License Plate Recognition

In [2]:

```
# !pip install pytorch_lightning torchmetrics torch-summary
```

In [3]:

```
from itertools import groupby
import os
from pathlib import Path
from typing import Any, Dict, List, Tuple, Optional, Union

import cv2
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, random_split
from torch.nn.utils.rnn import pad_sequence
import torchvision.transforms as transforms
from torchmetrics import ExactMatch, CharErrorRate, ConfusionMatrix
import pytorch_lightning as pl
from pytorch_lightning.callbacks import ModelCheckpoint

import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
import seaborn as sns

batch_size = 64
learning_rate = 1e-3

device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print(f"Device: {device}")
```

```
Device: cuda
```

Dataset:

Load dataset:

In [8]:

```
# !unzip /content/drive/MyDrive/OCR/dataset/CCPD2019-dl1.zip
```

In [9]:

```
root_dir = "/content"
img_dir = os.path.join(root_dir, "CCPD2019-dl1")
```

Build dictionary:

In [10]:

```
blank = ["-"]
provinces = ["皖", "沪", "津", "渝", "冀", "晋", "蒙", "辽", "吉", "黑", "苏", "浙", "京", "闽
             "琼", "川", "贵", "云", "藏", "陕", "甘", "青", "宁", "新", "警", "学", "O"]
alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'F
             'X', 'Y', 'Z', 'O']
numbers = [ '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

In [11]:

```
class Dictionary:
```

```python
    def __init__(self) -> None:
        self.char2idx: dict[str, int] = {}
        self.idx2char: list[str] = []

    def add_char(self, char: str) -> None:
        if char not in self.char2idx:
            self.idx2char.append(char)
            self.char2idx[char] = len(self.idx2char) - 1

    def __len__(self) -> int:
        return len(self.idx2char)
```

In [12]:

```python
dictionary = Dictionary()

for char in blank + provinces + alphabets + numbers:
  dictionary.add_char(char)
```

In [13]:

```python
class LabelConverter:
    def __init__(self, dictionary: Dictionary) -> None:
        self.dictionary = dictionary

    def encode(self, label: str) -> torch.Tensor:
        """Encodes string into tensor of corresponding indices"""
        return torch.LongTensor([self.dictionary.char2idx[char] for char in label])

    def decode(
        self,
        sequence: Union[List[int], torch.Tensor],
        prediction: bool = True,
        blank: int = 0,
    ) -> str:
        """Decodes tensor of indices into string"""
        # remove repetitions
        if prediction:
            sequence = [int(k) for k, _ in groupby(sequence)]
        return "".join(
            self.dictionary.idx2char[idx] for idx in sequence if idx != blank
        )
```

In [14]:

```python
label_converter = LabelConverter(dictionary)
```

Define Dataset:

In [15]:

```python
class LicencePlateDataset(Dataset):
    def __init__(
        self,
        img_dir: Path,
        dictionary: Dictionary,
        img_size: Tuple[int, int] = (128, 32),
        train: bool = True,
        transform: Union[None, transforms.Compose, transforms.ToTensor] = None,
    ):
        super().__init__()

        self.img_dir = os.path.join(img_dir, "train" if train else "test")
        self.images = [
            os.path.join(self.img_dir, img_filename)
            for img_filename in os.listdir(self.img_dir)
        ]
        self.labels = [img_filename.split("-")[-1][:-4] for img_filename in self.images]
```

```
        self.dictionary = dictionary
        self.img_size = img_size  # (width, height)
        self.transform = transform

    def __len__(self) -> int:
        return len(self.images)

    def __getitem__(self, idx: int) -> Tuple[torch.Tensor, torch.Tensor]:
        img = cv2.imread(self.images[idx])
        img = cv2.resize(img, self.img_size)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        if self.transform:
            img = self.transform(img)
        label = self.labels[idx]
        label_encoded = torch.LongTensor(
            [self.dictionary.char2idx[char] for char in label]
        )
        return img, label_encoded
```

Create datasets:

In [16]:

