Semantic Segmentation for Self-Driving Cars using U-net.

Miah, MD. Jihad (17-35375-3), Prosad, Bilash (17-35349-3), Arnob, MD. Abrer Shariar (17-35408-3) Jumo, Kazi Shanzida (17-34318-1)

Department of Computer Science, American International University-Bangladesh

Abstract

Self-driving cars are composed of three navigation subsystems: finding lanes, understanding the city landscape, and geographic positioning. This project introduces the semantic segmentation technology for understanding urban scenes that has a variety of implementation methods in recent years and proposes a new method for fast and accurate semantic segmentation. The architecture of the model was developed using Unet. This method of implementation provides an accuracy of about 99.42%.[1]

Keywords: Deep Learning, Computer Vision, Object detection, U-net, Segmentation, Semantic Segmentation, Machine Learning

Introduction

The purpose of this project is to develop a model with the highest semantic segmentation accuracy for autonomous vehicles. The data set we used contains data images and labeled semantic segments from CARLA, a self-driving car simulator. Udacity Challenge. This data set can be used to train machine learning algorithms to determine the semantic segmentation of cars, roads, etc. in images. The data contains 5 groups of 1000 images and corresponding labels. The data set contains RGB sets and corresponding semantic segments. ... This data set is trained in our model based on the Unet architecture. Over the years, several deep neural network architectures for semantic segmentation have been discovered, but in addition to efficiency, running time is the most important factor. Emphasize that real-time performance is often required because semantic markup is usually only used as a preliminary step for other time-sensitive tasks. With this in mind, we decided to adopt the UNet architecture.[2]

Literature Review

Semantic segmentation, also known as pixel-based classification, is an important task. We assign each pixel in the image to a specific class. In GIS, segmentation can be used to classify land cover or extract streets or buildings from satellite imagery. The purpose of semantic segmentation is the same as that of traditional remote sensing image classification, which is usually achieved by traditional machine learning methods such as random forest and maximum likelihood classifier. Like image classification, semantic segmentation has two inputs: a bitmap containing multiple bands and a label image containing a label for each pixel.

Unet was invented and first used for biomedical image segmentation. Its architecture can be broadly seen as an encoder network and then a decoder network. Compared with classification, the bottom line of the deep web is the only important thing. Semantic segmentation requires not only pixel-level differentiation, but also a mechanism to project the differentiation obtained at different stages of the encoder into the pixel space. The encoder is the first half in the architecture diagram (Figure 2). It usually is a pre-trained classification network like VGG/ ResNet where you apply convolution blocks followed by a max-pool down-sampling to encode the input image into feature representations at multiple different levels.[3]

• The decoder is the second half of the architecture. The purpose is to semantically project the discriminative parameters (lowest resolution) learned from the internal encoder (highest resolution) to facilitate dense classification. Regular folding operation.

