CamVid

Данные можно скачать по ссылке.

CamVid (Cambridge-driving Labeled Video Database) - это база данных для определения дорожно-транспортных происшествий, которая первоначально была записана в виде пяти видеорядов камерой с разрешением 960 × 720, установленной на приборной панели автомобиля. Эти последовательности были отобраны (четыре из них со скоростью 1 кадр в секунду и одна со скоростью 15 кадров в секунду), в результате чего получилось 701 кадр. Эти кадры были вручную аннотированы 32 классами, а именно: пустота, здание, стена, дерево, растительность, забор, тротуар, парковочный блок, колонна / столб, дорожный конус, мост, знак, различный текст, светофор, небо, туннель, арка, дорога, обочина, разметка полосы движения (для вождения), разметка полосы движения (для не вождения), животное, пешеход, ребенок, тележка с багажом, велосипедист, мотоцикл, автомобиль, внедорожник /пикап /грузовик, грузовик/автобус, поезд и другой движущийся объект.

Датасет видео с метками CamVid, созданная в Кембридже, - это первая коллекция видео с семантическими метками класса объектов в комплекте с метаданными. База данных предоставляет метки истинности, которые связывают каждый пиксель с одним из 32 семантических классов.

Применение

Датасет удовлетворяет потребность в экспериментальных данных для количественной оценки новых алгоритмов. В то время как большинство видеороликов сняты с помощью стационарных камер видеонаблюдения, наши данные были сняты с точки зрения движущегося автомобиля. Сценарий вождения увеличивает количество и неоднородность наблюдаемых классов объектов.

Еще немного о подготовке данных

Предоставляется более десяти минут видеоматериала высокого качества с частотой 30 Гц и соответствующими семантически помеченными изображениями с частотой 1 Гц и частично 15 Гц. Сначала семантическая сегментация более 700 изображений по пикселям была задана вручную, а затем проверена и подтверждена вторым лицом на предмет точности. Во-вторых, цветные видеоизображения высокого качества и большого разрешения в базе данных представляют собой ценные оцифрованные кадры увеличенной продолжительности для тех, кто интересуется сценариями вождения или эго-движением. В-третьих, мы сняли

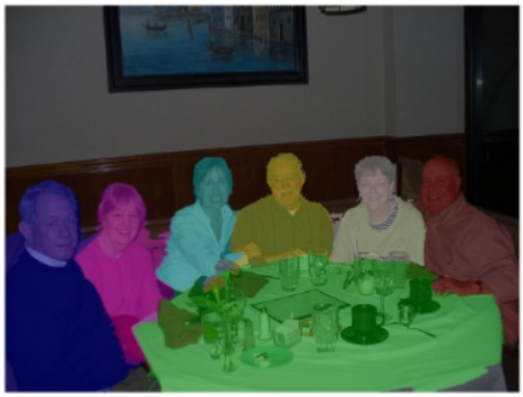
последовательности калибровки для цветовой характеристики камеры и внутренних характеристик и рассчитали позу 3D-камеры для каждого кадра в последовательностях. Наконец, в поддержку расширения этой или других баз данных мы предлагаем программное обеспечение для маркировки на заказ, которое поможет пользователям, желающим нанести точные классовые метки на другие изображения и видео. Оценена релевантность данных, измерив производительность алгоритма в каждой из трех различных областей: распознавание многоклассовых объектов, обнаружение пешеходов и распространение меток.

Semantic segmentation

Семантическая сегментация изображения означает присвоение каждому пикселю определенной метки. В этом заключается главное отличие от классификации, где всему изображению ставится в соответствие только одна метка. Сегментация работает со множеством объектов одного класса как с единым целым.

Инстанс-сегментация обрабатывает несколько объектов одного класса как различные объекты. Обычно инстанс-сегментация сложнее чем семантическая сегментация.

