Филиал Московского Государственного Университета
имени М.В.Ломоносова в городе Ташкенте

Факультет прикладной математики и информатики

Кафедра прикладной математики и информатики

Ким Семён Валерьевич

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему: «Оптимизация нейросетевого классификатора при его специализации»

по направлению 01.03.02 «Прикладная математика и информатика»

ВКР рассмотрена и рекомендована к защите

зав.кафедрой «ПМиИ», д.ф.-м.н., профессор_____Кудрявцев В.Б.

Научный руководитель

Кандидат физико-математических наук, доцент _____Часовских А. А.

«_____»_____ 2020г.

Ташкент 2020 г.

Оглавление

Аннотация

Построен генератор, который по заданному шаблону автомобильных номеров и шрифтам символов строит аугментированный набор данных для обучения CNN системы распознавания автомобильных номерных знаков.

Сгенерировав обучающие наборы номеров как без ограничения на позиции символов, так и с ограничениями на них. На каждом из таких наборов выполнено обучение нейронной сети. Эти сети протестированы на сгенерированной базе номеров. Оказалось, что специализация при обучении приводит к повышению качества распознавания с 78,5% до 99,7%.

A generator has been built which based on a given pattern of license plates and character fonts, builds an augmented data set for training CNN license plate recognition systems.

By generating training sets of license plates, both without restriction on the positions of characters, and with restrictions on them. On each of these sets, neural network training was performed. These networks are tested on the generated license plate base. It turned out that specialization in training leads to an increase in the quality of recognition from 78.5% to 99.7%.

§1 ВВЕДЕНИЕ

Актуальность работы. Автоматическое распознавание автомобильных номерных знаков является сложной и важной задачей, которая используется в управлении дорожным движением, цифровом видеонаблюдении, распознавании транспортных средств, управлении парковками в больших городах. Эта задача представляет собой сложную проблему из-за многих факторов, которые включают, но не ограничиваются: размытые изображения, плохие условия освещения, изменчивость номеров автомобильных номеров (включая специальные символы, *например,* логотипы для Китая, Японии), физическое воздействие (деформации), погодные условия.

Эта работа посвящена проблеме распознавания автомобильных номерных знаков и представляет метод end-to-end глубоких нейронных сетей, который предназначен для работы без предварительной сегментации и последующего распознавания символов. В данной работе не рассматривается проблема обнаружения номерных знаков.

В настоящее время существует большое множество различных вариаций автомобильных номерных знаков, различающиеся алфавитом используемых символов и их количеством находящихся на номерной плате, а также фиксированной позицией каждого символа в номере. В связи с этим при построении нейросетевого классификатора возникает зависимость результата распознавания автомобильного номера от количества символов на номерной плате.

Целью данной работы является оптимизация нейросетевого классификатора при его специализации.

Задачи данной работой являются:

Построение генератора автомобильных номерных знаков.

Построение (end-to-end) архитектуры глубокой нейронной сети.

Сравнение результатов работы нейронной сети при разграничении алфавита допустимых символов в автомобильном номерном знаке.

Научная новизна работы состоит в том, что было выполнено сравнение качества распознавания автомобильных номеров в зависимости от количества символов для каждой позиции номерного знака.

Практическая значимость состоит в том, что построенная end-to-end архитектура достаточно надежна, чтобы справляться со сложными случаями, такими как искажения, зависящие от перспективы и камеры, жесткие условия освещения и т. д.

Практическая часть дипломной работы проводилась на платформе Google Colaboratory [12]; На бесплатном графическом процессоре Tesla K80; на языке программирования Python версии 3; с использование библиотек: Keras**,** TensorFlow**,** PyTorch**,** OpenCV и др.

Работа организована следующим образом. §2 описывает общий алгоритм распознавания автомобильного номера. В §3 рассмотрены генератор автомобильных номеров, архитектура и детали модели end-to-end сети. §4 включает в себя результаты сравнения классификатора на номерных знаках, в зависимости от размера используемого алфавита символов и их позиций.

§2 Технологии распознавания автомобильных номерных знаков

В более ранних работах по общему распознаванию License Plate(автомобильных номеров), общий алгоритм состоял из этапов: [3]

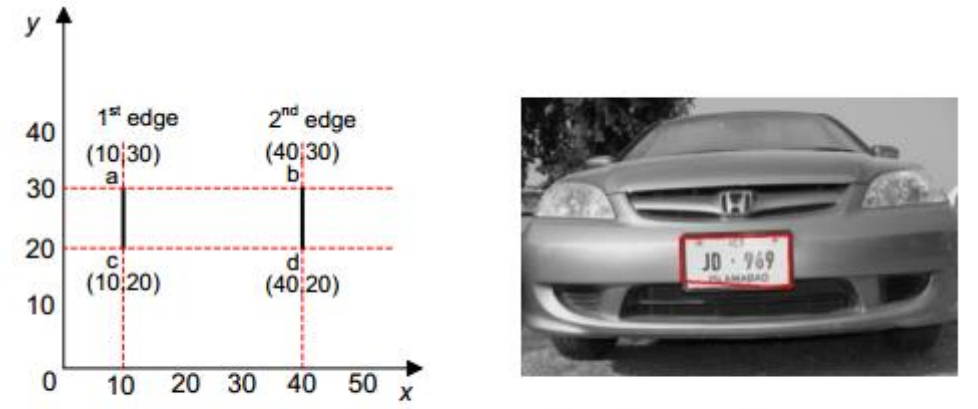
1. Предварительный поиск номера – обнаружение области, в которой содержится номер.
2. Нормализация номера – определение точных границ номера, нормализация контраста.
3. Распознавание символов – чтение всего, что нашлось в нормализованном изображении.

Алгоритмы предварительного поиска.

Анализ границ и фигур, контурный анализ. – поиск прямоугольного контура.

Рисунок 1 — Пример выделения границ для определения положения автомобильных номеров. Оригинальное изображение и результат.

Анализ только части границ. Выделяются контуры, после чего ищутся все вертикальные прямые. (рис.2).



(c) Edges having same or almost same x & y coordinates

(d) Vehicle number plate detected

Рисунок 2

Гистограммный анализ регионов. (рис. 3). частотная характеристика региона с номером отлична от частотной характеристики окрестности.

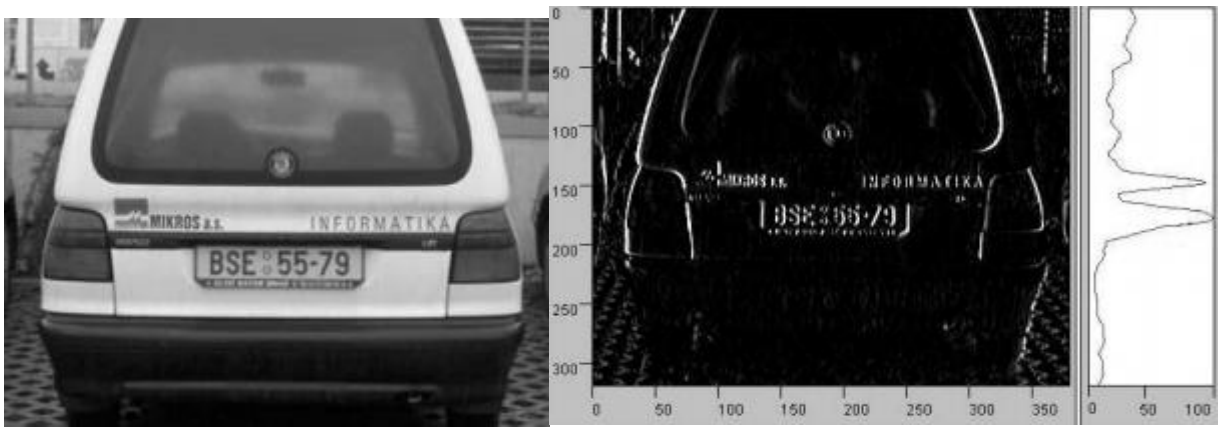


Рисунок 3

Статистический анализ. Минус всех предыдущих методов в том, что на реальных, запачканных грязью номерах нет ни выраженных границ. Также очень популярная архитектура CNN YOLO [5] или You Only Look Once.

Алгоритмы нормализации

Большинство указанных выше алгоритмов обнаруживают номер не точно и требуют дальнейшего уточнения его положения. Например, в данном случае требуется поворот и обрезка краёв:

Рисунок 4

Поворот номера в горизонтальную ориентацию.Самый простой фильтр, способный выделить такие прямые – преобразование Хафа (рис.5):

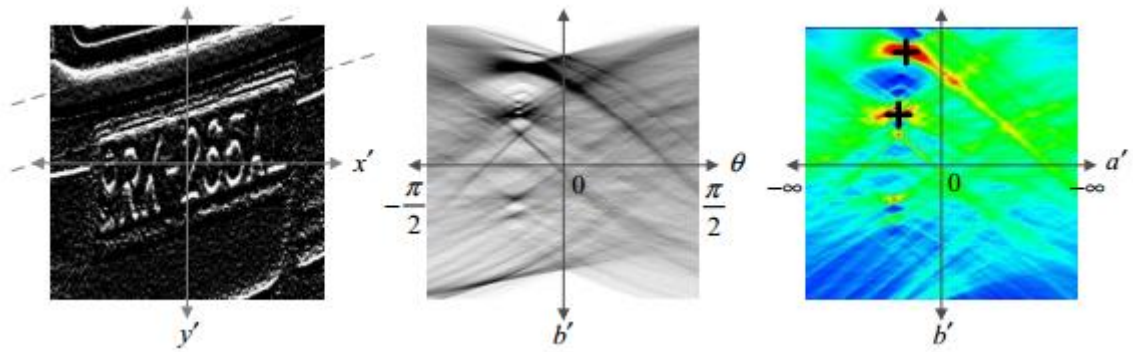


Рисунок 5

Преобразование Хаффа позволяет очень быстро выделить две главных прямых и обрезать по ним изображение (рис.6):

Рисунок 6

Увеличение контраста. (рис.7):

Рисунок 7

Разбиение на буквы. (рис.8).

