In [1]:
```
! pip install --quiet \
"setuptools==59.5.0" \
"pytorch-lightning>=1.4" \
"matplotlib" "torch>=1.8" \
"ipython[notebook]" \
"torchmetrics>=0.7" \
"torchvision" \
"seaborn" \
"opencv-python"
```

In [114]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import os
import torch
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
import torchvision
import string
from torchmetrics import CharErrorRate
import torch.nn.functional as F
```

In [ ]:

```
def accuracy(preds, target):
    N = len(preds)
    accurate = 0
    for i in range(len(preds)):
        if preds[i] == target[i]:
            accurate += 1
    return accurate/N
```

Соберем все символы, которые могут встретиться в датасете для последущего кодирования

In [182]:
```python
#Провинции в Китае, иероглифы которых могут встретиться на автомобильных номерах
dict_provinces = {
    '粤': 'Гуандун',
    '黑': 'Хэйлунцзян',
    '蒙': 'Автономный район Внутренняя Монголия',
    '琼': 'Хайнань',
    '辽': 'Ляонин',
    '鲁': 'Шаньдун',
    '贵': 'Фуцзянь',
    '浙': 'Чжецзян',
    '新': 'Синьцзян-Уйгурский Автономный Район',
    '吉': 'Провинция Цзилинь',
    '皖': 'Аньхой',
    '甘': 'Ганьсу',
    '闽': 'Гуйчжоу',
    '冀': 'Хэбэй',
    '豫': 'Хэнань',
    '鄂': 'Хубэй',
    '湘': 'Хунань',
    '苏': 'Цзянсу',
    '赣': 'Цзянси',
    '青': 'Цинхай',
    '陕': 'Шэньси',
    '晋': 'Шаньси',
    '川': 'Сычуань',
    '云': 'Юньнань',
    '桂': 'Гуанси-Чжуанский Автономный Округ',
```

```
    '桂': 'Гуанси-Чжуанский Автономный Округ',
    '宁': 'Нинся-Хуэйский Автономный Округ',
    '藏': 'Автономный Округ Тибет',
    '京': 'Beijing',
    '渝': 'Chongqing',
    '沪': 'Shanghai',
    '津': 'Tianjin'
}

NUMBER = list(string.digits)
ALPHABET = list(string.ascii_uppercase)
Hieroglyphics = list(dict_provinces.keys())
ALL_CHAR_SET = NUMBER + ALPHABET + Hieroglyphics
ALL_CHAR_SET_LEN = len(ALL_CHAR_SET)
```

In [183]:
```
ALL_CHAR_SET_LEN, len(Hieroglyphics)
```

Out[183]:
```
(67, 31)
```

In [184]:
```
data_path = '/Users/alenabuk/Downloads/CCPD2019-dl1'
```

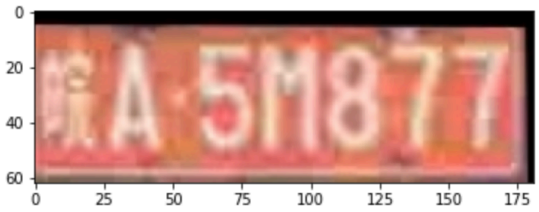
In [188]:
```
#Загрузим для примера картинку из train
img = cv2.imread(data_path + '/train/01-皖A5M877.jpg')
print(img.shape)
plt.imshow(img)
```

```
(62, 181, 3)
```

In [188]:
```
#Загрузим для примера картинку из train
img = cv2.imread(data_path + '/train/01-皖A5M877.jpg')
print(img.shape)
plt.imshow(img)
```

```
(62, 181, 3)
```

Out[188]:
```
<matplotlib.image.AxesImage at 0x7f94139c8eb0>
```




In [189]:
```
#Определим размеры картинок в train
cnt, mean_h, mean_w = 0, 0, 0
for i in os.listdir(data_path + '/train'):
    img = cv2.imread(data_path + '/train/' + f'{i}')

    if cnt == 0:
        min_shape_h, max_shape_h = img.shape[0], img.shape[0]
        min_shape_w, max_shape_w = img.shape[1], img.shape[1]

    if img.shape[0] > max_shape_h:
```

In [189]:

```python
#Определим размеры картинок в train
cnt, mean_h, mean_w = 0, 0, 0
for i in os.listdir(data_path + '/train'):
    img = cv2.imread(data_path + '/train/' + f'{i}')

    if cnt == 0:
        min_shape_h, max_shape_h = img.shape[0], img.shape[0]
        min_shape_w, max_shape_w = img.shape[1], img.shape[1]

    if img.shape[0] > max_shape_h:
        max_shape_h = img.shape[0]
        max_img_h = img
    if img.shape[0] < min_shape_h:
        min_shape_h = img.shape[0]
        min_img_h = img

    if img.shape[1] > max_shape_w:
        max_shape_w = img.shape[1]
        max_img_w = img
    if img.shape[1] < min_shape_w:
        min_shape_w = img.shape[1]
        min_img_w = img

    h, w = img.shape[0], img.shape[1]

    mean_h = ((mean_h * cnt) + h)/(cnt + 1)
    mean_w = ((mean_w * cnt) + w)/(cnt + 1)
    cnt += 1
```

```
    mean_h = ((mean_h * cnt) + h)/(cnt + 1)
    mean_w = ((mean_w * cnt) + w)/(cnt + 1)
    cnt += 1
```

In [190]:
```
# (наименьший, наибольший, средний) по высоте размер картинки
min_shape_h, max_shape_h, mean_h
```

Out[190]: (28, 239, 84.81559155915525)

In [191]:
```
# (наименьший, наибольший, средний) по ширине размер картинки
min_shape_w, max_shape_w, mean_w
```

Out[191]: (74, 719, 249.84530453045298)

In [192]:
```
#картинка с наибольшим значением по высоте
max_img_h.shape
```

Out[192]: (239, 692, 3)

In [193]:
```
#картинка с наибольшим значением по ширине
max_img_w.shape
```

Out[193]: (211, 719, 3)

In [194]:
```
#Посмотрим на самую высокую картинку
plt.imshow(transforms.ToPILImage()(transforms.ToTensor()(max_img_h)), interpolation="bicubic")
```

In [194]:
```
#Посмотрим на самую высокую картинку
plt.imshow(transforms.ToPILImage()(transforms.ToTensor()(max_img_h)), interpolation="bicubic")
```

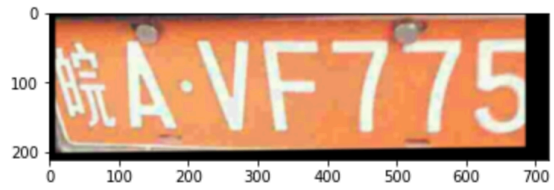
Out[194]: <matplotlib.image.AxesImage at 0x7f94175a4580>


In [195]:
```
картинка после приведения к наименьшим в train датасете параметрам
how(transforms.ToPILImage()(transforms.Resize((28, 74))(transforms.ToTensor()(max_img_h))), interpolation="bicubic")
```

Out[195]: <matplotlib.image.AxesImage at 0x7f9406799be0>

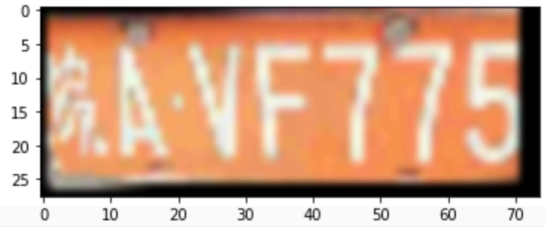
In [196]:
```
#Посмотрим на самую широкую картинку
plt.imshow(transforms.ToPILImage()(transforms.ToTensor()(max_img_w)), interpolation="bicubic")
```

Out[196]: <matplotlib.image.AxesImage at 0x7f93fd6868b0>




In [197]:
```
#Та же картинка после приведения к наименьшим в train датасете параметрам
plt.imshow(transforms.ToPILImage()(transforms.Resize((28, 74))(transforms.ToTensor()(max_img_w))), interpolation="bi
```

Out[197]: <matplotlib.image.AxesImage at 0x7f9405a7a130>

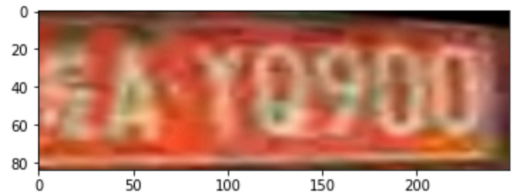


In [198]:
```
#Посмотрим на самую маленькую по ширине картинку
transforms.ToPILImage()(min_img_w)
```

Out[198]:

In [199]:
```
#Та же картинка после приведения к средним в train датасете параметрам
plt.imshow(transforms.ToPILImage()(transforms.Resize((84, 249))(transforms.ToTensor()(min_img_w))), \
           interpolation="bicubic")
```

Out[199]:
```
<matplotlib.image.AxesImage at 0x7f94128bab20>
```




Лучше приводить к наименьшим размерам, так как даже самые высокие и широкие картинки несильно теряют в качестве по сравнению с растяжением

В качестве дефолтных трансформаций будем использовать ToTensor() и Resize((28, 74))

В качестве дефолтных трансформаций будем использовать ToTensor() и Resize((28, 74))

In [201]:
```
max_img_w.shape
```

Out[201]:
```
(211, 719, 3)
```

In [202]:
```
transforms.ToTensor()(max_img_w).shape
```

Out[202]:
```
torch.Size([3, 211, 719])
```

In [203]:
```
transforms.Resize((28, 74))(transforms.ToTensor()(max_img_w)).shape
```

Out[203]:
```
torch.Size([3, 28, 74])
```

In [204]:
```
#Кодировщик лейблов в разреженный вектор
def encode(a):
    onehot = [0]*ALL_CHAR_SET_LEN
    idx = ALL_CHAR_SET.index(a)
    onehot[idx] += 1
    return onehot
```

In [216]:
```
class PlateDetDataset(Dataset):
    def __init__(self, path=None, train=False,
                 transform=transforms.Compose([transforms.ToTensor(), transforms.Resize((28, 74))])
                ):
        super(Dataset, self).__init__()
```

In [216]:

```
class PlateDetDataset(Dataset):
    def __init__(self, path=None, train=False,
                 transform=transforms.Compose([transforms.ToTensor(), transforms.Resize((28, 74))])
                ):
        super(Dataset, self).__init__()
        self.transform = transform
        self.train = train
        if path is None:
            return 'No path to file'
        else:
            self.data_path = path

            if self.train:
                self.train_file_names = [p for p in os.listdir(self.data_path + '/train')]
                self.train_labels = [p.split('-')[1].split('.')[0] for p in self.train_file_names]

            else:
                self.test_file_names = [p for p in os.listdir(self.data_path + '/test')]
                self.test_labels = [p.split('-')[1].split('.')[0] for p in self.test_file_names]

    def __len__(self):
        if self.train:
            return len(self.train_file_names)
        else:
            return len(self.test_file_names)

    def __getitem__(self, index):
        if self.train:
            self.label_train_en = []
```

```
            return len(self.test_file_names)

    def __getitem__(self, index):
        if self.train:
            self.label_train_en = []

            for i in self.train_labels[index]:
                self.label_train_en += encode(i)

            return self.transform(cv2.imread(self.data_path + '/train/' + self.train_file_names[index])), \
                   np.array(self.label_train_en), \
                   self.train_labels[index]

        else:
            self.label_test_en = []

            for i in self.test_labels[index]:
                    self.label_test_en += encode(i)

            return self.transform(cv2.imread(self.data_path + '/test/' + self.test_file_names[index])), \
                   np.array(self.label_test_en), \
                   self.test_labels[index]
```

In [318]:

```
#Сделаем dataloaders

train_dataset = PlateDetDataset(path=data_path, train=True)
data_train, data_val = torch.utils.data.random_split(
                            PlateDetDataset(path=data_path, train=True),
                            [int(len(train_dataset)*0.8), int(len(train_dataset)*0.2)])

data_test = PlateDetDataset(path=data_path, train=False)

try:
    #если картинки одинакового размера
    data_train_loader = DataLoader(data_train, batch_size=64, shuffle=True)
    data_val_loader = DataLoader(data_val, batch_size=64, shuffle=False)
    data_test_loader = DataLoader(data_test, batch_size=64, shuffle=False)

except:
    #используется, если картинки разного размера
    def my_collate(batch):
        data = [item[0] for item in batch]
        target = [item[1] for item in batch]
        return [data, target]

    data_train_loader = DataLoader(data_train, batch_size=64, collate_fn=my_collate, shuffle=True)
    data_val_loader = DataLoader(data_val, batch_size=64, collate_fn=my_collate, shuffle=False)
    data_test_loader = DataLoader(data_test, batch_size=64, collate_fn=my_collate, shuffle=False)
```

```
data_test_loader = DataLoader(data_test, batch_size=64, collate_fn=my_collate, shuffle=False)
```

In [218]:
```
data_train[0][0].shape
```

Out[218]:
```
torch.Size([3, 28, 74])
```

In [219]:
```
#torch.Size([64, 462]) - BATCH_SIZE, ALL_CHAR_SET_LEN * PIC_WORD_LEN
for batch_idx, samples in enumerate(data_train_loader):
    print(batch_idx, samples[0].shape, samples[1].shape, len(samples[2]))
    break
```

```
0 torch.Size([64, 3, 28, 74]) torch.Size([64, 469]) 64
```

In [220]:
```
#torch.Size([64, 462]) - BATCH_SIZE, ALL_CHAR_SET_LEN * PIC_WORD_LEN
for batch_idx, samples in enumerate(data_test_loader):
    print(batch_idx, samples[0].shape, samples[1].shape, len(samples[2]))
    break
```

