

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: «The Street View House Numbers»

Студентка гр. 7381

Кревчик А.Б.

Студентка гр. 7381

Процветкина А.В.

Преподаватель

Жукова Н.А.

Санкт-Петербург

2020

Описание датасета:

SVHN (The Street View House Numbers) представляет собой датасет изображений. Включает в себя изображения с цифрами, которые были взяты с номеров домов на улице.

Основные данные:

- 10 классов, по 1 на каждую цифру. Цифра «1» имеет метку 1, «9» имеет метку 9, а «0» имеет метку 10.
- 73257 цифр для обучения, 26032 цифры для тестирования
- Формат данных является MNIST-подобным и представляет собой изображения 32 на 32 пикселя. Данные находятся в файле с расширением .mat, при загрузке из которого создаются 2 переменные: X - это четырехмерная матрица, содержащая изображения, и y - вектор меток классов.

Решаемая задача:

Задача заключается в выделении на изображении цифр.

Зоны ответственности:

Кревчик А. — подготовка и анализ данных, написание Callback'а, разработка модели_1.

Процветкина А. — разработка модели_2, анализ качества работы моделей, создание ансамбля.

Анализ данных:

Загружаем данные с помощью функции load_data():

```
def load_data():  
    mat_train = sio.loadmat('train_32x32.mat')  
    train_X = mat_train['X']  
    train_y = mat_train['y']  
    mat_test = sio.loadmat('test_32x32.mat')  
    test_X = mat_test['X']  
    test_y = mat_test['y']  
    return train_X, train_y, test_X, test_y
```

`train_X` представляет собой четырехмерный тензор `[32,32,3,73257]`, т.е. изображения `32*32` пиксела, RGB(3 канала), и таких изображений 73257.

`test_X` представляет собой четырехмерный тензор `[32,32,3,26032]`, т.е. изображения `32*32` пиксела, RGB(3 канала), и таких изображений 26032.

`train_y`, `test_y` представляют собой тензоры `[73257,1]` и `[26032,1]` соответственно с метками класса.

Проверим, что изображения загрузились, выведя 1 изображение на экран. Для этого воспользуемся функцией `print_image(i)`.

```
def print_image(i):  
    plt.imshow(train_X[:, :, :, i])  
    plt.show()  
    print(train_y[i])
```

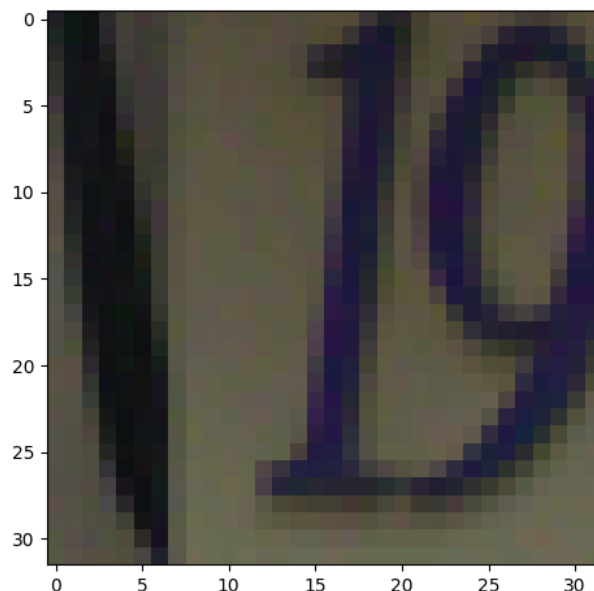


Рисунок 1 - Изображение `train_X[:, :, :, 0]` с меткой класса 1

Изучим, сколько каких изображений в нашем датасете. Для этого построим гистограмму частот меток классов. Воспользуемся функцией `plot_distribution_y(train_y, test_y)`.

```
def plot_distribution_y(train_y, test_y):  
    plt.hist(x=train_y, bins='auto', color='#0504aa', alpha=0.7,  
rwidth=0.85)  
    plt.grid(axis='y', alpha=0.75)  
    plt.xlabel('y')  
    plt.ylabel('Frequency')  
    plt.title('Distribution of train_y')  
    plt.show()
```

```
plt.hist(x=test_y, bins='auto', color='#0504aa', alpha=0.7,
rwidth=0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('y')
plt.ylabel('Frequency')
plt.title('Distribution of test_y')
plt.show()
```

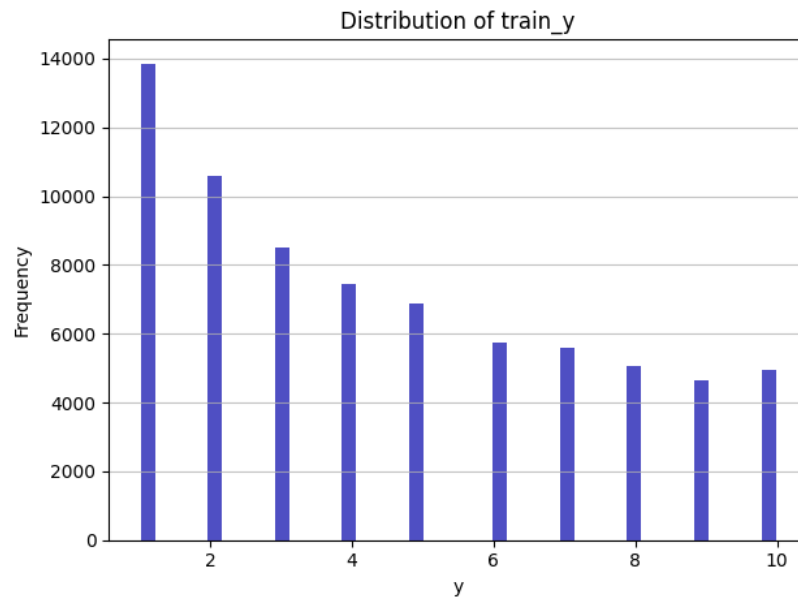


Рисунок 2 - Распределение тренировочных меток

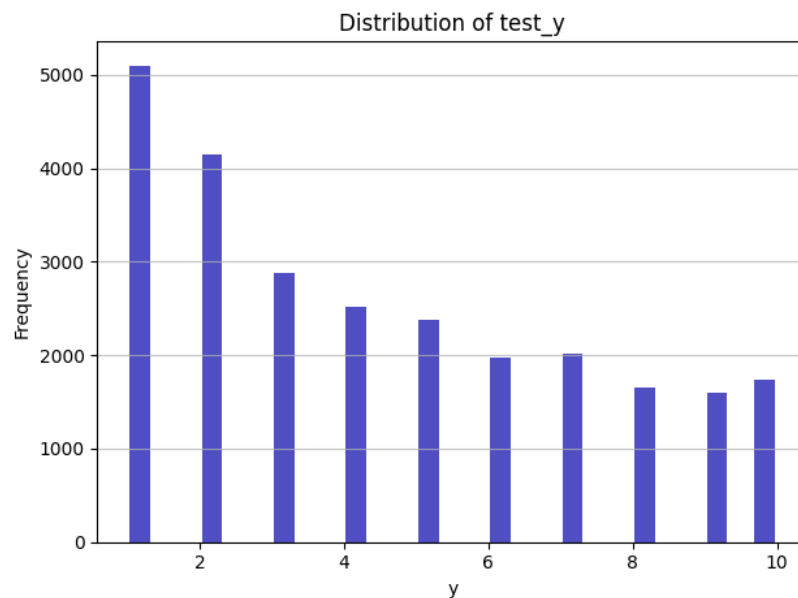


Рисунок 3 - Распределение тестовых меток

Так как наше изображение RGB, то мы имеем три канала цвета с метками от 0 до 255. Чтобы в этом убедиться, построим гистограмму цвета для изображения, представленного на рис.1.

