Computer Vision [H02A5a] : Final Project Presentation

Group 34

Authors:

- Chaitra Harsha (r0775320)
- Bavo Kempen (r0585283)
- Muhammad Farjad Malik (r0689729)
- Po-Kai Yang (r0774709)

Submitted Weights

Dropbox link: https://www.dropbox.com/s/excxlgyh1vq91y3/weights.zip?dl=0

Content

Introduction

First part: perform neural network on computer vision tasks, including classification and segmentation.

- Train model from scratch
- Train model via Fine-Tuning

Second part: find the weaknesses of the classification model

- Utilize White-Box Attack on the model

Environment: Google Colab GPU

Dataset: PASCAL VOC 2009 Dataset

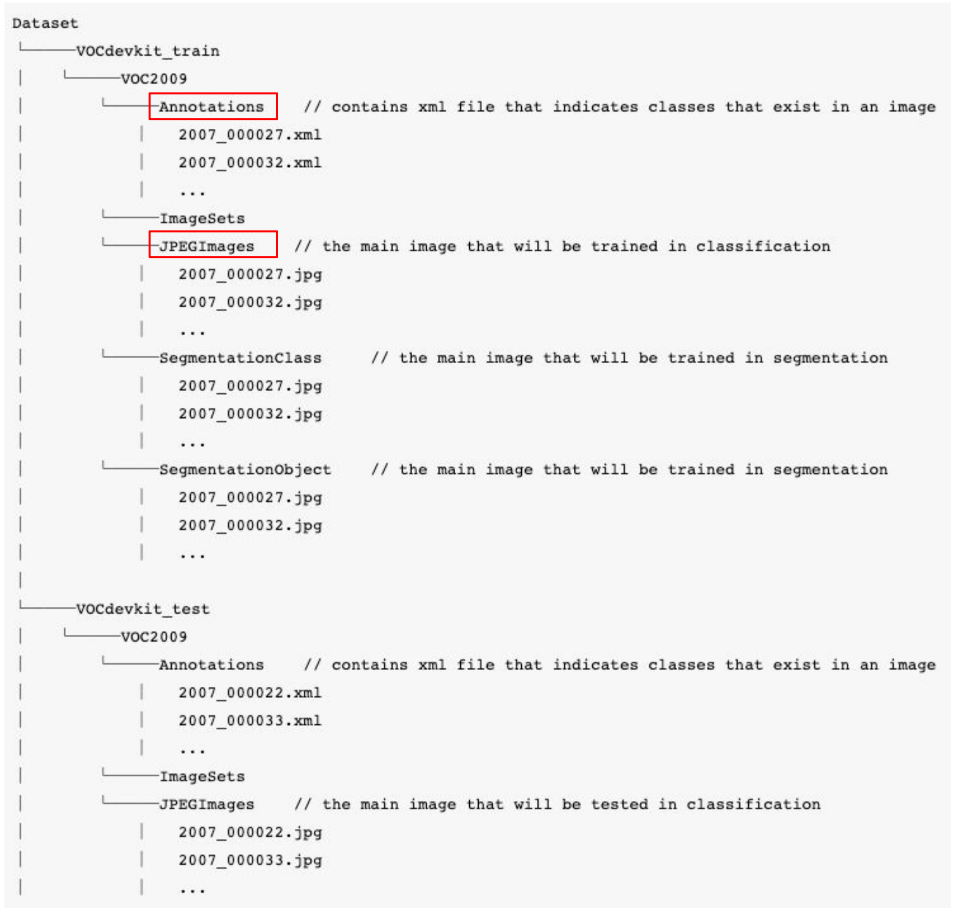
Dataset Preparation

Dataset: PASCAL VOC2009 Dataset

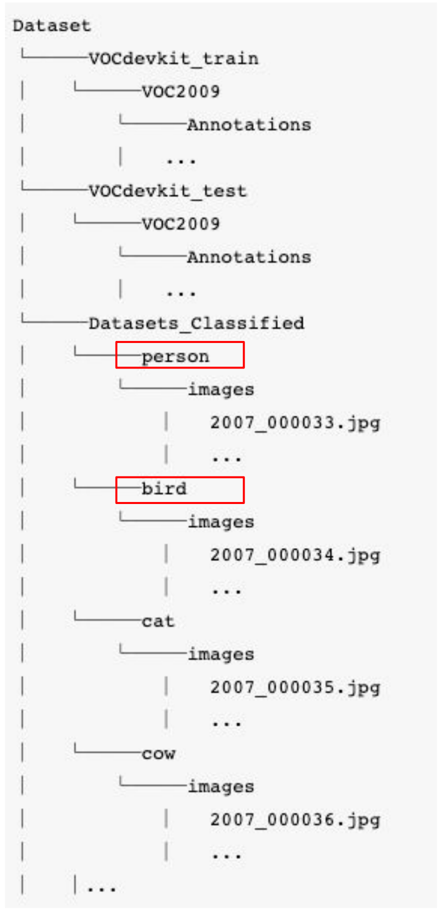
- Train/Validation Image:	7819 images
- Test Image:	6926 images

Classes: 20 classes

- person, bird, cat, cow, dog, horse,
- sheep, aeroplane, bicycle, boat, bus,
- car, motorbike, train, bottle, chair,
- diningtable, pottedplant, sofa, tvmonitor



```
Dataset
└────VOCdevkit_train
|    └────VOC2009
|        └────Annotations    // contains xml file that indicates classes that exist in an image
|            |  2007_000027.xml
|            |  2007_000032.xml
|            |  ...
|        └────ImageSets
|        └────JPEGImages    // the main image that will be trained in classification
|            |  2007_000027.jpg
|            |  2007_000032.jpg
|            |  ...
|        └────SegmentationClass     // the main image that will be trained in segmentation
|            |  2007_000027.jpg
|            |  2007_000032.jpg
|            |  ...
|        └────SegmentationObject    // the main image that will be trained in segmentation
|            |  2007_000027.jpg
|            |  2007_000032.jpg
|            |  ...
|
└────VOCdevkit_test
|    └────VOC2009
|        └────Annotations    // contains xml file that indicates classes that exist in an image
|            |  2007_000022.xml
|            |  2007_000033.xml
|            |  ...
|        └────ImageSets
|        └────JPEGImages    // the main image that will be tested in classification
|            |  2007_000022.jpg
|            |  2007_000033.jpg
|            |  ...
```



```
Dataset
└────VOCdevkit_train
|    └────VOC2009
|        └────Annotations
|        |   ...
└────VOCdevkit_test
|    └────VOC2009
|        └────Annotations
|        |   ...
└────Datasets_Classified
|    └────person
|        └────images
|            |  2007_000033.jpg
|            |  ...
|    └────bird
|        └────images
|            |  2007_000034.jpg
|            |  ...
|    └────cat
|        └────images
|            |  2007_000035.jpg
|            |  ...
|    └────cow
|        └────images
|            |  2007_000036.jpg
|            |  ...
|    |...
```

KU LEUVEN

VOC 2009 dataset is very imbalanced, e.g. person: 2641, bicycle: 299.

