The network architecture was taken from the article https://arxiv.org/abs/1507.05717

```
!pip install torchmetrics
```

```
import numpy as np
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
import zipfile
import cv2
import os
from sklearn.model_selection import train_test_split
import re
from torch.nn.utils.rnn import pad_sequence
from tqdm.notebook import trange, tqdm
from torch.optim import AdamW
from torch.nn import CTCLoss
from sklearn.metrics import accuracy_score
from torchmetrics import CharErrorRate
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Extract data

```
zip_file = '/content/drive/MyDrive/CCPD2019-dl1.zip'
z = zipfile.ZipFile(zip_file, 'r')
z.extractall()
```

Separation of validation and train parts

```
data_path = '/content/CCPD2019-dl1/train'
test_data_path = '/content/CCPD2019-dl1/test'
images = os.listdir(data_path)
train_images, val_images = train_test_split(images, test_size=0.2)

train_images = [os.path.join(data_path, image)
                for image in train_images]
val_images = [os.path.join(data_path, image)
                for image in val_images]
test_images = [os.path.join(test_data_path, image)
                for image in os.listdir(test_data_path)]
```

Define tokenizer

```
OOV_TOKEN = ''
CTC_BLANK = ''

class Tokenizer:
    def __init__(self, alphabet):
        self.id_to_symbol = dict(enumerate(alphabet, start = 1))
        self.id_to_symbol[0] = CTC_BLANK
        self.symbols_dict = {val: key for key, val in
self.id_to_symbol.items()}

    def encode(self, words_list):
      """Encode every word from words list into a list of symbolic
identifiers"""
      enc_words_list = []
      for word in words_list:
          enc_words_list.append([self.symbols_dict[s] if s in
self.symbols_dict
                                 else self.symbols_dict[CTC_BLANK]
                                 for s in word])
      return enc_words_list

    def decode(self, encoded_words_list):
        """Decode every encoded word from list into string form"""
        words_list = []
        for encoded_word in encoded_words_list:
            decoded_word = ''
            for i in range(len(encoded_word)):
                 if encoded_word[i] != encoded_word[i-1] or i == 0:
                    decoded_word += self.id_to_symbol[encoded_word[i]]
            words_list.append(decoded_word)
        return words_list
```

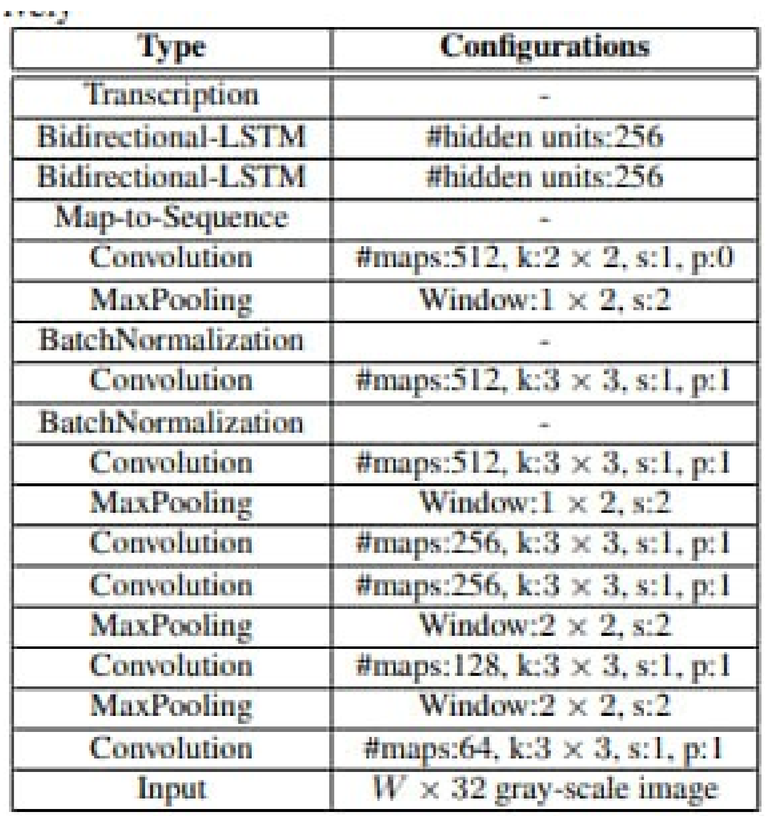
Create Dataset