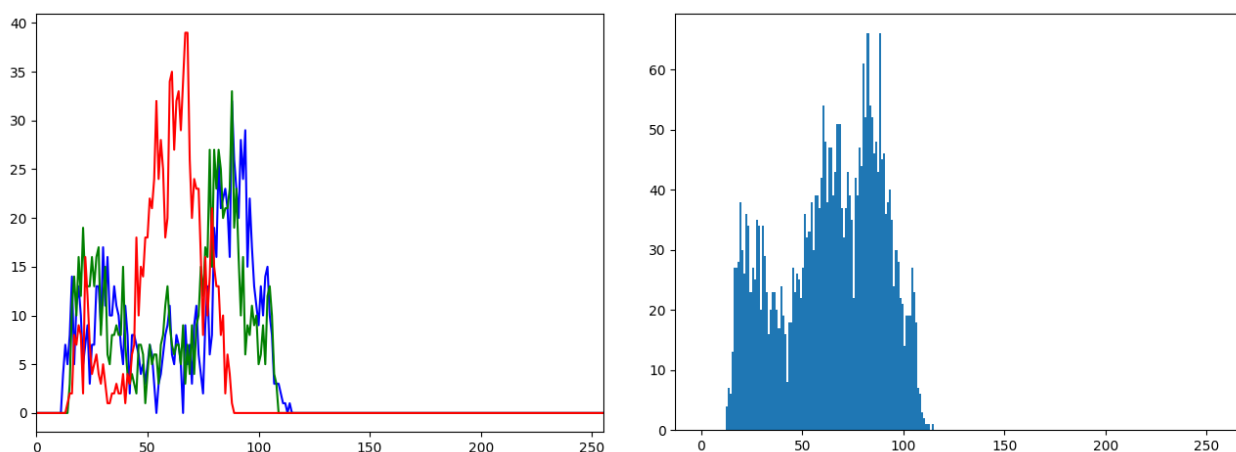


Рисунок 4 - Распределение тестовых меток

Для того чтобы использовать наши данные об изображения для сети, нам нужно их транспонировать и нормировать. Транспонировать нужно для того, чтобы первым измерением у нас стало количество изображений. Нормировка в нашем случае необходима, так как исходные данные имеют диапазон $[0, 255]$, а сеть работает с диапазоном $[0, 1]$. В качестве нормировки воспользуемся делением на максимальный элемент нашего тензора.

Что касается меток классов, то для начала заменим метку для нулевого класса (т.к. в исходных данных она равна 10) на 0. Затем переедем наш вектор класса в двоичную матрицу классов.

Все преобразования с исходными данными выполняются в функции `prepare_data(train_X, train_y, test_X, test_y, num_classes)`.

```
def prepare_data(train_X, train_y, test_X, test_y, num_classes):
    train_X = train_X.astype('float32').transpose((3, 0, 1, 2))
    test_X = test_X.astype('float32').transpose((3, 0, 1, 2))
    train_X /= np.max(train_X)
    test_X /= np.max(test_X)
    train_y[train_y == 10] = 0
    test_y[test_y == 10] = 0

    train_y = np_utils.to_categorical(train_y, num_classes)
    test_y = np_utils.to_categorical(test_y, num_classes)
    return train_X, train_y, test_X, test_y
```

Разработка модели:

Было решено создать 2 независимые модели и после объединить их в ансамбль.

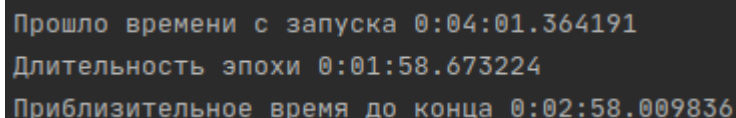
Для контроля процесса обучения был написан Callback, который выводит на экран общее время обучения сети, длительность обучения на каждой эпохе и время, прошедшее с начала обучения.

```
class show_time_callback(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
    def on_epoch_begin(self, epoch, logs=None):
        self.epoch_start = datetime.datetime.now()

    def on_epoch_end(self, epoch, logs=None):
        epoch_end = datetime.datetime.now()
        if (epoch != (self.params['epochs'] - 1)):
            print("Прошло времени с запуска", epoch_end -
self.start_train)
            print("Длительность эпохи", epoch_end- self.epoch_start)
            if(epoch != (self.params['epochs']-1)):
                print("Приблизительное время до конца", (epoch_end-
self.epoch_start)/(epoch+1)* (self.params['epochs'] - (epoch + 1)))

    def on_train_begin(self,epoch, logs={}):
        self.start_train=datetime.datetime.now()

    def on_train_end(self,epoch, logs={}):
        full_time = datetime.datetime.now() - self.start_train
        print("Общее время ",full_time)
```



```
Прошло времени с запуска 0:04:01.364191
Длительность эпохи 0:01:58.673224
Приблизительное время до конца 0:02:58.009836
```

Рисунок 5 - Пример работы show_time_callback

Модель 1

Так как задачей является классификация изображений, то для модели выбрана архитектура сверточной нейронной сети.

Модель 1.0

Первая модель представляет собой один слой свертки и субдискретизации. Решено было провести обучение на 5 эпохах. Т.к. датасет достаточно объемный, пакет `batch_size = 100`. Для избежания переобучения добавим слой Dropout с коэффициентом 0.4.

```
batch_size = 100
num_epochs = 5
```

```

kernel_size = 3
pool_size = 2
conv_depth_1 = 32
drop_prob_1 = 0.4
hidden_size = 512

inp = Input(shape=(depth, height, width))

conv_1 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(inp)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_1)
drop_1 = Dropout(drop_prob_1)(pool_1)
flat = Flatten()(drop_1)
dence_1 = Dense(hidden_size, activation='relu')(flat)
out = Dense(num_classes, activation='softmax')(dence_1)

model = Model(input=inp, output=out)

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

```

Т.к. классификация многоклассовая, в качестве функции потерь используем `categorical_crossentropy`.

