Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем
Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №4
«Реализация приложения по распознаванию номеров домов»

Магистрант:
гр. 956241
Шуба И.А.

Проверил:
Заливако С. С.

Минск, 2020

ХОД РАБОТЫ

Задание.

Данные: Набор изображений из Google Street View с изображениями номеров домов, содержащий 10 классов, соответствующих цифрам от 0 до 9.

- 73257 изображений цифр в обучающей выборке;
- 26032 изображения цифр в тестовой выборке;
- 531131 изображения, которые можно использовать как дополнение к обучающей выборке.

В двух форматах:

- Оригинальные изображения с выделенными цифрами;
- Изображения размером 32 × 32, содержащих одну цифру.

Данные первого формата можно скачать по ссылкам:

- http://ufldl.stanford.edu/housenumbers/train.tar.gz (обучающая выборка);
- http://ufldl.stanford.edu/housenumbers/test.tar.gz (тестовая выборка);
- http://ufldl.stanford.edu/housenumbers/extra.tar.gz (дополнительные данные).

Данные второго формата можно скачать по ссылкам:

- http://ufldl.stanford.edu/housenumbers/train_32x32.mat (обучающая выборка);
- http://ufldl.stanford.edu/housenumbers/test_32x32.mat (тестовая выборка);
- http://ufldl.stanford.edu/housenumbers/extra_32x32.mat (дополнительные данные).

Описание данных на английском языке доступно по ссылке: http://ufldl.stanford.edu/housenumbers/

Задание 1.

Реализуйте глубокую нейронную сеть (полносвязную или сверточную) и обучите ее на синтетических данных (например, наборы MNIST (http://yann.lecun.com/exdb/mnist/) или notMNIST).

Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf), видео на YouTube (https://www.youtube.com/watch?v=vGPI_JvLoN0).

Задание 2.

После уточнения модели на синтетических данных попробуйте обучить ее на реальных данных (набор Google Street View). Что изменилось в модели?

Задание 3.

Сделайте множество снимков изображений номеров домов с помощью смартфона на ОС Android. Также можно использовать библиотеки OpenCV, Simple CV или Pygame для обработки изображений с общедоступных камер видеонаблюдения (например, https://www.earthcam.com/). Пример использования библиотеки TensorFlow на смартфоне можете воспользоваться демонстрационным приложением от Google (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android).

Результат выполнения:

Задание 1. Реализуйте глубокую нейронную сеть (полносвязную или сверточную) и обучите ее на синтетических данных (например, наборы MNIST (http://yann.lecun.com/exdb/mnist/) или notMNIST). Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf), видео на YouTube (https://www.youtube.com/watch?v=vGPI_JvLoN0).

Объявим несколько констант и загрузим наборы данных MNIST.

```
(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()
```

```
print('Train: ',X_train.shape, y_train.shape)
print('Test: ', X_test.shape, y_test.shape)
```

```
Train:  (60000, 28, 28) (60000,)
Test:  (10000, 28, 28) (10000,)
```

```
RESOLUTION = 28
CLASSES = 11
NUMBERS_GEN = 3
BATCH_SIZE = 128
EPOCHS = 5
INPUT_SHAPE = (RESOLUTION, RESOLUTION * NUMBERS_GEN, 1)
```

Рисунок 1 – Загрузка данных и объявление констант

Отобразим несколько примеров изображений датафрейма. Для этого напишем функцию.

```
def plot_examples(X, y):
    fig, ax = plt.subplots(1, 3, figsize=(16, 4))
    for i,j in enumerate(np.random.randint(0, len(X), 3)):
        ax[i].imshow(X[j], 'binary')
        ax[i].set_title(str(y[j]))
    plt.show()
```

Рисунок 2 – Функция для отображения изображений их набора MNIST

```
plot_examples(X_train, y_train)
```