```
class CCPDataset(Dataset):
      def __init__(self, data, tokenizer, transform = None):
          super().__init__()
          self.images_paths = data
          self.texts = [re.split('/|-|\.', image)[-2]
                         for image in self.images_paths]
          self.labels = torch.LongTensor(tokenizer.encode(self.texts))
          self.transform = transform

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

      def get_image_as_array(self, idx):
          return cv2.imread(self.images_paths[idx])

      def __getitem__(self, idx):
          """Return resized and scaled (Min-Max Scaling) image in
grayscale"""
          width = 512
          height = 64
          image = cv2.imread(self.images_paths[idx],
cv2.IMREAD_GRAYSCALE)
          image = cv2.resize(image, (height, width))
          image = 
torch.unsqueeze(torch.from_numpy(image).to(torch.float), 0)
          image = (image - torch.min(image)) / (torch.max(image) - 
torch.min(image))
          if self.transform is not None:
              image = self.transform(image)
          label = self.labels[idx]
          text = self.texts[idx]
          return image, label, text
```

Initialize tokenizer, datasets and dataloaders

```
def collate_fn(batch):
    images, labels, texts = zip(*batch)
    images = torch.stack(images, 0)
    texts_lengths = torch.LongTensor([len(label) for label in labels])
    labels = pad_sequence(labels, batch_first=True, padding_value=0)
    return images, labels, texts, texts_lengths

def get_alphabet(data_path):
    labels = [re.split('/|-|\.', image)[-2]
             for image in os.listdir(os.path.join(data_path, 
'train'))]
    labels.extend([re.split('/|-|\.', image)[-2]
             for image in os.listdir(os.path.join(data_path, 
'test'))])
    return set(''.join(labels))

data_path = '/content/CCPD2019-dl1/'
alphabet = sorted(get_alphabet(data_path))
tokenizer = Tokenizer(alphabet)

train_dataset = CCPDataset(train_images, tokenizer)
val_dataset = CCPDataset(val_images, tokenizer)
test_dataset = CCPDataset(test_images, tokenizer)

train_loader = DataLoader(
    train_dataset,
    batch_size=64,
    shuffle=True,
    collate_fn=collate_fn
)
val_loader = DataLoader(
    val_dataset,
    batch_size=64,
    shuffle=True,
    collate_fn=collate_fn
)
test_loader = DataLoader(
    test_dataset,
    batch_size=64,
    shuffle=False,
    collate_fn=collate_fn
)
```

CRNN definition



Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	W × 32 gray-scale image

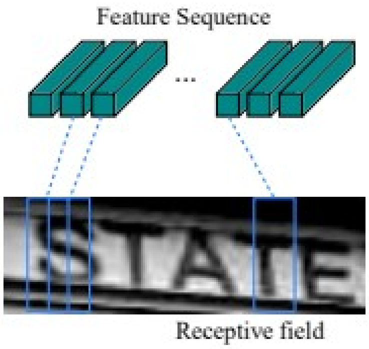
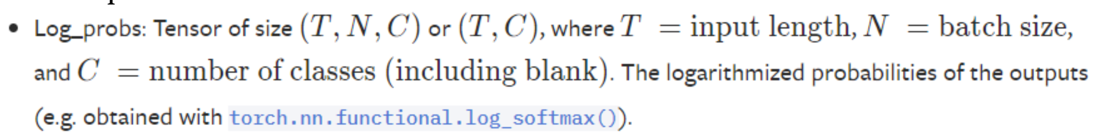


Figure 2. The receptive field. Each vector in the extracted feature sequence is associated with a receptive field on the input image, and can be considered as the feature vector of that field.