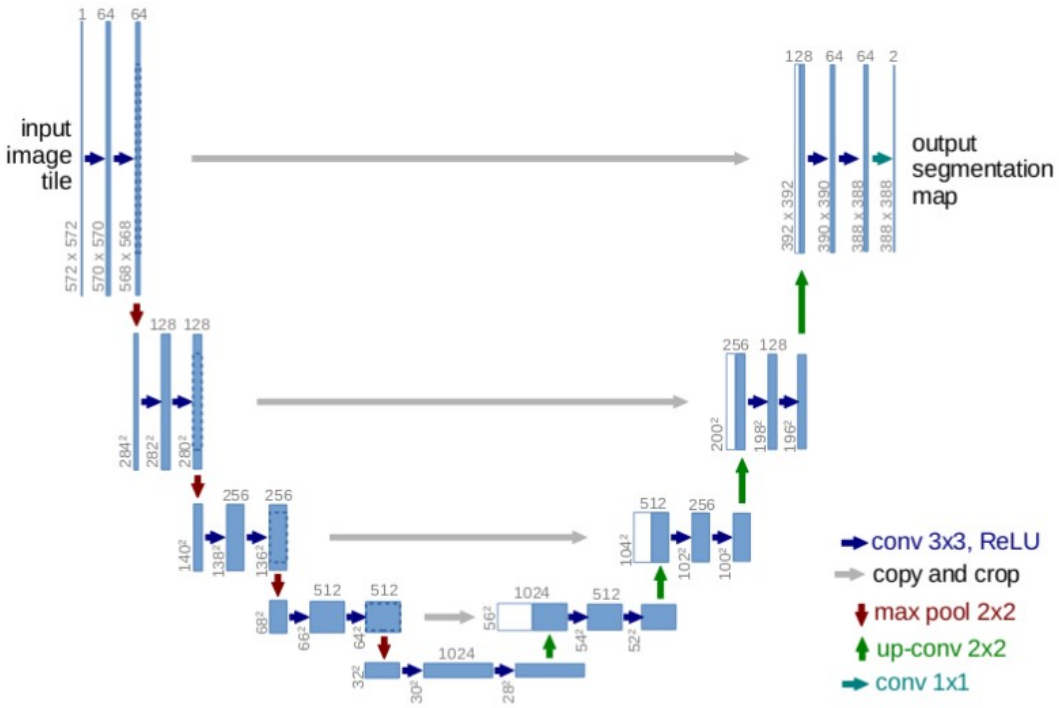
Semantic Segmentation



Instance Segmentation

U-Net

Сеть U-Net представляет из себя улучшение простой FCN архитектуры. Сеть skip-связи между выходами с блоков свертки и соответствующими им входами блока транспонированной свертки на том же уровне.



Skip-связи позволяют градиентам лучше распространяться и предоставлять информацию с различных масштабов размера изображения. Информация с больших масштабов (верхние слои) может помочь модели лучше классифицировать. В то время как информация с меньших масштабов (глубокие слои) помогает модели лучше сегментировать.

Загрузка и импортирование необходимых модулей

```
# !pip install pytorch_lightning
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import albumentations as A
from albumentations.pytorch import ToTensorV2
from albumentations.pytorch.functional import img_to_tensor
import cv2 as cv
from PIL import Image
from tqdm.notebook import tqdm
from pathlib import Path
import cv2
from dataclasses import dataclass
import pytorch_lightning as pl
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
import torchmetrics
```

```
import plotly.express as px
import torchvision
from torch.utils.tensorboard import SummaryWriter

# !unzip /content/drive/MyDrive/CW/archive.zip
```

Создание класса данных для более простой загрузки данных

```
@dataclass
class DataFrame:
    train_images = list(Path('./CamVid/train').glob('*'))
    train_labels = list(Path('./CamVid/train_labels').glob('*'))
    train_images.sort();train_labels.sort();

    classes = pd.read_csv('./CamVid/class_dict.csv')
    labels = dict(zip(range(len(classes)), classes['name'].tolist()))
    color_map =
dict(zip(range(len(classes)),classes.iloc[:,1:].values.tolist()))

    assert len(train_images) == len(train_labels)

    train_set = list(zip(train_images,train_labels))

    val_images = list(Path('./CamVid/val').glob('*'))
    val_labels = list(Path('./CamVid/val_labels').glob('*'))
    val_images.sort();val_labels.sort();

    assert len(val_images) == len(val_labels)

    val_set = list(zip(val_images,val_labels))

    test_images = list(Path('./CamVid/test').glob('*'))
    test_labels = list(Path('./CamVid/test_labels').glob('*'))
    test_images.sort();test_labels.sort();

    assert len(test_images) == len(test_labels)

    test_set = list(zip(test_images,test_labels))

df = DataFrame()
```

Классы и соответствующие им числовыеметки

```
df.labels
```

```
{0: 'Animal',
 1: 'Archway',
 2: 'Bicyclist',
 3: 'Bridge',
 4: 'Building',
 5: 'Car',
 6: 'CartLuggagePram',
```

```
 7: 'Child',
 8: 'Column_Pole',
 9: 'Fence',
 10: 'LaneMkgsDriv',
 11: 'LaneMkgsNonDriv',
 12: 'Misc_Text',
 13: 'MotorcycleScooter',
 14: 'OtherMoving',
 15: 'ParkingBlock',
 16: 'Pedestrian',
 17: 'Road',
 18: 'RoadShoulder',
 19: 'Sidewalk',
 20: 'SignSymbol',
 21: 'Sky',
 22: 'SUVPickupTruck',
 23: 'TrafficCone',
 24: 'TrafficLight',
 25: 'Train',
 26: 'Tree',
 27: 'Truck_Bus',
 28: 'Tunnel',
 29: 'VegetationMisc',
 30: 'Void',
 31: 'Wall'}
```

