Lecture 10:

Convolutional Neural Networks (CNN) – Part II

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Course Map

Lecture 1: Course Overview and Introduction to Machine Learning

Lecture 2: Bayesian Methods in Machine Learning

classic ML tools & algorithms

Lecture 3: Dimensionality Reduction: Principal Component Analysis

Lecture 4: Linear and Non-linear Regression and Classification

Lecture 5: Unsupervised Learning: K-Means, GMM, Trees

Lecture 6: Adaptive Learning and Gradient Descent Optimization Algorithms

Lecture 7: Introduction to Artificial Neural Networks - The Perceptron

ANNs/AI/Deep Learning

Lecture 8: Introduction to Artificial Neural Networks - Building Multiple Dense Layers

Lecture 9: Convolutional Neural Networks (CNNs) - Part I

Lecture 10: CNNs - Part II

Lecture 11: Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs)

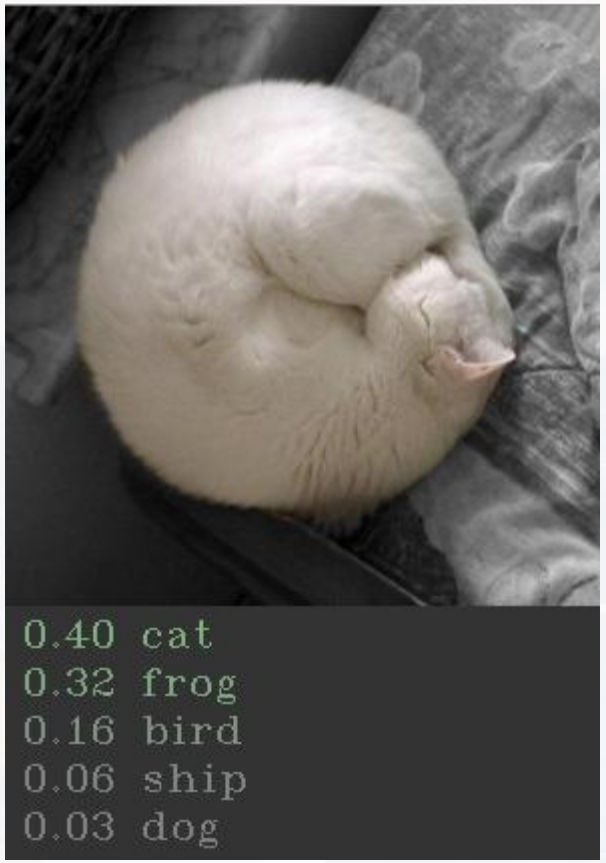
Lecture 12: Combining LSTMs and CNNs

Lecture 13: Running Models on GPUs and Parallel Processing