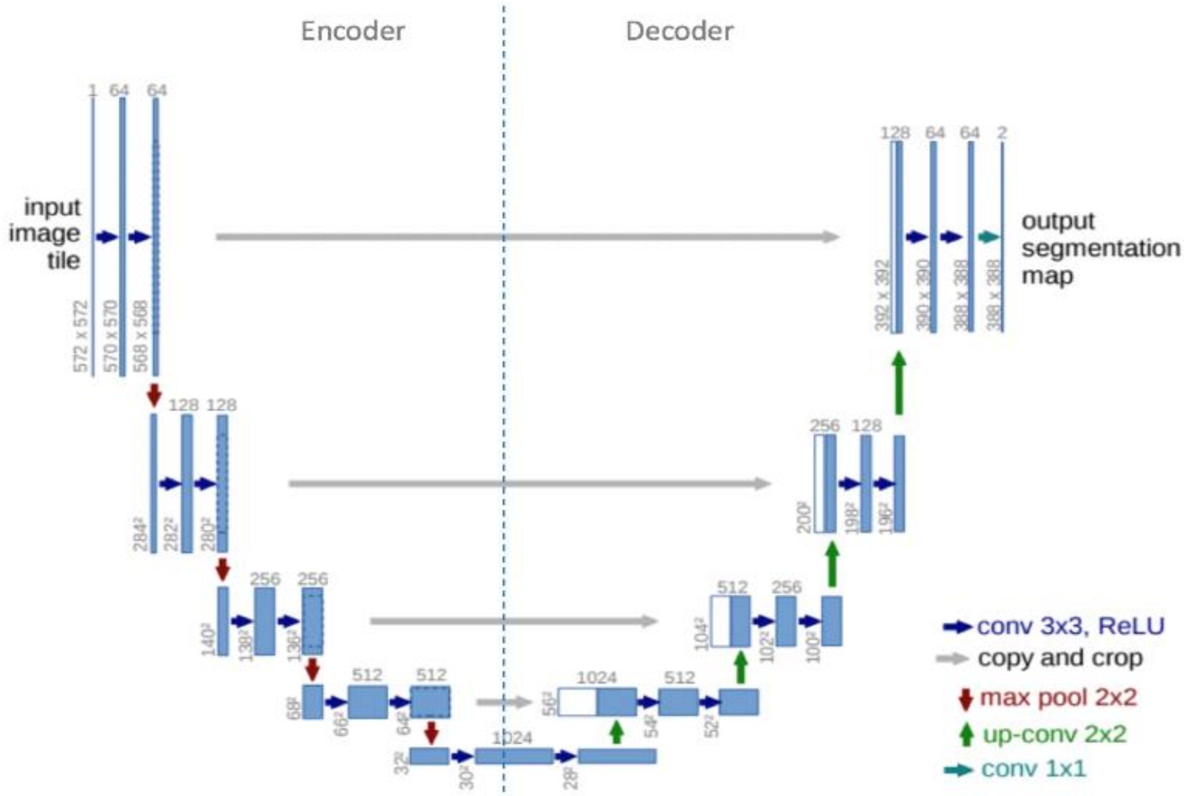


Figure 2. U-net architecture. Blue boxes represent multi-channel feature maps, while boxes represent copied feature maps. The arrows of different colors represent different operations.

Up-Sampling on CNN may be new to those familiar with object detection and classification architectures, but the idea is very simple. Intuition tells us that we want to restore the compressed feature map to the original size of the input image, thereby increasing the size of the object. Up-sampling is also called transposed convolution, bottom-up convolution, or deconvolution. There are several forms of higher sampling, such as B. nearest neighbor, bilinear interpolation, and convolution from the simplest to the most complex. More information can be found in the "Guide to Deep Learning Convolution Arithmetic" mentioned at the beginning.

In particular, we want to enlarge it to the same size as the corresponding link block on the left. You can see the gray and green arrows where we merge the two function cards. The main contribution of UNet in this regard is that when we up-sample the network, we also combine the higher resolution feature map of the encoder network with the up-sampling function to better understand the representation based on the following convolution. Since up-sampling rarely occurs, we need a good preview of the previous steps to better represent the position.

Therefore, unlike classification, the only important point is the final result of a very deep network. Semantic segmentation requires not only pixel-level differentiation, but also a mechanism for projecting the salient features obtained at different levels of the encoder. Pixel space.[4]

Building and Training the Model and Testing Accuracy

In [8]:
```
import tensorflow as tf
gpu_device = tf.config.experimental.list_physical_devices('GPU')
print(f"Number of GPU = {len(gpu_device)}")
tf.config.experimental.set_memory_growth(gpu_device[0], True)
```

```
Number of GPU = 1
```

In [9]:
```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
from sklearn.utils import shuffle
%matplotlib inline
```

Data Preprocessing

In [10]:
```python
image_paths = []
imseg_paths = []

for x in ['dataA', 'dataB', 'dataC', 'dataD', 'dataE']:
    image_path_dir = 'content/' + x + '/' + x + '/' + 'CameraRGB'
    imseg_path_dir = 'content/' + x + '/' + x + '/' + 'CameraSeg'

    for dirname, _, filenames in os.walk(image_path_dir):
        for filename in filenames:
            image_path = image_path_dir + '/' + filename
            image_paths.append(image_path)
            imseg_path = imseg_path_dir + '/' + filename
            imseg_paths.append(imseg_path)


num_images = len(image_paths)
print("Total number of images = ", num_images)
```

```
Total number of images =  5000
```