Рисунок 8

Алгоритмы распознавания символов

Задача распознавания текста или отдельных символов (optical character recognition, OCR) .

1. **Tesseract OCR.** 2. **K-nearest.** 3. **Нейросети.**

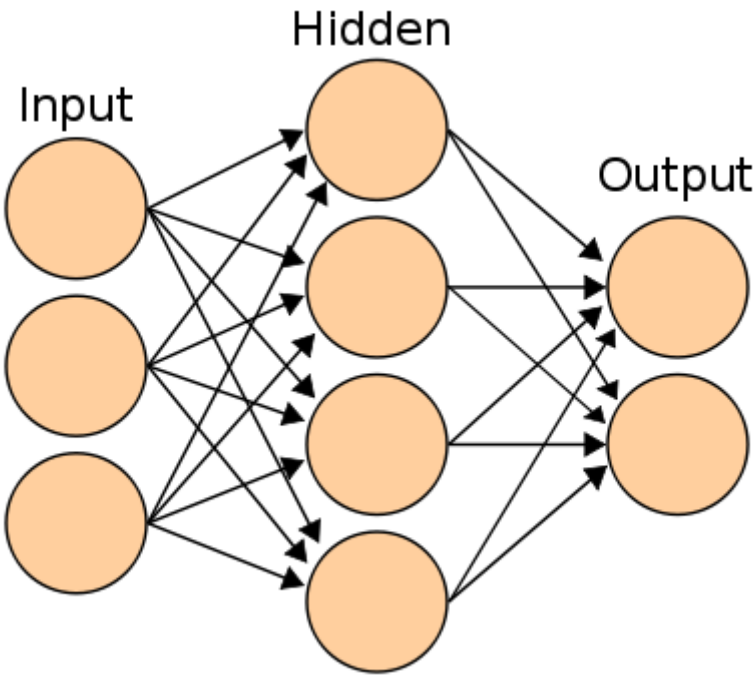


Рисунок 9

— deep-learning нейросети и сверточные сети. [2][6]

Замечание. Сегментация символов обычно использует различные алгоритмы ручной работы, сочетающие в себе проекции, связность и контурные компоненты изображения. В качестве входных данных используется двоичное изображение или промежуточное представление, поэтому на качество сегментации символов сильно влияют шум входного изображения, низкое разрешение, размытость или деформации.

Поскольку классификация следует за сегментацией символов, качество сквозного распознавания в значительной степени зависит от применяемого метода сегментации.

Чтобы решить проблему сегментации символов, были предложены комплексные решения на основе сверточных нейронных сетей (CNN), в которых весь LP-образ вводится как входной и создается выходная последовательность символов.

В моем подходе не были использованы созданные вручную функции над бинаризованным изображением – вместо этого были использованы необработанные пиксели RGB в качестве входных данных сверточной модели, выход которой интерпретируется как последовательность вероятностей символов. Для регуляризации использовался метод Batch Normalization.

§3 Разработка и реализация алгоритмов

3.1 Генератор автомобильных номеров

Наш конечный автомобильный номер будет принадлежать к региону: Россия. И иметь вид «тип 1 – однострочный». Согласно ГОСТ Р 50577-2018 «Знаки государственные регистрационные транспортных средств. Типы и основные размеры. Технические требования» большинство регистрационных знаков имеют вид (Тип 1 – регистрационные знаки для легковых, грузовых автомобилей и автобусов).

Рисунок 1 — Тип 1 (однострочный)

Стандартные размеры регистрационного знака: 520×112 мм. Символьные комбинации на **стандартных** номерных знаках определяются тремя буквами и тремя цифрами. Вышеприведенный ГОСТ Р 50577-2018 ссылается на использование 10 цифр (0-9) и 12 букв кириллицы, которые имеют графические аналоги в латинском алфавите: A, B, E, K, M, H, O, P, C, T, У и X. Также я добавил еще одну букву “D”, так как *буква D используется в регистрационных знаках транспортных средств дипломатических представительств и торговых представительств иностранных компаний.* Буквы по размеру шрифта меньше, чем цифры. Начертание символов определяется шрифтом ЖР5 по ГОСТ 3489.2-71 «Шрифты типографские. Гарнитура Журнальная рубленая». [10] (рис.2)

1234567890

ABCDEHKMOPTXY

Рисунок 2

3.1.1 Описание генератора

1. Получаем последовательность символов состоящей из 3 букв и 5 цифр используя функцию random;

2. Устанавливаем символы в соответствии с ГОСТ Р 50577-2018. (рис.3)

Рисунок 3

3. Добавляем искажения в изображение:

а.1) Аффинные преобразования со случайным выбором угла наклона до 20 градусов.

а.2) Аффинные преобразования со случайным выбором угла поворота до 5 градусов.

б.1) Добавление нечеткости посредством смешивания исходного изображения с маской нечеткости.

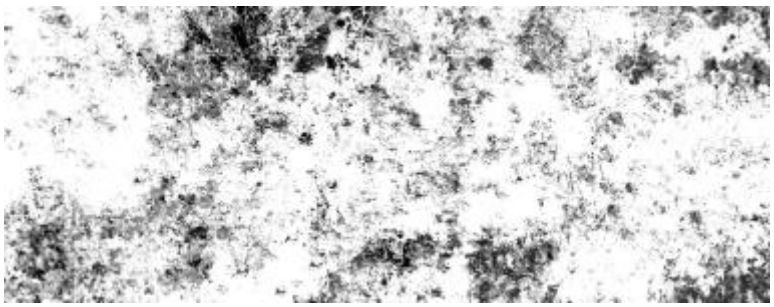


Рисунок 4 — Маска нечеткости

б.2) Добавление нечеткости посредством смешивания исходного изображения со случайной маской нечеткости из набора. (рис.5)

Рисунок 5 — Набор масок нечеткости

в) Добавление размытия blur.

г) Добавление шума в фильтры изображения RGB.

а.1

а.2

б.1

б.2

в

г

В результате получаем набор изображений: [7]

Рисунок 6

(Программная реализация см. приложение 1)

3.2 Архитектура нейронной сети

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 140, 30, 3)	0	
conv2d_1 (Conv2D)	(None, 140, 30, 32)	896	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 140, 30, 32)	128	conv2d_1[0][0]
activation_1 (Activation)	(None, 140, 30, 32)	0	batch_normalization_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 70, 15, 32)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 70, 15, 64)	18496	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNor	(None, 70, 15, 64)	256	conv2d_2[0][0]
activation_2 (Activation)	(None, 70, 15, 64)	0	batch_normalization_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 35, 7, 64)	0	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 35, 7, 128)	73856	max_pooling2d_2[0][0]
batch_normalization_3 (BatchNor	(None, 35, 7, 128)	512	conv2d_3[0][0]
activation_3 (Activation)	(None, 35, 7, 128)	0	batch_normalization_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 17, 3, 128)	0	activation_3[0][0]
conv2d_4 (Conv2D)	(None, 15, 1, 256)	295168	max_pooling2d_3[0][0]
batch_normalization_4 (BatchNor	(None, 15, 1, 256)	1024	conv2d_4[0][0]
activation_4 (Activation)	(None, 15, 1, 256)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 15, 1, 1024)	263168	activation_4[0][0]
batch_normalization_5 (BatchNor	(None, 15, 1, 1024)	4096	conv2d_5[0][0]
activation_5 (Activation)	(None, 15, 1, 1024)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 15, 1, 13)	13325	activation_5[0][0]
conv2d_7 (Conv2D)	(None, 15, 1, 10)	10250	activation_5[0][0]
activation_6 (Activation)	(None, 15, 1, 13)	0	conv2d_6[0][0]
activation_7 (Activation)	(None, 15, 1, 10)	0	conv2d_7[0][0]
reshape_1 (Reshape)	(None, 15, 13)	0	activation_6[0][0]
reshape_2 (Reshape)	(None, 15, 10)	0	activation_7[0][0]
norm_layer_1 (NormLayer)	(None, 13)	15	reshape_1[0][0]
norm_layer_2 (NormLayer)	(None, 10)	15	reshape_2[0][0]
norm_layer_3 (NormLayer)	(None, 10)	15	reshape_2[0][0]
norm_layer_4 (NormLayer)	(None, 10)	15	reshape_2[0][0]
norm_layer_5 (NormLayer)	(None, 13)	15	reshape_1[0][0]
norm_layer_6 (NormLayer)	(None, 13)	15	reshape_1[0][0]

norm_layer_7 (NormLayer)	(None, 10)	15	reshape_2[0][0]
norm_layer_8 (NormLayer)	(None, 10)	15	reshape_2[0][0]
out1 (Activation)	(None, 13)	0	norm_layer_1[0][0]
out2 (Activation)	(None, 10)	0	norm_layer_2[0][0]
out3 (Activation)	(None, 10)	0	norm_layer_3[0][0]
out4 (Activation)	(None, 10)	0	norm_layer_4[0][0]
out5 (Activation)	(None, 13)	0	norm_layer_5[0][0]
out6 (Activation)	(None, 13)	0	norm_layer_6[0][0]
out7 (Activation)	(None, 10)	0	norm_layer_7[0][0]
out8 (Activation)	(None, 10)	0	norm_layer_8[0][0]