Lecture 14: Project Presentations

Lecture 15: Transformer

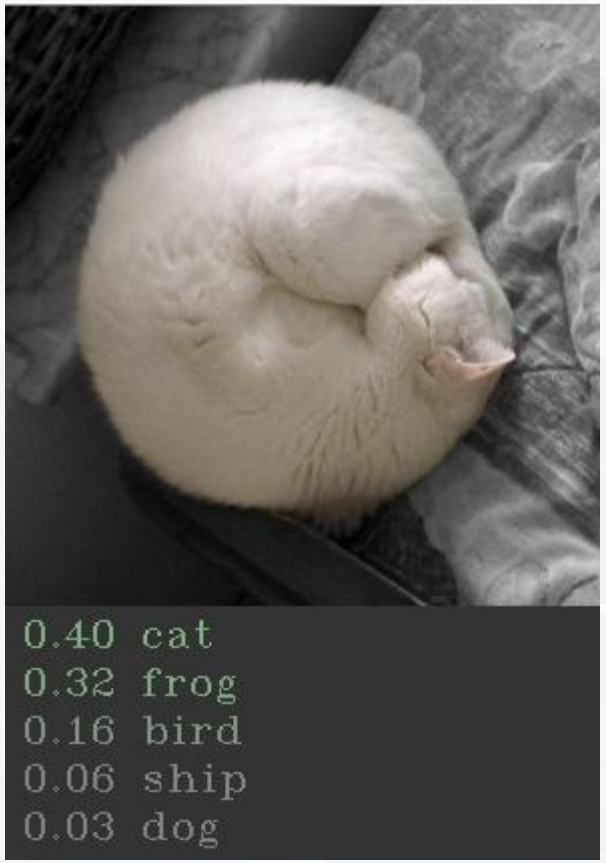
Lecture 16: GNN



Outline

- **Labeling Tools**

- **Calling a Pretrained CNN**



Outline

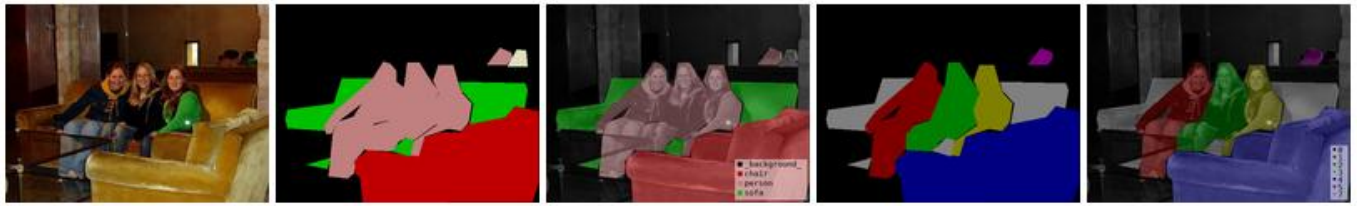
- Labeling Tools

- Calling a Pretrained CNN

"labelme"

Description

Labelme is a graphical image annotation tool inspired by http://labelme.csail.mit.edu.
It is written in Python and uses Qt for its graphical interface.

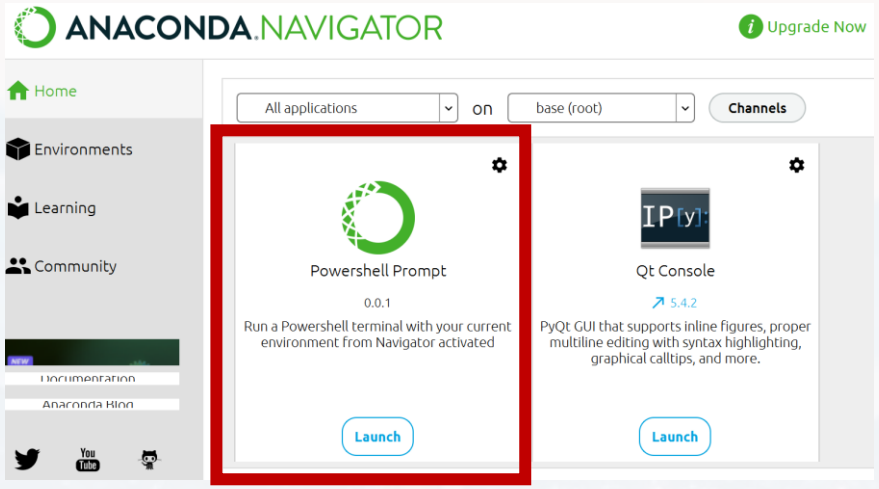


VOC dataset example of instance segmentation.

Other examples (semantic segmentation, bbox detection, and classification).

Various primitives (polygon, rectangle, circle, line, and point).

Convolutional Neural Networks:




```
conda install labelme
```

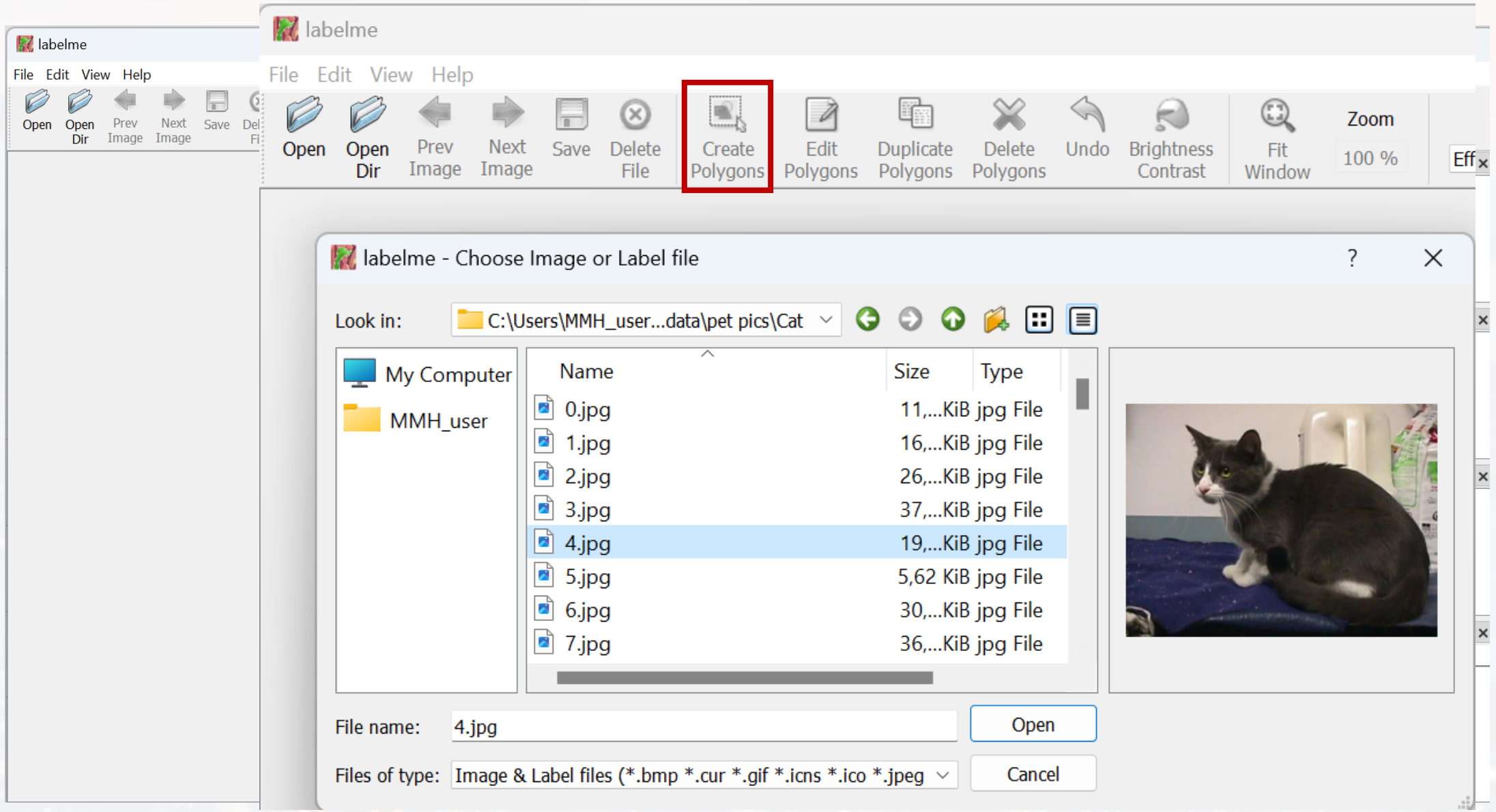
```
C:\WINDOWS\System32\Win
(base) PS C:\Users\MMH_user> conda create --name=labelme python=3
```

```
(base) PS C:\Users\MMH_user> conda activate labelme
```

```
(labelme) PS C:\Users\MMH_user> labelme
```