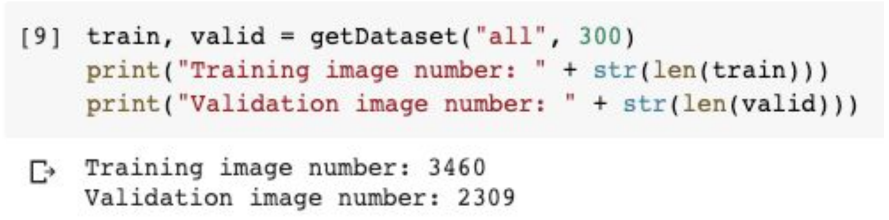
Only fetch the same amount of images to train our classifier

We only read max 300 images from a class directory:

- X < 300: fetch X images
- X >= 300: fetch 300 images

Training Validation Ratio: 0.6

3460 + 2309 = 5769 < 300 * 20 = 6000



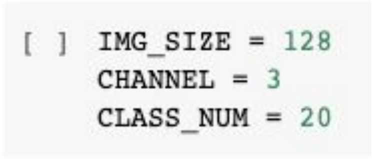
```
[9] train, valid = getDataset("all", 300)
    print("Training image number: " + str(len(train)))
    print("Validation image number: " + str(len(valid)))
```

```
Training image number: 3460
Validation image number: 2309
```

Classification

Training image size = 128, with RGB (Channel = 3)

Total class number = 20



```
[ ] IMG_SIZE = 128
    CHANNEL = 3
    CLASS_NUM = 20
```

We apply one hot encoding to the ylabel:

E.g. {aeroplane, bus, train}

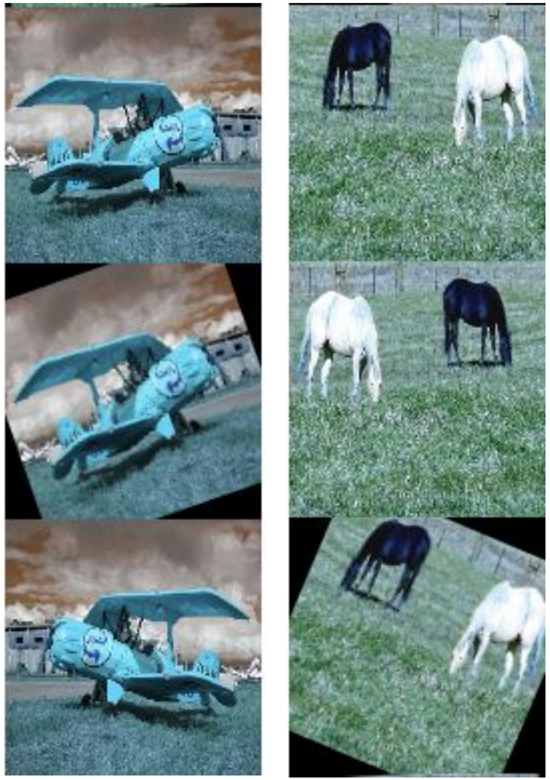
=> {0,5,18}

=> [1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0]

Data Augmentation:

- Image Rotation: **-25 / +25 degree**
- Image Flipping: **Horizontal Flip**

Random choose some augmentation functions from the 3 above and apply:



KU LEUVEN

Train Model from Scratch

Batch Size : 32

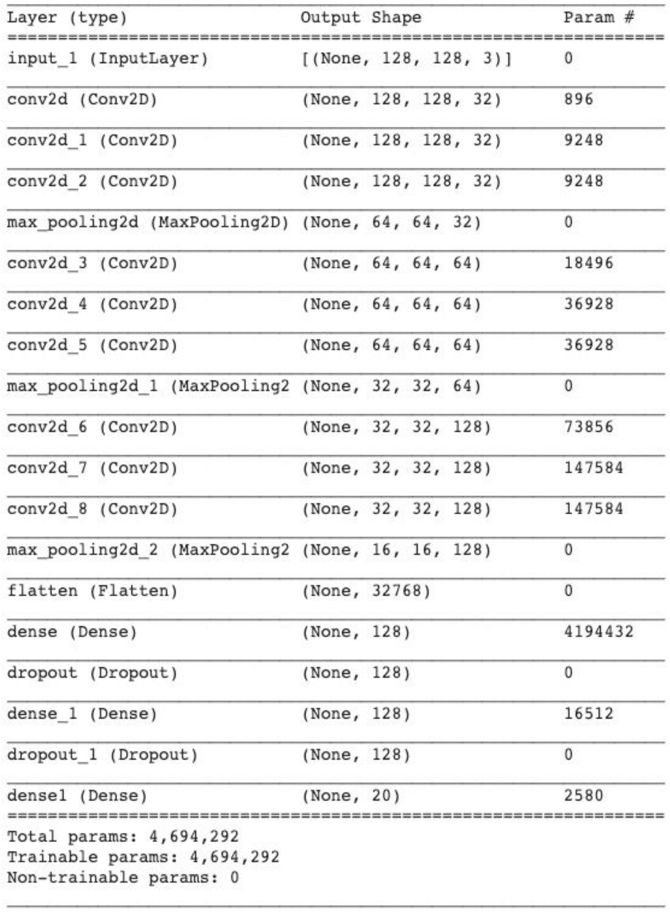
Early Stopping with max epoch=100

15 layers

- 3 max pooling layer
- 3 fully-connected layer

Multi-Label Classifier:

- loss function = 'binary crossentropy'
- Final Activation Function = 'sigmoid'