Total params: 681,295
Trainable params: 678,287
Non-trainable params: 3,008

Таблица 1 – Архитектура (e2e) нейронной сети

Входное изображение размера 140 x 30 (RGB). Предложенная архитектура (таблица 1) имеет в общей сложности 6 слоев свёртки, где у первых 4 слоев размер свёрточных фильтров фиксируются в 3x3. У двух последних размер фильтров 1x1 [9][11]. Свертки с размером фильтра 1x1 имеют ряд преимуществ, которые и были использованы в данной архитектуре. После 1 x 1 свертки мы значительно уменьшаем размер по глубине. Скажем, если исходный вход имеет 200 каналов, свертка 1 x 1 объединит эти каналы (функции) в один канал. Другое преимущество заключается в том, что после свертки 1 x 1 может быть добавлена нелинейная активация, такая как ReLU. Нелинейность позволяет сети изучать более сложные функции. Активации ReLU используются во всей сети после слоев свертки. Есть 3 max pooling слоя (рис.7) размером 2 × 2 и шаг 2, который уменьшает входную размерность в 8 раз. Наконец, Normlayer который по сути является реализацией полносвязного слоя, где матрица весов размерности 1 × 15 умножается на входную карту признаков размерности 15 × 1 × 13 (10). На выходе получим размерность 1 × 1 × 13 или 1 × 1 × 10, то

есть 13 нейронов на каждую букву и 10 нейронов на выходе на каждую цифру. После чего нейронная сеть выдает свои вероятности ответов.

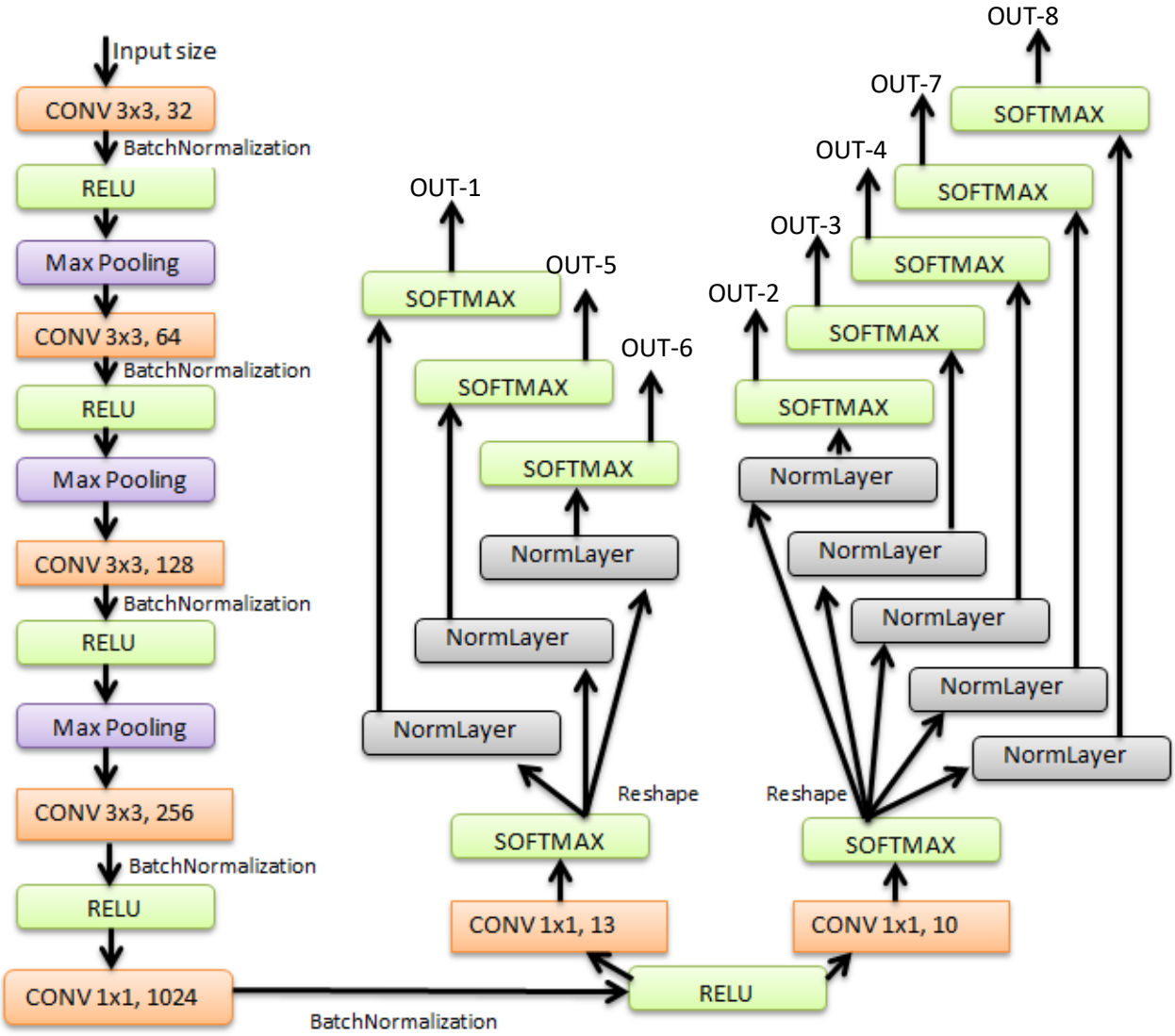


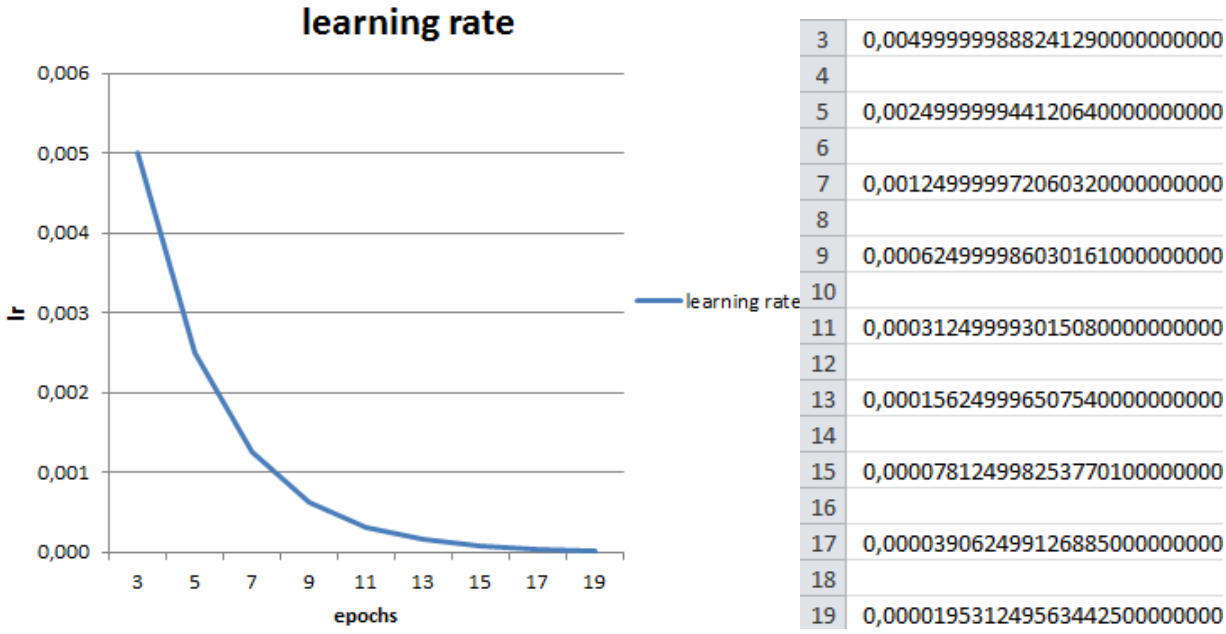
Рисунок 7

Для входного изображения с высотой H и шириной W и заданным шагом сети при $Ns = 2^3$ (3 слоя max pooling) карта характеристик выходного сигнала сети состоит из объема M × N × 128, где M = [H / Ns] и N = [W / Ns]. Далее после слоя свёртки размером фильтра 3x3 с параметром padding= VALID выходной сигнал сети состоит из объема $M' \times N' \times 256$, где $M' = M - 2 = 15$ и $N' = N - 2 = 1$. И после двух слоев свертки размером фильтра 1x1 получаем два выхода с объемом 15 × 1 × 13 для букв и 15 × 1 × 10 для цифр. Наконец применяя Reshape к выходам и пользовательский слой

NormLayer на выходе получаем 3 карты с глубиной 13 и 5 карт с глубиной 10 для букв и цифр соответственно. Для каждого символа на автомобильном номере на выходе сети есть 8 нейронов отвечающие за классификацию каждого символа в отдельности, как показано на рисунке выше.