In [11]:
```python
def read_image(path):
    image = cv2.imread(path)
    image = image_resize(image)
    return np.array(image)

def read_imseg(path):
    imseg = np.array(cv2.imread(path))
    imseg = image_resize(imseg)
    imseg = np.array([max(imseg[i, j]) for i in range(imseg.shape[0]) for j in range(imseg.shape[1])]).reshape(imseg.shape[0], imseg.shape[1])
    return imseg

def image_resize(image):
    height, width = (224, 224)
    return np.array(cv2.resize(image, (width, height), cv2.INTER_AREA))

def imseg2roadseg(imseg):
    height, width = imseg.shape
    imseg_road = np.zeros((height, width, 1), dtype=np.int8)
    imseg_road[np.where(imseg==7)[0], np.where(imseg==7)[1]] = 1
    return np.array(imseg_road)

def pipeline(X_path, y_path):
    image_BGR = read_image(X_path)
    imseg = read_imseg(y_path)
    imseg_road = imseg2roadseg(imseg)
    return image_BGR, imseg_road
```

In [12]:
```python
def read_data(image_paths,imseg_paths):
    height, width = (224, 224)
    images = np.zeros((len(image_paths), height, width, 3), dtype=np.int16)
    imsegs_road = np.zeros((len(image_paths), height, width, 1), dtype=np.int8)
    for index in tqdm(range(len(image_paths))):
        X_path, y_path = image_paths[index], imseg_paths[index]
        images[index], imsegs_road[index] = pipeline(X_path, y_path)
    return images, imsegs_road

X, y = read_data(image_paths,imseg_paths)
```

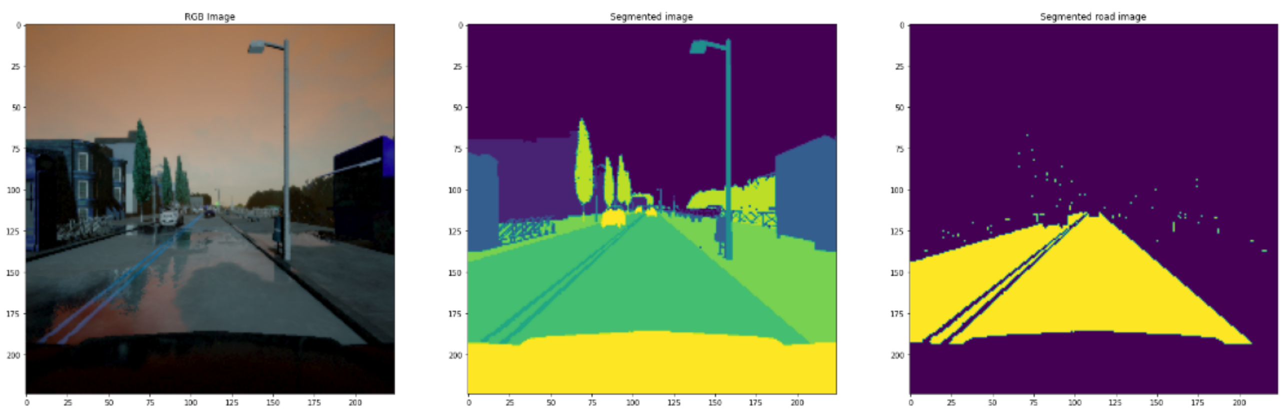
```
100%|██████████| 5000/5000 [08:09<00:00, 10.21it/s]
```

Sample Image Visualization

In [13]:
```
from random import randint
index = randint(0,len(image_paths))
height, width = (224, 224)
segment = 7

image = read_image(image_paths[index])
imseg = read_imseg(imseg_paths[index])
imseg_road = imseg2roadseg(imseg)

print(image.shape)
print(imseg.shape)
print(imseg_road.shape)

fig, axes = plt.subplots(1, 3, figsize=(30,20))
axes[0].imshow(image)
axes[0].set_title('RGB Image')
axes[1].imshow(imseg)
axes[1].set_title('Segmented image')
axes[2].imshow(imseg_road.reshape(height,width))
axes[2].set_title('Segmented road image')
```

```
(224, 224, 3)
(224, 224)
(224, 224, 1)
```