В качестве оптимизатора выберем `adam` с параметрами по умолчанию.

Графическое представление модели представлено на рис. 6

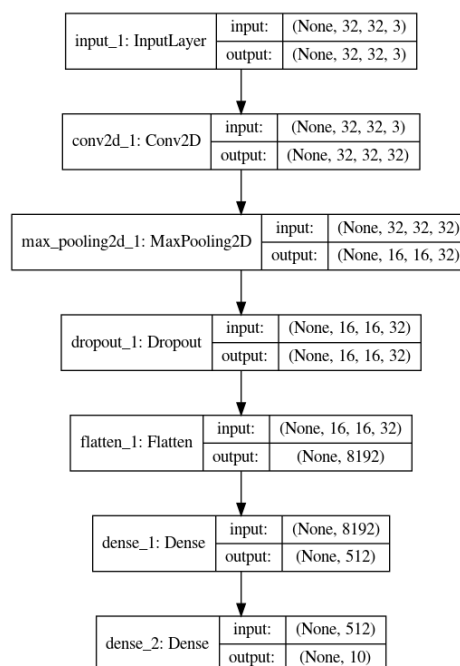


Рисунок 6 - графическое представление модели 1.0

График точности и ошибок представлен на рис.7.

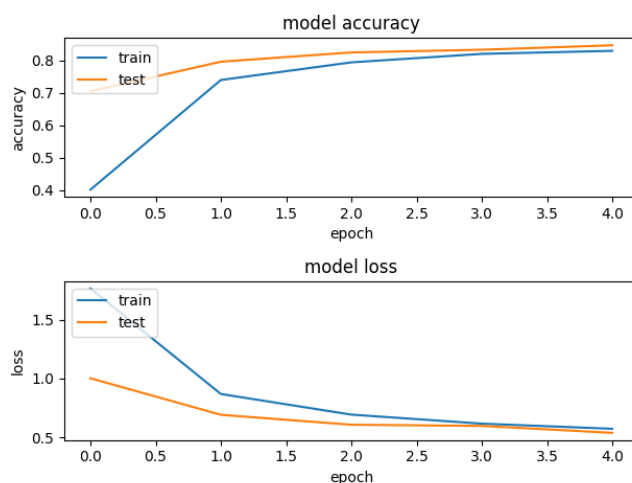


Рис. 7 - График точности и ошибок модели 1.0

Потери на тестовых данных: 0.64

Точность на тестовых данных: 0.82

Модель 1.1

Так как потери нашей модели довольно большие, нам нужно подкорректировать архитектуру. Для этого добавим еще слой свертки, субдискретизации и Dropout. Параметры слоев

```
kernel_size = 3
pool_size = 2
drop_prob_1 = 0.4
conv_depth_2 = 64
```

```
conv_2 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_1)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_2 = Dropout(drop_prob_1)(pool_2)
```

Графическое представление модели представлено на рис. 8.

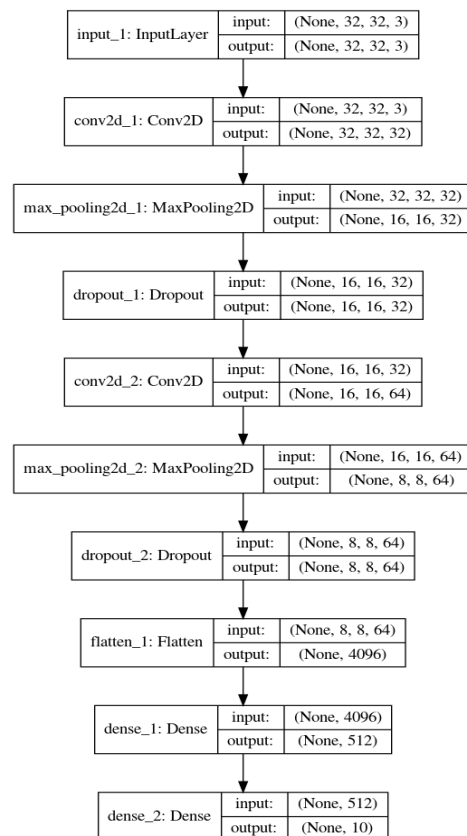


Рисунок 8 - Графическое представление модели 1.1

График точности и ошибок представлен на рис.9.

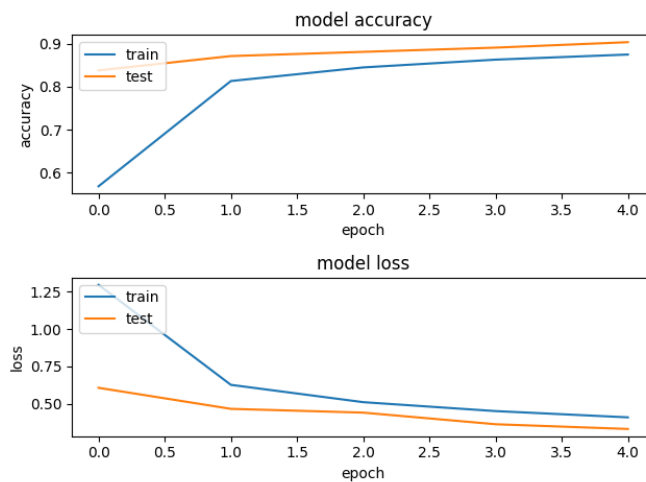


Рис. 9 - График точности и ошибок модели 1.1

Потери на тестовых данных: 0.38

Точность на тестовых данных: 0.89

Как мы видим, потери стали значительно меньше.

Модель 1.2

Попробуем еще больше улучшить сеть.

Добавим еще пару слоев :

```
conv_3 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_2)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_3)
```

Судя по графикам, наша сеть не переобучается, поэтому попробуем уменьшить коэффициент в слое Dropout и сделаем `drop_prob_1 = 0.2`. Также уменьшим `batch_size = 50`, чтобы четь чаще корректировалась.

Так как для проверки стабильности сети запуск делается несколько раз, в один из запусков на графиках было выявлено незначительное переобучение, которое не влияло на точность и потери, но все равно было решено сократить количество эпох до 4.