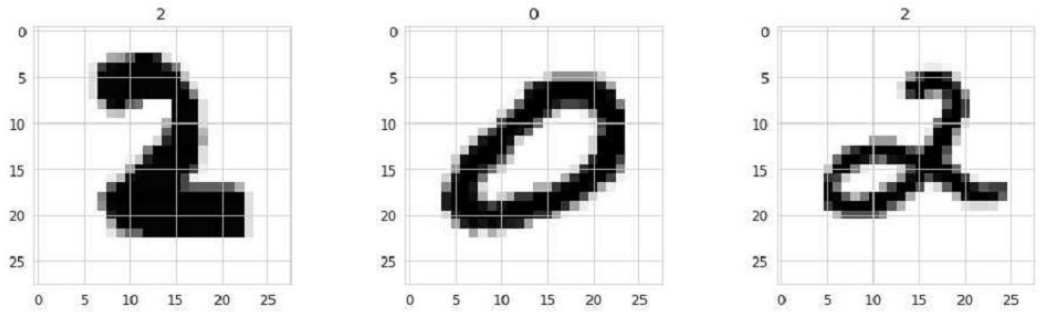


Рисунок 3 – Примеры изображений

Для того, чтобы данные из набора MNIST подходили для обучения, необходимо создать рандомизированные последовательности из чисел из датасета. Выберем максимальное количество цифр в последовательности NUMBERS_GEN = 3, как как в реальной жизни встречаются номера домов, в основном, не превышающие 3 цифры. Теперь напишем функцию, генерирующую последовательности чисел.

Последовательности должны содержать как 1 цифру, так и все. Введем 10 лейбл для пустых мест.

```
def generate_data(X, y, numbers_gen=3):
    x_train_gen, y_train_gen = [], []
    for _ in range(len(X)):
        shape_gen_image = np.zeros((X.shape[1], X.shape[2] * numbers_gen))
        count_numbers = np.random.randint(1, numbers_gen + 1)
        offset = np.random.randint(0, numbers_gen - count_numbers + 1)
        labels = [10] * numbers_gen
        for i in range(offset, offset + count_numbers):
            ind = random.randint(0, X.shape[0] - 1)
            rand_digit = X[ind]
            shape_gen_image[:, i * X.shape[2]: i * X.shape[2] + X.shape[2]] = rand_digi
t
            labels[i] = y[ind]
        x_train_gen.append(shape_gen_image)
        y_train_gen.append(labels)

    return np.array(x_train_gen), np.array(y_train_gen)
```

Рисунок 4 – Функция для создания последовательностей из чисел

Применим написанную функцию к нашим наборам данных и отобразим несколько примеров.

```
x_train_gen, y_train_gen = generate_data(X_train, y_train)
x_test_gen, y_test_gen = generate_data(X_test, y_test)
```

```
plot_examples(x_train_gen, y_train_gen)
```

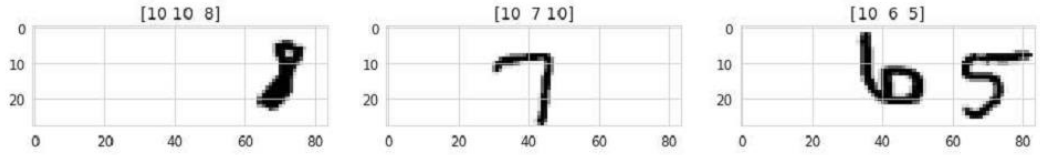


Рисунок 5 – Применение функции для создания последовательностей и отображение примеров

Разделим полученные данные на обучающую, валидационную и тестовую выборки, а далее приведем в вид, который требует сеть (INPUT_SHAPE).

```
X_train, X_val, y_train, y_val = train_test_split(x_train_gen, y_train_gen, test_size=
0.25)
```

```
X_train = X_train.reshape(-1, RESOLUTION, RESOLUTION*NUMBERS_GEN, 1)
y_train = [y_train.T[i] for i in range(NUMBERS_GEN)]
```

```
X_val = X_val.reshape(-1, RESOLUTION, RESOLUTION*NUMBERS_GEN, 1)
y_val = [y_val.T[i] for i in range(NUMBERS_GEN)]
```

```
X_test = x_test_gen.reshape(-1, RESOLUTION, RESOLUTION*NUMBERS_GEN, 1)
y_test = [y_test_gen.T[i] for i in range(NUMBERS_GEN)]
```

```
print('Train: ',X_train.shape, len(y_train))
print('Validation: ', X_val.shape, len(y_val))
print('Test: ', X_test.shape, len(y_test))
```

```
Train:  (45000, 28, 84, 1) 3
Validation:  (15000, 28, 84, 1) 3
Test:  (10000, 28, 84, 1) 3
```

Рисунок 7 – Разделение данных на обучающую, валидационную и тестовую выборки, а также изменение формы данных