Классы и соответствующие им цвета в маске

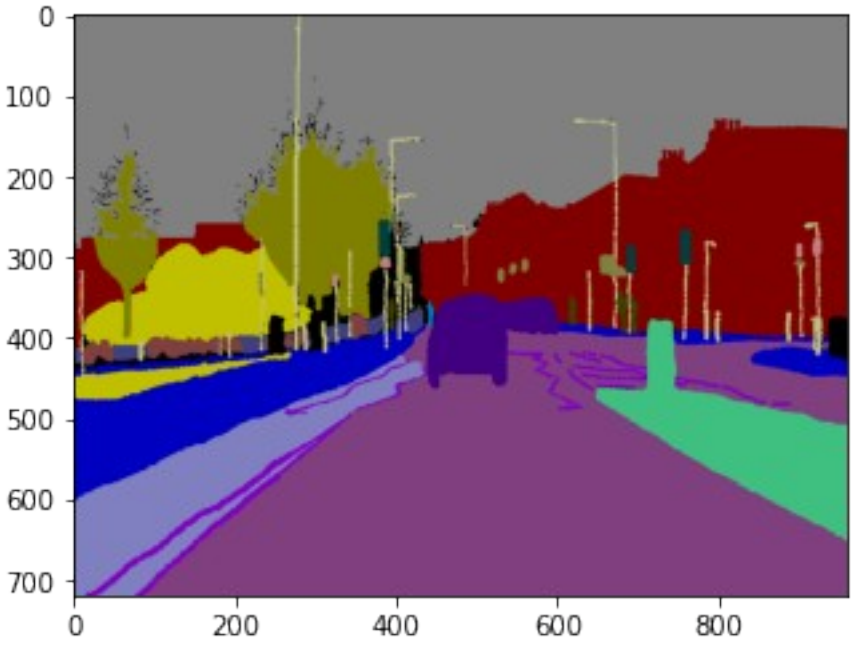
```
list(zip(df.color_map.values(),df.labels.values()))
```

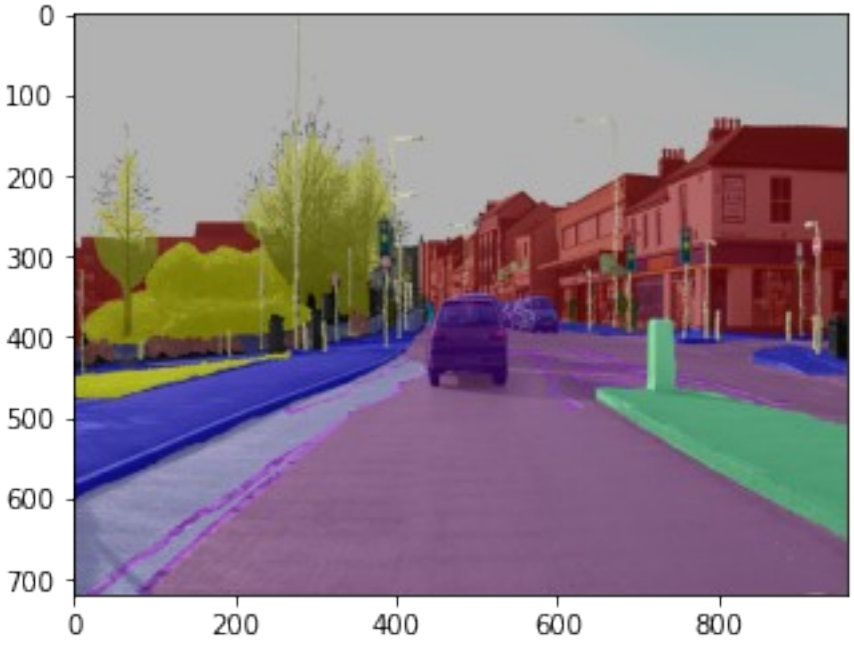
```
[([64, 128, 64], 'Animal'),
 ([192, 0, 128], 'Archway'),
 ([0, 128, 192], 'Bicyclist'),
 ([0, 128, 64], 'Bridge'),
 ([128, 0, 0], 'Building'),
 ([64, 0, 128], 'Car'),
 ([64, 0, 192], 'CartLuggagePram'),
 ([192, 128, 64], 'Child'),
 ([192, 192, 128], 'Column_Pole'),
 ([64, 64, 128], 'Fence'),
 ([128, 0, 192], 'LaneMkgsDriv'),
 ([192, 0, 64], 'LaneMkgsNonDriv'),
 ([128, 128, 64], 'Misc_Text'),
 ([192, 0, 192], 'MotorcycleScooter'),
 ([128, 64, 64], 'OtherMoving'),
 ([64, 192, 128], 'ParkingBlock'),
 ([64, 64, 0], 'Pedestrian'),
 ([128, 64, 128], 'Road'),
 ([128, 128, 192], 'RoadShoulder'),
 ([0, 0, 192], 'Sidewalk'),
 ([192, 128, 128], 'SignSymbol'),
```

```
 ([128, 128, 128], 'Sky'),
 ([64, 128, 192], 'SUVPickupTruck'),
 ([0, 0, 64], 'TrafficCone'),
 ([0, 64, 64], 'TrafficLight'),
 ([192, 64, 128], 'Train'),
 ([128, 128, 0], 'Tree'),
 ([192, 128, 192], 'Truck_Bus'),
 ([64, 0, 64], 'Tunnel'),
 ([192, 192, 0], 'VegetationMisc'),
 ([0, 0, 0], 'Void'),
 ([64, 192, 0], 'Wall')]
```