```
class CRNN(nn.Module):
    def __init__(self, alphabet_len, lstm_input_size=256,
                 lstm_hidden_size=256, lstm_num_layers=2):
        super().__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(1, 64, (3,3), padding=1),
            nn.MaxPool2d((2,2), 2),
            nn.Conv2d(64, 128, (3,3), padding=1),
            nn.MaxPool2d((2,2),2),
            nn.Conv2d(128, 256, (3,3),padding=1),
            nn.Conv2d(256, 256, (3,3),padding=1),
            nn.MaxPool2d((1,2),2),
            nn.Conv2d(256, 512, (3,3),padding=1),
            nn.BatchNorm2d(512),
            nn.Conv2d(512, 512, (3,3),padding=1),
            nn.BatchNorm2d(512),
            nn.MaxPool2d((1,2),2),
            nn.Conv2d(512, 512, (2,2))
        )
        self.adaptive_avg_pool = nn.AdaptiveAvgPool2d(
            (lstm_input_size, lstm_input_size))
        self.BiLSTM = nn.LSTM(input_size=lstm_input_size,
                              hidden_size=lstm_hidden_size,
                              num_layers=lstm_num_layers,
                              batch_first=True, bidirectional=True)
        self.transcription = nn.Linear(lstm_hidden_size *
lstm_num_layers,
                                       alphabet_len)

    def forward(self, x):
        x = self.feature_extractor(x)
        # input for LSTM: (N,L,H_{in}), N - batch_size, L -
sequence_length, H_{in} - input_size
        b, c, h, w = x.shape
        x = torch.reshape(x, (b, c * h, w)) # map to sequence
        x = self.adaptive_avg_pool(x)
        x, _ = self.BiLSTM(x)
        x = self.transcription(x)
        x = nn.functional.log_softmax(x, dim=2)
        # (N, L, C), N - batch_size,
        # L - sequence_length (rectangles count),
        # C - number of classes
        return x
```

Train loop

Format of predictions for CTCLoss function:



- Log_probs: Tensor of size (T, N, C) or (T, C), where $T = \text{input length}, N = \text{batch size}$, and $C = \text{number of classes (including blank)}$. The logarithmized probabilities of the outputs (e.g. obtained with `torch.nn.functional.log_softmax()`).

```
!pip install neptune-client

import neptune.new as neptune
run = neptune.init(
    api_token= os.getenv('NEPTUNE_API_TOKEN'),
    project = 'misha/ocr-recognition-carplates'
)

DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def train_epoch(model, train_loader, criterion, optimizer):
    model.train()
    total_loss = 0
    batches_count = 0
    for batch in tqdm(train_loader):
        model.zero_grad()
        images, targets, _, target_lengths = batch
        images=images.to(DEVICE)
        targets=targets.to(DEVICE)
        target_lengths=target_lengths.to(DEVICE)
        predictions = model(images).permute(1, 0, 2)
        input_lengths = torch.full(
            (predictions.shape[1],),
            predictions.shape[0]
        )
        loss = criterion(predictions, targets, input_lengths,
target_lengths)
        loss.backward()
        optimizer.step()
        run["train/loss"].log(loss.item())
        total_loss += loss.item()
        batches_count += 1
    return total_loss/batches_count

def eval_epoch(model, val_loader, criterion, tokenizer):
    model.eval()
    total_loss = 0
    batches_count = 0
    total_epoch_accuracy = 0
    total_epoch_cer = 0
    for batch in tqdm(val_loader):
        images, targets, texts, target_lengths = batch
        images=images.to(DEVICE)
        targets=targets.to(DEVICE)
        target_lengths=target_lengths.to(DEVICE)
        predictions = model(images)
        input_lengths = torch.full(
            (predictions.shape[0],),
            predictions.shape[1]
        )
        loss = criterion(torch.permute(predictions, (1, 0, 2)),
                         targets, input_lengths, target_lengths)
        predicted_words = torch.argmax(predictions.detach().cpu(),
dim=2).numpy()
        decoded_words = tokenizer.decode(predicted_words)
        accuracy = accuracy_score(texts, decoded_words)
        cer_score = CharErrorRate()
        cer = cer_score(texts, decoded_words)
        total_loss += loss.item()
        total_epoch_accuracy += accuracy
        total_epoch_cer += cer
        batches_count += 1
        run["evaluation/loss"].log(loss.item())
        run["evaluation/accuracy"].log(accuracy)
        run["evaluation/charecter_error_rate"].log(cer)
    return total_loss/batches_count,
total_epoch_accuracy/batches_count, \
           total_epoch_cer/batches_count