```
train_dataset = LicencePlateDataset(img_dir=img_dir, dictionary=dictionary, train=True, tr
test_dataset = LicencePlateDataset(img_dir=img_dir, dictionary=dictionary, train=False, tr

# split train dataset into train and valid sets
n_val = int(0.05 * len(train_dataset))
n_train = len(train_dataset) - n_val
train_dataset, val_dataset = random_split(train_dataset, [n_train, n_val])

print(f"Number of images in train set: {len(train_dataset)}")
print(f"Number of images in valid set: {len(val_dataset)}")
print(f"Number of images in test set: {len(test_dataset)}")
```

```
Number of images in train set: 189981
Number of images in valid set: 9999
Number of images in test set: 9999
```

Let's have a look to 5 random images from train dataset:

In [17]:

```
# the following code applies Chinese font to matplotlib
def change_matplotlib_font(font_download_url):
    FONT_PATH = "chinese_font"

    font_download_cmd = f"wget {font_download_url} -O {FONT_PATH}.zip"
    unzip_cmd = f"unzip -o {FONT_PATH}.zip -d {FONT_PATH}"
    os.system(font_download_cmd)
    os.system(unzip_cmd)

    font_files = fm.findSystemFonts(fontpaths=FONT_PATH)
    for font_file in font_files:
        fm.fontManager.addfont(font_file)

    font_name = fm.FontProperties(fname=font_files[0]).get_name()
    matplotlib.rc("font", family=font_name)

font_download_url = "https://fonts.google.com/download?family=Noto%20Sans%20SC"
change_matplotlib_font(font_download_url)

ids = np.random.randint(low=0, high=len(train_dataset), size=5)
plt.figure(figsize=(18, 6))
for i in range(5):
  plt.subplot(1, 5, i+1)
  plt.imshow(train_dataset[ids[i]][0].squeeze(), cmap="gray")
```

```
    plt.title("".join([dictionary.idx2char[idx] for idx in train_dataset[ids[i]][1]]))
    plt.axis("off")
```

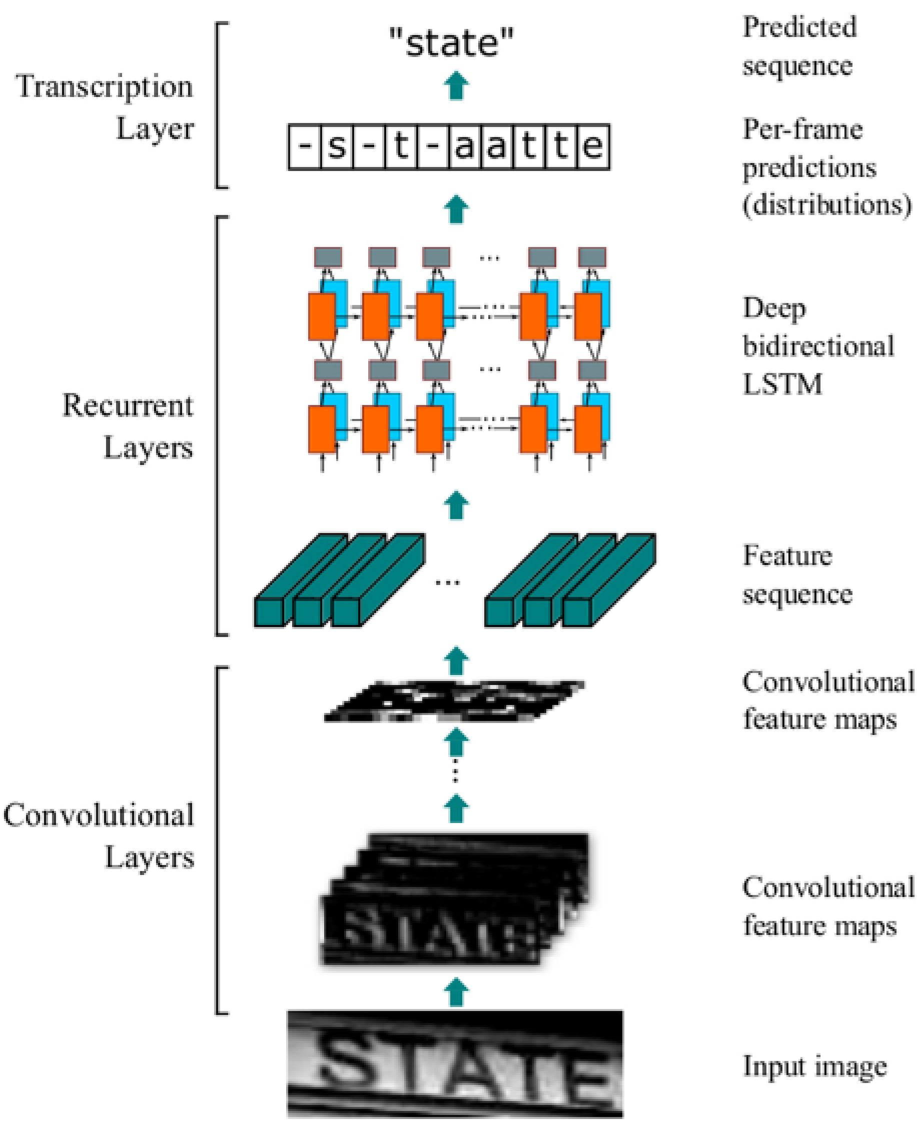
Create DataLoaders:

In [18]:
```
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [19]:
```
images, labels = next(iter(train_dataloader))
print(images.size())
print(labels.size())
```

```
torch.Size([64, 1, 32, 128])
torch.Size([64, 7])
```