Небольшой пример данных, маски и данных с наложеной поверх них маской

```
img,mask = df.train_set[100]
plt.imshow(Image.open(img))
plt.show()
plt.imshow(Image.open(mask))
plt.show()
plt.imshow(Image.open(img))
plt.imshow(Image.open(mask),alpha=0.6)
plt.show()
```





Анимация для визуализации карты сегментации набора данных

```
from matplotlib import animation, rc
rc('animation', html='jshtml')

def create_anim(save=True):
    fig, [ax1,ax2] = plt.subplots(1,2,dpi=150)
    ax1.axis('off')
    ax2.axis('off')
    images = []
```

```
    for i in tqdm(range(100)):
        im1,im2 = df.train_set[i]
        sample = Image.open(im1).convert('RGB')
        mask = Image.open(im2).convert('RGB')
        im1 = ax1.imshow(sample, animated=True)
        im2 = ax2.imshow(mask, animated=True)
        if i==0:
            ax1.imshow(sample)
            ax2.imshow(mask)
        images.append([im1,im2])


    ani = animation.ArtistAnimation(fig, images, interval=100,
blit=True,
                                    repeat_delay=1000)
    plt.close()

    if save:
        writer = animation.PillowWriter(fps=10)

ani.save('train_sample.gif',writer=writer,savefig_kwargs={'facecolor':
'white'})

    return ani

# create_anim(save=True)
```

Создание датасета

- использование albumentations для выполнения всех преобразований на изображении и карте сегментации
- изображения, считываемые с помощью open-cv, обрабатываемые с помощью numpy
- карта сегментации RGB затем изолируется до 32 отдельных двоичных масок для каждого класса, например, 17 - это road [потребовалась вечность, чтобы разобраться в этом :/]
- конвертируем в torch tensors

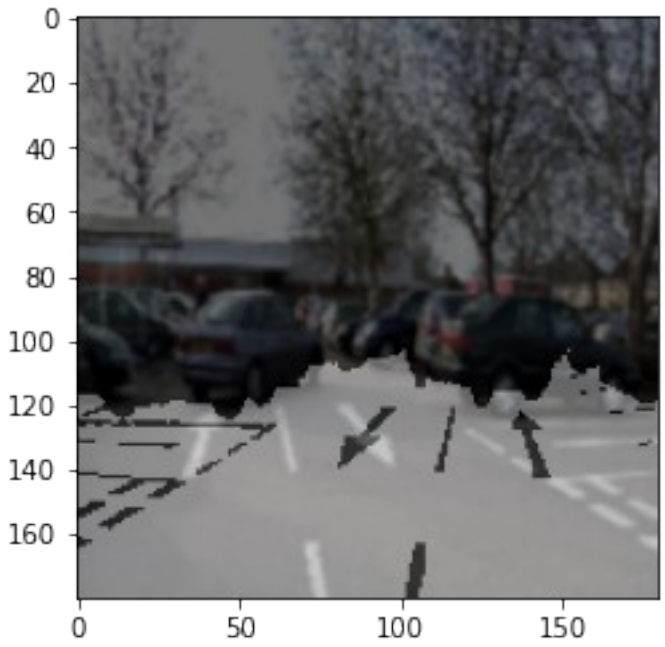
```
class CamVidDataset:
    def __init__(self, split, cmap, labels, val=False, single=None):
        self.split = split
        self.cmap = cmap
        self.labels = {y:x for x,y in labels.items()}
        self.tfms = A.Compose([
            A.CLAHE(),
            A.Resize(180,180),
            A.HorizontalFlip(),
            A.RandomBrightnessContrast(),
            A.GaussianBlur(blur_limit=(1,5))
        ])
```

```
        self.val_tfms = A.Compose([
            A.Resize(180,180)
        ])
        self.val = val
        self.single = self.labels[single] if single is not None else
None


    def __len__(self):
        return len(self.split)

    def __getitem__(self,idx):

        sample,seg = self.split[idx]

        sample = cv.imread(str(sample))
        sample = cv.cvtColor(sample,cv.COLOR_BGR2RGB)

        seg = cv.imread(str(seg))
        seg = cv.cvtColor(seg,cv.COLOR_BGR2RGB)

        if self.val:
            aug = self.val_tfms(image=sample,mask=seg)
        else:
            aug = self.tfms(image=sample,mask=seg)

        sample,seg = aug['image'],aug['mask']

        mask_shape = *seg.shape[:2],len(self.cmap)
        mask = np.zeros(mask_shape)

        for i,color in self.cmap.items():
            mask[:,:,i] = np.all(np.equal(seg,color),axis=-1)


        if self.single is not None:
            mask = mask[:,:,self.single]
            mask =
torch.tensor(mask,dtype=torch.float32).unsqueeze(2).permute(2,0,1)
        else:
            mask =
torch.tensor(mask,dtype=torch.float32).permute(2,0,1)

        sample = img_to_tensor(sample)

        return sample,mask
```