def train(model, train_loader, val_loader, tokenizer, num_epochs):
  model.to(DEVICE)
  criterion = torch.nn.CTCLoss(blank=0)
  optimizer = AdamW(model.parameters(), lr=0.001, weight_decay=0.01)
  run['model/parameters/n_epochs'] = num_epochs
  scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
              optimizer=optimizer, mode='min', factor=0.5, patience=2)
  for epoch in trange(num_epochs):
      train_loss = train_epoch(model,
                               train_loader,
                               criterion,
                               optimizer
                               )
      val_loss, mean_epoch_accuracy, mean_epoch_cer = eval_epoch(
                                                                model,


val_loader,

criterion,

tokenizer
                                                                )
      scheduler.step(val_loss)
      run['train/epoch/loss'].log(train_loss)
      run['evaluation/epoch/loss'].log(val_loss)
      run['evaluation/epoch/accuracy'].log(mean_epoch_accuracy)
      run['evaluation/epoch/charecter_error_rate'].log(mean_epoch_cer)
      save_dir = '/content/drive/MyDrive/crnn_weights/'
      model_save_path = os.path.join(save_dir,
                                     f'model-{epoch}-
{mean_epoch_cer:.4f}.ckpt')
      torch.save(model.state_dict(), model_save_path)

model = CRNN(len(alphabet))
num_epochs = 15
train(model, train_loader, val_loader, tokenizer, num_epochs)
```

```
{"version_major":2,"version_minor":0,"model_id":"8c2a22c8874d47c79ab35
8a6527c376f"}

{"version_major":2,"version_minor":0,"model_id":"af8d1a6eb5cc483b85730
1105947175a"}

{"version_major":2,"version_minor":0,"model_id":"9074e82f0b89405c84e68
c9ffe8a6b49"}

{"version_major":2,"version_minor":0,"model_id":"9b16912a207b44b8ae996
fc46f51a6a0"}

{"version_major":2,"version_minor":0,"model_id":"254d2ab1748240d9ba831
cac91ab68d2"}

{"version_major":2,"version_minor":0,"model_id":"14425effc285429282b37
4001b5c641d"}

{"version_major":2,"version_minor":0,"model_id":"755696a2e8984aceb2bd5
4c6eda44ae0"}

{"version_major":2,"version_minor":0,"model_id":"33d5098df1a74653ae75a
5419a6907c0"}

{"version_major":2,"version_minor":0,"model_id":"a49c31c85d964e62aed9e
6469ef50e6c"}

{"version_major":2,"version_minor":0,"model_id":"4d1d2190c4ee44b5ad67c
124a283f441"}

{"version_major":2,"version_minor":0,"model_id":"8ea49121a2cb43718cd58
6a1049c0cc5"}

{"version_major":2,"version_minor":0,"model_id":"7e73d267548e4db6914ea
7f3db2fdcdc"}