В результате модель имеет вид:

```
batch_size = 50
num_epochs = 4

kernel_size = 5
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.2
drop_prob_2 = 0.4
hidden_size = 512

inp = Input(shape=(depth, height, width))

conv_1 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(inp)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_1)
drop_1 = Dropout(drop_prob_1)(pool_1)

conv_2 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_1)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_2 = Dropout(drop_prob_1)(pool_2)

conv_3 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_2)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_3)

flat = Flatten()(pool_2)
```

```

dense_1 = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(dense_1)
out = Dense(num_classes, activation='softmax')(drop_3)

model = Model(input=inp, output=out)

```

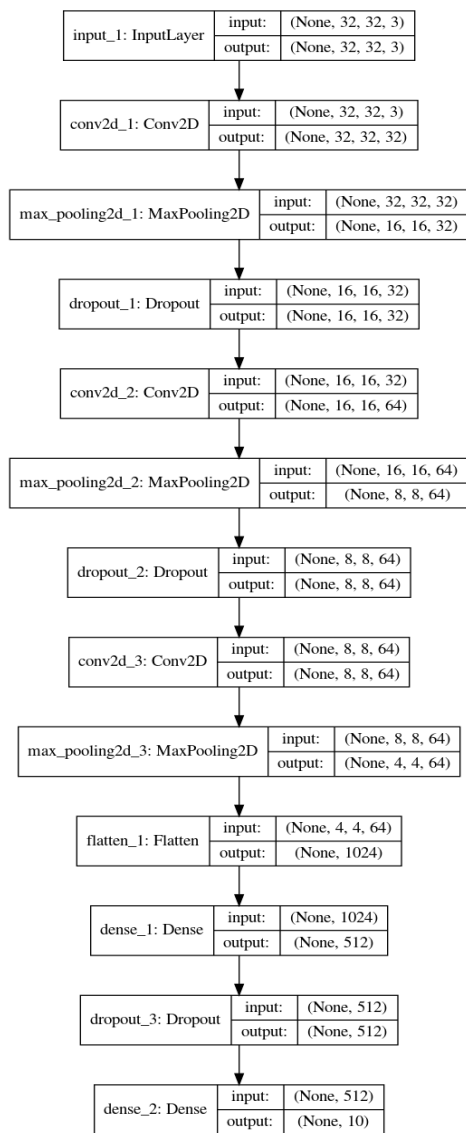


Рисунок 10 - Графическое представление модели 1.2

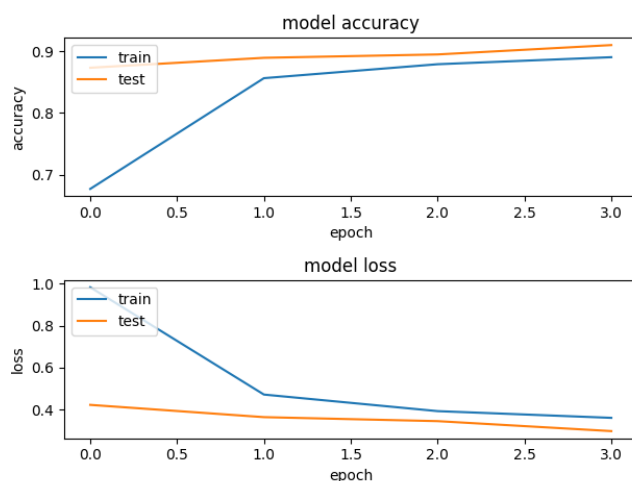


Рис.11 - График точности и ошибок модели 1.2

Потери на тестовых данных:0.32

Точность на тестовых данных: 0.91

Модель выбрана в качестве итоговой.

Модель 2.

Модель 2.1

Начальная версия модели 2 содержала два «блока» из двух сверток с последующим MaxPooling и два полносвязных слоя.

Планировалось создать очень сложную модель с последующим упрощением, но первая модель все равно была недостаточно сложной.

Конфигурация модели:

```
keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu', input_shape=(32, 32, 3)),
keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),

keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),

keras.layers.Flatten(),
keras.layers.Dense(256, activation='relu'),
keras.layers.Dense(10, activation='softmax')
```

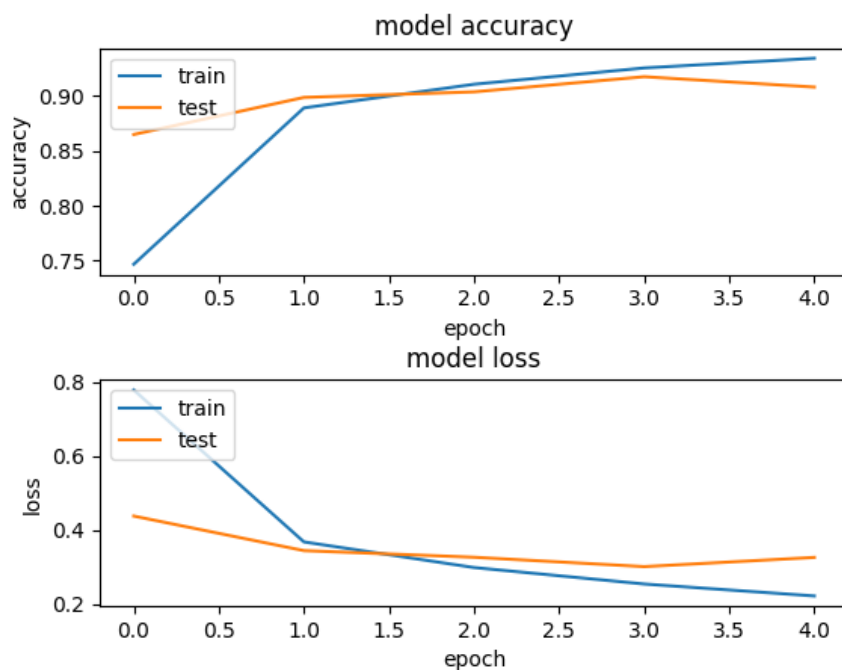


Рисунок 12 - График точности и ошибок модели 2.1

Потери на тестовых данных: 0.33

Точность на тестовых данных: 0.90

В процессе проверки обучения на устойчивость несколько раз наблюдалось переобучение.

Модель 2.2

В целях избавления от переобучения и увеличения точности добавим dropout и batch_normalization.

```
keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu', input_shape=(32, 32, 3)),
keras.layers.BatchNormalization(),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.1),
keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.1),

keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
keras.layers.BatchNormalization(),
keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.1),
```

```

keras.layers.Flatten(),
keras.layers.Dense(256, activation='relu'),
keras.layers.Dropout(0.1),
keras.layers.Dense(10, activation='softmax')