Создание даталоадеров

```
train_ds =
CamVidDataset(df.train_set,df.color_map,df.labels,single='Road')
```

```
val_ds = CamVidDataset(df.val_set,
df.color_map,df.labels,single='Road',val=True)
test_ds = CamVidDataset(df.test_set,
df.color_map,df.labels,single='Road',val=True)
output_ds =
CamVidDataset(df.test_set[100:201],df.color_map,df.labels,single='Road
',val=True)

train_ds[0][0].shape

torch.Size([3, 180, 180])

plt.imshow(train_ds[50][0].permute(1,2,0))
plt.imshow(train_ds[50][1].permute(1,2,0),alpha=0.4,cmap='gray') #
road
plt.show()
```




```
train_loader = torch.utils.data.DataLoader(train_ds, shuffle=True,
batch_size=8)
valid_loader = torch.utils.data.DataLoader(val_ds, shuffle=False,
batch_size=8)

output_loader = torch.utils.data.DataLoader(output_ds, shuffle=False,
batch_size=16)
```

Предобработка модели

- используется UNET из репозитория @aladdinpersson

```
!git clone https://github.com/aladdinpersson/Machine-Learning-
Collection.git
```

```
Cloning into 'Machine-Learning-Collection'...
remote: Enumerating objects: 1291, done.ote: Counting objects: 100%
(252/252), done.ote: Compressing objects: 100% (143/143), done.ote:
Total 1291 (delta 105), reused 218 (delta 87), pack-reused 1039
```

```
import sys
sys.path.append('./Machine-Learning-Collection/ML/Pytorch/image_segmen
tation/semantic_segmentation_unet')
from model import UNET
```

Для тренировки используется PyTorch Lightning

```
class BinaryUNetModel(pl.LightningModule):

    def __init__(self, pretrained=True, in_channels = 3, num_classes =
1, lr=3e-4):
        super(BinaryUNetModel, self).__init__()
        self.in_channels = in_channels
        self.num_classes = num_classes
        self.lr = lr

        self.model = UNET(self.in_channels, self.num_classes)

        self.loss_fn = nn.BCEWithLogitsLoss()

        self.train_acc = torchmetrics.Accuracy(task='binary')
        self.val_acc = torchmetrics.Accuracy(task='binary')

        self.outputs = []


    def forward(self, x):
        return self.model(x)


    def configure_optimizers(self):
        optimizer = torch.optim.AdamW(self.model.parameters(),
lr=self.lr)
        return optimizer


    def training_step(self, batch, batch_idx):

        x, y = batch

        preds = self.model(x)

        loss = self.loss_fn(preds, y)
        self.train_acc(preds, y)

        self.log('train_loss', loss.item(), on_epoch=True)
```

```
        self.log('train_acc', self.train_acc, on_epoch=True)

        return loss


    def validation_step(self, batch, batch_idx):

        x,y = batch

        preds = self.model(x)

        loss = self.loss_fn(preds, y)
        self.val_acc(preds,y)

        self.log('val_loss', loss.item(), on_epoch=True)
        self.log('val_acc', self.val_acc, on_epoch=True)


    def test_step(self, batch, batch_idx):

        x,y = batch
        preds = self.model(x)

        self.outputs.append(preds.detach().cpu())
```