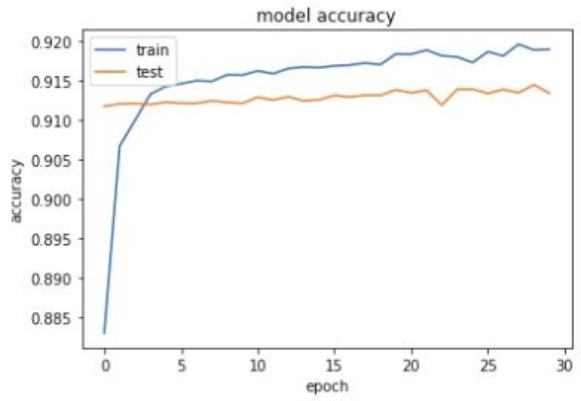


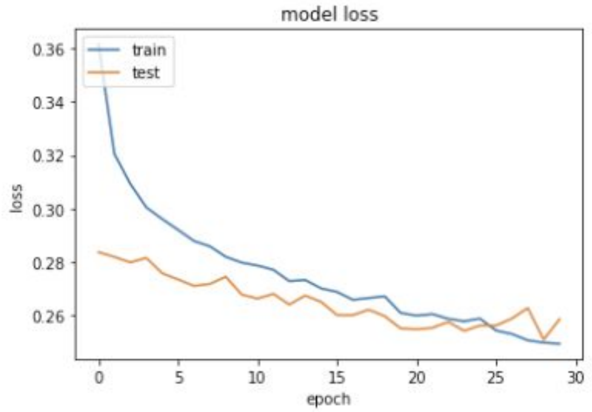
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 128)               4194432
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense1 (Dense)               (None, 20)                2580
=================================================================
Total params: 4,694,292
Trainable params: 4,694,292
Non-trainable params: 0
_________________________________________________________________
```

Evaluate on non-trained model / trained model:

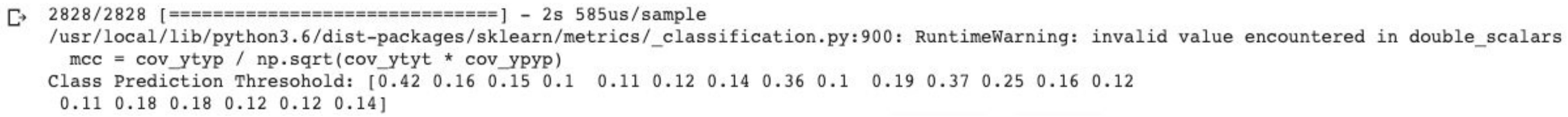
```
Untrained model:
11278/11278 - 12s - loss: 1.0569 - accuracy: 0.4183
Restored model:
11278/11278 - 5s - loss: 0.2148 - accuracy: 0.9238
```

Plot the training history:





Next, we try to construct the threshold for each class on the training result. The threshold indicate the minimum probability to determine if an image contains the target object.



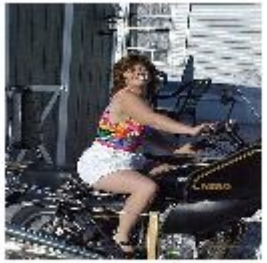
```
2828/2828 [==============================] - 2s 585us/sample
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:900: RuntimeWarning: invalid value encountered in double_scalars
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
Class Prediction Thresohold: [0.42 0.16 0.15 0.1  0.11 0.12 0.14 0.36 0.1  0.19 0.37 0.25 0.16 0.12
 0.11 0.18 0.18 0.12 0.12 0.14]
```

If the output of a prediction is

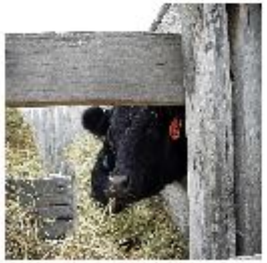
[0.3, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.13, 0.13]

It indicates that the image contains the object 2 and 19.

Result:

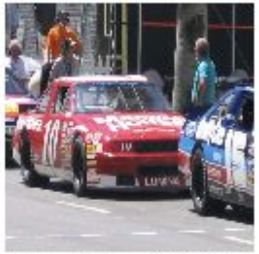


```
['0.61', '0.08', '0.02', '0.01', '0.08', '0.02
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
['person', 'bicycle', 'motorbike', 'bottle']
----------------------------------
```

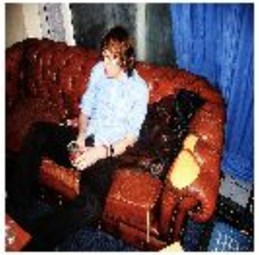


```
['0.25', '0.10', '0.02', '0.11', '0.05', '0.11
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
['cow', 'train']
```

```
----------------------------------
```



```
['0.35', '0.01', '0.00', '0.00', '0.00'
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
['bus', 'car']
----------------------------------
```



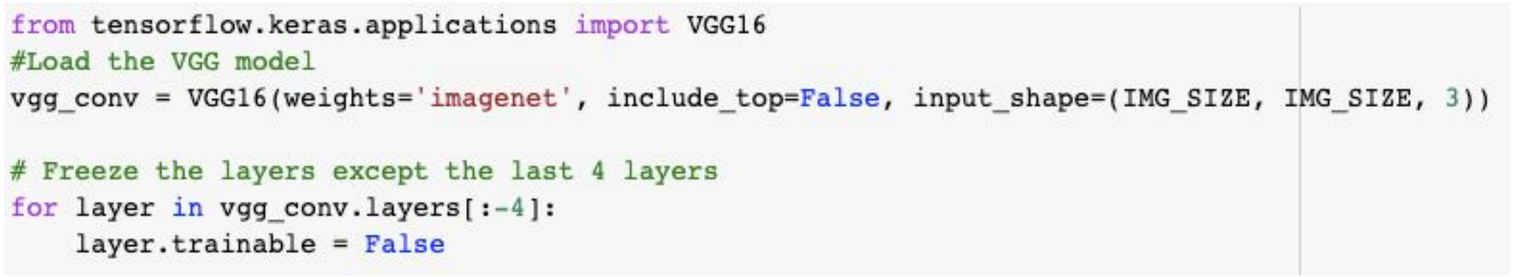
```
['0.52', '0.16', '0.03', '0.01', '0.07'
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
['person', 'bicycle']
```

Model trained from Fine-Tuning

Apply fine-tuning on : VGG 16 model

- CNN model
- Test accuracy of 92.7% in ImageNet dataset

Since in our notebook, we use VOC 2009 dataset, our test image will be different than the training image trained by the VGG16 model.



```
from tensorflow.keras.applications import VGG16
#Load the VGG model
vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))

# Freeze the layers except the last 4 layers
for layer in vgg_conv.layers[:-4]:
    layer.trainable = False
```

Batchsize = 32

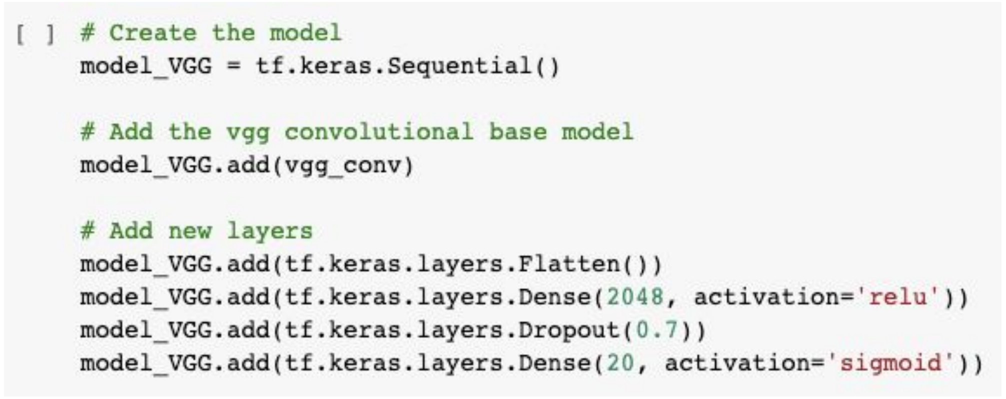
Early Stopping with max epoch=100

Last Activation Function:

- Sigmoid

Loss Function:

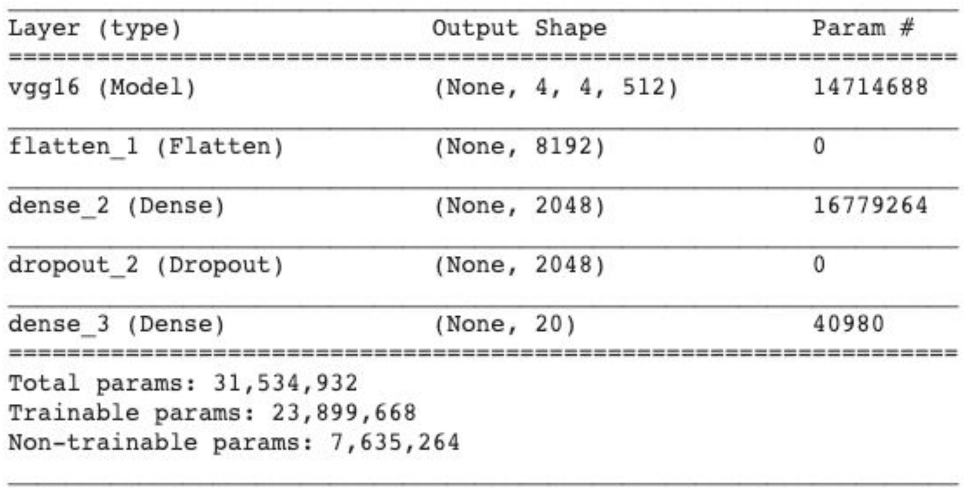
- Binary Cross Entropy