Далее необходимо построить архитектуру сети для обработки таких последовательностей. В статье http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf содержится информация о модели, а именно: «Our best architecture consists of eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. All connections are feedforward and go from one layer to the next (no skip connections). The first hidden layer contains maxout units (Goodfellow et al., 2013) (with three filters per unit) while the others contain rectifier units (Jarrett et al., 2009; Glorot et al., 2011). The number of units

at each spatial location in each layer is [48, 64, 128, 160] for the first four layers and 192 for all other locally connected layers. The fully connected layers contain 3,072 units each. Each convolutional layer includes max pooling and subtractive normalization. The max pooling window size is 2 × 2. The stride alternates between 2 and 1 at each layer, so that half of the layers don't reduce the spatial size of the representation. All convolutions use zero padding on the input to preserve representation size. The subtractive normalization operates on 3x3 windows and preserves representation size. All convolution kernels were of size 5 × 5. We trained with dropout applied to all hidden layers but not the input».

Отсюда следует, что в модели 8 скрытых слоев: 2 полносвязных и 6 сверточных. Количество нейронов для сверточных слоев - 48, 64, 128, 160, 192, 192 и в каждом слое размер ядра 5×5. Также перед каждым сверточным слоем следуем слой нормализации, а после слой, который реализует операцию максимального пуллинга с размером ядра 2 × 2, однако в каждом нечетном слое еще происходит изменение «stride» от 2 до 1, так что половина слоев не уменьшает пространственный размер представления. Также сказано, что все сверточные слои используют «same padding». После каждого слоя с операцией максимального пуллинга следуют сброс нейронов с увеличением процента сброса. Стоит учесть, что, так как обучение будет происходить на последовательности из 3 чисел, то выходов у нейронной сети должно быть тоже 3.

Согласно всему вышесказанному, разработаем архитектуру глубокой нейронной сети.

```
inputs = keras.Input(INPUT_SHAPE)

x = layers.BatchNormalization()(inputs)
x = layers.Conv2D(48, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.1)(x)

x = layers.BatchNormalization()(x)
x = layers.Conv2D(64, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2), strides=(1, 1))(x)
x = layers.Dropout(0.1)(x)

x = layers.BatchNormalization()(x)
x = layers.Conv2D(128, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.25)(x)

x = layers.BatchNormalization()(x)
x = layers.Conv2D(160, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2), strides=(1, 1))(x)
x = layers.Dropout(0.25)(x)

x = layers.BatchNormalization()(x)
x = layers.Conv2D(192, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.5)(x)

x = layers.BatchNormalization()(x)
x = layers.Conv2D(192, 5, activation='relu', padding='same')(x)
x = layers.MaxPool2D(pool_size=(2, 2), strides=(1, 1))(x)
x = layers.Dropout(0.5)(x)

x = layers.Flatten()(x)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense(1024, activation='relu')(x)

outputs = []
for i in range(NUMBERS_GEN):
    outputs.append(layers.Dense(CLASSES, activation='softmax')(x))

model = keras.Model(inputs=inputs, outputs=outputs)

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

model.summary()
```

Рисунок 7 – Полученная архитектура глубокой нейронной сети

Обучим полученную ахритектуру.

```
history = model.fit(X_train, y_train,
                    epochs=5,
                    batch_size=BATCH_SIZE,
                    validation_data=(X_val, y_val),
                    verbose=1)
```