Тренировка

```
model = BinaryUNetModel()
trainer = pl.Trainer(accelerator='cuda',
                     max_epochs=8,
                     callbacks=[
                         EarlyStopping(monitor="val_loss",
                                       mode="min",
                                       patience=1,
                                      )
                         ],
                     log_every_n_steps=25
                    )
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda),
used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False,
using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False,
using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False,
using: 0 HPUs
```

```
trainer.fit(model, train_loader, valid_loader)
```

```
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger
folder: /content/lightning_logs
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name      | Type              | Params
-----------------------------------------------
0 | model     | UNET              | 31.0 M
1 | loss_fn   | BCEWithLogitsLoss | 0
2 | train_acc | BinaryAccuracy    | 0
3 | val_acc   | BinaryAccuracy    | 0
-----------------------------------------------
31.0 M    Trainable params
0         Non-trainable params
31.0 M    Total params
124.151   Total estimated model params size (MB)
```

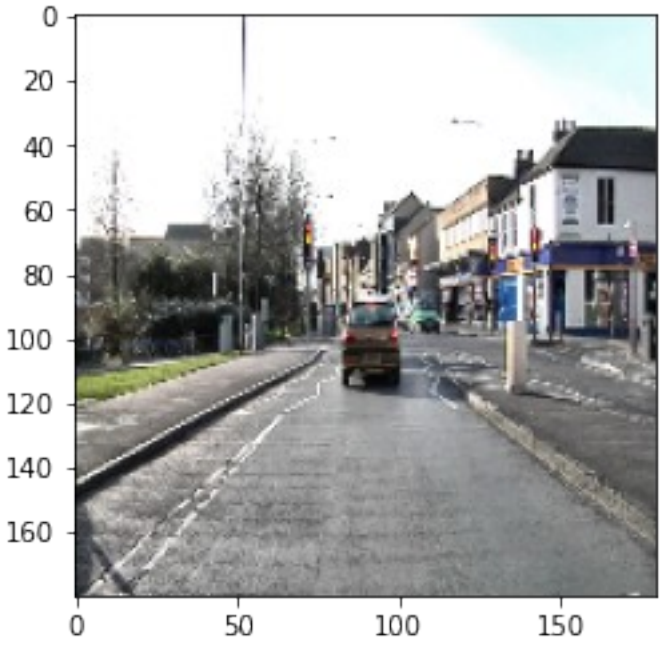

```
{"model_id":"6365709141a74974b52bba494c577d24","version_major":2,"vers
ion_minor":0}

{"model_id":"15f1feb958c94584830457b3ba94e250","version_major":2,"vers
ion_minor":0}

{"model_id":"3bc3c5e832af425fa3772f1eece06557","version_major":2,"vers
ion_minor":0}

{"model_id":"f2ab8663f5094fc9abd25eb958859ad0","version_major":2,"vers
ion_minor":0}
```