Model architecture (CRNN)



Refer to https://arxiv.org/pdf/1507.05717.pdf for more details.

In [20]:
```
class BidirectionalLSTM(nn.Module):
    def __init__(
        self,
```

```
        dictionary_size: int,
        ninp: int,
        nhid: int,
        nlayers: int,
    ) -> None:
        super().__init__()
        self.rnn = nn.LSTM(
            input_size=ninp, hidden_size=nhid, num_layers=nlayers, bidirectional=True
        )
        self.decoder = nn.Linear(2 * nhid, dictionary_size)

        self.dictionary_size = dictionary_size

    def forward(self, input: torch.Tensor) -> torch.Tensor:
        output, _ = self.rnn(input)
        seq_len, batch_size, _ = output.size()
        decoded = self.decoder(output)
        decoded = decoded.view(seq_len, batch_size, self.dictionary_size)
        return decoded


class CRNN(nn.Module):
    def conv_layer(
        self,
        in_channels: int,
        out_channels: int,
        kernel_size: int,
        padding: int,
        batch_norm: bool = False,
    ) -> nn.Sequential:
        layers: list[Any] = [
            nn.Conv2d(
                in_channels=in_channels,
                out_channels=out_channels,
                kernel_size=kernel_size,
                padding=padding,
            )
        ]
        if batch_norm:
            layers.append(nn.BatchNorm2d(num_features=out_channels))
        layers.append(nn.ReLU())
        return nn.Sequential(*layers)

    def __init__(self, dictionary_size: int):
        super().__init__()

        self.conv0 = self.conv_layer(
            in_channels=1, out_channels=64, kernel_size=3, padding=1
        )  # 1x32x128 -> 64x32x128
        self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2)  # 64x32x128 -> 64x16x64
        self.conv1 = self.conv_layer(
            in_channels=64, out_channels=128, kernel_size=3, padding=1
        )  # 64x16x64 -> 128x16x64
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # 128x16x64 -> 128x8x32
        self.conv2 = self.conv_layer(
            in_channels=128, out_channels=256, kernel_size=3, padding=1
        )  # 128x8x32 -> 256x8x32
        self.conv3 = self.conv_layer(
            in_channels=256, out_channels=256, kernel_size=3, padding=1
        )  # 256x8x32 -> 256x8x32
        self.pool2 = nn.MaxPool2d(kernel_size=(1, 2), stride=2)  # 256x8x32 -> 256x4x16
        self.conv4 = self.conv_layer(
            in_channels=256, out_channels=512, kernel_size=3, padding=1, batch_norm=True
        )  # 256x4x16 -> 512x4x16
        self.conv5 = self.conv_layer(
            in_channels=512, out_channels=512, kernel_size=3, padding=1, batch_norm=True
```

```
        )  # 512x4x16 -> 512x4x16
        self.pool3 = nn.MaxPool2d(
            kernel_size=(1, 2), stride=(2, 1), padding=(0, 1)
        )  # 512x4x16 -> 512x2x17
        self.conv6 = self.conv_layer(
            in_channels=512, out_channels=512, kernel_size=2, padding=0, batch_norm=True
        )  # 512x2x17 -> 512x1x16

        self.rnn = BidirectionalLSTM(
            dictionary_size=dictionary_size, ninp=512, nhid=256, nlayers=2
        )

    def forward(self, input: torch.Tensor) -> torch.Tensor:
        # CNN layers
        l0 = self.conv0(input)
        l1 = self.conv1(self.pool0(l0))
        l2 = self.conv2(self.pool1(l1))
        l3 = self.conv3(l2)
        l4 = self.conv4(self.pool2(l3))
        l5 = self.conv5(l4)
        l6 = self.conv6(self.pool3(l5))

        # (seq_len=16, batch_size, features=512)
        map_to_sequence = l6.squeeze(2).permute(2, 0, 1)

        # RNN layers
        output = self.rnn(map_to_sequence)

        return output
```