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/5
45000/45000 [==============================] - 503s 11ms/sample - loss: 1.
5417 - dense_22_loss: 0.4911 - dense_23_loss: 0.5355 - dense_24_loss: 0.51
36 - dense_22_acc: 0.8308 - dense_23_acc: 0.8075 - dense_24_acc: 0.8215 -
val_loss: 2.8214 - val_dense_22_loss: 0.9151 - val_dense_23_loss: 1.0333 -
val_dense_24_loss: 0.8731 - val_dense_22_acc: 0.7839 - val_dense_23_acc:
0.7387 - val_dense_24_acc: 0.7953
```

Рисунок 8 – Обучение нейронной сети

Построим графики обучения из history.

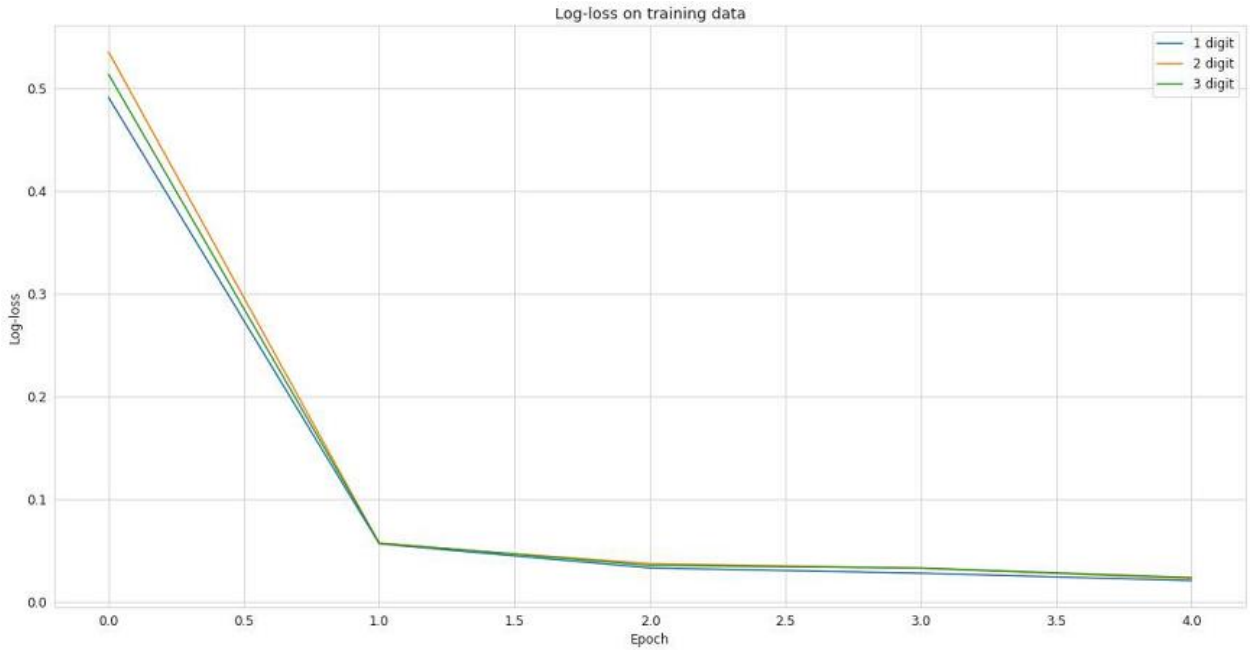


Рисунок 9 – График Log-loss от количества эпох на обучающей выборке