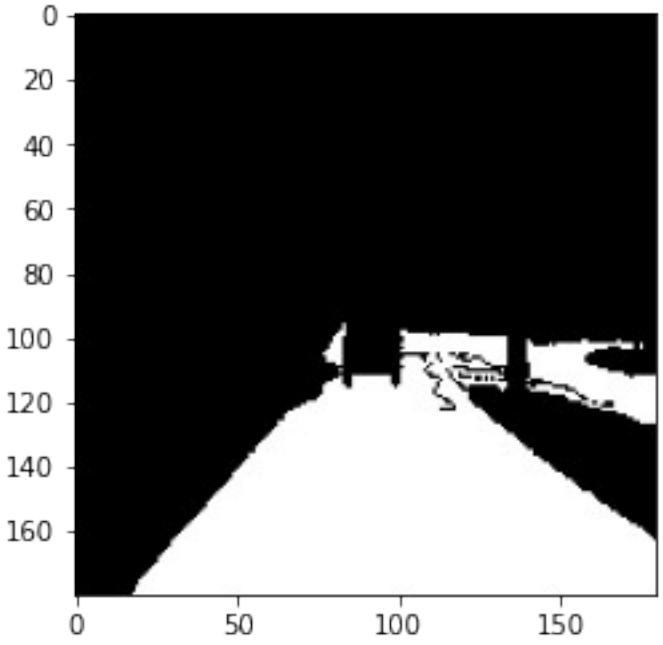

```
trainer.logged_metrics
```

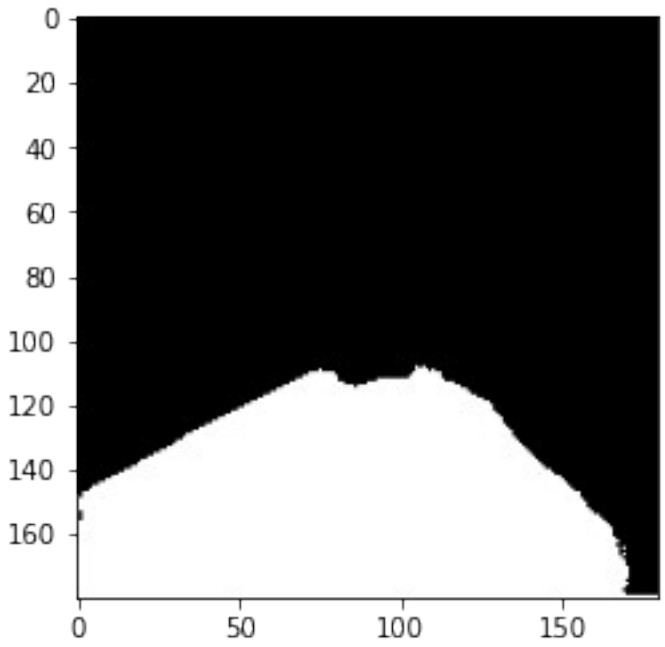
```
{'train_loss_step': tensor(0.2328),
 'train_acc_step': tensor(0.9602),
 'val_loss': tensor(0.3453),
 'val_acc': tensor(0.9044),
 'train_loss_epoch': tensor(0.2841),
 'train_acc_epoch': tensor(0.9232)}
```

Сравнение прогназирования выборки из тренировки с истинной маской

```
x,y = train_ds[100]
y_pred = model(x.unsqueeze(0)).squeeze(0)
plt.imshow(x.permute(1,2,0))
plt.show()
plt.imshow(y.permute(1,2,0),cmap='gray')
plt.show()
plt.imshow((torch.sigmoid(y_pred.permute(1,2,0)) >
0.5).detach().cpu().numpy(),cmap='gray')
plt.show()
```


0
20
40
60
80
100
120
140
160
0
50
100
150


0
20
40
60
80
100
120
140
160
0
50
100
150



Использование тестового набора данных для прогнозирования дорог

```
trainer.test(model,output_loader)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
```

```
{"model_id":"6d2bf908a4e743cbbc46ff181837a031","version_major":2,"vers
ion_minor":0}
```

```
[{}]
```

```
outputs = model.outputs
outputs = torch.sigmoid(torch.vstack(outputs)) > 0.5
print('torch',outputs.shape)
outputs = outputs.permute(0,2,3,1).numpy()
print('numpy',outputs.shape)
```

```
torch torch.Size([101, 1, 180, 180])
numpy (101, 180, 180, 1)
```

```
img_outputs = []
for i in range(outputs.shape[0]):
    img_outputs.append(outputs[i,:,:,:])
```

```
original_images = []
for i in range(100,201):
    img = cv.imread(str(df.test_images[i]))
    img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
    img = cv.resize(img,(180,180))
    img = np.array(img)
    original_images.append(img)
```

```
len(original_images)
```

```
101
```

```
original_images[0].dtype, img_outputs[0].dtype
```

```
(dtype('uint8'), dtype('bool'))
```

Анимация тестовых данных

```
def create_out_anim(save=True):
    fig, [ax1,ax2] = plt.subplots(1,2,dpi=150)
    ax1.axis('off')
    ax2.axis('off')
    images = []
    for i in tqdm(range(len(original_images))):
        im1 = img_outputs[i].astype(np.uint8)
        im2 = original_images[i]
        im1f = ax1.imshow(im1, animated=True)
        im2f = ax2.imshow(im2, animated=True)
        if i==0:
            ax1.imshow(im1.astype(np.uint8))
            ax2.imshow(im2)
        images.append([im1f,im2f])


    ani = animation.ArtistAnimation(fig, images, interval=100,
blit=True,
                                    repeat_delay=1000)
    plt.close()

    if save:
        writer = animation.PillowWriter(fps=10)

ani.save('output_sample.gif',writer=writer,savefig_kwargs={'facecolor'
:'white'})

    return ani

create_out_anim(save=True)
```

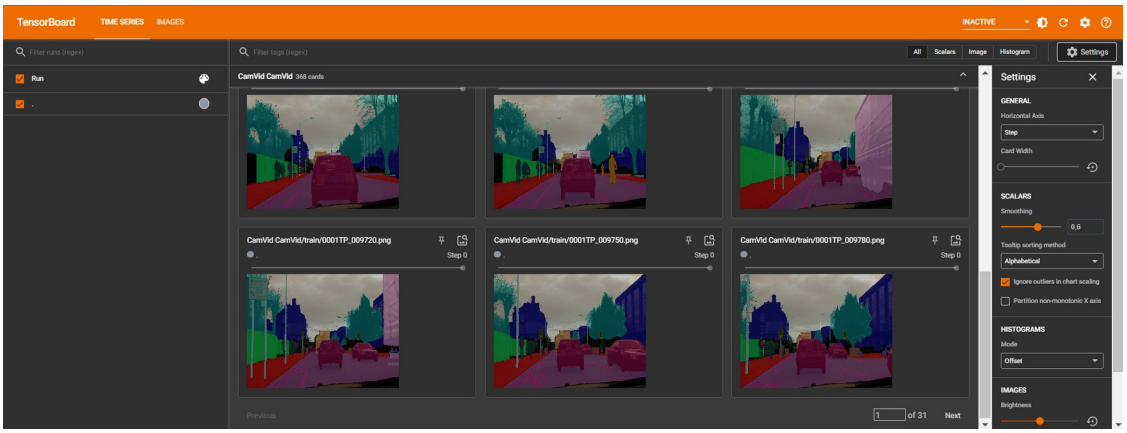
```
{"model_id":"172b1ca34d284d37829628e5c5dfd69b","version_major":2,"vers
ion_minor":0}
```