```
0 torch.Size([64, 3, 28, 74]) torch.Size([64, 469]) 64
```

In [221]:
```
len(data_train), len(data_val), len(data_test)
```

Out[221]:
```
(159984, 39996, 9999)
```

Модель: Попробуем классическую сверточную нейросеть с остаточными (residule) слоями, она будет хуже rnn или трансформеров, но остается не очень много времени

Модель: Попробуем классическую сверточную нейросеть с остаточными (residule) слоями, она будет хуже rnn или трансформеров, но остается не очень много времени

In [376]:
```
class ResBlock(nn.Module):
    def __init__(self, n_chans):
        super(ResBlock, self).__init__()
        self.conv = nn.Conv2d(n_chans, n_chans, kernel_size=3,
                              padding=1, bias=False)  # <1>
        self.batch_norm = nn.BatchNorm2d(num_features=n_chans)
        torch.nn.init.kaiming_normal_(self.conv.weight,
                                      nonlinearity='relu')  # <2>
        torch.nn.init.constant_(self.batch_norm.weight, 0.5)
        torch.nn.init.zeros_(self.batch_norm.bias)

    def forward(self, x):
        out = self.conv(x)
        out = self.batch_norm(out)
        out = torch.relu(out)
        return out + x
```

```
        return out + x
```

In [379]:
```
class NetResDeep(nn.Module):
    def __init__(self, n_chans1=64, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(64 * 3 * 9, 512)
        self.fc = nn.Linear(in_features=512, out_features=ALL_CHAR_SET_LEN*MAX_LEN, bias=True)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 64 * 3 * 9)
        out = torch.relu(self.fc1(out))
        out = self.fc(out)
        return out
```

In [380]:
```
#Проверим вывод для одной картинки
x = transforms.Resize((28, 74))(transforms.ToTensor()(max_img_h))
NetResDeep().forward(x.unsqueeze(0)).shape
```

Out[380]:
```
torch.Size([1, 469])
```

In [382]:

```
from torchvision import models
import torch.nn as nn
# from fastai.vision import Path
import torch
from torch.autograd import Variable

MAX_LEN = 7 #число символов на картинке

# model = models.resnet18(pretrained=False)
# model.conv1 = nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
# model.fc = nn.Linear(in_features=512, out_features=ALL_CHAR_SET_LEN*MAX_LEN, bias=True)
model = NetResDeep(n_blocks=100)
loss_func = nn.MultiLabelSoftMarginLoss()
optm = torch.optim.Adam(model.parameters(), lr=0.001)

#Обучение модели
#Очень долго считается без gru, поэтому стоит 1 эпоха
cnt_new = 0
for epoch in range(1):
    for step, i in enumerate(data_train_loader):
        img, label_oh, label = i
        img = Variable(img)
        label_oh = Variable(label_oh.float())
        pred = model(img)
        loss = loss_func(pred, label_oh)
        optm.zero_grad()
        loss.backward()
```

```python
# model.fc = nn.Linear(in_features=512, out_features=ALL_CHAR_SET_LEN*MAX_LEN, bias=True)
model = NetResDeep(n_blocks=100)
loss_func = nn.MultiLabelSoftMarginLoss()
optm = torch.optim.Adam(model.parameters(), lr=0.001)

#Обучение модели
#Очень долго считается без gru, поэтому стоит 1 эпоха
cnt_new = 0
for epoch in range(1):
    for step, i in enumerate(data_train_loader):
        img, label_oh, label = i
        img = Variable(img)
        label_oh = Variable(label_oh.float())
        pred = model(img)
        loss = loss_func(pred, label_oh)
        optm.zero_grad()
        loss.backward()
        optm.step()
        print('epoch:', epoch+1, 'step:', step+1, 'loss:', loss.item())
```

```
epoch: 1 step: 1664 loss: 0.009112097322940826
epoch: 1 step: 1665 loss: 0.008505580015480518
epoch: 1 step: 1666 loss: 0.006594761740416288
epoch: 1 step: 1667 loss: 0.006857633590698242
epoch: 1 step: 1668 loss: 0.00733698345720768
epoch: 1 step: 1669 loss: 0.007240569218993187
epoch: 1 step: 1670 loss: 0.007505036424845457
epoch: 1 step: 1671 loss: 0.006932688411325216
```

```
        optm.step()
        print('epoch:', epoch+1, 'step:', step+1, 'loss:', loss.item())
```