{"version_major":2,"version_minor":0,"model_id":"030ec4802f7f420684c51
3a788a43242"}
```

{"version_major":2,"version_minor":0,"model_id":"f7e73a6cc638454e89e65a76aed347a7"}

{"version_major":2,"version_minor":0,"model_id":"a9d74bda82fd41c48841bdfe72db0798"}

{"version_major":2,"version_minor":0,"model_id":"d222060995cb48a08e0f92b5d4ef523e"}

{"version_major":2,"version_minor":0,"model_id":"c0910eca43f846e2a8d17b30eba82aca"}

{"version_major":2,"version_minor":0,"model_id":"413f09e853aa47d0bb25f57ca5d627ad"}

{"version_major":2,"version_minor":0,"model_id":"fa03685530054ce8b0ce49ee1ef6ed24"}

{"version_major":2,"version_minor":0,"model_id":"111b14b60ac841ffb7eb92579a5e595e"}

{"version_major":2,"version_minor":0,"model_id":"c112205c61b64f51b88b6d0e63307ee5"}

{"version_major":2,"version_minor":0,"model_id":"612b2e8ab6c24d27b374ac739ccf0570"}

{"version_major":2,"version_minor":0,"model_id":"5d54be91d16847d0b7320b0f33c57e95"}

{"version_major":2,"version_minor":0,"model_id":"c4db5120971b4246bf0d0e7f86c3c8ab"}

{"version_major":2,"version_minor":0,"model_id":"0d5d2a2b1b084b1da4e10ad236e1fe9b"}

{"version_major":2,"version_minor":0,"model_id":"7e11c2c0bdad4ea98794a16e5544adde"}

{"version_major":2,"version_minor":0,"model_id":"39fb5c44fed948f4a16c05dafd865667"}

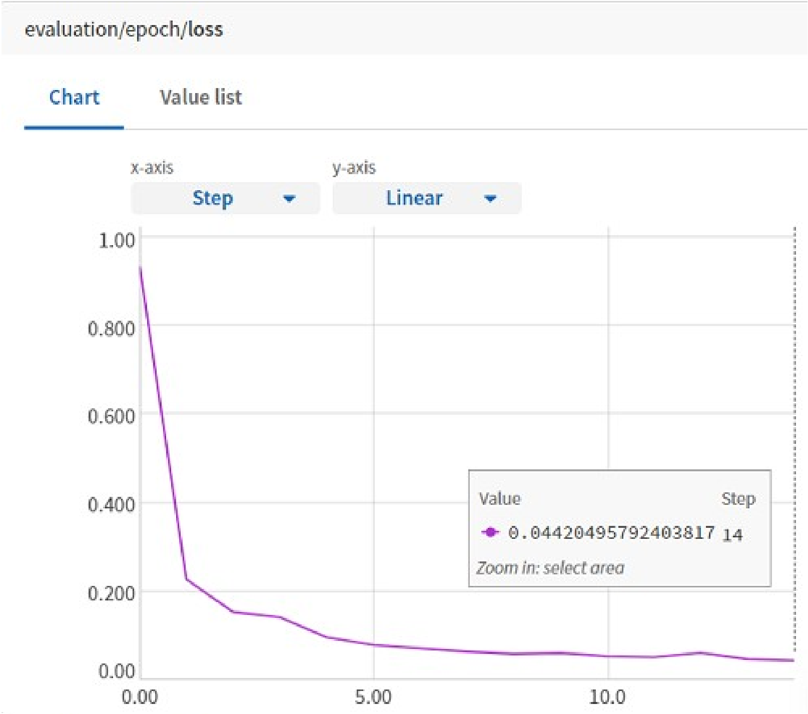
{"version_major":2,"version_minor":0,"model_id":"74fe328c65ee482b87ad9257def0da95"}

{"version_major":2,"version_minor":0,"model_id":"cceac8e6cbff42429e4afc039065c775"}

{"version_major":2,"version_minor":0,"model_id":"99e7db6438f14aa3842b220d22699026"}

{"version_major":2,"version_minor":0,"model_id":"ba14202edc1747c290d7b56794907acf"}

The model was trained for 15 epochs in total. Below are the graphs of the loss function and metrics.



evaluation/epoch/loss

Chart Value list

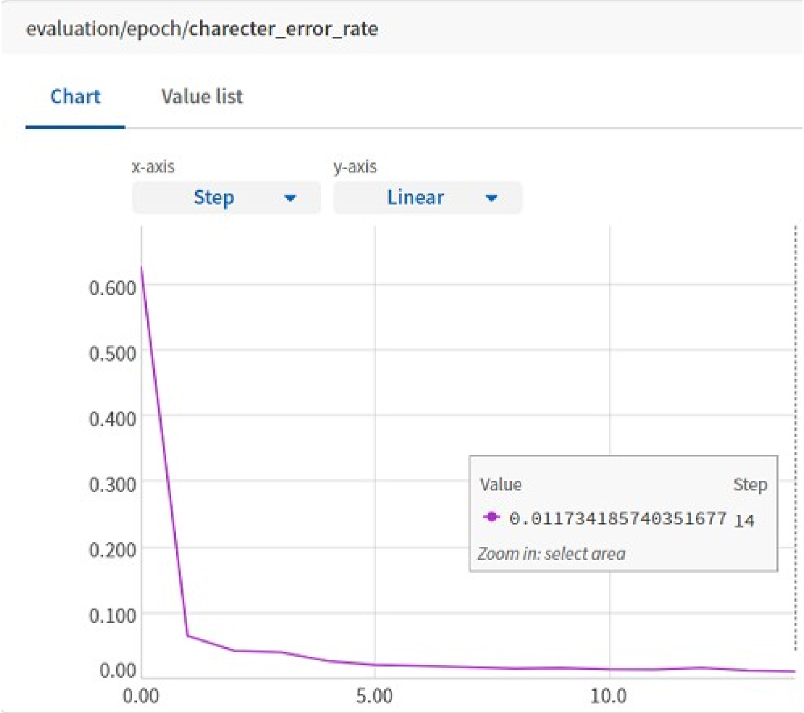
x-axis y-axis

Step Linear

Value Step

0.04420495792403817 14

Zoom in: select area



evaluation/epoch/charecter_error_rate

Chart Value list

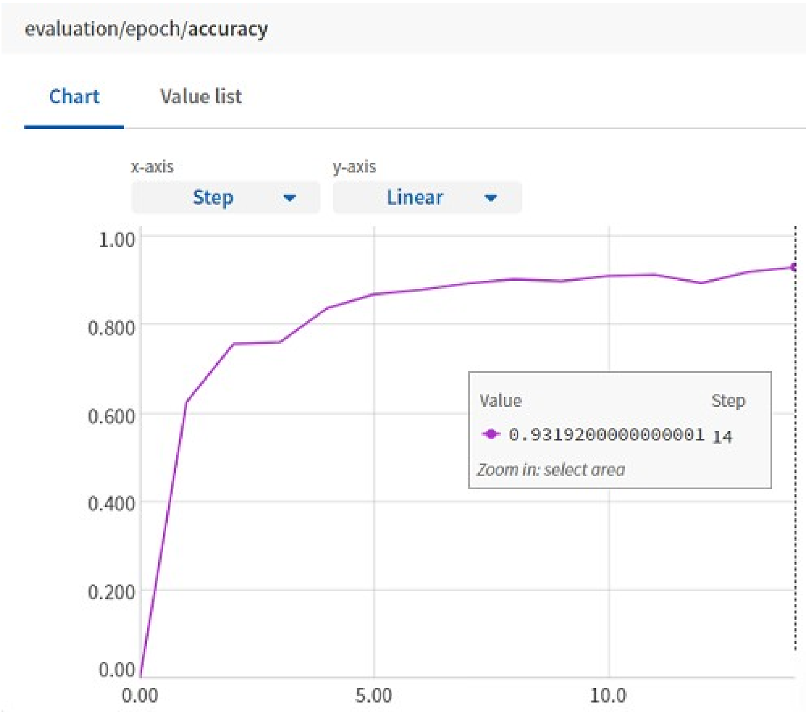
x-axis y-axis

Step Linear

Value Step

0.011734185740351677 14

Zoom in: select area



Load model and make predictons on test dataset