In [21]:

```
from torchsummary import summary

model = CRNN(dictionary_size=len(dictionary)).to(device)
summary(model, (1, 32, 128))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 64, 32, 128]         --
|    └─Conv2d: 2-1                       [-1, 64, 32, 128]         640
|    └─ReLU: 2-2                         [-1, 64, 32, 128]         --
├─MaxPool2d: 1-2                         [-1, 64, 16, 64]          --
├─Sequential: 1-3                        [-1, 128, 16, 64]         --
|    └─Conv2d: 2-3                       [-1, 128, 16, 64]         73,856
|    └─ReLU: 2-4                         [-1, 128, 16, 64]         --
├─MaxPool2d: 1-4                         [-1, 128, 8, 32]          --
├─Sequential: 1-5                        [-1, 256, 8, 32]          --
|    └─Conv2d: 2-5                       [-1, 256, 8, 32]          295,168
|    └─ReLU: 2-6                         [-1, 256, 8, 32]          --
├─Sequential: 1-6                        [-1, 256, 8, 32]          --
|    └─Conv2d: 2-7                       [-1, 256, 8, 32]          590,080
|    └─ReLU: 2-8                         [-1, 256, 8, 32]          --
├─MaxPool2d: 1-7                         [-1, 256, 4, 16]          --
├─Sequential: 1-8                        [-1, 512, 4, 16]          --
|    └─Conv2d: 2-9                       [-1, 512, 4, 16]          1,180,160
|    └─BatchNorm2d: 2-10                 [-1, 512, 4, 16]          1,024
|    └─ReLU: 2-11                        [-1, 512, 4, 16]          --
├─Sequential: 1-9                        [-1, 512, 4, 16]          --
|    └─Conv2d: 2-12                      [-1, 512, 4, 16]          2,359,808
|    └─BatchNorm2d: 2-13                 [-1, 512, 4, 16]          1,024
|    └─ReLU: 2-14                        [-1, 512, 4, 16]          --
├─MaxPool2d: 1-10                        [-1, 512, 2, 17]          --
├─Sequential: 1-11                       [-1, 512, 1, 16]          --
|    └─Conv2d: 2-15                      [-1, 512, 1, 16]          1,049,088
```

```
|    └─BatchNorm2d: 2-16                [-1, 512, 1, 16]          1,024
|    └─ReLU: 2-17                       [-1, 512, 1, 16]          --
├─BidirectionalLSTM: 1-12               [-1, 2, 69]               --
|    └─LSTM: 2-18                       [-1, 2, 512]              3,153,920
|    └─Linear: 2-19                     [-1, 2, 69]               35,397
==========================================================================================
Total params: 8,741,189
Trainable params: 8,741,189
Non-trainable params: 0
Total mult-adds (M): 559.53
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 5.13
Params size (MB): 33.34
Estimated Total Size (MB): 38.49
==========================================================================================
```