Out[13]: `Text(0.5, 1.0, 'Segmented road image')`



In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, shuffle =True,  random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, shuffle =True,  random_state=42)

print("Training images = ", len(X_train), "Validation images= ", len(X_val), "Test images = ", len(X_test))
```

```
Training images =  3825 Validation images=  675 Test images =  500
```

Implementation of Unet, using Keras

In [15]:
```
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dropout, Lambda
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, BatchNormalization
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
```

In [16]:
```
def encoder_block(input, filters):
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (input)
  c = BatchNormalization()(c)
  c = Dropout(0.1) (c)
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c)
  c = BatchNormalization()(c)
  p = MaxPooling2D((2, 2)) (c)
  return p, c
```

In [17]:
```
def bridge_block(input, filters):
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (input)
  c = BatchNormalization()(c)
  c = Dropout(0.3) (c)
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c)
  c = BatchNormalization()(c)
  return c
```

In [18]:
```
def decoder_block(input, filters, skip_features):
  u = Conv2DTranspose(filters/2, (2, 2), strides=(2, 2), padding='same') (input)
  u = concatenate([u, skip_features])
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (u)
  c = BatchNormalization()(c)
  c = Dropout(0.2) (c)
  c = Conv2D(filters, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c)
  c = BatchNormalization()(c)
  return c
```

In [19]:
```
# Build U-Net model
def unet_model(image_shape):
  inputs = Input((image_shape))
  s = Lambda(lambda x: x / 255) (inputs)

  # Encoder
  p1, c1 = encoder_block(s, 32)
  p2, c2 = encoder_block(p1, 64)
  p3, c3 = encoder_block(p2, 128)
  p4, c4 = encoder_block(p3, 256)

  # Bridge
  c5 = bridge_block(p4, 512)

  # Decoder
  c6 = decoder_block(c5, 256, c4)
  c7 = decoder_block(c6, 128, c3)
  c8 = decoder_block(c7, 64, c2)
  c9 = decoder_block(c8, 32, c1)

  outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
  model = Model(inputs=[inputs], outputs=[outputs])
  return model
```

In [20]:
```python
from tensorflow.keras.models import Model
image_shape = (224, 224, 3)
num_classes = 1
model = unet_model(image_shape)

#compile the model
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 224, 224, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 224, 224, 32) 896         lambda[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 224, 224, 32) 128         conv2d[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 224, 224, 32) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 224, 224, 32) 9248        dropout[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 224, 224, 32) 128         conv2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 112, 112, 32) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 112, 112, 64) 18496       max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 112, 112, 64) 256         conv2d_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 112, 112, 64) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 112, 112, 64) 36928       dropout_1[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 112, 112, 64) 256         conv2d_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 56, 56, 64)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 56, 56, 128)  73856       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 56, 56, 128)  512         conv2d_4[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 56, 56, 128)  0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 56, 56, 128)  147584      dropout_2[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 56, 56, 128)  512         conv2d_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 28, 28, 128)  0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 28, 28, 256)  295168      max_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 28, 28, 256)  1024        conv2d_6[0][0]
__________________________________________________________________________________________________
```