3.2.1 Процедура обучения

Параметры обучения. Для обучения модели было создано 20000 изображений. Из них 18000 экземпляров для train и 2000 для valid.
validation_split=0.1;
Количество эпох epochs=20;
Размер батча batch_size=256;
Оптимизатор RMSPROP, где скорость обучения (learning rate) изменяется планировщиком скорости обучения (LearningRateScheduler) через каждые 2 эпохи; (рисунок 8)
Мера производительности модели классификации categorical_crossentropy с весами loss=[1,1,1,1,1,1,1] равными по всем выходам модели;
Метрика metrics=[‘accuracy’].



3	0,004999999888241290000000000
4	
5	0,002499999944120640000000000
6	
7	0,001249999972060320000000000
8	
9	0,000624999986030161000000000
10	
11	0,000312499993015080000000000
12	
13	0,000156249996507540000000000
14	
15	0,000078124998253770100000000
16	
17	0,000039062499126885000000000
18	
19	0,000019531249563442500000000

Рисунок 8 — График изменения learningrate

Процесс обучения:

```
Train on 18000 samples, validate on 2000 samples
Epoch 1/20
18000/18000 [======]  - loss: 31.0475  - acc:1,26E-08   - val_loss: 27.1350 - val_acc:6,172E-09

Epoch 2/20
18000/18000 [======]  - loss: 14.8178 - acc:0,0038 - val_loss: 13.1357 - val_acc:0,0078

Epoch 3/20
18000/18000 [======]  - loss: 5.4509 - acc: 0,5025 - val_loss: 5.7633 - val_acc: 0,3200

Epoch 4/20
18000/18000 [======]  - loss: 2.1769 - acc:0,9062 - val_loss: 11.8180 - val_acc: 0,01318

Epoch 5/20
18000/18000 [======]  - loss: 1.0476 - acc:0,9866  - val_loss: 1.1558 - val_acc: 0,9305

...

Epoch 16/20
18000/18000 [======]  - loss: 0.2354 - acc:0,9996  - val_loss: 0.2447 - val_acc:0,9990


Epoch 17/20
18000/18000 [======]  - loss: 0.2329 - acc: 0,9996 - val_loss: 0.2436 - val_acc:0,99850

Epoch 18/20
18000/18000 [======]  - loss: 0.2312 - acc:0,9995  - val_loss: 0.2418 - val_acc: 0,9990

Epoch 19/20
18000/18000 [======]  - loss: 0.2300 - acc:0,9996  - val_loss: 0.2411 - val_acc: 0,9990

Epoch 20/20
18000/18000 [======]  - loss: 0.2292 - acc:0,9997 - val_loss: 0.2401 - val_acc: 0,9990
```

Таблица 2 – Процесс обучения нейронной сети на 20000 изображениях.

Результат обучения:

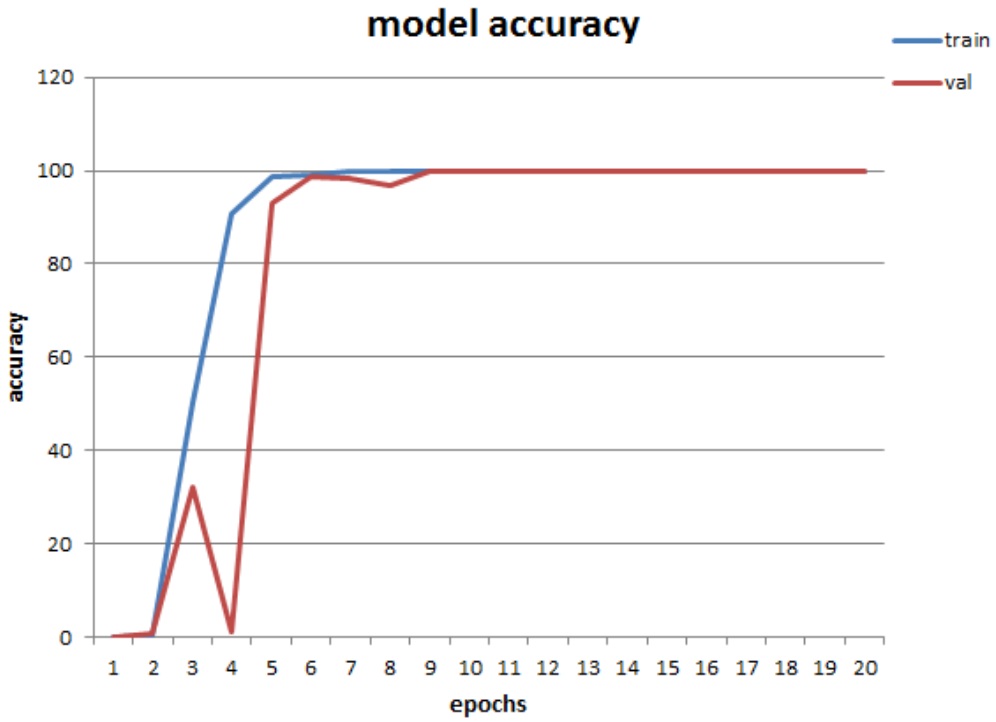


Рисунок 9 — График правильных ответов на выборках обучения и валидации.

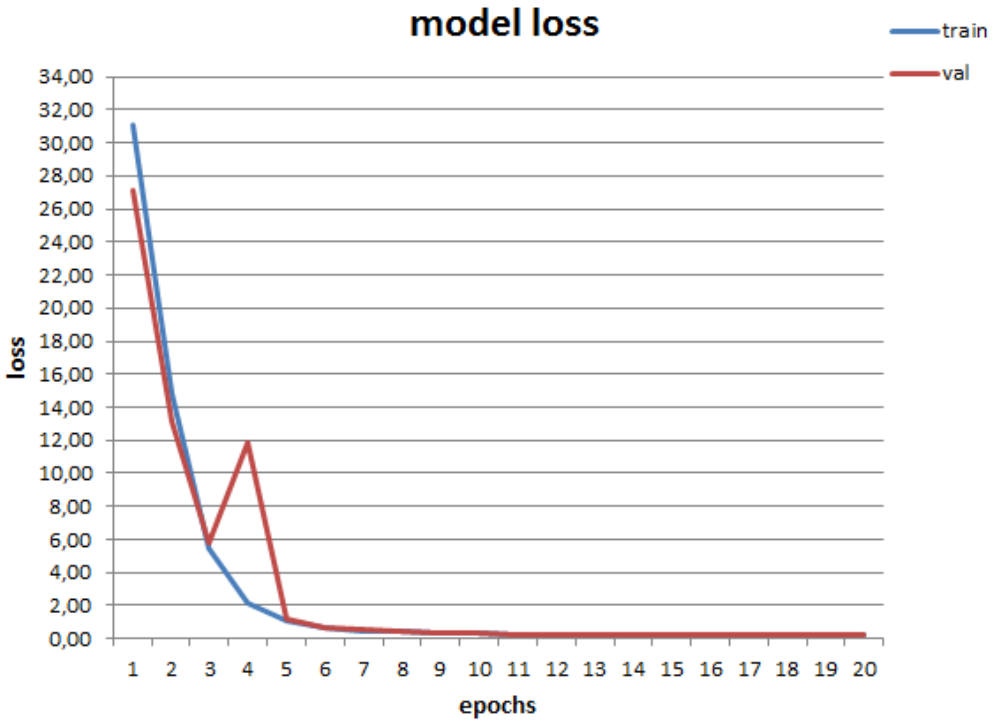


Рисунок 10 — График ответов функции потерь на выборках обучения и валидации.

(Программная реализация нейронной сети см. приложение 2)

§4 Оптимизация нейросетевого классификатора

Наша задача получить наивысший результат классификатора при распознавании российского номерного знака. Как отмечалось выше, в автомобильных номерах могут участвовать различное количество символов. Поэтому сначала рассмотрим задачу распознавания автомобильного номера с общим (не ограниченном) алфавитом допустимых символов. При том, что последовательность символов на номерном знаке может быть различной с длиной последовательности равной 8. Пусть у нас есть классификатор, предназначенный для распознавания таких номеров общего алфавита. И попробуем его обучить на таких наборах изображений.

Сначала сгенерируем изображения автомобильных номеров. Для определенности за основу автомобильного номера выберем Тип 1 (однострочный) из §3 Рисунок 1. Определим алфавит, который будет содержать все символы, использующиеся при создании номерных знаков:

```
PLATE_CHARS_LETTER = {"Ф", "Ц", "W", "Д", "Q", "Ш", "A", "B", "C", "D", "E",
"F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "R", "S", "T", "U", "V", "X", "Z",
"Б", "Г", "З", "И", "Л", "П", "У", "Ч", "Ь", "Э", "Я"}
Убрал: W Ж Д Q Ы Ю Ц У Ё Й Ъ
PLATE_CHARS_DIGIT = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
```

Алфавит символов состоит из части английского и части русского алфавитов, а также цифр 0-9. Некоторые буквы были убраны из-за их нестандартного размера или сильной похожести на уже выбранные буквы, например: Ё и Е.