Out[21]:


```
==========================================================================================
Layer (type:depth-idx)                  Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                       [-1, 64, 32, 128]         --
|    └─Conv2d: 2-1                      [-1, 64, 32, 128]         640
|    └─ReLU: 2-2                        [-1, 64, 32, 128]         --
├─MaxPool2d: 1-2                        [-1, 64, 16, 64]          --
├─Sequential: 1-3                       [-1, 128, 16, 64]         --
|    └─Conv2d: 2-3                      [-1, 128, 16, 64]         73,856
|    └─ReLU: 2-4                        [-1, 128, 16, 64]         --
├─MaxPool2d: 1-4                        [-1, 128, 8, 32]          --
├─Sequential: 1-5                       [-1, 256, 8, 32]          --
|    └─Conv2d: 2-5                      [-1, 256, 8, 32]          295,168
|    └─ReLU: 2-6                        [-1, 256, 8, 32]          --
├─Sequential: 1-6                       [-1, 256, 8, 32]          --
|    └─Conv2d: 2-7                      [-1, 256, 8, 32]          590,080
|    └─ReLU: 2-8                        [-1, 256, 8, 32]          --
├─MaxPool2d: 1-7                        [-1, 256, 4, 16]          --
├─Sequential: 1-8                       [-1, 512, 4, 16]          --
|    └─Conv2d: 2-9                      [-1, 512, 4, 16]          1,180,160
|    └─BatchNorm2d: 2-10                [-1, 512, 4, 16]          1,024
|    └─ReLU: 2-11                       [-1, 512, 4, 16]          --
├─Sequential: 1-9                       [-1, 512, 4, 16]          --
|    └─Conv2d: 2-12                     [-1, 512, 4, 16]          2,359,808
|    └─BatchNorm2d: 2-13                [-1, 512, 4, 16]          1,024
|    └─ReLU: 2-14                       [-1, 512, 4, 16]          --
├─MaxPool2d: 1-10                       [-1, 512, 2, 17]          --
├─Sequential: 1-11                      [-1, 512, 1, 16]          --
|    └─Conv2d: 2-15                     [-1, 512, 1, 16]          1,049,088
|    └─BatchNorm2d: 2-16                [-1, 512, 1, 16]          1,024
|    └─ReLU: 2-17                       [-1, 512, 1, 16]          --
├─BidirectionalLSTM: 1-12               [-1, 2, 69]               --
|    └─LSTM: 2-18                       [-1, 2, 512]              3,153,920
|    └─Linear: 2-19                     [-1, 2, 69]               35,397
==========================================================================================
Total params: 8,741,189
Trainable params: 8,741,189
Non-trainable params: 0
Total mult-adds (M): 559.53
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 5.13
Params size (MB): 33.34
Estimated Total Size (MB): 38.49
==========================================================================================
```


In [22]:

```python
inputs, labels = next(iter(train_dataloader))
print(f"Input shape: {inputs.size()}, labels shape: {labels.size()}")
model = CRNN(dictionary_size=len(dictionary))
```

```
    output = model(inputs)
    print(f"Output shape: {output.size()}")
```

```
Input shape: torch.Size([64, 1, 32, 128]), labels shape: torch.Size([64, 7])
Output shape: torch.Size([16, 64, 69])
```

Training

Кроме Accuracy (в качестве Accuarcy я использовала метрику ExactMatch) и Character Error Rate для тестовой выборки также построим Confusion Matrix, чтобы посмотреть, какие символы модель путает между собой:

In [23]:

```
class OCRModule(pl.LightningModule):
    def __init__(
        self,
        learning_rate: float,
        dictionary_size: int,
        label_converter: LabelConverter
    ) -> None:
        super().__init__()

        self.save_hyperparameters("learning_rate", "dictionary_size")

        self.learning_rate = learning_rate
        self.dictionary_size = dictionary_size
        self.label_converter = label_converter

        self.model = CRNN(dictionary_size=dictionary_size)
        self.criterion = nn.CTCLoss()
        self.test_accuracy = ExactMatch(
            task="multiclass", num_classes=dictionary_size, ignore_index=0
        )
        self.test_char_error_rate = CharErrorRate()
        self.test_confusion_matrix = ConfusionMatrix(
            task="multiclass", num_classes=dictionary_size, ignore_index=0
        )

    def forward(self, images: torch.Tensor) -> torch.Tensor:
        return self.model(images)

    def step(
        self, stage: str, batch: Tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> Dict[str, torch.Tensor]:
        images, targets = batch
        logits = self(images)
        log_probs = F.log_softmax(logits, dim=-1)

        T, N, C = log_probs.size() # T - sequence length, N - batch size, C - number of c

        input_lengths = torch.LongTensor([T for _ in range(N)])
        target_lengths = torch.LongTensor([targets.size(-1) for _ in range(N)])

        loss = self.criterion(log_probs, targets, input_lengths, target_lengths)

        return {"loss": loss, "logits": logits}

    def training_step(
        self, batch: Tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> torch.Tensor:
        results = self.step("train", batch, batch_idx)
        self.log(
            "train_loss", results["loss"], on_step=False, on_epoch=True, prog_bar=True
        )
```

```python
        return results["loss"]

    def validation_step(
        self, batch: Tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> torch.Tensor:
        results = self.step("val", batch, batch_idx)
        self.log("val_loss", results["loss"], prog_bar=True)
        return results["loss"]

    def test_step(
        self, batch: Tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> Dict[str, Any]:
        results = self.step("test", batch, batch_idx)
        self.log("test_loss", results["loss"], prog_bar=True)

        batch_size = batch[0].size(0)
        targets = batch[1]
        targets_decoded = [
            self.label_converter.decode(targets[i][:], prediction=False)
            for i in range(batch_size)
        ]

        preds = results["logits"].argmax(-1)  # seq_len x batch_size
        preds = preds.permute(1, 0)  # batch_size x seq_len
        preds_decoded = [
            self.label_converter.decode(preds[i][:]) for i in range(batch_size)
        ]  # for CER

        preds_encoded = pad_sequence(
            [self.label_converter.encode(label) for label in preds_decoded],
            batch_first=True,
        ).to(self.device) # for accuracy

        if preds_encoded.size(1) != 7:
            targets = F.pad(targets, (0, preds_encoded.size(1) - 7), "constant", 0).to(sel

        # compute and log accuracy
        self.test_accuracy.update(preds_encoded, targets)
        self.log("test_acc", self.test_accuracy, prog_bar=True)

        # compute and log character error rate
        self.test_char_error_rate.update(preds_decoded, targets_decoded)
        self.log("test_char_error_rate", self.test_char_error_rate, prog_bar=True)

        return {"loss": results["loss"], "outputs": preds_encoded, "targets": targets}

    def test_epoch_end(self, outs: List[Dict[str, Any]]) -> None:
        # compute and display confusion matrix
        # some batches have different dimensions, so we have to align them
        # in order to compute confusion matrix
        outputs, targets = [], []
        for i in range(len(outs)):
            batch_size = outs[i]["outputs"].size(0)
            for j in range(batch_size):
                outputs.append(outs[i]["outputs"][j][:])
                targets.append(outs[i]["targets"][j][:])

        outputs_padded = pad_sequence(outputs, batch_first=True).to(
            self.device
        )
        targets_padded = pad_sequence(targets, batch_first=True).to(
            self.device
        )

        self.test_confusion_matrix(outputs_padded, targets_padded)
        computed_conf_mat = (
```

```python
            self.test_confusion_matrix.compute().cpu().numpy().astype(int)
        )

        # plot confusion matrix
        df = pd.DataFrame(
            computed_conf_mat,
            index=list(dictionary.char2idx.keys()),
            columns=list(dictionary.char2idx.keys()),
        )

        # display DataFrame
        with pd.option_context("display.max_rows", None, "display.max_columns", None):
            display(df)

        plt.figure(figsize=(35, 35))
        fig = sns.heatmap(df, annot=True, cmap="Spectral").get_figure()
        plt.xlabel("Predicted", fontsize=18)
        plt.ylabel("Actual", fontsize=18)
        plt.close(fig)

        self.logger.experiment.add_figure("test_confusion_matrix", fig)

    def configure_optimizers(self) -> torch.optim.Optimizer:
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
        return optimizer
```