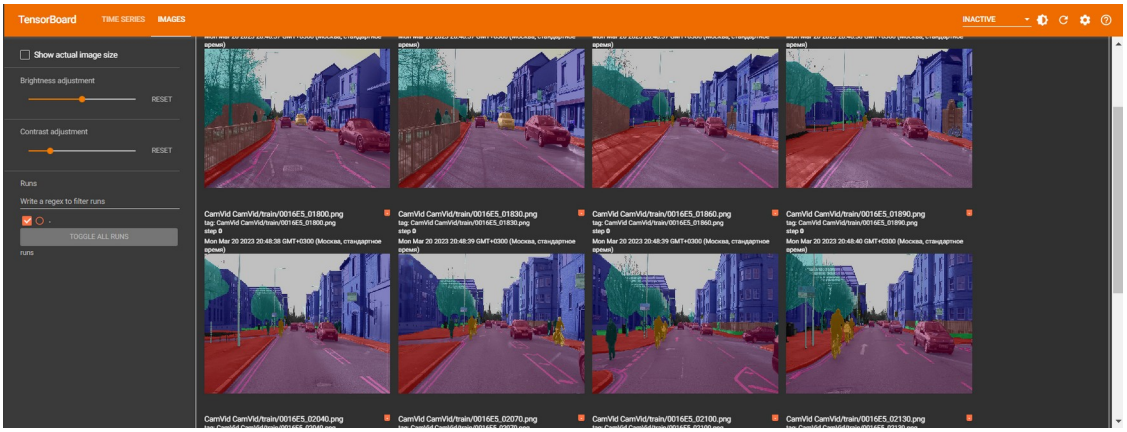
```
WARNING:matplotlib.animation:Animation size has reached 20976243
bytes, exceeding the limit of 20971520.0. If you're sure you want a
larger animation embedded, set the animation.embed_limit rc parameter
to a larger value (in MB). This and further frames will be dropped.
```

```
import os

test_data = os.listdir('./CamVid/test')
```

```
train_data = os.listdir('./CamVid/train')
val_data = os.listdir('./CamVid/val')

data = list()

for test in test_data:
    data.append(['test', 1])
for train in train_data:
    data.append(['train', 1])
for val in val_data:
    data.append(['val', 1])

data = pd.DataFrame(data, columns=['type', 'count'])

fig = px.pie(data, values='count', names='type')
fig.show()
```


```
writer = SummaryWriter('runs/')

for img, mask in df.train_set:
    src1 = cv2.imread(str(img), cv2.IMREAD_COLOR)
    src2 = cv2.imread(str(mask), cv2.IMREAD_COLOR)
    dst = cv2.addWeighted(src1, 0.5, src2, 0.5, 0.0)

    img_grid = torchvision.transforms.functional.to_tensor(dst)
    writer.add_image(f'CamVid {str(img)}', img_grid)

%load_ext tensorboard

tensorboard --logdir=runs

<IPython.core.display.Javascript object>
```


TensorBoard
TIME SERIES
IMAGES
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run
CamVid CamVid 368 cards
CamVid CamVid/train/0001TP_009720.png
CamVid CamVid/train/0001TP_009750.png
CamVid CamVid/train/0001TP_009780.png
Step 0
Previous
1
of 31
Next
Settings
GENERAL
Horizontal Axis
Step
Card Width
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset
IMAGES
Brightness


TensorBoard
TIME SERIES
IMAGES
INACTIVE
Show actual image size
Brightness adjustment
RESET
Contrast adjustment
RESET
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
runs
CamVid CamVid/train/0016E5_01800.png
CamVid CamVid/train/0016E5_01830.png
CamVid CamVid/train/0016E5_01860.png
CamVid CamVid/train/0016E5_01890.png
CamVid CamVid/train/0016E5_02040.png
CamVid CamVid/train/0016E5_02070.png
CamVid CamVid/train/0016E5_02100.png
CamVid CamVid/train/0016E5_02130.png