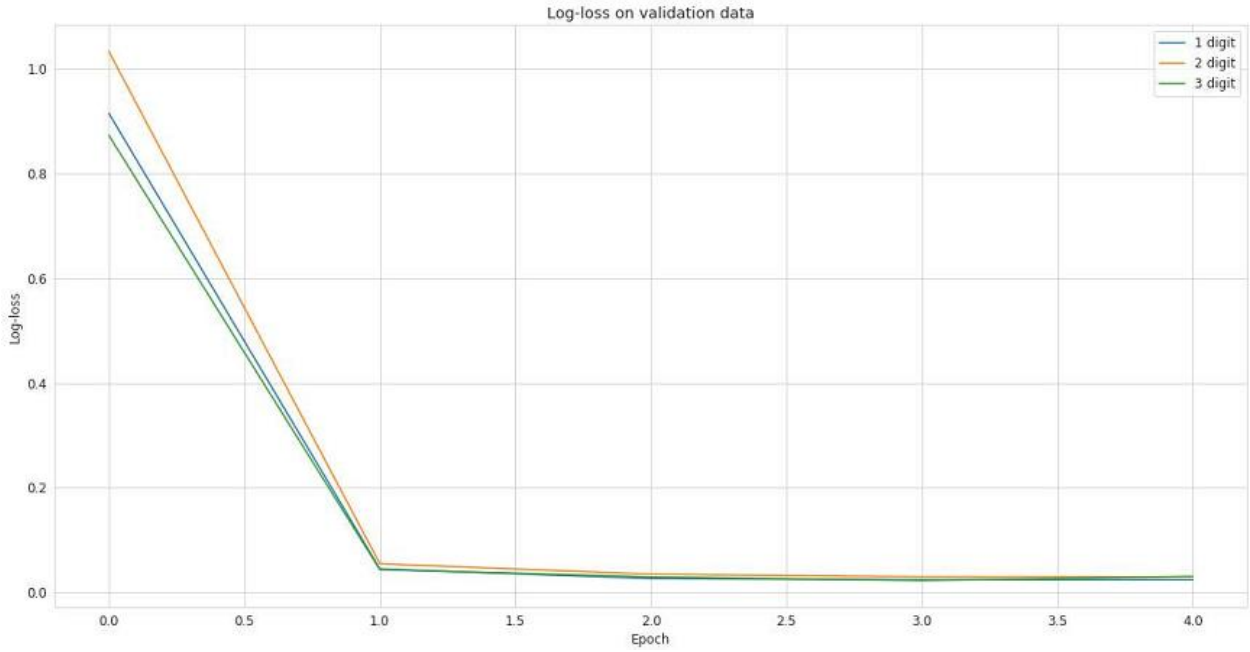


Рисунок 10 – График Log-loss от количества эпох на валидационной выборке

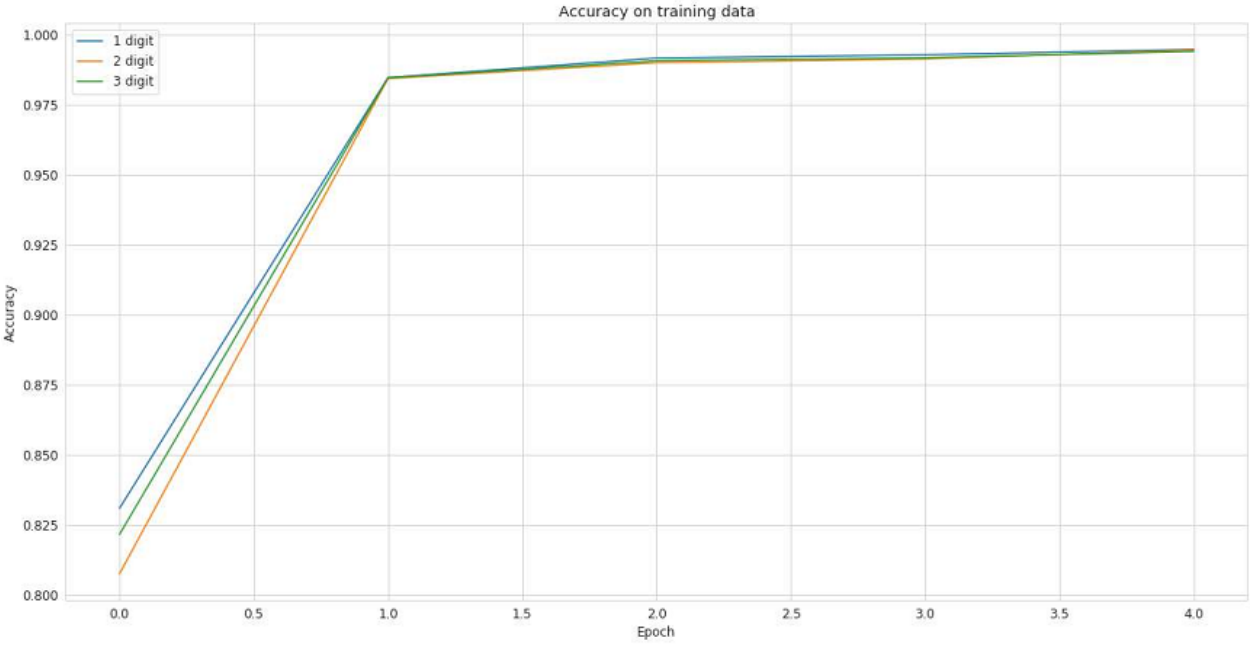


Рисунок 11 – График Accuracy от количества эпох на обучающей выборке

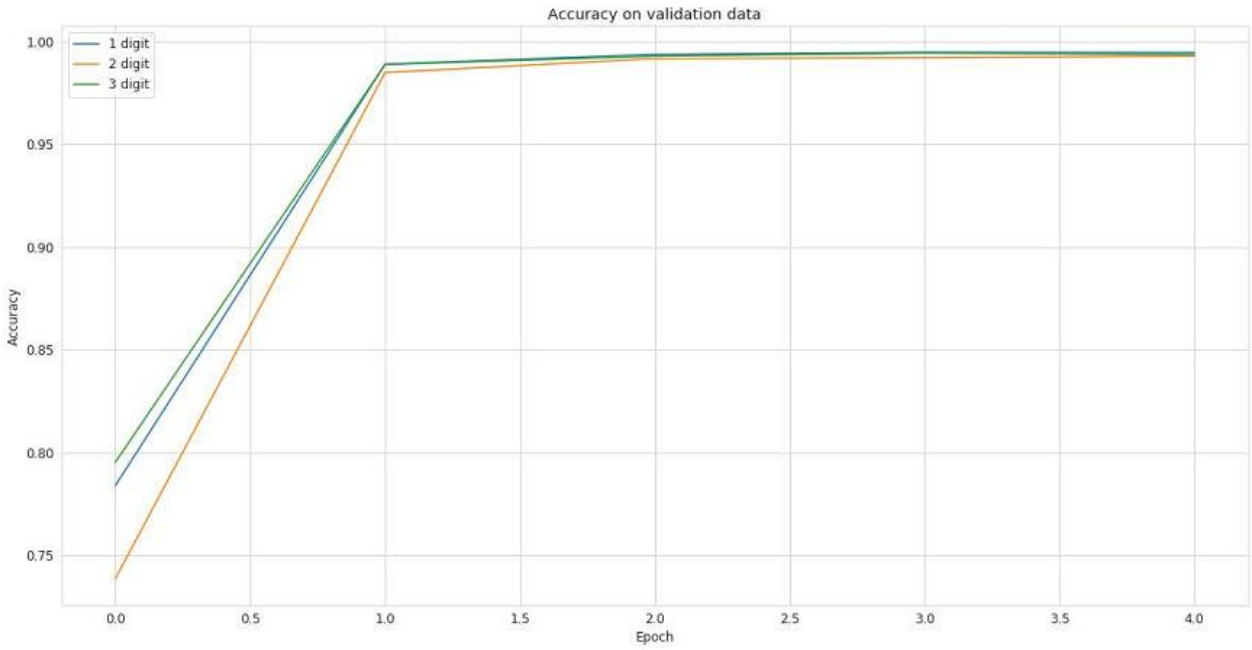


Рисунок 12 – График Accuracy от количества эпох на валидационной выборке

Как видно из графиков, модель не переобучена, уже после 1 эпохи, веса нейронной сети настроились достаточно хорошо. Увеличение числа эпох бы не принесло увеличения точности, а возможно, даже ухудшило результат.

Проверим точность модели на контрольной выборке.