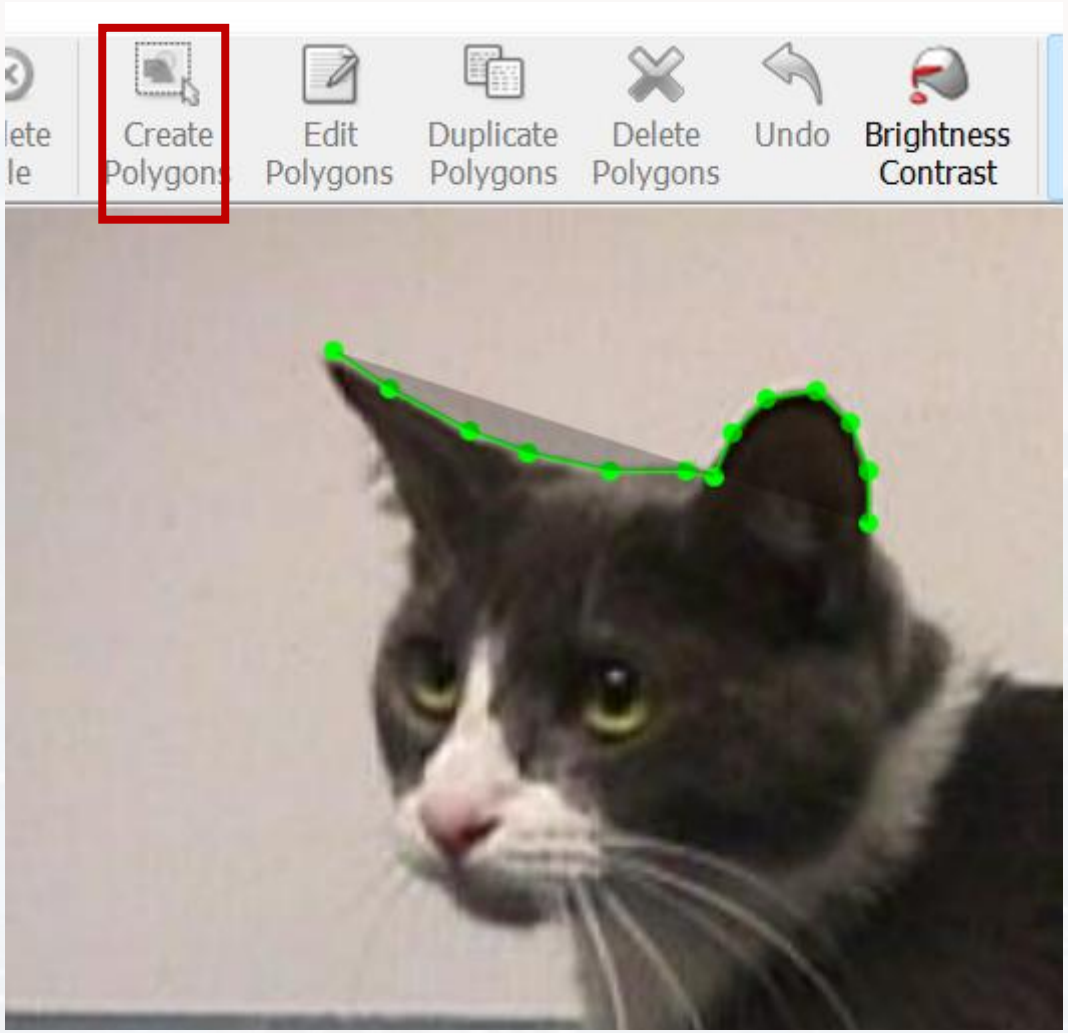
Convolutional Neural Networks:

Labeling



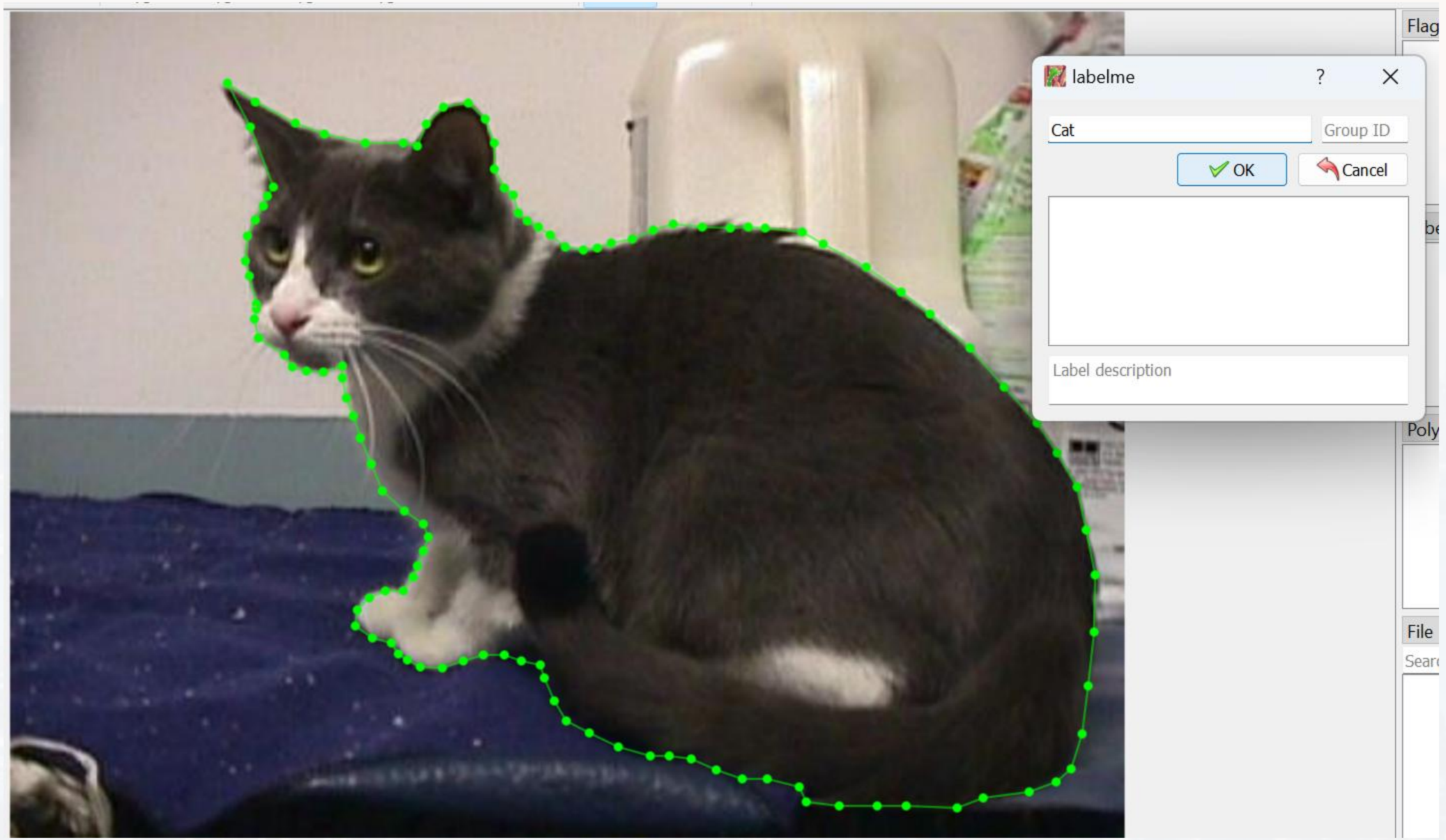
Convolutional Neural Networks:

Labeling



Convolutional Neural Networks:

Labeling



Convolutional Neural Networks:

Labeling

image is saved as `.json`

run within the labelme prompt:

```
labelme_export_json .\Cat4label.json -o Cat4label_json
```

Convolutional Neural Networks:

PNG File

img

label

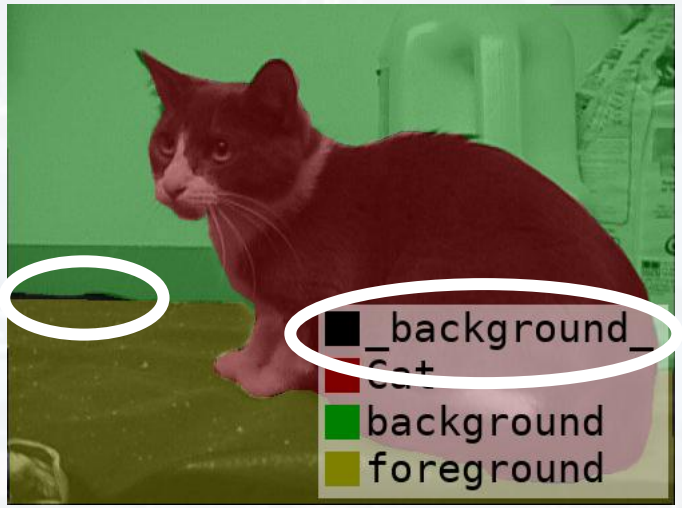
label_viz

Text Document

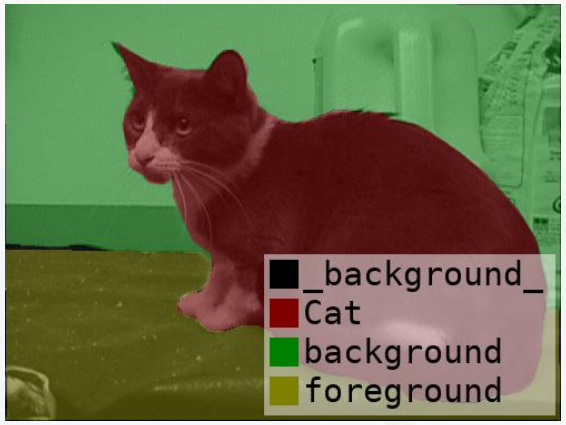
label_names

File Edit View

```
_background_
Cat
background
foreground
```



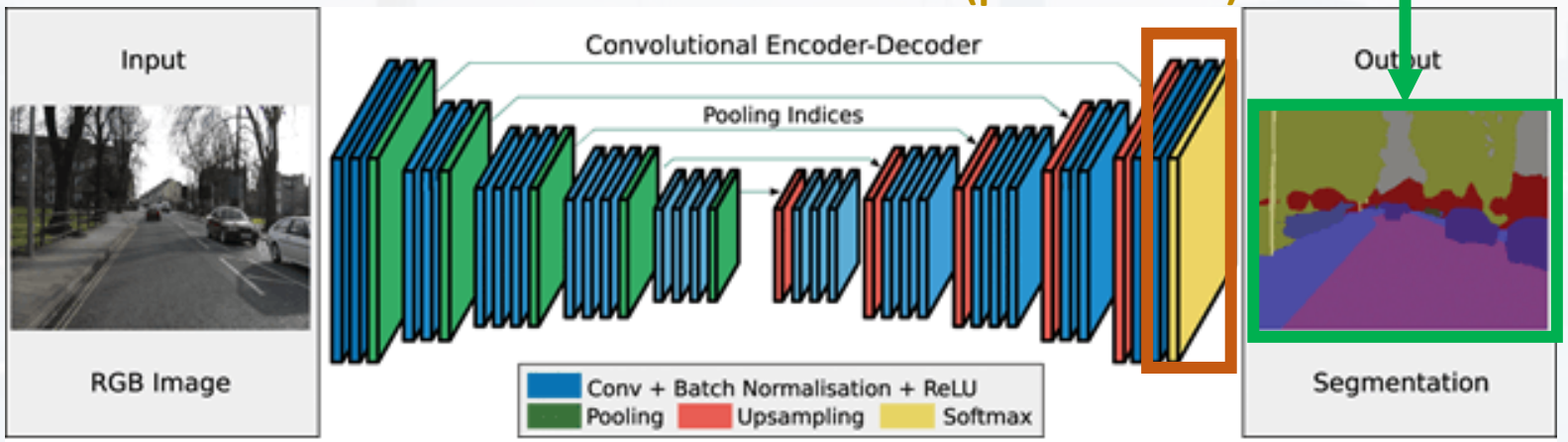
Convolutional Neural Networks:



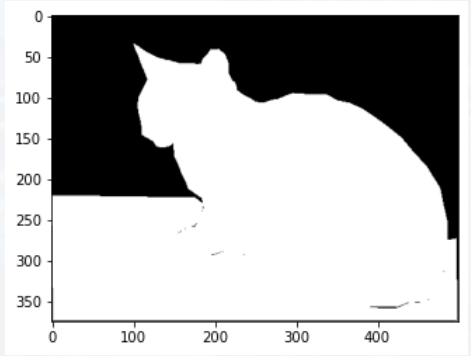

```
I = plt.imread('Cat/Cat4label_json/label.png')
```

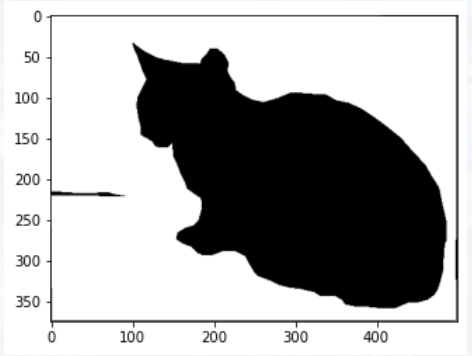
```
I.shape
(375, 500, 4)
```

softmax
(probabilities)



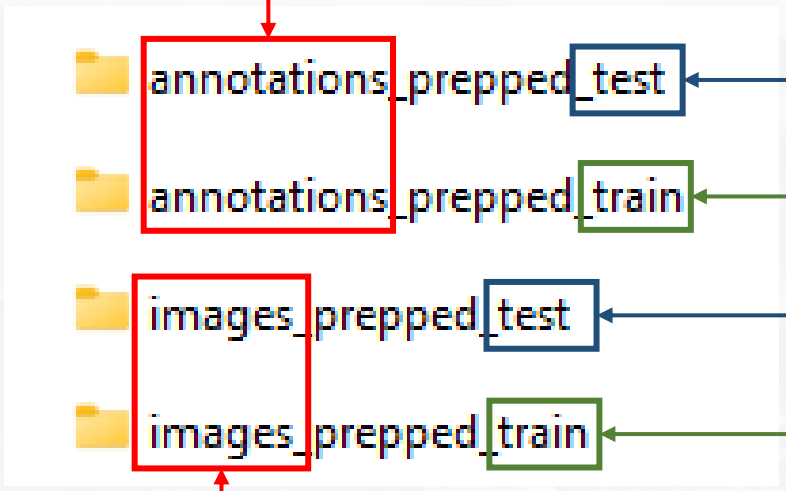
labels assigned during training vs actual labels





Convolutional Neural Networks:

Labeling



the labelled images

test data to evaluate training

training data

test data to evaluate training

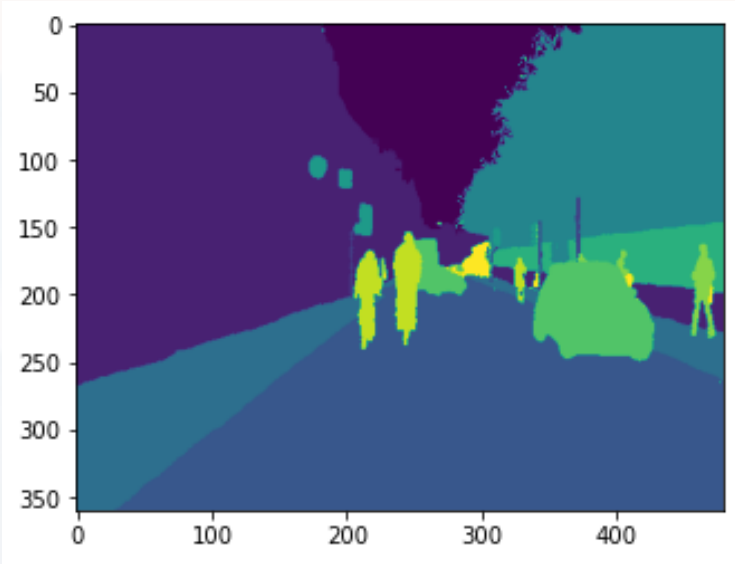
training data

the actual images

Convolutional Neural Networks: Labeling

[nice dataset]

