
ALL_CLASS_NAMES           ['BG', 'wolf']
BACKBONE                  resnet101
BACKBONE_STRIDES          [4, 8, 16, 32, 64]
BATCH_SIZE                2
BBOX_STD_DEV              [0.1 0.1 0.2 0.2]
CLASS_NAMES               ['wolf']
DETECTION_MAX_INSTANCES   100
DETECTION_MIN_CONFIDENCE  0.9
DETECTION_NMS_THRESHOLD   0.3
GPU_COUNT                 1
GRADIENT_CLIP_NORM        5.0
IMAGES_PER_GPU            2
LEARNING_RATE             0.001
LOSS_WEIGHTS              {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0,
'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE            14
MASK_SHAPE                [28, 28]
MAX_GT_INSTANCES          100
MEAN_PIXEL                [123.7 116.8 103.9]
MINI_MASK_SHAPE           (56, 56)
NAME                      wolf
NUM_CLASSES               2
POOL_SIZE                 7
POST_NMS_ROIS_INFERENCE   1000
POST_NMS_ROIS_TRAINING    2000
ROI_POSITIVE_RATIO        0.33
RPN_ANCHOR_RATIOS         [0.5, 1, 2]
RPN_ANCHOR_SCALES         (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE         1
RPN_BBOX_STD_DEV          [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD         0.7
RPN_TRAIN_ANCHORS_PER_IMAGE   256
STEPS_PER_EPOCH           100
TRAINING_VERBOSE          1
TRAIN_BN                  False
TRAIN_ROIS_PER_IMAGE      200
USE_MINI_MASK             True
USE_RPN_ROIS              True
VALIDATION_STEPS          50
WEIGHT_DECAY              0.0001
```

Example Training Output

```
Epoch 61/75100/100 [==============================] - 67s 674ms/step - loss: 0.1700 - rpn_class_loss: 0.0018 -
rpn_bbox_loss: 0.0581 - mrcnn_class_loss: 0.0085 - mrcnn_bbox_loss: 0.0302 - mrcnn_mask_loss: 0.0714 - val_loss:
0.7379 - val_rpn_class_loss: 0.0055 - val_rpn_bbox_loss: 0.2347 - val_mrcnn_class_loss: 0.0662 -
val_mrcnn_bbox_loss: 0.1594 - val_mrcnn_mask_loss: 0.2722
Epoch 62/75100/100 [==============================] - 67s 669ms/step - loss: 0.2485 - rpn_class_loss: 0.0029 -
rpn_bbox_loss: 0.1213 - mrcnn_class_loss: 0.0101 - mrcnn_bbox_loss: 0.0364 - mrcnn_mask_loss: 0.0779 - val_loss:
0.7327 - val_rpn_class_loss: 0.0047 - val_rpn_bbox_loss: 0.2240 - val_mrcnn_class_loss: 0.0743 -
val_mrcnn_bbox_loss: 0.1654 - val_mrcnn_mask_loss: 0.2642
Epoch 63/75100/100 [==============================] - 67s 669ms/step - loss: 0.1893 - rpn_class_loss: 0.0023 -
rpn_bbox_loss: 0.0708 - mrcnn_class_loss: 0.0091 - mrcnn_bbox_loss: 0.0341 - mrcnn_mask_loss: 0.0730 - val_loss:
0.7406 - val_rpn_class_loss: 0.0039 - val_rpn_bbox_loss: 0.2385 - val_mrcnn_class_loss: 0.0793 -
val_mrcnn_bbox_loss: 0.1763 - val_mrcnn_mask_loss: 0.2426
Epoch 64/75100/100 [==============================] - 67s 674ms/step - loss: 0.1678 - rpn_class_loss: 0.0014 -
rpn_bbox_loss: 0.0549 - mrcnn_class_loss: 0.0072 - mrcnn_bbox_loss: 0.0314 - mrcnn_mask_loss: 0.0728 - val_loss:
0.7117 - val_rpn_class_loss: 0.0049 - val_rpn_bbox_loss: 0.2295 - val_mrcnn_class_loss: 0.0665 -
val_mrcnn_bbox_loss: 0.1654 - val_mrcnn_mask_loss: 0.2454
Epoch 65/75100/100 [==============================] - 68s 677ms/step - loss: 0.1649 - rpn_class_loss: 0.0016 -
rpn_bbox_loss: 0.0548 - mrcnn_class_loss: 0.0050 - mrcnn_bbox_loss: 0.0317 - mrcnn_mask_loss: 0.0717 - val_loss:
0.7583 - val_rpn_class_loss: 0.0041 - val_rpn_bbox_loss: 0.2411 - val_mrcnn_class_loss: 0.0660 -
val_mrcnn_bbox_loss: 0.1838 - val_mrcnn_mask_loss: 0.2634
```

Evaluating Loss Results

- I captured the history data return by Keras and plotted the loss functions of the different networks in the mode.



Detection

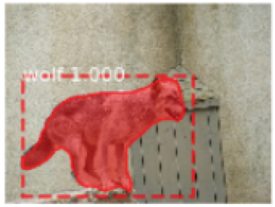
- Example for running detection on a single image

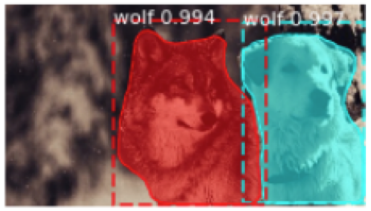
```
# Set up paths and directory variable (not shown for readability)

from mrcnn import utils
import mrcnn.model as modellib
import csci_e89_project.det as det