Рисунок 1 — Пример выборки с номерами общего алфавита

Далее запустим обучение классификатора на 20000 изображениях.

```
Train on 18000 samples, validate on 2000 samples
Epoch 1/20
18000/18000 [======]  - loss: 36.6905  - acc:7.76E-14  - val_loss: 35.1175 - val_acc:3.595E-14

Epoch 2/20
18000/18000 [======]  - loss: 29.6739 - acc:2.478E-07 - val_loss: 31.7566 - val_acc:1.795E-11

Epoch 3/20
18000/18000 [======]  - loss: 23.311 - acc: 0.0010 - val_loss: 29.0472 - val_acc:3.599E-07

Epoch 4/20
18000/18000 [======]  - loss: 17.2349 - acc:0.0264 - val_loss: 24.6103 - val_acc: 0.00013

Epoch 5/20
18000/18000 [======]  - loss: 12.6845 - acc: 0.12833 - val_loss: 13.0675 - val_acc: 0.1123

...

Epoch 16/20
18000/18000 [======]  - loss: 4.5396 - acc:0.8428 - val_loss: 4.676 - val_acc: 0.8309

Epoch 17/20
18000/18000 [======]  - loss: 4.5155 - acc:0.8429  - val_loss: 4.6627 - val_acc: 0.8300

Epoch 18/20
18000/18000 [======]  - loss: 4.5003 - acc:0.8432  - val_loss: 4.6483 - val_acc: 0.8300

Epoch 19/20
18000/18000 [======]  - loss: 4.4878 - acc:0.8430  - val_loss: 4.6414 - val_acc: 0.8300

Epoch 20/20
18000/18000 [======]  - loss: 4.48 - acc:0.8431 - val_loss: 4.6333 - val_acc: 0.8300
```

Рисунок 2 — Процесс обучения классификатора на наборе вида рис.1

Результат обучения:

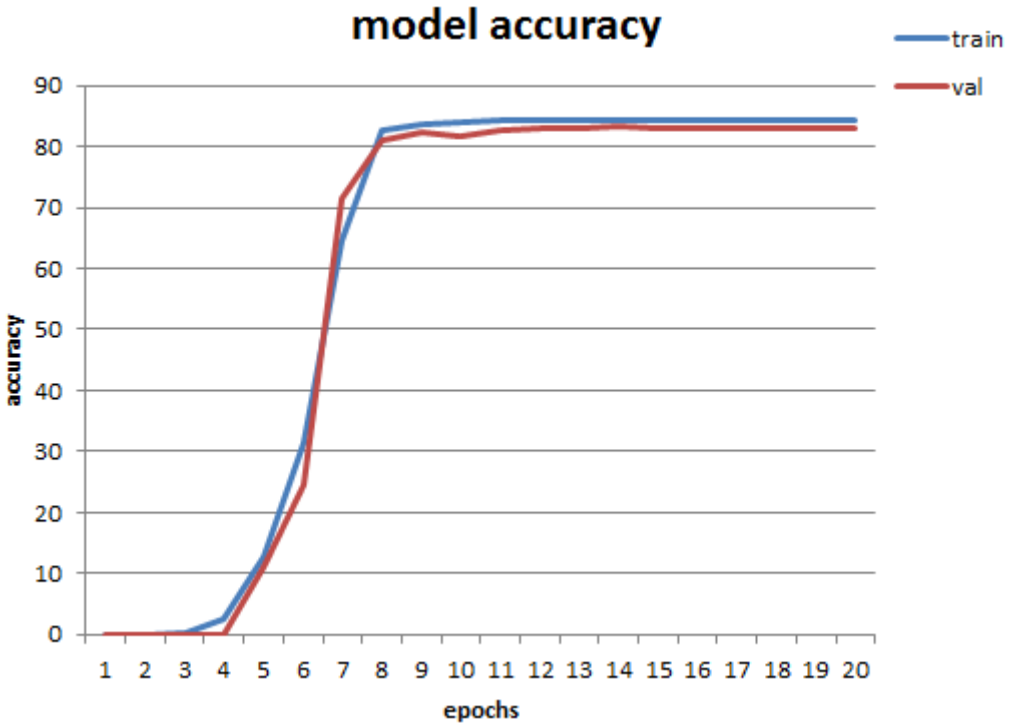


Рисунок 3 — График результата правильных ответов классификатора на выборке рис.1

Как видно результат правильных ответов на части train составляет ~84.3%, а на validation ~83%.

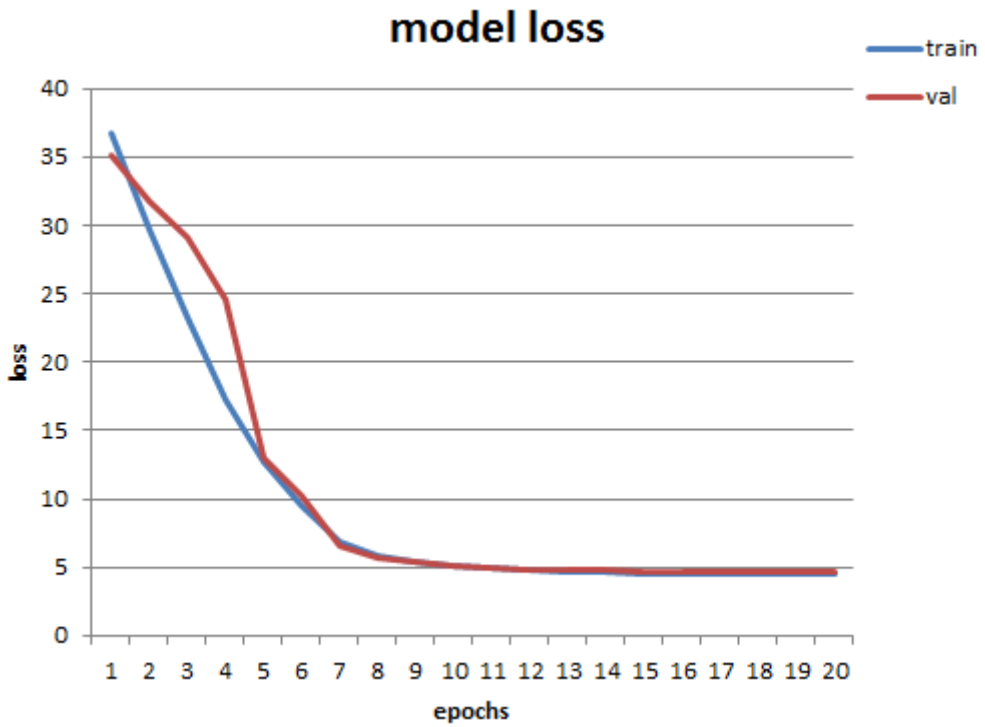


Рисунок 4 — График результата функции потерь сети на выборке рис.1

Сохраним веса модели и проверим данный классификатор, обученный на изображениях рис.1, на изображениях с фиксированной позицией букв/цифр и ограниченном алфавите символов вида §3 рисунок 6.

Результат = 0,78569 (~78,5% величина правильных ответов из 10000 изображений).

Следующим шагом попробуем улучшить наш классификатор, ограничив используемый алфавит. Так как мы рассматриваем задачу распознавания номерных знаков конкретного региона: Россия. А в этом регионе используемый алфавит символов на номерных знаках сильно ограничен, что помогает оптимизации классификатора. Соответственно наш алфавит примет вид:

```
PLATE_CHARS_LETTER = {"A", "B", "C", "D", "E", "H", "K", "M", "O", "P", "T", "X", "Y"}
PLATE_CHARS_DIGIT = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
```

Он состоит из 12 русских букв, 1буквы “D” английского алфавита и 10 цифр (0-9).

Рисунок 5 — Пример выборки с ограниченным алфавитом символов

Напоминаю, что в сгенерированном наборе класс букв и класс цифр имеют произвольные позиции в последовательности номера. То есть в автомобильном номере буквы и цифры не имеют фиксированной позиции и могут стоять в любом порядке.

Аналогично запускаем обучение на 20000 изображениях вида рисунок 4.

```
Train on 18000 samples, validate on 2000 samples
Epoch 1/20
18000/18000 [======]  - loss: 35.0184  - acc: 6.086E-11 - val_loss: 31.9679 - val_acc: 1.51E-11

Epoch 2/20
18000/18000 [======]  - loss: 19.8622 - acc:0.00067 - val_loss: 21.5233 - val_acc: 0.00008

Epoch 3/20
18000/18000 [======]  - loss: 10.346 - acc:0.5309  - val_loss: 11.8886 - val_acc: 0.1465

Epoch 4/20
18000/18000 [======]  - loss: 6.4651 - acc:0.9519 - val_loss: 6.3871 - val_acc: 0.6642

Epoch 5/20
18000/18000 [======]  - loss: 4.6504 - acc:0.9826  - val_loss: 4.5885 - val_acc: 0.8895

...

Epoch 16/20
18000/18000 [======]  - loss: 1.995 - acc:0.9993  - val_loss: 2.0526 - val_acc: 0.9960

Epoch 17/20
18000/18000 [======]  - loss: 1.9794 - acc: 0.9993 - val_loss: 2.0426 - val_acc: 0.9960

Epoch 18/20
18000/18000 [======]  - loss: 1.9697 - acc:0.9993  - val_loss: 2.0331 - val_acc: 0.9960

Epoch 19/20
18000/18000 [======]  - loss: 1.9619 - acc:0.9994  - val_loss: 2.0285 - val_acc: 0.9960

Epoch 20/20
18000/18000 [======]  - loss: 1.9574 - acc:0.9994 - val_loss: 2.024 - val_acc:  0.9965
```

Рисунок 6 — Процесс обучения классификатора на наборе вида рис.5

Результат обучения:

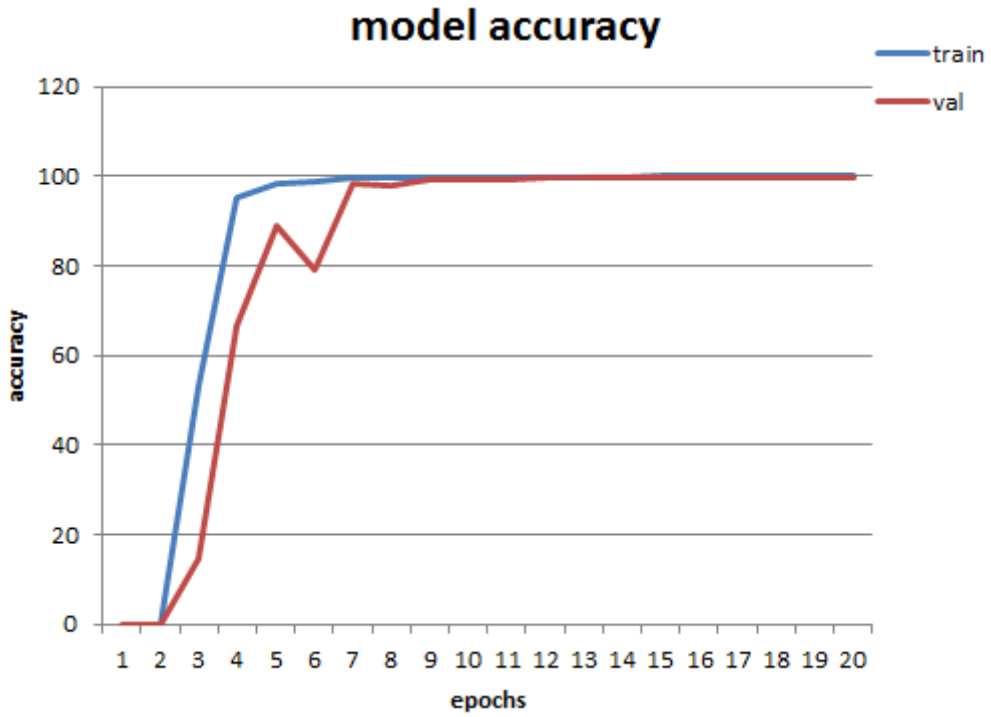


Рисунок 7 — График результата правильных ответов классификатора на выборке вида рис.5

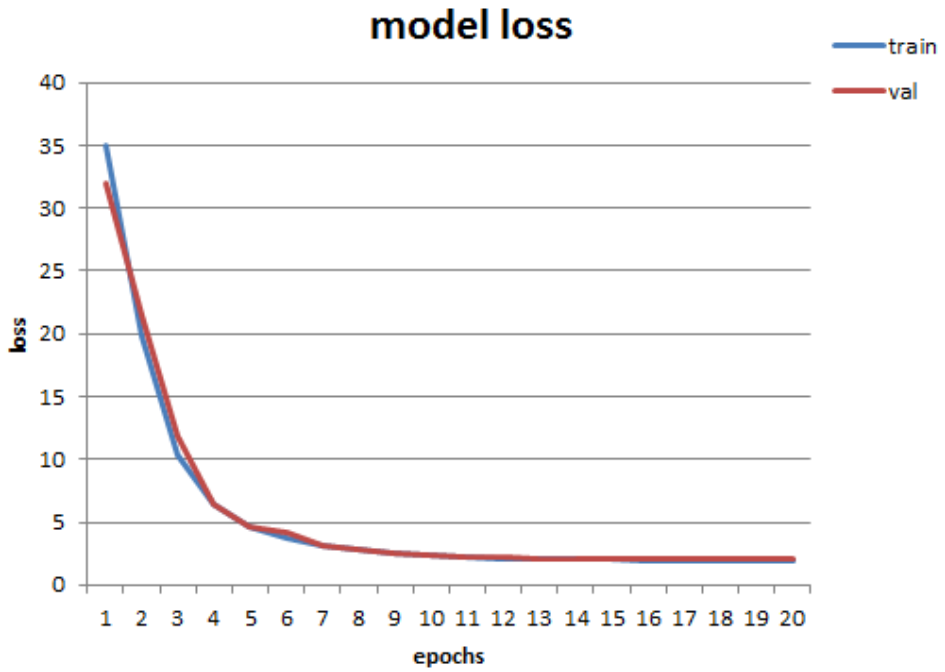


Рисунок 8 — График результата функции потерь сети на выборке вида рис.5

Сохраним веса модели и проверим данный классификатор, обученный на изображениях вида рис.5, на изображениях с фиксированной позицией букв/цифр и ограниченном алфавите символов вида §3 рисунок 6. Результат = 0,9912 (~99,1% величина правильных ответов из 10000 изображений). Нам удалось значительно улучшить результат классификатора за счет уменьшения количества символов в алфавите.

Следующее замечание, рассматриваемый автомобильный номер Тип1(однострочный) (§3 рис. 1) имеет еще одну особенность. В нем буквы и цифры имеют определенный порядок - Б ЦЦЦ ББ ЦЦ, где Б-буква и Ц-цифра. На первой позиции номерного знака всегда одна буква, затем три цифры и две буквы, и в конце код региона состоящий из цифр. Длина номерного знака равна 8. В итоге беря во внимание это свойство при обучении классификатора, мы приходим к генератору автомобильных номеров с фиксированными позициями символов и ограниченным алфавитом, описанному в параграфе *4.1 Генератор автомобильных номеров* и получаем модель нейронной сети, которая была описана в *4.2 Архитектура нейронной сети.*

В таблице 1 оценивается результат работы сети по всем 8 символам номерного знака. Считается, что номерной знак правильно распознается, если все 8 символов распознались правильно. В качестве тестовой базы изображений использовались 10000 сгенерированных изображений, описанных в *§3.1.1 описание генератора.*

Классификатор	Величина правильных ответов
A	0,785696261
B	0,991211417
C	0,997302918

Таблица 1 — Сравнение трёх типов классификаторов на одном наборе изображений соответствующих ГОСТ Р 50577-2018 номера тип-1 однострочный

A. Классификатор, обученный на множестве из 20000 изображений, где алфавит допустимых символов состоит из части английского, части русского алфавитов и 10 цифр (0-9), всего 50 символов. При том, что последовательность букв и цифр случайная (позиции классов символов не фиксированы).
B. Классификатор, обученный на множестве из 20000 изображений, где алфавит допустимых символов состоит из части русского алфавита (“D” + 12 букв) и 10 цифр (0-9), всего 23 символов. При том, что последовательность букв и цифр случайная (позиции классов символов не фиксированы).
C. Классификатор, обученный на множестве из 20000 изображений, где алфавит допустимых символов состоит из части русского алфавита (“D” + 12 букв) и 10 цифр (0-9), всего 23 символов. При том, что последовательность букв и цифр фиксированная: Б ЦЦЦ ББ ЦЦ.

Итог: Обнаружена прямая зависимость между количеством символов используемых в номерном знаке и качеством распознавания автомобильных номеров (e2e) нейронной сетью. Впоследствии чего удалость улучшить результат распознавания.

ЗАКЛЮЧЕНИЕ

Таким образом, были решены все задачи и достигнута цель данной работы. А именно:

- Построен генератор автомобильных номеров с добавлением искажений на изображения, который способен генерировать десятки тысяч экземпляров одним прогоном. Генератор строит изображение номеров по заданному шаблону и набору допустимых символов, что может быть использовано для моделирования номеров для различных стран.
- Построена сверточная нейронная сеть для классификации номерных знаков, подтвердившая свою эффективность на тестовом наборе данных. Преимуществом данной сети является легкая переносимость модели на номера других регионов, не только России.
- Было выполнено сравнение качества распознавания нейронной сети с разными наборами символов.
- А также удалось оптимизировать работу классификатора при распознавании автомобильного номера посредством его специализации с 78,5% до 99,7%.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. https://keras.io/
2. https://en.wikipedia.org/wiki/Deep_learning
3. https://habr.com/ru/company/recognitor/blog/221891/ - Распознавание номеров: от А до 9.
4. https://developers.ghttps://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmaxoogle.com/machine-learning/crash-course/multi-class-neural-networks/softmax - о функции softmax.
5. https://web.inf.ufpr.br/vri/publications/layout-independent-alpr/ - An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO Detector
6. https://github.com/sergiomsilva/alpr-unconstrained - ALPR in Unscontrained Scenarios
7. https://github.com/LCorleone/A-Simple-Chinese-License-Plate-Generator-and-Recognition-Framework - hyperlpr-train_e2e
8. http://neuralnetworksanddeeplearning.com/chap1.html -Using neural nets to recognize handwritten digits
9. https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/
10. https://m.habr.com/ru/post/148840/ - Шрифт для регистрационных номерных знаков.
11. http://yann.lecun.com/ex/research/index.html
12. https://colab.research.google.com/

Приложения

Приложение 1: код генератора автомобильных номерных знаков на языке python.