```
dropout_3 (Dropout)            (None, 28, 28, 256)  0           batch_normalization_6[0][0]
________________________________________________________________________________
conv2d_7 (Conv2D)              (None, 28, 28, 256)  590080      dropout_3[0][0]
________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 28, 28, 256)  1024       conv2d_7[0][0]
________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 14, 14, 256)  0          batch_normalization_7[0][0]
________________________________________________________________________________
conv2d_8 (Conv2D)              (None, 14, 14, 512)  1180160     max_pooling2d_3[0][0]
________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 14, 14, 512)  2048       conv2d_8[0][0]
________________________________________________________________________________
dropout_4 (Dropout)            (None, 14, 14, 512)  0           batch_normalization_8[0][0]
________________________________________________________________________________
conv2d_9 (Conv2D)              (None, 14, 14, 512)  2359808     dropout_4[0][0]
________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 14, 14, 512)  2048       conv2d_9[0][0]
________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 28, 28, 128)  262272     batch_normalization_9[0][0]
________________________________________________________________________________
concatenate (Concatenate)      (None, 28, 28, 384)  0           conv2d_transpose[0][0]
                                                                 batch_normalization_7[0][0]
________________________________________________________________________________
conv2d_10 (Conv2D)             (None, 28, 28, 256)  884992      concatenate[0][0]
________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 28, 28, 256)  1024       conv2d_10[0][0]
________________________________________________________________________________
dropout_5 (Dropout)            (None, 28, 28, 256)  0           batch_normalization_10[0][0]
________________________________________________________________________________
conv2d_11 (Conv2D)             (None, 28, 28, 256)  590080      dropout_5[0][0]
________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 28, 28, 256)  1024       conv2d_11[0][0]
________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 56, 56, 64)   65600      batch_normalization_11[0][0]
________________________________________________________________________________
concatenate_1 (Concatenate)    (None, 56, 56, 192)  0           conv2d_transpose_1[0][0]
                                                                 batch_normalization_5[0][0]
________________________________________________________________________________
conv2d_12 (Conv2D)             (None, 56, 56, 128)  221312      concatenate_1[0][0]
________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 56, 56, 128)  512        conv2d_12[0][0]
________________________________________________________________________________
dropout_6 (Dropout)            (None, 56, 56, 128)  0           batch_normalization_12[0][0]
________________________________________________________________________________
conv2d_13 (Conv2D)             (None, 56, 56, 128)  147584      dropout_6[0][0]
________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 56, 56, 128)  512        conv2d_13[0][0]
________________________________________________________________________________
conv2d_transpose_2 (Conv2DTrans (None, 112, 112, 32) 16416      batch_normalization_13[0][0]
________________________________________________________________________________
concatenate_2 (Concatenate)    (None, 112, 112, 96) 0           conv2d_transpose_2[0][0]
                                                                 batch_normalization_3[0][0]
________________________________________________________________________________
conv2d_14 (Conv2D)             (None, 112, 112, 64) 55360       concatenate_2[0][0]
________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 112, 112, 64) 256        conv2d_14[0][0]
________________________________________________________________________________
dropout_7 (Dropout)            (None, 112, 112, 64) 0           batch_normalization_14[0][0]
________________________________________________________________________________
conv2d_15 (Conv2D)             (None, 112, 112, 64) 36928       dropout_7[0][0]
________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 112, 112, 64) 256        conv2d_15[0][0]
________________________________________________________________________________
conv2d_transpose_3 (Conv2DTrans (None, 224, 224, 16) 4112       batch_normalization_15[0][0]
________________________________________________________________________________
concatenate_3 (Concatenate)    (None, 224, 224, 48) 0           conv2d_transpose_3[0][0]
                                                                 batch_normalization_1[0][0]
________________________________________________________________________________
conv2d_16 (Conv2D)             (None, 224, 224, 32) 13856       concatenate_3[0][0]
________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 224, 224, 32) 128        conv2d_16[0][0]
________________________________________________________________________________
dropout_8 (Dropout)            (None, 224, 224, 32) 0           batch_normalization_16[0][0]
________________________________________________________________________________
conv2d_17 (Conv2D)             (None, 224, 224, 32) 9248        dropout_8[0][0]
________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 224, 224, 32) 128        conv2d_17[0][0]
________________________________________________________________________________
conv2d_18 (Conv2D)             (None, 224, 224, 1)  33          batch_normalization_17[0][0]
================================================================================
Total params: 7,031,793
Trainable params: 7,025,905
Non-trainable params: 5,888
________________________________________________________________________________
```

In [21]:

```
earlystopper = EarlyStopping(patience=5, verbose=1)
checkpointer = ModelCheckpoint('unet_model_road_seg_checkpoint.h5', verbose=1, save_best_only=True)
```

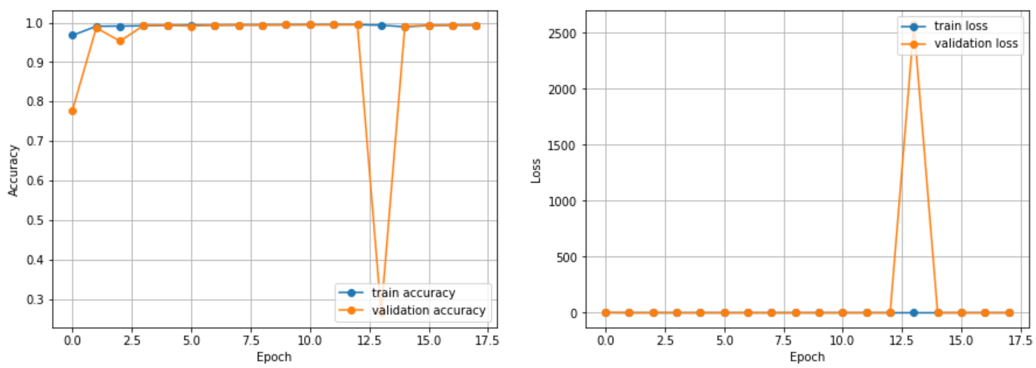
Training Unet Model

In [22]:
```
results = model.fit(x=X_train, y=y_train, validation_split=0.1, batch_size=16, epochs=50, verbose=1, callbacks=[earlystopper, checkpo
inter])
```

```
Epoch 1/50
216/216 [==============================] - 93s 260ms/step - loss: 0.1162 - accuracy: 0.9676 - val_loss: 2.8118 - val_accuracy: 0.7783

Epoch 00001: val_loss improved from inf to 2.81176, saving model to unet_model_road_seg_checkpoint.h5
Epoch 2/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0377 - accuracy: 0.9905 - val_loss: 0.0537 - val_accuracy: 0.9867

Epoch 00002: val_loss improved from 2.81176 to 0.05374, saving model to unet_model_road_seg_checkpoint.h5
Epoch 3/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0340 - accuracy: 0.9911 - val_loss: 0.6937 - val_accuracy: 0.9529

Epoch 00003: val_loss did not improve from 0.05374
Epoch 4/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0312 - accuracy: 0.9918 - val_loss: 0.0280 - val_accuracy: 0.9926

Epoch 00004: val_loss improved from 0.05374 to 0.02798, saving model to unet_model_road_seg_checkpoint.h5
Epoch 5/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0268 - accuracy: 0.9930 - val_loss: 0.0256 - val_accuracy: 0.9933

Epoch 00005: val_loss improved from 0.02798 to 0.02562, saving model to unet_model_road_seg_checkpoint.h5
Epoch 6/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0266 - accuracy: 0.9930 - val_loss: 0.0323 - val_accuracy: 0.9912

Epoch 00006: val_loss did not improve from 0.02562
Epoch 7/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0246 - accuracy: 0.9935 - val_loss: 0.0233 - val_accuracy: 0.9938

Epoch 00007: val_loss improved from 0.02562 to 0.02327, saving model to unet_model_road_seg_checkpoint.h5
Epoch 8/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0232 - accuracy: 0.9939 - val_loss: 0.0228 - val_accuracy: 0.9939

Epoch 00008: val_loss improved from 0.02327 to 0.02279, saving model to unet_model_road_seg_checkpoint.h5
Epoch 9/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0227 - accuracy: 0.9940 - val_loss: 0.0222 - val_accuracy: 0.9941

Epoch 00009: val_loss improved from 0.02279 to 0.02220, saving model to unet_model_road_seg_checkpoint.h5
Epoch 10/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0217 - accuracy: 0.9942 - val_loss: 0.0213 - val_accuracy: 0.9943

Epoch 00010: val_loss improved from 0.02220 to 0.02126, saving model to unet_model_road_seg_checkpoint.h5
Epoch 11/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0212 - accuracy: 0.9943 - val_loss: 0.0206 - val_accuracy: 0.9944

Epoch 00011: val_loss improved from 0.02126 to 0.02061, saving model to unet_model_road_seg_checkpoint.h5
Epoch 12/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0206 - accuracy: 0.9945 - val_loss: 0.0202 - val_accuracy: 0.9945