```

Потери на тестовых данных: 0.27

Точность на тестовых данных: 0.926

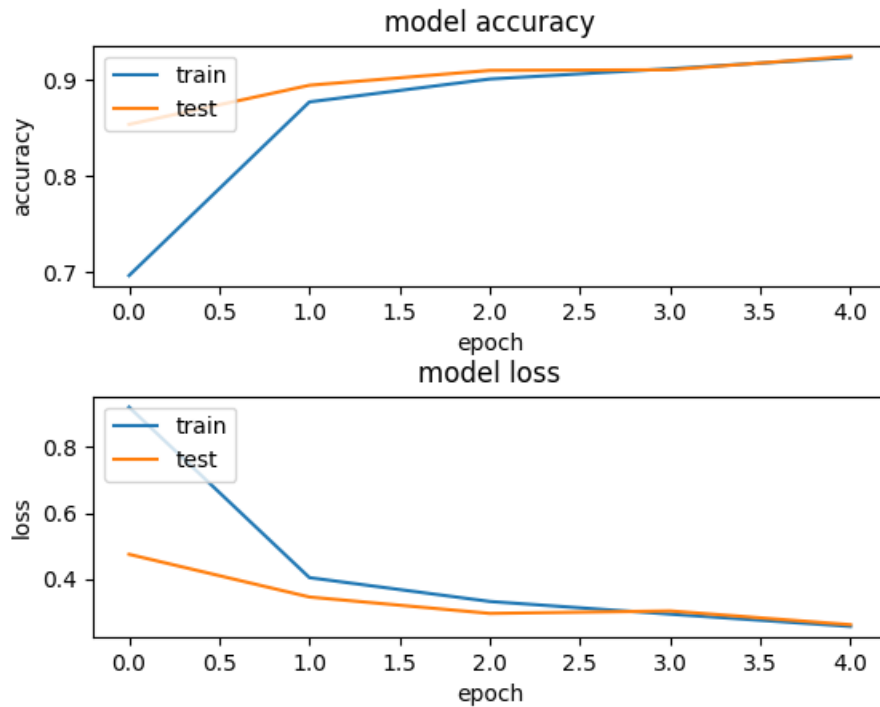


Рисунок 13 - График точности и ошибок модели 2.2

Модель 2.3

Попробуем добиться большей точности усложнением модели. Добавим еще один «блок» сверток и dropout.

```

model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu', input_shape=(32, 32, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.3),

    keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
    keras.layers.BatchNormalization(),

```

```

keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.3),

keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
keras.layers.BatchNormalization(),
keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.3),

keras.layers.Flatten(),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dropout(0.4),
keras.layers.Dense(10, activation='softmax')
])

```

Результат:

Потери на тестовых данных: 0.215

Точность на тестовых данных: 0.945

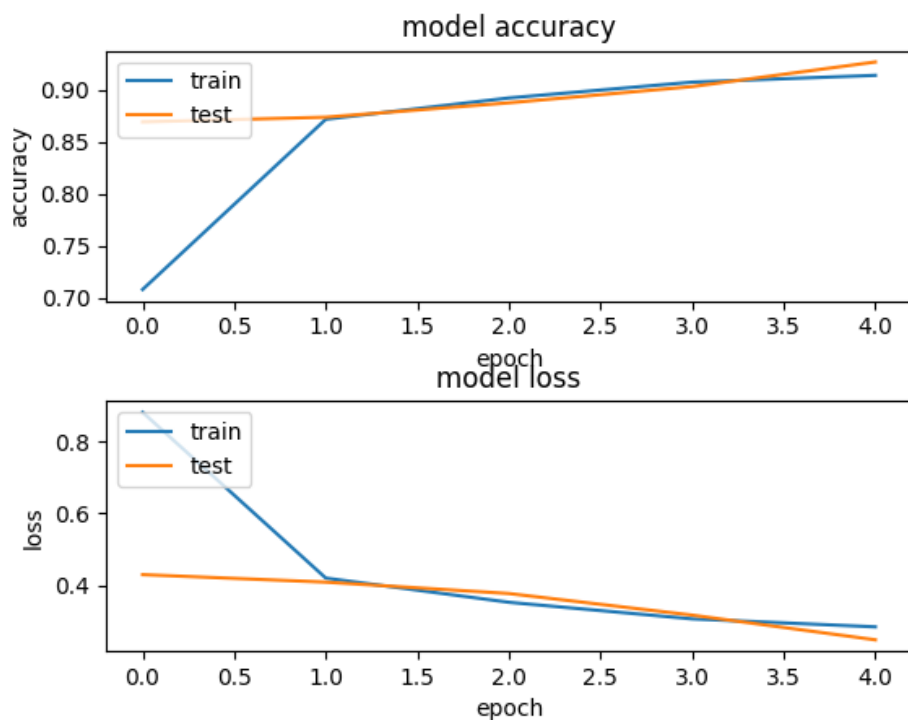


Рисунок 14 - График точности и ошибок модели 2.3

Создание ансамбля.

Покажем для начала графически, как обе модели по отдельности справляются с классификацией на тестовом датасете.

На рисунках ниже приведены confusion matrix для моделей 1 и 2 соответственно. Из них можно извлечь следующую полезную информацию - в чем каждая модель ошибается чаще всего.

Точность модели 1 ниже, но она гораздо легче. Можно заметить что эта модель справляется с определением единицы на картинках заметно лучше второй (разница на тестовых датасетах - 100 раз).

Однако же с остальными метками усложненная модель 2 справляется чуть лучше.