In [24]:

```python
%reload_ext tensorboard
%tensorboard --logdir=/content/logs
```

In [25]:

```python
module = OCRModule(learning_rate=learning_rate, dictionary_size=len(dictionary), label_cor

logger = pl.loggers.TensorBoardLogger(save_dir=os.path.join(root_dir, "logs"), name="exper
checkpoint_callback = ModelCheckpoint(monitor="val_loss", save_top_k=1, dirpath=os.path.jo
trainer = pl.Trainer(accelerator="auto", logger=logger, max_epochs=10, callbacks=[checkpoi
trainer.fit(model=module, train_dataloaders=train_dataloader, val_dataloaders=val_dataloac
trainer.test(model=module, dataloaders=test_dataloader, ckpt_path="best")
```

```
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
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger folder: /content/logs/experim
ent1
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name                  | Type                      | Params
--------------------------------------------------------------------
0 | model                 | CRNN                      | 8.7 M
1 | criterion             | CTCLoss                   | 0
2 | test_accuracy         | MulticlassExactMatch      | 0
3 | test_char_error_rate  | CharErrorRate             | 0
4 | test_confusion_matrix | MulticlassConfusionMatrix | 0
--------------------------------------------------------------------
8.7 M     Trainable params
0         Non-trainable params
8.7 M     Total params
34.965    Total estimated model params size (MB)
```

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=10` reached.
INFO:pytorch_lightning.utilities.rank_zero:Restoring states from the checkpoint path at /c
ontent/checkpoints/best_checkpoint.ckpt
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.utilities.rank_zero:Loaded model weights from checkpoint at /conten
t/checkpoints/best_checkpoint.ckpt
```

	-	皖	沪	津	渝	冀	晋	蒙	辽	吉	黑	苏	浙	京	闽	赣	鲁	豫	鄂	湘	粤	桂	琼	川	贵	云	藏
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
皖	0	9474	1	0	1	0	0	5	0	0	0	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
沪	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
津	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
渝	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
冀	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
晋	0	0	0	0	1	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
蒙	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
辽	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
吉	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
黑	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
苏	0	0	0	0	1	0	0	7	0	1	0	203	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
浙	0	0	0	0	1	0	0	0	0	0	0	0	79	0	0	0	0	3	0	0	0	0	0	0	0	0	0
京	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
闽	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
赣	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0
鲁	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
豫	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0
鄂	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0
湘	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
粤	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	25	0	0	0	0	0	0
桂	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
琼	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
川	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
贵	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
云	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
藏	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
陕	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
甘	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
青	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	-	皖	沪	津	渝	冀	晋	蒙	辽	吉	黑	苏	浙	京	闽	赣	鲁	豫	鄂	湘	粤	桂	琼	川	贵	云	藏
宁	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
新	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
警	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
学	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	-	皖	沪	津	渝	冀	晋	蒙	辽	吉	黑	苏	浙	京	闽	赣	鲁	豫	鄂	湘	粤	桂	琼	川	贵	云	藏
7	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Test metric	DataLoader 0
test_acc	0.979997992515564
test_char_error_rate	0.0035574985668063164
test_loss	0.016605110839009285

Out[25]:

```
[{'test_loss': 0.016605110839009285,
  'test_acc': 0.979997992515564,
  'test_char_error_rate': 0.0035574985668063164}]
```

Анализ ошибок

Accuracy на тестовом датасете: 0.9799

Char Error Rate: 0.0036

Confusion Matrix:
На Confusion Matrix мы можем увидеть, что символ "皖" встречается на 9491 объектах (из 9999 объектов тестового датасета), из которых модель корректно распознает 9474.
Остальные иероглифы встречаются сильно реже. Например, символ "黑" встречается всего 3 раза, из которых только 1 раз модель распознала символ верно, а 2 раза допустила ошибку.
Но в общем, можно сказать, что модель неплохо справляется с распознаванием. Например, символ "辽" встречается всего 4 раза и все 4 раза модель распознала этот символ. То же верно и для символа "川".
Возможно, модель распознаёт их хорошо, несмотря на малое кол-во объектов, потому что эти символы визуально сильно отличаются от остальных. Есть символы очень сложные и очень похожие друг на друга, в таком случае может помочь добавление в обучающую выборку бОльшего кол-ва изображений с редковстречающимися символами.
С буквами латинского алфавита и цифрами модель справляется хорошо. Ошибки допускает относительно редко, но обучающий датасет в целом не очень большой и всё обучение заняло не более часа.

Дополнительные наблюдения: для девяти объектов из 9999 модель предсказывала последовательность длиной 8 символов вместо ожидаемых 7 (впоследствии я убрала ту часть кода, где выводила такие последовательности, чтобы код выглядел менее беспорядочно). Можно попробовать использовать CrossEntropyLoss вместо CTCLoss и посмотреть, как это скажется на итоговом качестве.

Загрузим нашу модель и попробуем сделать предсказание:

In [26]:

```
checkpoint = torch.load("/content/checkpoints/best_checkpoint.ckpt", map_location=device)
hyper_parameters = checkpoint["hyper_parameters"]

model = OCRModule(**hyper_parameters, label_converter=label_converter)

model_weights = checkpoint["state_dict"]

model.load_state_dict(model_weights)
model.eval()
```

```
Out[26]: OCRModule(
  (model): CRNN(
    (conv0): Sequential(
      (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (pool0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv1): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (conv3): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (pool2): MaxPool2d(kernel_size=(1, 2), stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (conv4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
      (2): ReLU()
    )
    (conv5): Sequential(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
      (2): ReLU()
    )
    (pool3): MaxPool2d(kernel_size=(1, 2), stride=(2, 1), padding=(0, 1), dilation=1, ceil
_mode=False)
    (conv6): Sequential(
      (0): Conv2d(512, 512, kernel_size=(2, 2), stride=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
      (2): ReLU()
    )
    (rnn): BidirectionalLSTM(
      (rnn): LSTM(512, 256, num_layers=2, bidirectional=True)
      (decoder): Linear(in_features=512, out_features=69, bias=True)
    )
  )
  (criterion): CTCLoss()
  (test_accuracy): MulticlassExactMatch()
  (test_char_error_rate): CharErrorRate()
  (test_confusion_matrix): MulticlassConfusionMatrix()
)
```

In [35]:

```
def predict(model, converter, img_path):
  img = cv2.imread(img_path)
  img = cv2.resize(img, (128,32))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  transform = transforms.ToTensor()
  img = transform(img)

  with torch.no_grad():
    pred = model(img.unsqueeze(0))

  print(f"predicted label: {converter.decode(pred.argmax(-1))}")
```

In [36]:

```
img_path = "/content/license_plate.jpg"
img = cv2.imread(img_path)
plt.imshow(img, cmap="gray")
plt.axis("off")
```

Out[36]:

```
(-0.5, 1199.5, 396.5, -0.5)
```

In [37]:

```
predict(model, label_converter, img_path)
```

```
predicted label: 闽GGL883
```