```
path_to_model = '/content/model-14-0.0117.ckpt'
model = CRNN(len(alphabet))
model.load_state_dict(torch.load(path_to_model))
```

```
<All keys matched successfully>
```

```
def test_model(model, test_loader):
    model.eval()
    model.to(DEVICE)
    texts = []
    predictions = torch.FloatTensor([])
    for batch in tqdm(test_loader):
        batch_images, _, batch_texts, _ = batch
        batch_images=batch_images.to(DEVICE)
        predictions=torch.cat((predictions,
model(batch_images).detach().cpu()))
        texts.extend(list(batch_texts))
    predicted_words = torch.argmax(predictions.detach().cpu(),
dim=2).numpy()
    decoded_words = tokenizer.decode(predicted_words)
    accuracy = accuracy_score(texts, decoded_words)
    cer_score = CharErrorRate()
    cer = cer_score(texts, decoded_words)
    return accuracy, cer, texts, decoded_words
```

```
accuracy, cer, texts, decoded_words = test_model(model, test_loader)
print(f"Accuracy on test dataset: {accuracy}")
print(f"Character Error Rate on test dataset: {cer}")
```

{"version_major":2,"version_minor":0,"model_id":"967c368ee8154e79a6927
42c73c01629"}

```
Accuracy on test dataset: 0.8588858885888588
Character Error Rate on test dataset: 0.026025692000985146
```

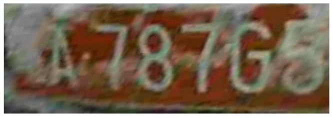
Model error analysis