```
[ ] # Create the model
    model_VGG = tf.keras.Sequential()

    # Add the vgg convolutional base model
    model_VGG.add(vgg_conv)

    # Add new layers
    model_VGG.add(tf.keras.layers.Flatten())
    model_VGG.add(tf.keras.layers.Dense(2048, activation='relu'))
    model_VGG.add(tf.keras.layers.Dropout(0.7))
    model_VGG.add(tf.keras.layers.Dense(20, activation='sigmoid'))
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 2048)              16779264
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                40980
=================================================================
Total params: 31,534,932
Trainable params: 23,899,668
Non-trainable params: 7,635,264
_________________________________________________________________
```

```
Untrained model:
11288/11288 - 10s - loss: 6.6013 - accuracy: 0.4759
Restored model:
11288/11288 - 10s - loss: 0.1695 - accuracy: 0.9425
```

```
2828/2828 [==============================] - 4s 1ms/sample
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:900: RuntimeWarning: invalid value encountered in double_scalars
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
[0.41 0.69 0.41 0.35 0.24 0.24 0.52 0.26 0.36 0.52 0.23 0.21 0.26 0.82
 0.1  0.31 0.11 0.22 0.24 0.12]
```

```
['0.86', '0.00', '0.00', '0.00', '0.
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
['person', 'bus']
```

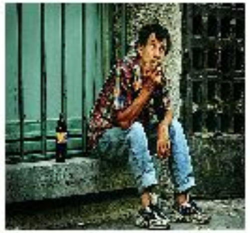
```
['0.98', '0.00', '0.00', '0.00', '0.
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
['person']
```

```
['0.06', '0.02', '0.13', '0.02', '0.
[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
['dog', 'sofa']
```

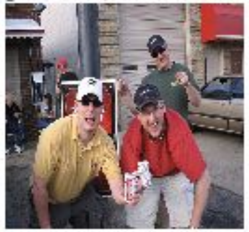
```
['0.03', '0.00', '0.00', '0.10', '0.
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
['horse']
```

Compare two model results

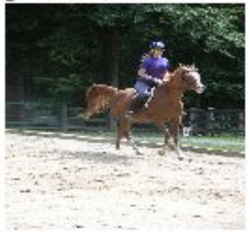
```
Self_Trained Model Prediction:
person, dog, bicycle, motorbike
VGG + Fine-Tuning Model Prediction:
```



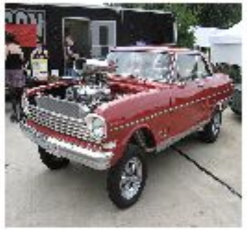
```
Self_Trained Model Prediction:
person, bottle, chair, sofa, tvmonitor
VGG + Fine-Tuning Model Prediction:
person, bottle
```



```
Self_Trained Model Prediction:
person, dog
VGG + Fine-Tuning Model Prediction:
person, horse
```

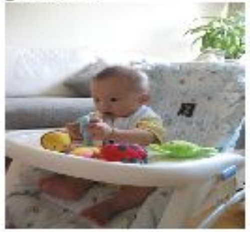


```
Self_Trained Model Prediction:
person, bus, car
VGG + Fine-Tuning Model Prediction:
car
```

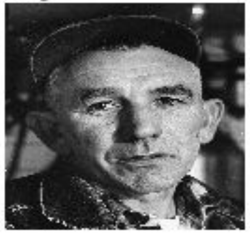


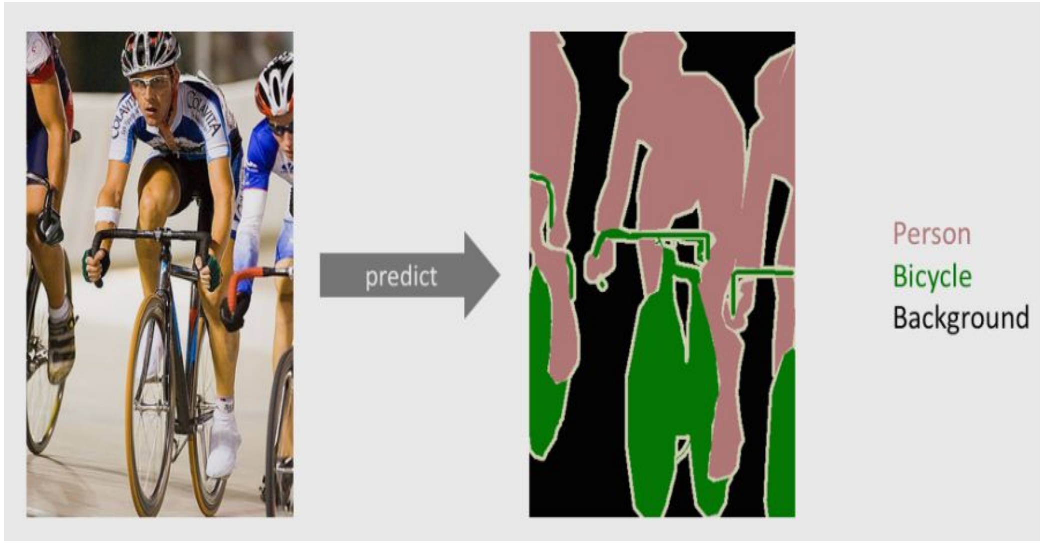
```
Self_Trained Model Prediction:

VGG + Fine-Tuning Model Prediction:
person
```



```
Self_Trained Model Prediction:
bird, cat, dog
VGG + Fine-Tuning Model Prediction:
dog
```

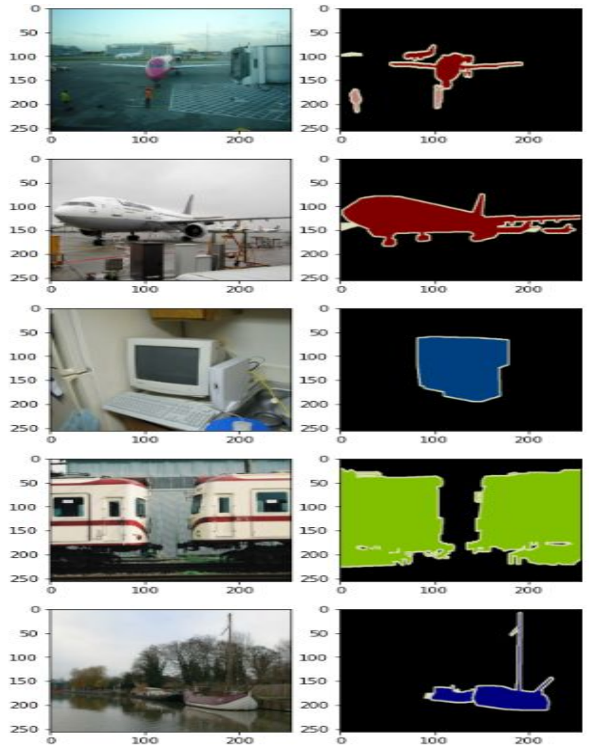




Segmentation

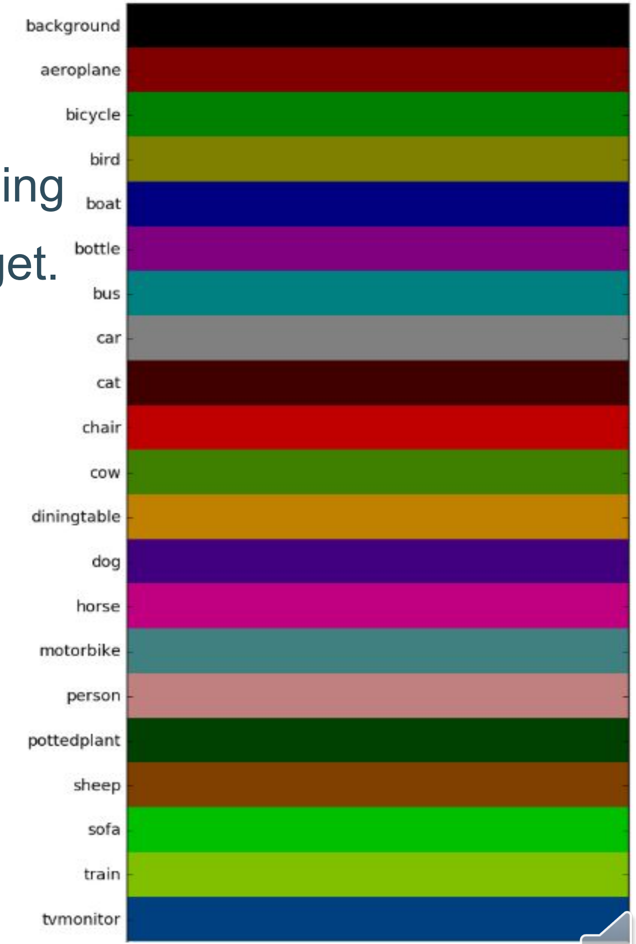
"WHAT" as well as "WHERE" information

Annotation



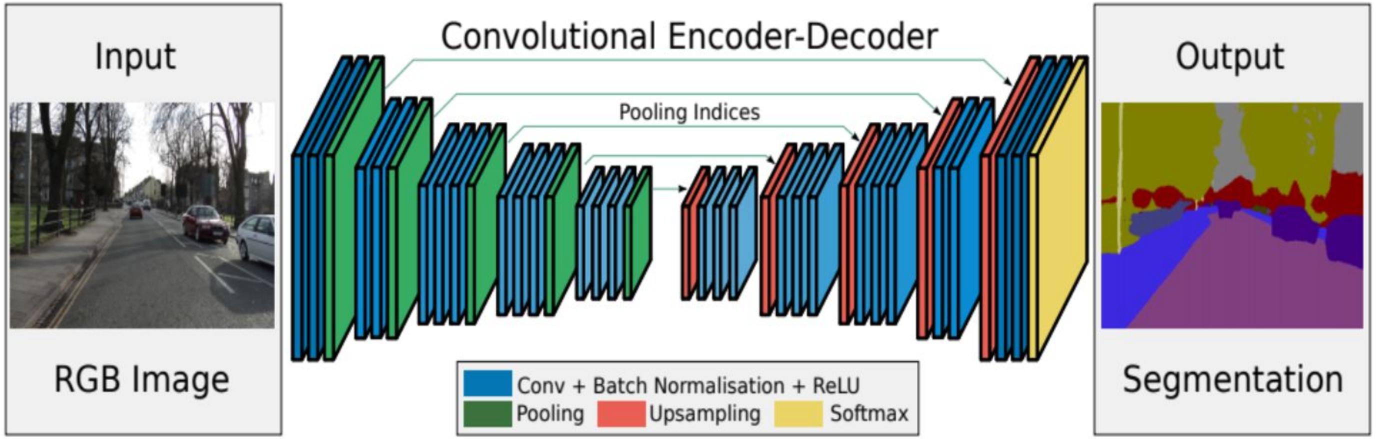
Create one hot encoding of class labels for target.

21 classes including background.



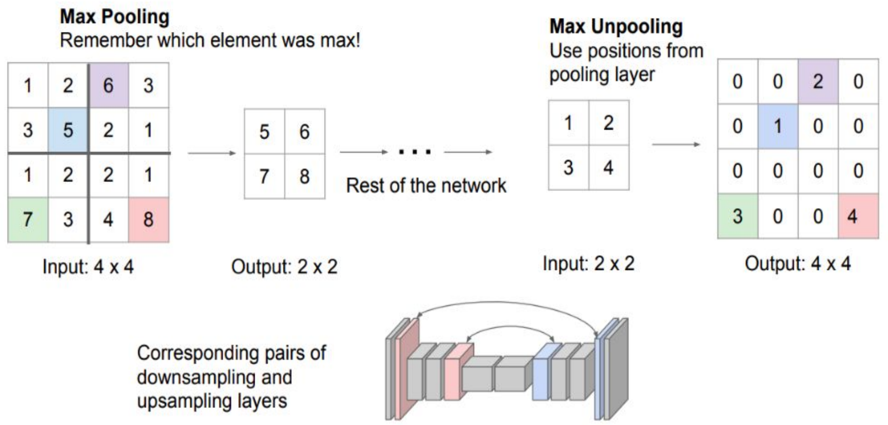
Segmentation Architecture and Base model

- UNET, PSPNET, FCN, SegNet etc
- Base models – VGG16, ResNet etc
- Using SegNet architecture with VGG16 as base model



SegNet

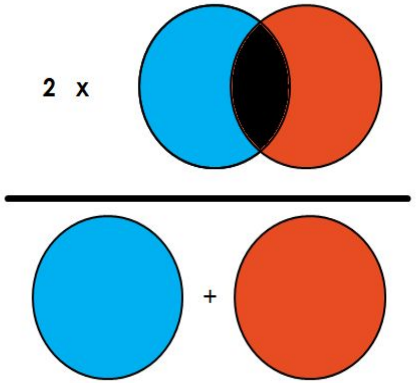
- Segnet is simple and efficient for pixel wise sematic segmentation.
- 13 convolution layers in VGG16 network for object classification as encoder.
- Each encoder layer corresponds to a decoder layer, so 13 decoder layers.



Metrics

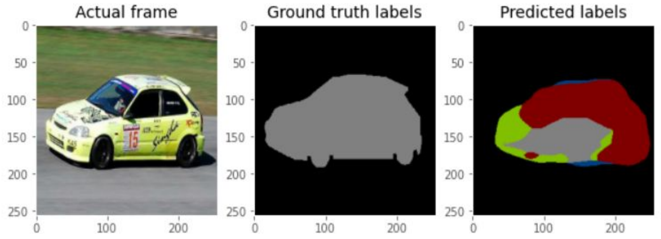
- Adam optimizer
- Categorical Cross entropy loss
- Dice coefficient metrics
- Early stopping with patience=15

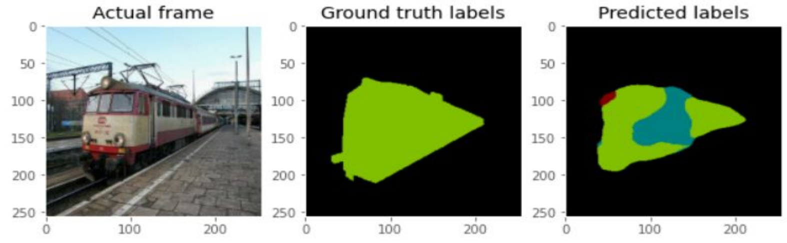
- Dice Coefficient

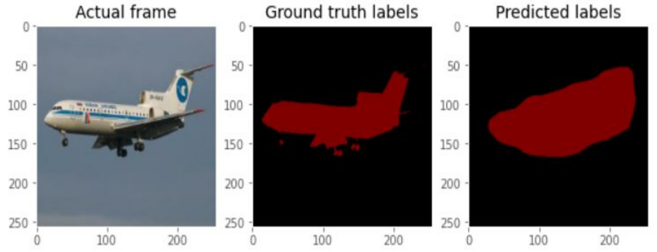


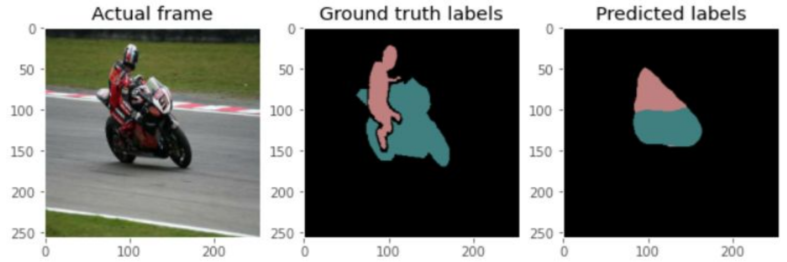
- Training set: 0.8
- Validation set: 0.7

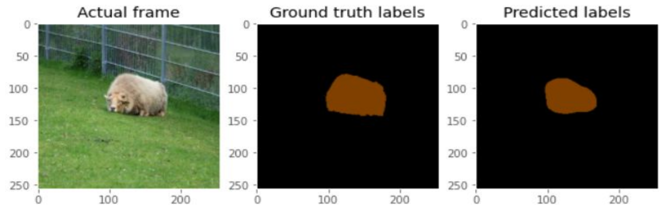