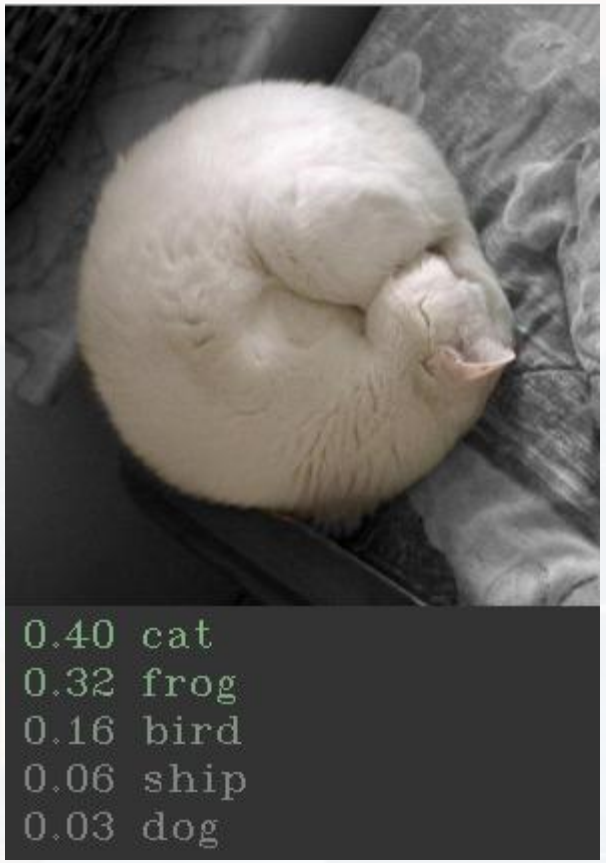
images_prepped_test



annotations_prepped_test

```
I = plt.imread('segmentation/pics/annotations_prepped_test/0016E5_07959.png')
```

```
I.shape
(360, 480)
```



Outline

- Labeling Tools

- Calling a Pretrained CNN

demonstrating realistic segmentation with reasonable results will take a few hours

→ check out my code on GitHub

Convolutional Neural Networks: Segmentation

```
from keras_segmentation.models.unet import *
```

Type	Names
VGG	'vgg16' 'vgg19'
ResNet	'resnet18' 'resnet34' 'resnet50' 'resnet101' 'resnet152'
SE-ResNet	'seresnet18' 'seresnet34' 'seresnet50' 'seresnet101' 'seresnet152'
ResNeXt	'resnext50' 'resnext101'
SE-ResNeXt	'seresnext50' 'seresnext101'
SENet154	'senet154'
DenseNet	'densenet121' 'densenet169' 'densenet201'
Inception	'inceptionv3' 'inceptionresnetv2'
MobileNet	'mobilenet' 'mobilenetv2'
EfficientNet	'efficientnetb0' 'efficientnetb1' 'efficientnetb2' 'efficientnetb3' 'efficientnetb4' 'efficientnetb5' efficientnetb6' efficientnetb7'

```
model = unet(n_classes = n_classes,\
                         input_height = 416, input_width = 608)

model.train(
        train_images       = my_path + r"images_prepped_train//",
        train_annotations  = my_path + r"annotations_prepped_train//",
        checkpoints_path   = my_path + r"checkpoints//",
        do_augment                   = True,
        gen_use_multiprocessing = True,
        auto_resume_checkpoint   = True,
        epochs = 5)
```

calling the specific network

saves current weights

Keras provides an augmentation routine

```
model = vgg_unet(n_classes = n_classes,\
                         input_height = 416, input_width = 608)

model.train(
        train_images       = my_path + r"images_prepped_train//",
        train_annotations  = my_path + r"annotations_prepped_train//",
        checkpoints_path   = my_path + r"checkpoints//",
        do_augment                   = True,
        gen_use_multiprocessing = True,
        auto_resume_checkpoint   = True,
        epochs = 5)
```

Keras provides an augmentation routine

Note: I always run my **own augmentation** routine, see e.g. AugmentMyImages.py

```
run:

S = SegmentMyImages()

S.Training()
```

Convolutional Neural Networks: Segmentation

```
S = SegmentMyImages()

S.Training()
```

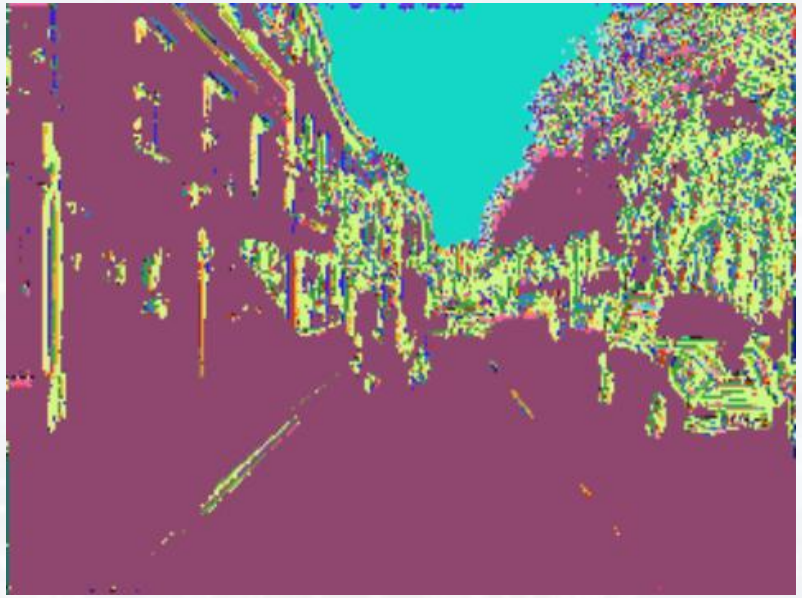
```
Dataset verified!
Epoch 1/5
512/512 [==============================] - ETA: 0s - loss: 4.1406 - accuracy: 0.0353     saved  ../data/segmentation
pics/checkpoints//.0
512/512 [==============================] - 3888s 8s/step - loss: 4.1406 - accuracy: 0.0353
Epoch 2/5
512/512 [==============================] - ETA: 0s - loss: 3.7805 - accuracy: 0.1636     saved  ../data/segmentation
pics/checkpoints//.1
512/512 [==============================] - 4133s 8s/step - loss: 3.7805 - accuracy: 0.1636
Epoch 3/5
512/512 [==============================] - ETA: 0s - loss: 3.4534 - accuracy: 0.3338     saved  ../data/segmentation
pics/checkpoints//.2
512/512 [==============================] - 4072s 8s/step - loss: 3.4534 - accuracy: 0.3338
Epoch 4/5
512/512 [==============================] - ETA: 0s - loss: 3.1926 - accuracy: 0.3980     saved  ../data/segmentation
pics/checkpoints//.3
512/512 [==============================] - 3363s 7s/step - loss: 3.1926 - accuracy: 0.3980
Epoch 5/5
512/512 [==============================] - ETA: 0s - loss: 3.0071 - accuracy: 0.4318   saved  ../data/segmentation
pics/checkpoints//.4
512/512 [==============================] - 3616s 7s/step - loss: 3.0071 - accuracy: 0.4318
```

```
MyModel = S.TrainedModel
```