create_train_data.py :

```
import numpy as np
from genplate_advanced import *
import os
import pandas as pd
import pickle
import cv2
import os

index = {"A": 0, "B": 1, "C": 2, "D": 3, "E": 4, "H": 5, "K": 6,
         "M": 7, "O": 8, "P": 9, "T": 10, "X": 11, "Y": 12,
         "0": 13, "1": 14, "2": 15, "3": 16, "4": 17, "5": 18,
         "6": 19, "7": 20, "8": 21, "9": 22}

chars = [u"A", u"B", u"C", u"D", u"E", u"H", u"K", u"M", u"O", u"P",
         u"T", u"X", u"Y", u"0", u"1", u"2", u"3", u"4", u"5",
         u"6", u"7", u"8", u"9"]


def rand_range(lo, hi):
    return lo + r(hi - lo)


def r(val):
    return int(np.random.random() * val)


def gen_rand():
    name = ""
    label = []
    label.append(rand_range(0, 13))
    for i in range(3):
        label.append(rand_range(13, 23))
    for i in range(2):
        label.append(rand_range(0, 13))
```

```
    for i in range(2):
            label.append(rand_range(13, 23))

    for i in range(8):
        name += chars[label[i]]

    return name, label


def gen_sample(genplate_advanced, width, height):
    name, label = gen_rand()
    img = genplate_advanced.generate(name)
    img = cv2.resize(img, (width, height))
    return label, name, img


def genBatch(batchSize, outputPath):
    if not os.path.exists(outputPath):
        os.makedirs(outputPath)
    label_store = []
    for i in range(batchSize):
        print('create num:' + str(i))
        label, name, img = gen_sample(genplate_advanced, 140,30)
        label_store.append(label)
        filename = os.path.join(outputPath, str(i).zfill(4) + ".jpg")
        cv2.imwrite(filename, img)
    np.savetxt('label.txt', label_store)



batchSize = 20000
path = './data/train_data'
font_en = './font/RoadNumbers2.0.ttf'
bg_dir = './NoPlates'
```

```
genplate_advanced = GenPlate(font_en, bg_dir)
genBatch(batchSize=batchSize, outputPath=path)

# create train label
a = np.loadtxt('label.txt')
b = np.zeros([batchSize, 23])
for i in range(batchSize):
    for j in range(8):
        b[i, int(a[i, j])] = int(a[i, j])

# create image train data
img_data = np.zeros([batchSize, 30, 140, 3])
for i in range(batchSize):
    img_path = path + '/' + str(i).zfill(4) + ".jpg"
    img_temp = cv2.imread(img_path)
    img_temp = np.reshape(img_temp, (30, 140, 3))
    img_data[i, :, :, :] = img_temp


print(b)

output = open('train_data.pkl', 'wb')
pickle.dump(img_data, output)

```

genplate_advanced.py :

```
import os
import argparse
from math import *
import numpy as np
import cv2
from PIL import Image, ImageFont, ImageDraw
from PlateCommon import *

class GenPlate:
    def __init__(self, fontEng, NoPlates):
        self.fontE = ImageFont.truetype(fontEng, 116, 0)
        self.fontE2 = ImageFont.truetype(fontEng, 86, 0)
        self.img = np.array(Image.new("RGB", (520, 112), (255, 255, 255)))
        self.bg = cv2.resize(cv2.imread("./images/lpr.png"), (520, 112))
        self.smu = cv2.imread("./images/smu2.jpg")
        self.noplates_path = []
        for parent, parent_folder, filenames in os.walk(NoPlates):
            for filename in filenames:
                path = parent + "/" + filename
                self.noplates_path.append(path)

    def draw(self, val):
        space1=14
        space2=5
        space3=0
        self.img[0: 94, 36:36 + 50] = GenCh1(self.fontE, val[0])
        self.img[0: 94, 36 + 50+space1:36 + 50*2+space1] = GenCh1(self.fontE, val[1])
        self.img[0: 94, 36 + 50+50+space1+space2:
        36 + 50+50+space1+50+space2] = GenCh1(self.fontE, val[2])
        self.img[0: 94, 36 + 50+50+14+50+space2+space2:
        36 + 50+50+14+50+space2+space2+50] = GenCh1(self.fontE, val[3])

        self.img[0: 94, 36 + 50+50+space1+50+space2+space2+50+space1-2:
        36 + 50*5+space1+space2+space2-2+space1] = GenCh1(self.fontE, val[4])

        self.img[0: 94, 36 + 50*5+space1+space2+space2+space1+space3-2:
        36 + 50*6+space1*2-2+space2*2+space3] = GenCh1(self.fontE, val[5])

        self.img[0: 73, 36 + 50*6+space1+space2+space2-2+space1+space3+29:
        36 + 50*6-2+space1*2+space2*2+space3+29+40] = GenCh2(self.fontE2, val[6])

        self.img[0: 73, 36 + 50*6+space1*2+space2*2-2+space3+29+40+space3:
        50*6+space1*2+space2*2+space3*2+63+2*40] = GenCh2(self.fontE2, val[7])

        return self.img

    def generate(self, text):
        if len(text) == 8:
            fg = self.draw(text)
            com = cv2.bitwise_and(fg, self.bg)
            com = rot(com, r(40) - 20, com.shape, 20)
            com = rotRandrom(com, 5, (com.shape[1], com.shape[0]))
            com = AddSmudginess(com, self.smu)
            com = random_envirment(com, self.noplates_path)
            com = AddGauss(com, 1 + r(2))
            com = addNoise(com)
            return com

```

PlateCommon.py :

```
35
36  def AddSmudginess(img, Smu):
37      img_h, img_w = img.shape[:2]
38      rows = r(Smu.shape[0] - img_h)
39
40      cols = r(Smu.shape[1] - img_w)
41      adder = Smu[rows:rows + img_h, cols:cols + img_w]
42      adder = cv2.resize(adder, (img_w, img_h))
43      adder = cv2.bitwise_not(adder)
44      val = random.random() * 0.5
45      img = cv2.addWeighted(img, 1 - val, adder, val, 0.0)
46      return img
47
48
49  def rot(img, angel, shape, max_angel):
50      size_o = [shape[1],shape[0]]
51      size = (shape[1] + int(shape[0] * sin((float(max_angel) / 180) * 3.14)), shape[0])
52      # print size
53      interval = abs(int(sin((float(angel) / 180) * 3.14) * shape[0]))
54
55      pts1 = np.float32([[0, 0], [0, size_o[1]], [size_o[0], 0], [size_o[0], size_o[1]]])
56      if(angel > 0):
57          pts2 = np.float32([[interval, 0], [0, size[1]],
58          [size[0], 0], [size[0] - interval, size_o[1]]])
59      else:
60          pts2 = np.float32([[0, 0], [interval, size[1]],
61          [size[0] - interval, 0], [size[0], size_o[1]]])
62
63      M = cv2.getPerspectiveTransform(pts1, pts2)
64      dst = cv2.warpPerspective(img, M, size)
65
66      return dst
67
68
```

```
69  def rotRandrom(img, factor, size):
70      shape = size
71      pts1 = np.float32([[0, 0], [0, shape[0]], [shape[1], 0], [shape[1], shape[0]]])
72      pts2 = np.float32([[r(factor), r(factor)],
73                         [ r(factor), shape[0] - r(factor)],
74                         [shape[1] - r(factor), r(factor)],
75                         [shape[1] - r(factor), shape[0] - r(factor)]])
76      M = cv2.getPerspectiveTransform(pts1, pts2)
77      dst = cv2.warpPerspective(img, M, size)
78      return dst
79
80
81  def random_envirment(img, data_set):
82      index = r(len(data_set))
83      env = cv2.imread(data_set[index])
84      env = cv2.resize(env, (img.shape[1], img.shape[0]))
85      val = random.random() * 0.4
86      img = cv2.addWeighted(img, 1 - val, env, val, 0.0)
87      return img
88
89
90  def GenCh1(f, val):
91      img = Image.new("RGB", (50, 94), (255, 255, 255))
92      draw = ImageDraw.Draw(img)
93      draw.text((0, 36), val, (0, 0, 0), font=f)
94      A = np.array(img)
95      return A
96
97  def GenCh2(f, val):
98      img = Image.new("RGB", (40, 73), (255, 255, 255))
99      draw = ImageDraw.Draw(img)
100     draw.text((0, 26), val, (0, 0, 0), font=f)
101     A = np.array(img)
102     return A
```

```
103
104 def AddGauss(img, level):
105     return cv2.blur(img, (level * 2 + 1, level * 2 + 1))
106
107
108 def r(val):
109     return int(np.random.random() * val)
110
111
112 def AddNoiseSingleChannel(single):
113     diff = 255 - single.max()
114     noise = np.random.normal(0, 1 + r(6), single.shape)
115     noise = (noise - noise.min()) / (noise.max() - noise.min())
116     noise = diff * noise
117     noise = noise.astype(np.uint8)
118     dst = single + noise
119     return dst
120
121
122 def addNoise(img, sdev=0.5, avg=10):
123     img[:, :, 0] = AddNoiseSingleChannel(img[:, :, 0])
124     img[:, :, 1] = AddNoiseSingleChannel(img[:, :, 1])
125     img[:, :, 2] = AddNoiseSingleChannel(img[:, :, 2])
126     return img
127
```

Приложение 2: код CNN нейронной сети.