Results of model developed from scratch

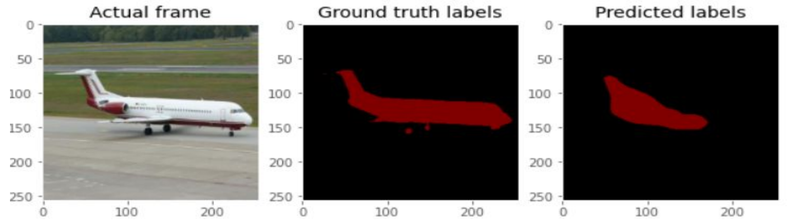






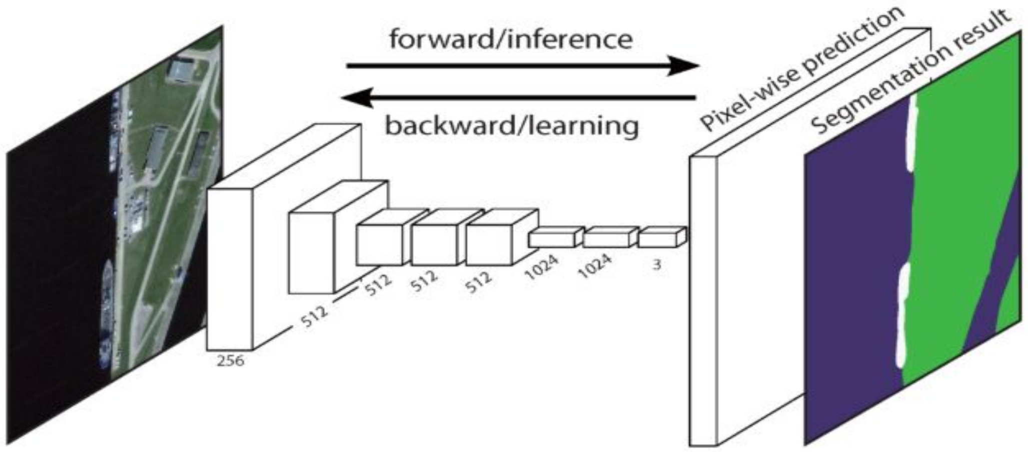




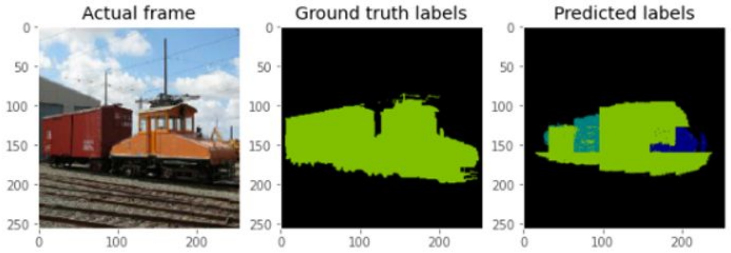


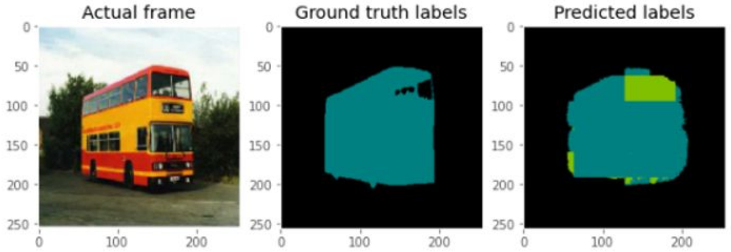
Transfer learning

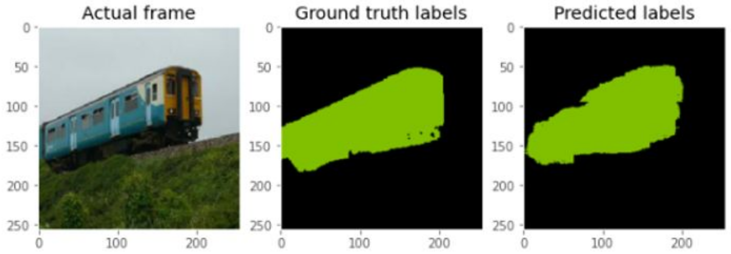
- FCN with VGG 16 pretrained weights
- Reuse convolution layers of pretrained models in encoder
- Dice coefficient improved to 0.9 and 0.8 respectively for training and validation set.

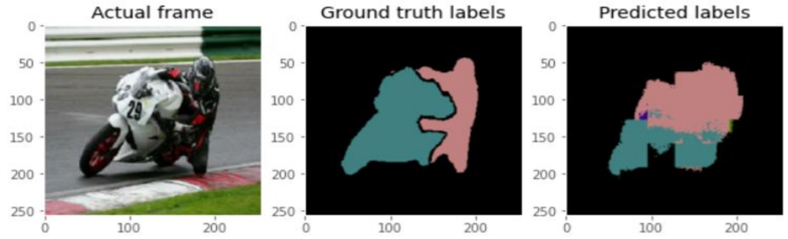


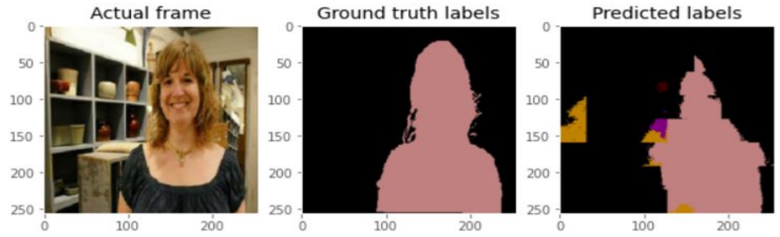
Results

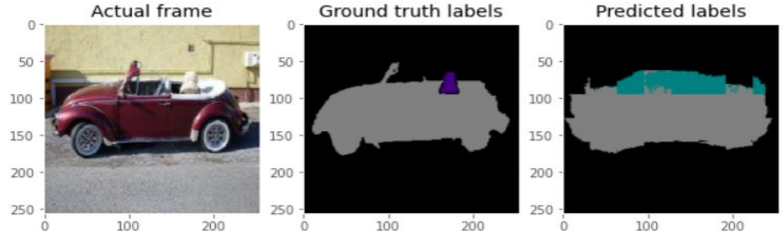

Actual frame
Ground truth labels
Predicted labels


Actual frame
Ground truth labels
Predicted labels


Actual frame
Ground truth labels
Predicted labels

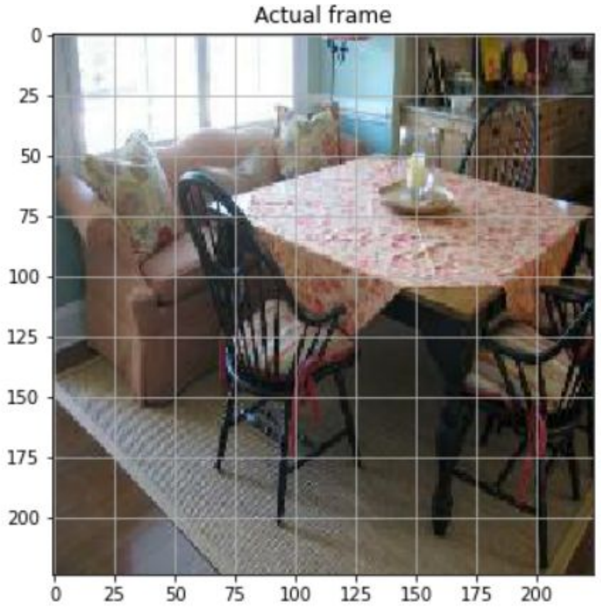

Actual frame
Ground truth labels
Predicted labels

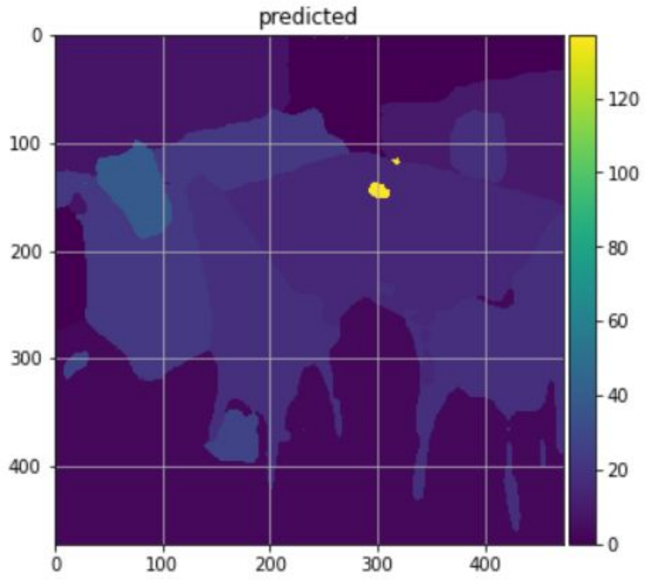

Actual frame
Ground truth labels
Predicted labels


Actual frame
Ground truth labels
Predicted labels

Transfer learning

- Prediction using pretrained segmentation model on ADE20k dataset





Part Two: Adverserial Examples

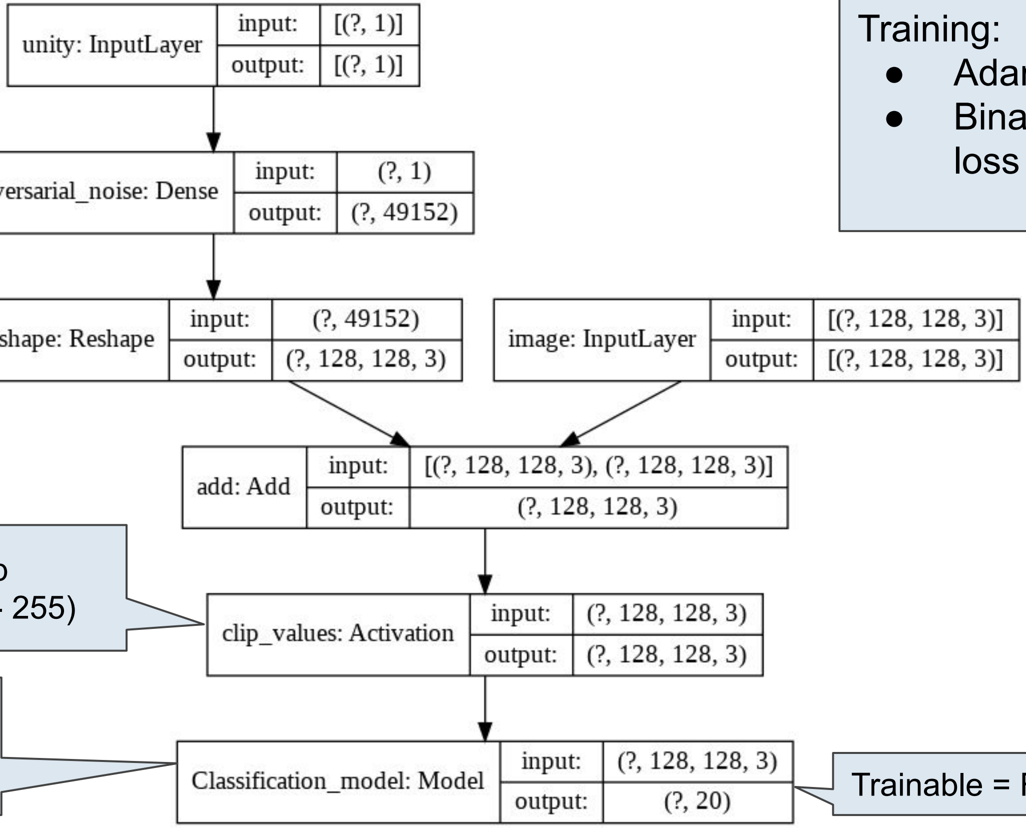
- White-box Evasion Attack
  - = Attack on testing phase
  - Full knowledge of model
  - Perturbation selection