```
epoch: 1 step: 2482 loss: 0.0050456352503926754
epoch: 1 step: 2483 loss: 0.002866320312023163
epoch: 1 step: 2484 loss: 0.0037900081370025873
epoch: 1 step: 2485 loss: 0.00343899754807353
epoch: 1 step: 2486 loss: 0.0030810730531811714
epoch: 1 step: 2487 loss: 0.003203991334885359
epoch: 1 step: 2488 loss: 0.0031308967154473066

epoch: 1 step: 2489 loss: 0.0027209664694964886
epoch: 1 step: 2490 loss: 0.004158376716077328
epoch: 1 step: 2491 loss: 0.004608082585036755
epoch: 1 step: 2492 loss: 0.002816480817273259
epoch: 1 step: 2493 loss: 0.003016311675310135
epoch: 1 step: 2494 loss: 0.0026756608858704567
epoch: 1 step: 2495 loss: 0.003271259367465973
epoch: 1 step: 2496 loss: 0.004935352131724358
epoch: 1 step: 2497 loss: 0.0037575310561805964
epoch: 1 step: 2498 loss: 0.004013530444353819
epoch: 1 step: 2499 loss: 0.0039054432418197393
epoch: 1 step: 2500 loss: 0.0026149749755859375
```

In [383]:
```
#Теперь прогоним на test датасете
cnt = 0
model.eval();
cnt, mean_acc, mean_CER = 0, 0, 0
```

```
epoch: 1 step: 2499 loss: 0.003905443241819739
epoch: 1 step: 2500 loss: 0.0026149749755859375
```

In [383]:
```
#Теперь прогоним на test датасете
cnt = 0
model.eval();
cnt, mean_acc, mean_CER = 0, 0, 0
for step, (img, label_oh, label) in enumerate(data_test_loader):
    img = Variable(img)
    pred = model(img)
    pred = torch.reshape(pred, (pred.shape[0], MAX_LEN, ALL_CHAR_SET_LEN))
    max_elems = torch.max(pred, 2)[1]
    pred_l = [''.join([ALL_CHAR_SET[i] for i in max_elems[j, :]]) for j in range(max_elems.shape[0])]
    metric = CharErrorRate()
    mean_acc += accuracy(pred_l, label)
    mean_CER += metric(pred_l, label)
#     print(f'CharErrorRate: {metric(pred_l, label)}, Accuracy: {accuracy(pred_l, label)}')
    cnt += 1
print(f'CharErrorRate: {mean_CER/cnt}, Accuracy: {mean_acc/cnt}')
```

```
CharErrorRate: 0.6857007741928101, Accuracy: 0.0
```

In [385]:
```
#сохраним веса модели
torch.save(model.state_dict(), '/Users/alenabuk/Tink_saved_model/tink_model.pt')
```

In [402]:
```
#Модель переобучилась
pred_l[0:10]
```

```
CharErrorRate: 0.6857007741928101, Accuracy: 0.0
```

In [385]:
```
#сохраним веса модели
torch.save(model.state_dict(), '/Users/alenabuk/Tink_saved_model/tink_model.pt')
```

In [402]:
```
#Модель переобучилась
pred_l[0:10]
```

Out[402]:
```
['皖AK889F',
 '皖AK889F',
 '皖AK809F',
 '皖AK889F',
 '皖AK889F',
 '皖AK889F',
 '皖AK889F',
 '皖AK889F',
 '皖AK889F',
 '皖AK889F']
```

Так делать не стоило

In [405]:
```
img = cv2.imread('/Users/alenabuk/Downloads/funny-sad-cat-Favim.com-7790179.jpg')
plt.imshow(img)
```

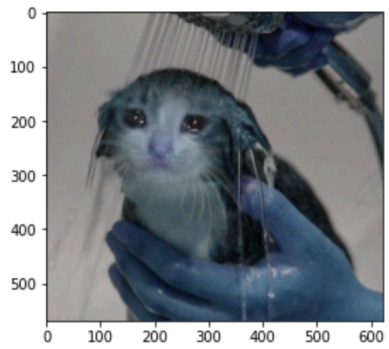
Out[405]:
```
<matplotlib.image.AxesImage at 0x7f9403e24250>
```

```
 '皖AK889F',
 '皖AK889F']
```

Так делать не стоило

In [405]:
```
img = cv2.imread('/Users/alenabuk/Downloads/funny-sad-cat-Favim.com-7790179.jpg')
plt.imshow(img)
```

Out[405]:
```
<matplotlib.image.AxesImage at 0x7f9403e24250>
```




Но на models.resnet18 вообще неплохо выходило, где-то 0.9 accuracy, ~0.14 CharErrorRate, просто тут реализация не такой хорошей получилась