```
def get_images_with_great_error(N, model, test_dataset):
    """Get top N images with max Character Error Rate metric"""
    error_image_map = {}
    cer_score = CharErrorRate()
    # get and put into dictionary Character Error Rate score for every
item in test_dataset
    for i in trange(len(test_dataset)):
        image, label, text = test_dataset[i]
        image=image.unsqueeze(0).to(DEVICE)
        prediction = model(image)
        predicted_word = torch.argmax(prediction.detach().cpu(),
dim=2).numpy()
        decoded_word = tokenizer.decode(predicted_word)
        error_image_map[i] = (cer_score(text, decoded_word),
                              f"{text} CER: {cer_score(text,
decoded_word):.4f} pred: {decoded_word}")
    sorted_error_image_map = sorted(error_image_map.items(),
key=lambda x: x[1])[:-N:-1]
    plt.figure(figsize=(18, 4*(N//3+1)))
    for i, item in enumerate(sorted_error_image_map):
        key, value = item
        cer_score, title = value
        plt.subplot(N//3+1, 3, i+1, title=title)
        plt.axis("off")
        plt.imshow(test_dataset.get_image_as_array(key))
    plt.show();
    return sorted_error_image_map
```

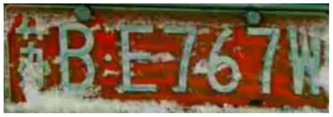
```
sorted_error_image_map = get_images_with_great_error(50, model,
test_dataset)
```

{"version_major":2,"version_minor":0,"model_id":"72cb586f095c4cca8c9c2
f94e0b33146"}

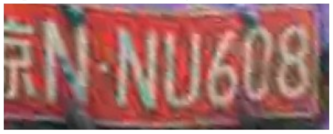
τÜûA787G5 CER: 0.8889 pred: ['τ╡ÖF06Y53']



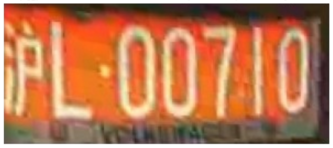
ΦïÅBE767W CER: 0.8889 pred: ['τÜûB6J3J4']



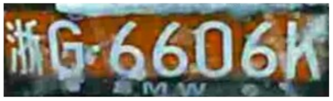
Σ║¼NNU608 CER: 0.7778 pred: ['τÜûN9K088']



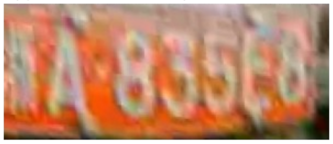
µ▓¬L00710 CER: 0.7778 pred: ['µÜûL98J78']



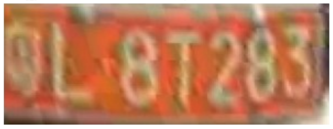
µ╡ÖG6606K CER: 0.7778 pred: ['Φ▒½E66D26']



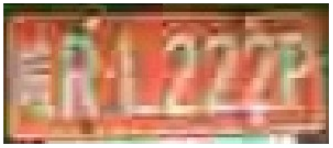
µ╡ÖA835E8 CER: 0.7778 pred: ['τÜûAG2958']



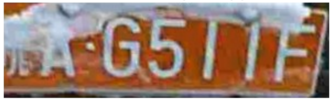
τ▓ñL8T283 CER: 0.6667 pred: ['τÜûL20793']



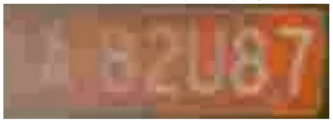
τÜûRL222P CER: 0.6667 pred: ['τÜûA17770']



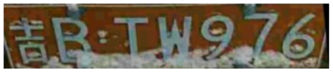
τÜûAG511F CER: 0.6667 pred: ['µ╡¬CS511E']



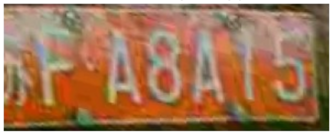
τÜûA82U87 CER: 0.6667 pred: ['µ╡ÖJC8U87']



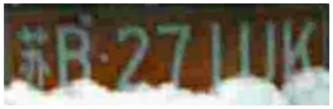
σÉëBTW976 CER: 0.6667 pred: ['ΦÜÅBYW879']



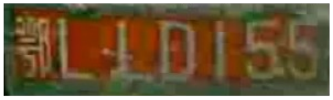
ΦïÅFA8A15 CER: 0.6667 pred: ['µ╡¬C58A1S']



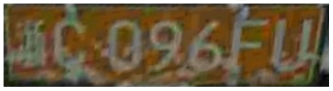
ΦïÅB271UK CER: 0.6667 pred: ['τÜûQ271H8']



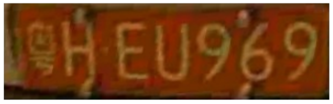
ΘäéLLD155 CER: 0.6667 pred: ['ΦïÅLL9129']



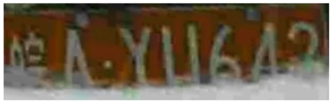
µ╡ÖC096FU CER: 0.6667 pred: ['Φ▒½SQ96FH']



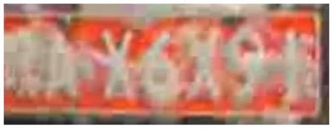
τ▓ñHEU969 CER: 0.5556 pred: ['τÜûMEH967']



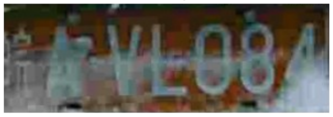
τÜûAYU642 CER: 0.5556 pred: ['τÜûAJVMJ6']



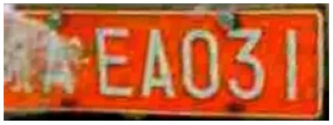
τÜûAX6X91 CER: 0.5556 pred: ['τÜûC25X6T']



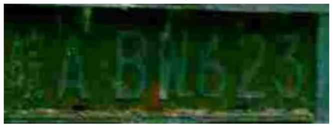
τÜûAVL084 CER: 0.5556 pred: ['τÜûP8F8Q4']



τÜûAEA031 CER: 0.5556 pred: ['τÜûPLN921']



τÜûABW623 CER: 0.5556 pred: ['τÜûAGX936']



τÜûA771X8 CER: 0.5556 pred: ['τÜûC1112R']

τÜûA0Y064 CER: 0.5556 pred: ['τÜûAYH2Q3']

Φ▒½RBS767 CER: 0.5556 pred: ['µ╡ÖBB9767']