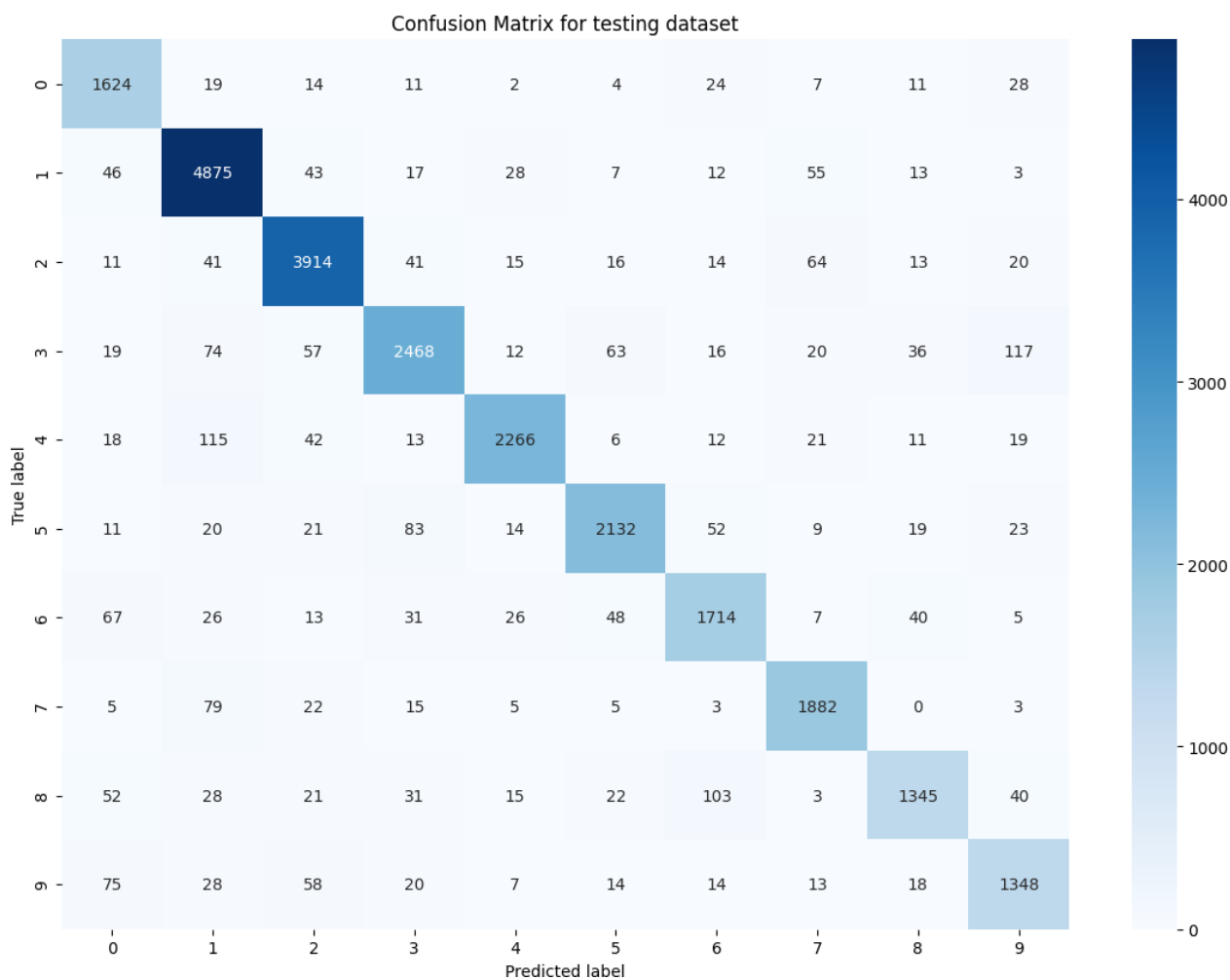


Рисунок 15 - Модель 1

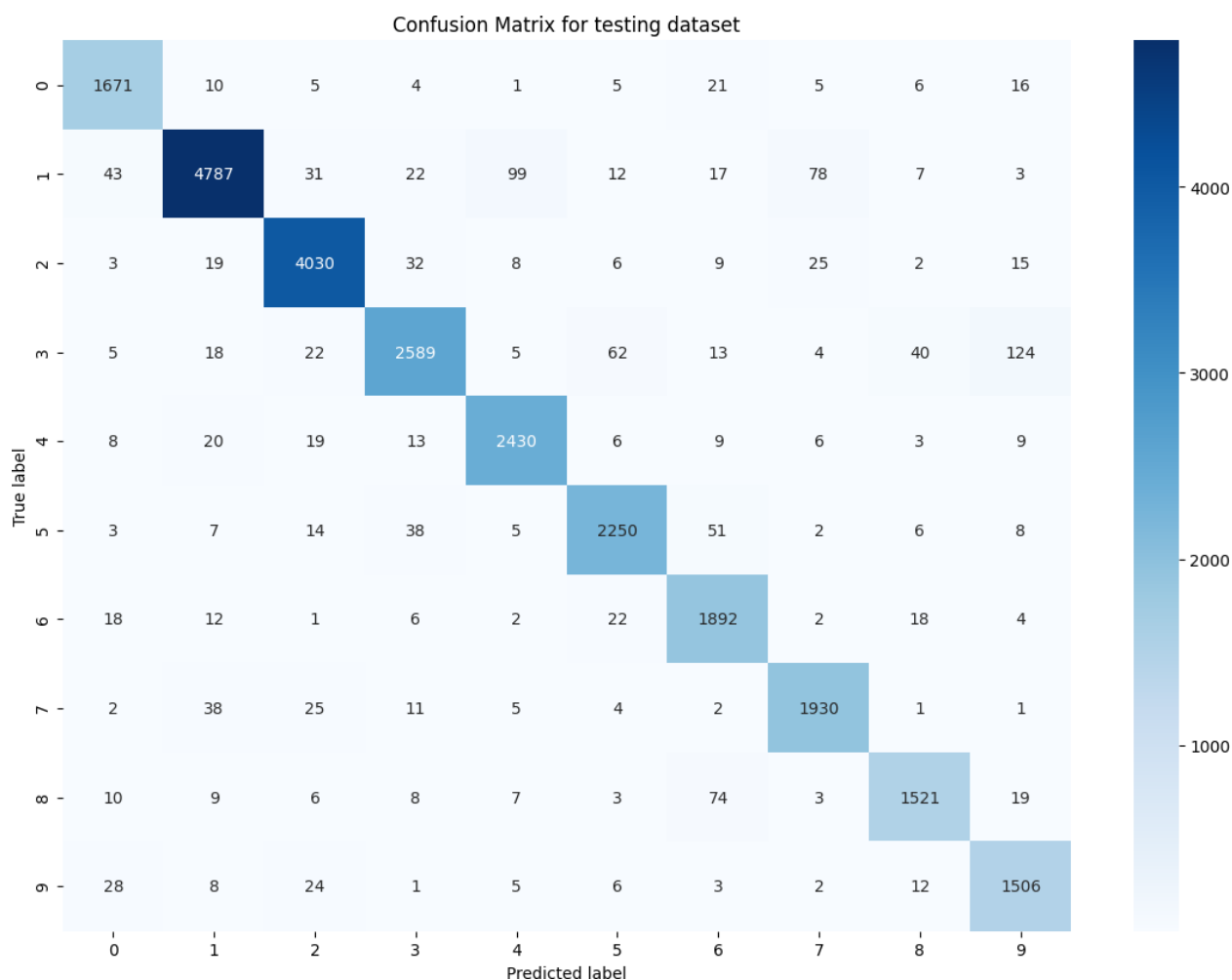


Рисунок 16 - Модель 2

Модели создавались независимо друг от друга, и все-таки имеют различия в предсказаниях, поэтому было решено объединить их в ансамбль. Наиболее удачным выбором оказались равные веса, т.е. предсказание - среднее арифметическое каждой модели.

Большее доверие какой-либо модели ухудшало точность.

Точность ансамбля составила 94.88%, а точность каждой отдельной модели – 90.53% и 94.52% соответственно.

Confusion matrix для ансамбля приведена на рис. 17.