```
model.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 37s 4ms/sample - loss: 0.14
21 - dense_22_loss: 0.0479 - dense_23_loss: 0.0515 - dense_24_loss: 0.0425
- dense_22_acc: 0.9923 - dense_23_acc: 0.9896 - dense_24_acc: 0.9925

Out[376]:

[0.14210880066910758,
 0.04789872,
 0.05145028,
 0.04253537,
 0.9923,
 0.9896,
 0.9925]
```

Рисунок 13 – Результаты обучения на контрольной выборке

Из рисунка 13 следует, что все 3 числа в последовательности обучились достаточно хорошо, что точность составляет: 0,9923, 0,9896, 0,9925.

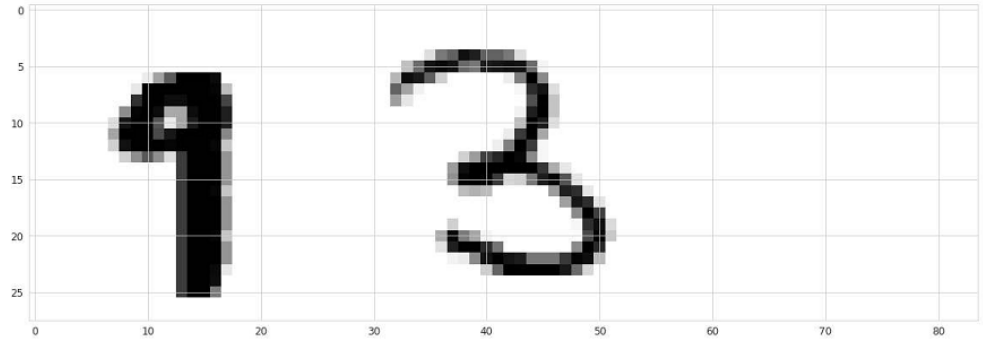
Приведем примеры работы полученной модели. Напишем функцию для отображения.