```
sorted_error_image_map
```

```
chinese_characters = {}
for text in tqdm(train_dataset.texts):
    if text[:3] not in chinese_characters:
        chinese_characters[text[:3]] = 1
    else:
        chinese_characters[text[:3]] +=1
```

{"version_major":2,"version_minor":0,"model_id":"0d3bd47116f44605a8d20c5812148d3b"}

```
chinese_characters
```

```
{'τÜû': 153505,
 'Φ▒½': 327,
 'ΦïÅ': 2638,
 'μ╣ÿ': 71,
 'Θù╜': 169,
 'μ╡Ö': 1064,
 'Σ║¼': 230,
 'μ▓¬': 540,
 'σåÇ': 144,
 'Θ▓ü': 202,
 'Θ¥Æ': 11,
 'τ▓ñ': 287,
 'Φ╛╜': 47,
 'σ╖¥': 95,
 'Φ╡ú': 118,
 'μ╕¥': 61,
 'Θäé': 224,
 'μÖï': 53,
 'μ┤Ñ': 48,
 'Θ╗æ': 14,
 'ΘÖò': 41,
 'μíé': 9,
 'τöÿ': 16,
 'Σ║æ': 12,
 'σÉë': 10,
 'μû░': 11,
 'Φ┤╡': 12,
 'ΦÆÖ': 15,
 'σ«ü': 4,
 'τÉ╝': 5,
 'ΦùÅ': 1}
```

Выводы

После просмотра изображений с худшими значениями метрики CER становится ясно, что большинство ошибок модели вызваны следующими причинами:

- плохая видимость номерного знака, размытие, недостаток света на фотографии, низкое разрешение фотографии
- ошибки в парах похожих символов, например ('S', '5'), ('9', 'S'), ('2', 'Z')
- ошибки в китайских символах, которые крайне редко встречались в тренировочной выборке ('τ▓ñ', 'Φ▒½', 'Θ▓ü')

Варинты решения:

- Искусственное увеличение количества обучающих примеров с редкими китайскими символами или буквами и цифрами, которые модель путает
- Попробовать использовать обработку изображений перед подачей в модель, увеличение контрастности, например, или попробовать применять фильтры для увеличения резкости

Скорее всего большой прирост в качестве распознавания можно было бы получить от использования другой архитектуры модели.
Стоит отметить, что на некоторых изображениях видимость номера настолько плохая, что нельзя ожидать адекватного распознавания такого номера.