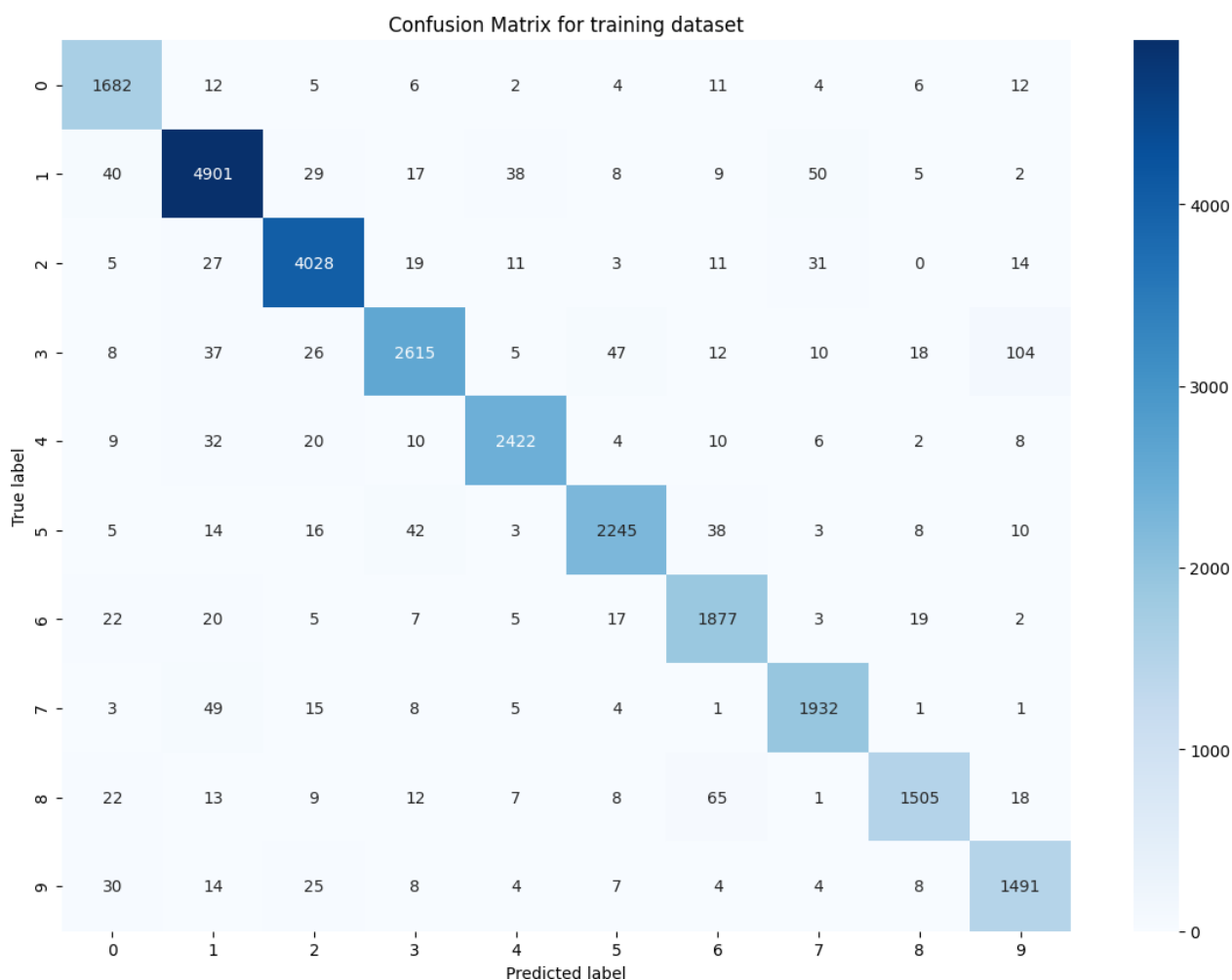


Рисунок 17 - Ансамбль

Анализ моделей

Модель 2 очень громоздка и долго обучается для достижения точности в 94%. Вероятно, глубину модели можно сократить, если попробовать использовать в модели некоторые иные техники, например resNet.

Поскольку модель 1 гораздо лучше справляется с меткой "1", логичнее было бы доверять ей больше, когда предсказываемый ею класс "1", в остальных же случаях следует доверять чуть больше модели 2. Текущий ансамбль просто взвешивает оба предсказания, что может быть не совсем логичным. В условиях ограниченного времени идею полностью строгой и логичной реализации ансамбля пришлось оставить. Теоретически же, эта идея может еще повысить точность.

Выводы.

В ходе лабораторной работы был разработан ансамбль из двух моделей для выделения на изображении цифр. Результирующая модель дает точность 94.88% при классификации изображений.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД MAIN.PY

```
import numpy as np
from sklearn.metrics import accuracy_score

from plots import print_image, plots, plot_distribution_y,
plot_distribution_X, plot_heatmap
from models import build_model_1, build_model_2, save_model,
load_model, ensemble_prediction
from data import load_data, prepare_data
from my_callback import show_time_callback
from keras import optimizers
from keras.utils.vis_utils import plot_model

batch_size = 50
num_epochs = 4

# Загрузка данные
train_X, train_y, test_X, test_y = load_data()

# Построение графиков распределения для X и y
plot_distribution_y(train_y, test_y)
plot_distribution_X(train_X[:, :, :, 0], [0, 255])

# Вывод загруженного изображения
print_image(train_X, train_y, 0)

num_classes = np.unique(train_y).shape[0]

# Подготовка данных (нормирование и т.д.)
train_X, train_y, test_X, test_y = prepare_data(train_X, train_y,
test_X, test_y, num_classes)
num_train, depth, height, width = train_X.shape

# Построение и обучение модели 1
model_1 = build_model_1(depth, height, width, num_classes)
history = model_1.fit(train_X, train_y, batch_size=batch_size,
nb_epoch=num_epochs, verbose=1, validation_split=0.1,
callbacks=[show_time_callback()])
plots(history)

# Построение и обучение модели 2
batch_size = 64
num_epochs = 5

model_2 = build_model_2()
history = model_1.fit(train_X, train_y, batch_size=batch_size,
nb_epoch=num_epochs, verbose=1, validation_split=0.1,
callbacks=[show time callback()])
```

```

plots(history)

# Сохранение модели.
# i - индекс для названия файлов.
save_model(model_1, i=11)

# Загрузка модели 1 из файлов
loaded_model_1 = load_model('model_1.json', 'model_1.h5')
loaded_model_1.compile(loss='categorical_crossentropy',
optimizer='rmsprop', metrics=['accuracy'])

# Загрузка модели 2 из файлов
loaded_model_2 = load_model('model_2.json', 'model_2.h5')
optimizer = optimizers.Adam(lr=1e-3, amsgrad=True)
loaded_model_2.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

print(model_1.evaluate(test_X, test_y, verbose=1))
print(loaded_model_1.evaluate(test_X, test_y, verbose=1))

print(model_2.evaluate(test_X, test_y, verbose=1))
print(loaded_model_2.evaluate(test_X, test_y, verbose=1))

# Построение confusion matrix
y_pred = model_1.predict(test_X)
y_pred = np.argmax(y_pred, axis=1)
y_test = np.argmax(test_y, axis=1)
plot_heatmap(y_test, y_pred)

y_pred = model_2.predict(test_X)
y_pred = np.argmax(y_pred, axis=1)
plot_heatmap(y_test, y_pred)

# Ансамблирование
y_hat_eq = ensemble_prediction([model_1, model_2], [0.5, 0.5],
test_X)
y_test = np.argmax(test_y, axis=1)
plot_heatmap(y_test, y_hat_eq)
acc = accuracy_score(y_test, y_hat_eq)
print("Точность ансамбля:%.2f%%" % (acc * 100))

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД DATA.PY

```
import scipy.io as sio
import numpy as np
from keras.utils import np_utils
```