  - Exploit gaps in Decision Boundaries learned a priori
- How?
  - Preprocess data
  - Prepend subnetwork to classifier
  - Learn perturbation vector δ_X so that:

$$\arg\min_{\delta_X} ||\delta_X|| \text{ s.t. } \mathbf{F(X + \delta_X) = Y}$$, with Y the desired misprediction

Preprocess Data: get_relevant_data()

1) Specify X (original) and Y (target) classes
2) Select image set I, where X but not Y is present
3) $\forall$ i $\in$ I, create one-hot encoded label vector with label Y
4) split data:
    a) x_train, y_train (70 %)
    b) x_validate, y_validate (15 %)
    c) x_test, y_test (15 %)

System:



L1/L2 regularization + ReLu activation

Custom Clip function (0 - 255)

Finetuned VGG / Own model from scratch

Trainable = False

Training:

- Adam optimizer
- Binary Crossentropy loss

KU LEUVEN

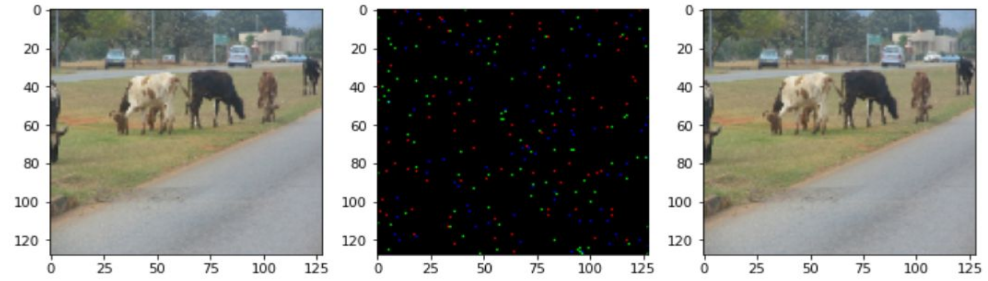
Adversarial Attacks using Train Data

The results received were very much dependent on the type of labels being misclassified

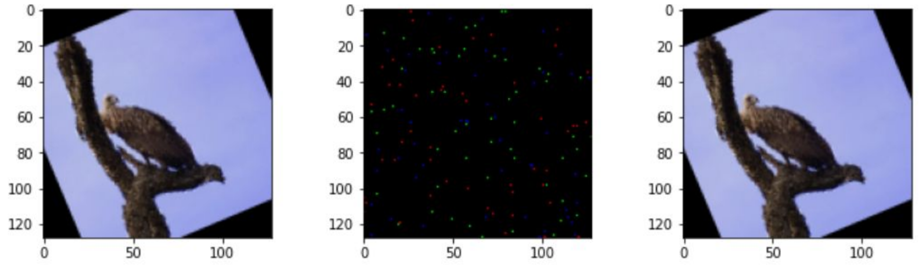
- Cow -> Person = 70 % Succesfully misclassified
- Bird -> Person = 65% Succesfully misclassified

Example: Here you can see an original image on the left and then the noise added to it resulting it a final image which is very similar to the original one and then the corresponding misclassified labels