Epoch 00012: val_loss improved from 0.02061 to 0.02021, saving model to unet_model_road_seg_checkpoint.h5
Epoch 13/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0203 - accuracy: 0.9945 - val_loss: 0.0199 - val_accuracy: 0.9946

Epoch 00013: val_loss improved from 0.02021 to 0.01985, saving model to unet_model_road_seg_checkpoint.h5
Epoch 14/50
216/216 [==============================] - 55s 257ms/step - loss: 0.0241 - accuracy: 0.9933 - val_loss: 2573.3591 - val_accuracy: 0.2
649

Epoch 00014: val_loss did not improve from 0.01985
Epoch 15/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0390 - accuracy: 0.9892 - val_loss: 0.0315 - val_accuracy: 0.9912

Epoch 00015: val_loss did not improve from 0.01985
Epoch 16/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0275 - accuracy: 0.9926 - val_loss: 0.0259 - val_accuracy: 0.9931

Epoch 00016: val_loss did not improve from 0.01985
Epoch 17/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0248 - accuracy: 0.9933 - val_loss: 0.0237 - val_accuracy: 0.9936

Epoch 00017: val_loss did not improve from 0.01985
Epoch 18/50
216/216 [==============================] - 55s 256ms/step - loss: 0.0235 - accuracy: 0.9937 - val_loss: 0.0223 - val_accuracy: 0.9940

Epoch 00018: val_loss did not improve from 0.01985
Epoch 00018: early stopping
```

Visualizing Training Results

In [23]:
```python
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(results.history['accuracy'], 'o-', label='train accuracy')
plt.plot(results.history['val_accuracy'], 'o-', label = 'validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend(loc='lower right')

plt.subplot(1,2,2)
plt.plot(results.history['loss'], 'o-', label='train loss')
plt.plot(results.history['val_loss'], 'o-', label='validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend(loc='upper right')

plt.show()
```



Testing Unet Model

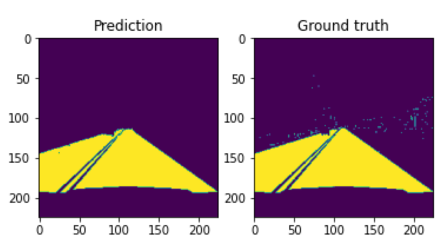
In [24]:
```python
score = model.evaluate(X_test, y_test, verbose=2, batch_size=16)
```

```
32/32 - 3s - loss: 0.0214 - accuracy: 0.9942
```

Predicting An Image

In [25]:
```python
NUMBER = 0

X_test_cast = X_test.astype(float)
y_pred = model.predict(X_test_cast, batch_size=16)
my_preds = y_pred[0].flatten()
my_preds = np.array([1 if i >= 0.5 else 0 for i in my_preds])
fig, ax = plt.subplots(nrows=1, ncols=2)
ax[0].imshow(my_preds.reshape(224, 224))
ax[0].set_title('Prediction')
ax[1].imshow(y_test[NUMBER].reshape(224, 224))
ax[1].set_title('Ground truth')
```

Out[25]:
```
Text(0.5, 1.0, 'Ground truth')
```



Discussion

The sensitivity of the project is necessary to have good concomitant precision and precision and high fps. However, since this project has not been applied for a period of time and completely relies on the static data set available on the Internet, the model achieved an accuracy of 99.42%, a loss of 2.14%. This shows that the model is economical, which is needed for this type of work. [5]

Conclusion

In summary, it should be noted that the model we created can be considered valid and implemented in a real-time scene for segmentation. The accuracy rate is very high, which is a good indicator, it will not deteriorate too much in real-time scenes. In fact, this project can be considered successful and effective.

References

[1]*https://smartlabai.medium.com/deep-learning-based-semantic-segmentation-in-simulation-and-real-world-for-autonomous-vehicles-e7fe25cef816*

[2] *https://blog.jovian.ai/semantic-segmentation-in-self-driving-cars-3cb89aa08e9b*

[3] *https://arxiv.org/pdf/1505.04597v1.pdf*

[4] *https://analyticsindiamag.com/my-experiment-with-unet-building-an-image-segmentation-model/*

[5] *https://developers.arcgis.com/python/guide/how-unet-works/*