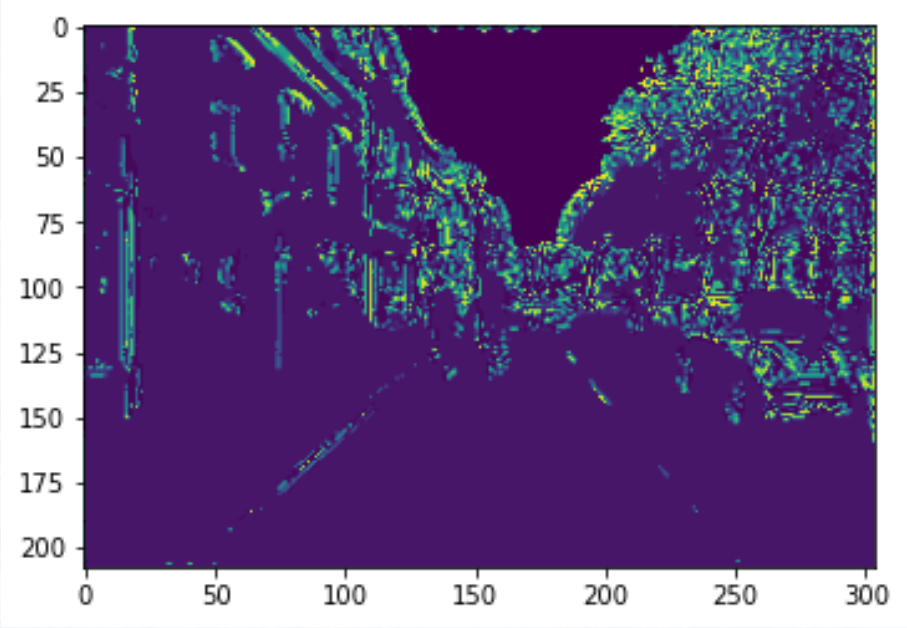
```
MyModel.summary()
```
→ returns the structure of the CNN

applying the trained CNN to an image:

```
out = S.ApplyTrainedNetwork()
```



```
plt.imshow(out)
```



recovering model from checkpoints:

```
MyModel = S.TrainedModel
out     = S.ApplyTrainedNetwork()

S.RecoverFromCheckpoint()
```

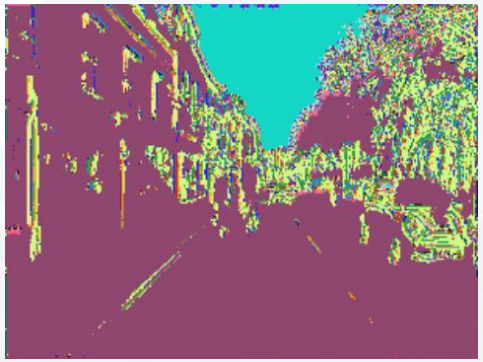
applying the trained CNN to an image:

```
#loading untrained CNN
model = self.model

if not image_name:
    image_name = '0016E5_07965.png'
    ....
#calling input from checkpoints
latest = tf.train.latest_checkpoint(self.checkpoint_path)
model.load_weights(latest)
```

untrained model (just CNN itself)

transfer the saved weights to untrained network → now it starts from latest training state



Convolutional Neural Networks: Segmentation

visualizing weights:

→ see `model.layers`

nice example

Thank you very much for your attention!