```
# функция для считывания датасета
```

```
def load_data():
    mat_train = sio.loadmat('train_32x32.mat')
    train_X = mat_train['X']
    train_y = mat_train['y']
    mat_test = sio.loadmat('test_32x32.mat')
    test_X = mat_test['X']
    test_y = mat_test['y']
    return train_X, train_y, test_X, test_y
```

```
# подготовка данных
```

```
def prepare_data(train_X, train_y, test_X, test_y, num_classes):
    # транспонируем из формы [32,32,3,73257] в [73257, 32, 32, 3]
    train_X = train_X.astype('float32').transpose((3, 0, 1, 2))
    test_X = test_X.astype('float32').transpose((3, 0, 1, 2))
```

```
    train_X /= np.max(train_X) # нормируем
    test_X /= np.max(test_X)
```

```
на 0    train_y[train_y == 10] = 0 # т.к. у 0 метка 10, заменяем ее
```

```
    test_y[test_y == 10] = 0
```

```
    train_y = np_utils.to_categorical(train_y, num_classes)
    test_y = np_utils.to_categorical(test_y, num_classes )
    return train_X, train_y, test_X, test_y
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД MODELS.PY

```
from keras.models import Model
from keras.layers import Input, Convolution2D, MaxPooling2D,
Dense, Dropout, Flatten
from keras.models import model_from_json
from keras.utils.vis_utils import plot_model
import keras
import numpy as np

kernel_size = 5
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.2
drop_prob_2 = 0.4
hidden_size = 512

# Построение модели
def build_model_1(depth, height, width, num_classes):
    inp = Input(shape=(depth, height, width))

    conv_1 = Convolution2D(conv_depth_1, kernel_size,
kernel_size,
                                border_mode='same',
activation='relu')(inp)
    pool_1 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_1)
    drop_1 = Dropout(drop_prob_1)(pool_1)

    conv_2 = Convolution2D(conv_depth_2, kernel_size,
kernel_size,
                                border_mode='same',
activation='relu')(drop_1)
    pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_2)
    drop_2 = Dropout(drop_prob_1)(pool_2)

    conv_3 = Convolution2D(conv_depth_2, kernel_size,
kernel_size,
                                border_mode='same',
activation='relu')(drop_2)
    pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_3)

    flat = Flatten()(pool_2)
    dense_1 = Dense(hidden_size, activation='relu')(flat)
    drop_3 = Dropout(drop_prob_2)(dense_1)
```

```

out = Dense(num_classes, activation='softmax')(drop_3)

model = Model(input=inp, output=out)

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

# Схематичное изображение модели
plot_model(model, to_file='model_11_plot.png',
show_shapes=True, show_layer_names=True)
return model

def build_model_2():
    model = keras.Sequential([
        keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu', input_shape=(32, 32, 3)),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(32, (3, 3), padding='same',
activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Dropout(0.3),

        keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Dropout(0.3),

        keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Dropout(0.3),

        keras.layers.Flatten(),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(10, activation='softmax')
    ])
    optimizer = keras.optimizers.Adam(lr=1e-3, amsgrad=True)
    model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
    plot_model(model, to_file='model_2_plot.png',
show_shapes=True, show_layer_names=True)
    return model

```



```

def ensemble_prediction(models, weights, X):
    pred1 = models[0].predict(X)
    pred2 = models[1].predict(X)
    weighted_res = pred1*weights[0] + pred2*weights[1]
    return np.argmax(weighted_res, axis=1)

# Сохранение модели
# i - индекс для названия файлов
def save_model(model, i):
    model_json = model.to_json()
    with open("model_"+str(i)+".json", "w") as json_file:
        json_file.write(model_json)
    model.save_weights("model_"+str(i)+".h5")
    print("Модель " + str(i) + " сохранена на диск")

# Загрузка модели
def load_model(model_name, weights_name):
    json_file = open(model_name, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights(weights_name)
    print("Модель и веса успешно загружены")
    return loaded_model

```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД MY_CALLBACK.PY

```
import tensorflow as tf
import datetime

class show_time_callback(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
    def on_epoch_begin(self, epoch, logs=None):
        self.epoch_start = datetime.datetime.now()

    def on_epoch_end(self, epoch, logs=None):
        epoch_end = datetime.datetime.now()

        # Если эпоха не последняя,
        # то выводится время, прошедшее от запуска,
        # и время, которое осталось
        if (epoch != (self.params['epochs'] - 1)):
            print("Прошло времени с запуска", epoch_end -
self.start_train)
            print("Приблизительное время до конца",
                    (epoch_end - self.epoch_start) / (epoch + 1) *
(self.params['epochs'] - (epoch + 1)))
            print("Длительность эпохи", epoch_end- self.epoch_start)

    def on_train_begin(self,epoch, logs={}):
        self.start_train=datetime.datetime.now()

    def on_train_end(self,epoch, logs={}):
        full_time = datetime.datetime.now() - self.start_train
        print("Общее время ",full_time)
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД PLOTS.PY

```
import matplotlib.pyplot as plt
#import cv2
from sklearn.metrics import confusion_matrix
import seaborn as sns
import pandas as pd
import numpy as np

# Вывод изображения после считывания
def print_image(train_X, train_y,i):
    plt.imshow(train_X[:, :, :, i])
    plt.show()
    print(train_y[i])

# Графики потерь и точности
def plots(history):
    plt.subplot(211)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')

    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

# Гистограмма распределения меток
def plot_distribution_y(train_y, test_y):
    plt.hist(x=train_y, bins='auto', color='#0504aa', alpha=0.7,
rwidth=0.85)
    plt.grid(axis='y', alpha=0.75)
    plt.xlabel('y')
    plt.ylabel('Frequency')
    plt.title('Distribution of train_y')
    plt.show()

    plt.hist(x=test_y, bins='auto', color='#0504aa', alpha=0.7,
rwidth=0.85)
    plt.grid(axis='y', alpha=0.75)
```

```

plt.xlabel('y')
plt.ylabel('Frequency')
plt.title('Distribution of test_y')
plt.show()

# Гистограмма цветов изображения
def plot_distribution_X(train_X, range):
    # Закрашенная гистограмма
    img = train_X
    plt.hist(img.ravel(), 256, [0, 256]);
    plt.show()

    # Гистограмма с RGB линиями
    color = ('b', 'g', 'r')
    for i, col in enumerate(color):
        histr = cv2.calcHist([img], [i], None, [256], [0, 256])
        plt.plot(histr, color=col)
        plt.xlim(range)
    plt.show()

def plot_heatmap(y_train, y_pred):
    matrix = confusion_matrix(y_train, y_pred)
    df_cm = pd.DataFrame(matrix, columns=np.unique(y_train),
index=range(10))
    fig, ax = plt.subplots(figsize=(14, 12))
    sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='d', ax=ax)
    plt.title('Confusion Matrix for testing dataset')
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()

```