# Path to trained weights
PROJECT_WEIGHTS_PATH = "models/wolf20180508T1124_mask_rcnn_wolf_0075.h5”

config = InferenceConfig()
config.display()

model = modellib.MaskRCNN(mode="inference",  model_dir=MODEL_DIR, config=config)
model.load_weights(weights_path, by_name=True)

test_img_filename = TEST_IMAGE_DIR + 'wolf_pack.jpg'
test_img = plt.imread(test_img_filename)

results = model.detect([test_img], verbose=1)
```

Sample Result

Other Results





Summary of Mask RCNN framework Process

- Stage 1: Regional Proposal Network (RPN)
  - RPN Targets
  - RPN Predictions
- Stage 2: For each proposal
  - Proposal Classification
  - Process detection
    - Filtering out background
    - Filter by score
    - Apply non-max suppression (NMS)
  - Apply Bounding Box Refinement
- Stage 3: Generating Masks

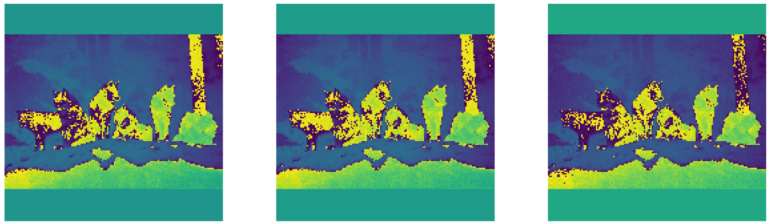
Visualizing Activations

Original

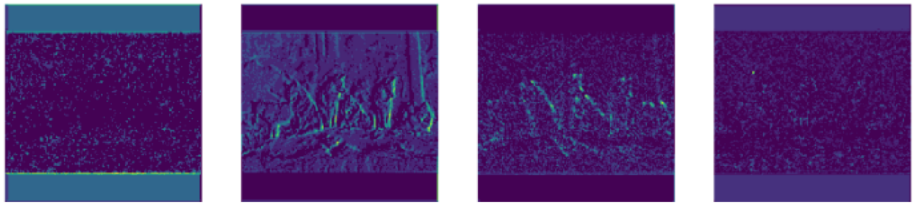
Layer

Activations

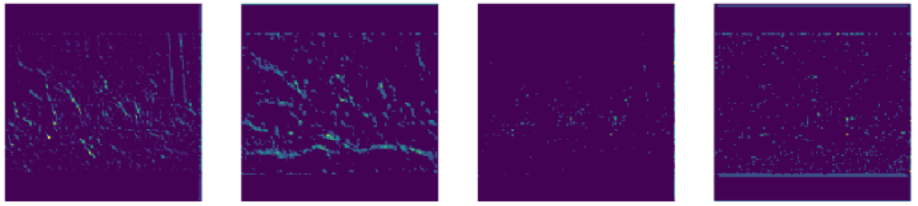
input_image



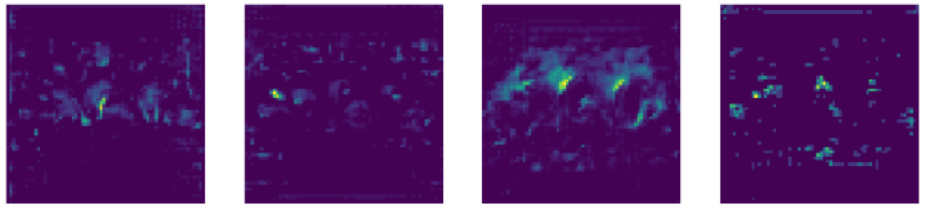
res2c_out



res3c_out



res4w_out



Summary of Results

- Mask RCNN framework is well written and easy to use. Some changes are required to
- Even with a small data set, a model trained with a single class showed impressive results.
- The overfitting could be attributed to training using 137 images, validation using 34. Additional images needed.
- The VIA annotation tools was simple and easy to use, and works well with small data sets.
- Detection time 0.250 seconds on GPU, 10 seconds on CPU

Lessons Learned

- With small or limited data sets, leveraging pre-trained models (transfer learning) can still result in good performing models.
- The area of object detection and instance segmentation pulls together many concepts and networks to produce an end-end system.
- Interconnecting different modules (RPN, FPN, etc) is a powerful architectural feature.
- Not suitable for real-time. Detection on my Mac's CPU were approx 5-10 seconds.
- Other approaches such as single shot detectors and yolo are more promising for real time, but they do not support instance segmentation. Also, for accuracy, Faster R-CNN which is what Mask R-CNN is based is currently a good choice.
- VIA is not sufficient for complicated workflows or teams. Other tools should be considered.

YouTube URLs, Last Page

- Two minute (short):
- 15 minutes (long):