train.py :

```
import numpy as np
import pickle
from keras.layers import Dense, Input, BatchNormalization,
Conv2D, Flatten, MaxPooling2D, Activation, Reshape, Layer, Lambda
from keras.models import Model
from keras.callbacks import TensorBoard, ModelCheckpoint, LearningRateScheduler
from keras.optimizers import RMSprop
from keras import backend as K
import tensorflow as tf
import keras.backend.tensorflow_backend as KTF
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "0"
def get_session():

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    return tf.Session(config=config)


class NormLayer(Layer):

    def __init__(self, **kwargs):
        super(NormLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernal = self.add_weight(name='NormLayer', shape=(
            1, 15), initializer='ones', trainable=True)
        super(NormLayer, self).build(input_shape)

    def call(self, inputs):
        # out = self.kernal * inputs
        print(inputs.shape)
```

```
        out = K.dot(self.kernal, inputs)
        out = K.permute_dimensions(out, (1, 0, 2))
        print(out.shape)
        return out[:, 0, :]


    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[2])

    def get_config(self):
        base_config = super(NormLayer, self).get_config()
        return dict(list(base_config.items()))



def get_session():

    config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth = True
    return tf.compat.v1.Session(config=config)


class train_e2e:

    def __init__(self):
        self.first_num = 13
        self.other_num = 10
        self.shape = (140, 30, 3)
        self.init_lr = 0.01

    def load_data(self):
        label_path = './label.txt'
        self.tem_label = np.loadtxt(label_path)
```

```
        row, col = self.tem_label.shape
        self.label1 = np.zeros([row, self.first_num])
        self.label2 = np.zeros([row, self.other_num])
        self.label3 = np.zeros([row, self.other_num])
        self.label4 = np.zeros([row, self.other_num])
        self.label5 = np.zeros([row, self.first_num])
        self.label6 = np.zeros([row, self.first_num])
        self.label7 = np.zeros([row, self.other_num])
        self.label8 = np.zeros([row, self.other_num])
        for i in range(row):
            self.label1[i, int(self.tem_label[i, 0])] = 1
            self.label2[i, int(self.tem_label[i, 1]) - 13] = 1
            self.label3[i, int(self.tem_label[i, 2]) - 13] = 1
            self.label4[i, int(self.tem_label[i, 3]) - 13] = 1
            self.label5[i, int(self.tem_label[i, 4])] = 1
            self.label6[i, int(self.tem_label[i, 5])] = 1
            self.label7[i, int(self.tem_label[i, 6]) - 13] = 1
            self.label8[i, int(self.tem_label[i, 7]) - 13] = 1


        img_path = './train_data.pkl'
        self.img_data = np.load(img_path, allow_pickle=True)
        self.img_data = self.img_data.transpose(0, 2, 1, 3)

    def step_decay(self, epoch):
        if epoch % 2 == 0 and epoch != 0:
            lr = K.get_value(self.model.optimizer.lr)
            K.set_value(self.model.optimizer.lr, lr * .5)
            print("lr changed to {}".format(lr * .5))
        return K.get_value(self.model.optimizer.lr)

    def __build_network(self):
        tf.compat.v1.keras.backend.set_session(get_session())
        input_img = Input(shape=self.shape)
        base_conv = 32
        x = input_img
        for i in range(3):
            x = Conv2D(base_conv * (2 ** (i)), (3, 3), padding="same")(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Conv2D(256, (3, 3))(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Conv2D(1024, (1, 1))(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        first_subject = Conv2D(self.first_num, (1, 1))(x)
        other_subject = Conv2D(self.other_num, (1, 1))(x)
        first_subject = Activation('softmax')(first_subject)
        other_subject = Activation('softmax')(other_subject)
        first_subject = Reshape((-1, self.first_num))(first_subject)
        other_subject = Reshape((-1, self.other_num))(other_subject)
        x1 = NormLayer()(first_subject)
        x2 = NormLayer()(other_subject)
        x3 = NormLayer()(other_subject)
        x4 = NormLayer()(other_subject)
        x5 = NormLayer()(first_subject)
        x6 = NormLayer()(first_subject)
        x7 = NormLayer()(other_subject)
        x8 = NormLayer()(other_subject)
        out1 = Activation('softmax', name='out1')(x1)
        out2 = Activation('softmax', name='out2')(x2)
        out3 = Activation('softmax', name='out3')(x3)
        out4 = Activation('softmax', name='out4')(x4)
        out5 = Activation('softmax', name='out5')(x5)
```

```
        out6 = Activation('softmax', name='out6')(x6)
        out7 = Activation('softmax', name='out7')(x7)
        out8 = Activation('softmax', name='out8')(x8)
        rmsprop = RMSprop(lr=self.init_lr)
        self.model = Model(input_img, [out1, out2, out3, out4, out5, out6, out7, out8])
        self.model.compile(loss=['categorical_crossentropy', 'categorical_crossentropy',
        'categorical_crossentropy', 'categorical_crossentropy', 'categorical_crossentropy',
        'categorical_crossentropy', 'categorical_crossentropy', 'categorical_crossentropy']
            optimizer=rmsprop, metrics=['accuracy'], loss_weights=[1, 1, 1, 1, 1, 1, 1, 1])
        print(self.model.summary())

    def train(self):
        self.__build_network()
        lrate = LearningRateScheduler(self.step_decay)
        self.model.fit(self.img_data, [self.label1, self.label2, self.label3, self.label4,
            self.label5, self.label6, self.label7, self.label8], epochs=20, batch_size=256,
            verbose=1, callbacks=[lrate], validation_split=0.1)
        self.model.save("e2e_model.h5")


if __name__ == "__main__":
    my_train_nn = train_e2e()
    my_train_nn.load_data()
    my_train_nn.train()
```

test.py :

```
import numpy as np
import pickle
from keras.layers import Dense, Input, BatchNormalization,
Conv2D, Flatten, MaxPooling2D, Activation, Reshape, Layer
from keras.models import Model, load_model, model_from_json
from keras.callbacks import TensorBoard, ModelCheckpoint, LearningRateScheduler
from keras.optimizers import RMSprop
from keras import backend as K
import tensorflow as tf
import keras.backend.tensorflow_backend as KTF

gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.4)
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(gpu_options=gpu_options))

chars = ["A", "B", "C", "D", "E", "H", "K", "M", "O", "P", "T", "X", "Y",
         "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]


class NormLayer(Layer):

    def __init__(self, **kwargs):
        super(NormLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernal = self.add_weight(name='NormLayer', shape=(
            1, 15), initializer='ones', trainable=True)
        super(NormLayer, self).build(input_shape)

    def call(self, inputs):
        # out = self.kernal * inputs
        out = K.dot(self.kernal, inputs)
        out = K.permute_dimensions(out, (1, 0, 2))
        return out[:, 0, :]
```

```
    def compute_output_shape(self, input_shape):
        return (input_shape[0], 23)

    def get_config(self):
        # config = {}
        base_config = super(NormLayer, self).get_config()
        return dict(list(base_config.items()))
        # return dict(list(base_config.items()) + list(config.items()))


e2e_model = load_model('e2e_model.h5', custom_objects={'NormLayer': NormLayer})
label_path = 'label.txt'
tem_label = np.loadtxt(label_path)
row, col = tem_label.shape
label1 = np.zeros([row, 13])
label2 = np.zeros([row, 10])
label4 = np.zeros([row, 10])
label3 = np.zeros([row, 10])
label5 = np.zeros([row, 13])
label6 = np.zeros([row, 13])
label7 = np.zeros([row, 10])
label8 = np.zeros([row, 10])
for i in range(row):
    label1[i, int(tem_label[i, 0])] = 1
    label2[i, int(tem_label[i, 1]) - 13] = 1
    label3[i, int(tem_label[i, 2]) - 13] = 1
    label4[i, int(tem_label[i, 3]) - 13] = 1
    label5[i, int(tem_label[i, 4])] = 1
    label6[i, int(tem_label[i, 5])] = 1
    label7[i, int(tem_label[i, 6]) - 13] = 1
    label8[i, int(tem_label[i, 7]) - 13] = 1


img_path = './train_data.pkl'
img_data = np.load(img_path, allow_pickle=True)
img_data = img_data.transpose(0, 2, 1, 3)


e2e_predict = e2e_model.predict(img_data[0:10, :, :, :])



def print_trueLabel(num):
    print(np.array([np.argmax(label1[num, :]), np.argmax(label2[num, :]) + 13,
    np.argmax(label3[num, :]) + 13, np.argmax(label4[num, :]) + 13,
    np.argmax(label5[num, :]), np.argmax(label6[num, :]), np.argmax(label7[num, :]) + 13,
    np.argmax(label8[num, :]) + 13]))
    print(chars[np.argmax(label1[num, :])] + chars[np.argmax(label2[num, :]) + 13] +
    chars[np.argmax(label3[num, :]) + 13] +  chars[np.argmax(label4[num, :]) + 13] +
    chars[np.argmax(label5[num, :])] + chars[np.argmax(label6[num, :])] +
          chars[np.argmax(label7[num, :]) + 13] + chars[np.argmax(label8[num, :]) + 13])

def print_predictLabel(x):
    num = x[0].shape[0]
    sort = np.zeros([num, len(x)])
    for i in range(len(x)):
        temp = x[i]
        sort[:, i] = np.argmax(temp, 1)
    for i in range(num):
        print(np.array([int(sort[i, 0]), int(sort[i, 1] + 13), int(sort[i, 2] + 13),
        int(sort[i, 3] + 13), int(sort[i, 4]), int(sort[i, 5]), int(sort[i, 6] + 13),
        int(sort[i, 7] + 13)]))
        print(chars[int(sort[i, 0])] + chars[int(sort[i, 1] + 13)] +
        chars[int(sort[i, 2] + 13)] + chars[int(sort[i, 3] + 13)] + chars[int(sort[i, 4])]+
        chars[int(sort[i, 5])] + chars[int(sort[i, 6] + 13)]+ chars[int(sort[i, 7] + 13)])

```

```
104
105  print_trueLabel(0)
106  print_trueLabel(1)
107  print_trueLabel(2)
108  print_trueLabel(3)
109  print_trueLabel(4)
110  print_trueLabel(5)
111  print_trueLabel(6)
112  print_trueLabel(7)
113  print_trueLabel(8)
114  print_trueLabel(9)
115  print_predictLabel(e2e_predict)
```