```
PREDICTED LABELS:
['person', 'car']
################################
```



```
PREDICTED LABELS:
['person']
################################
```



Self-made Classifier vs VGG Model

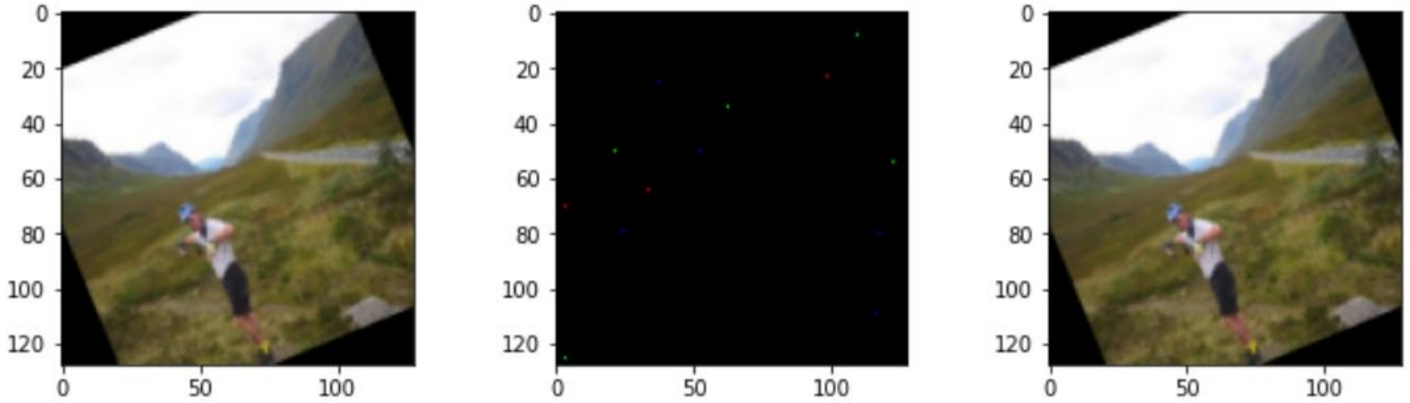
We observe that the misprediction success drastically dropped for the same data and classes from 70 % to 6 % for the self-made classification model and VGG fine-tuned model respectively.

Adversarial Attacks using Test Data

The results obtained from test data were less accurate as the test data has no initial labels so we have to assign these initial labels ourselves.

- Person -> Bird = 10% misclassified

```
PREDICTED LABELS:
['bird']
################################
```



Concluding Discussions

Classification:

- Higher accuracy due to multi-class classification
- Model predicts either too many or too little classes but still a solid 94%
- Fine-tuned VGG results is much more accurate at 97%

Segmentation:

- Dice coefficient of .87 (images with single class objects like car,train,aeroplane,sheep,person are predicted accurately)
- Improved by doing transfer learning

Adversarial Attacks:

- Succesfully managed to mispredict one label “cow” as “person”
- Not robust enough for all combinations of labels

Next Steps

Our model performs quite well on most classification and segmentation problems but given our take on these adversarial attacks and the limited amount of training set we propose before we launch this into a real-world setting we make sure we get enough training and test data so that our classification models can learn more and not only be more efficient but more resilient to adversarial attacks. Given that most likely with more sophisticated adversarial attacks, performance distortion may become even worse.