```python
def check_model(x_test_gen, model):
    for _ in range(5):
        test_image = random.choice(x_test_gen)
        prediction = model.predict(test_image.reshape(1,RESOLUTION,RESOLUTION*NUMBERS_G
EN,1))
        print(f"Prediction: {[np.argmax(i) for i in prediction]}")
        plt.imshow(test_image, cmap='binary')
        plt.show()
```

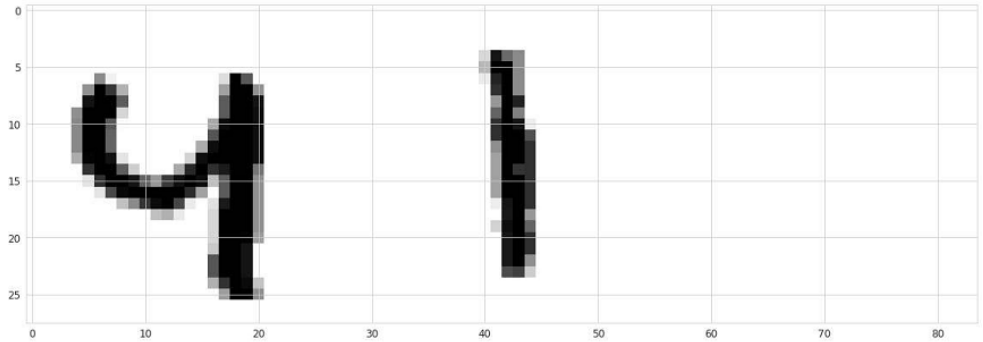
Рисунок 14 – Функция для отображения результатов

```python
check_model(x_test_gen, model)
```

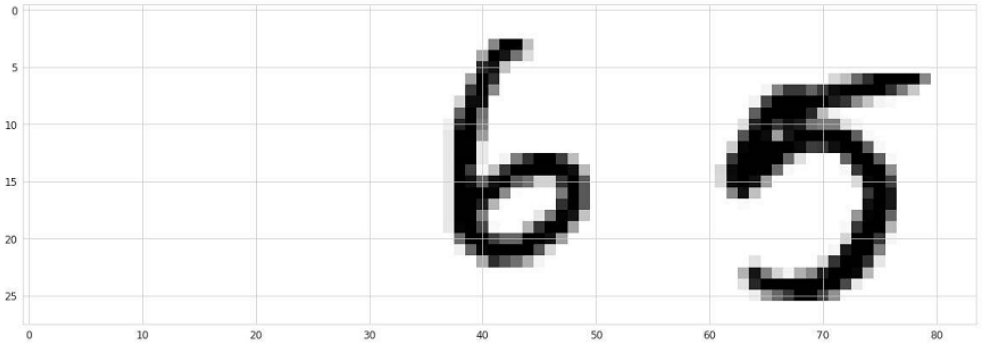
```
Prediction: [9, 3, 10]
```



```
Prediction: [4, 1, 10]
```



```
Prediction: [10, 6, 5]
```



```
Prediction: [4, 6, 0]
```

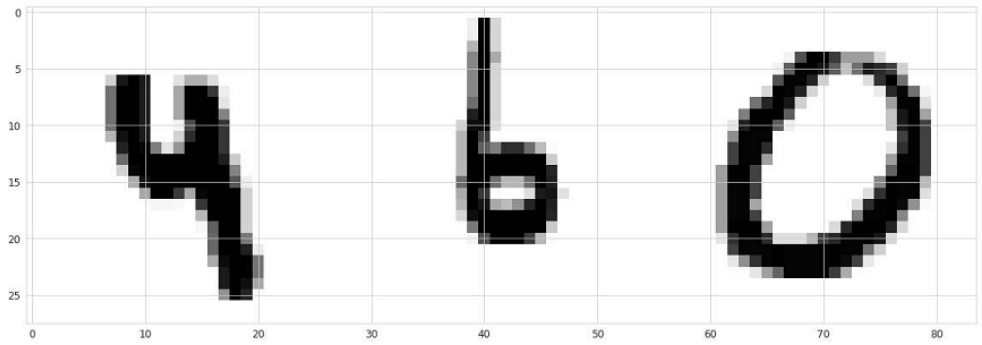


Рисунок 15 – Примеры на обученной модели

Задание 2. После уточнения модели на синтетических данных попробуйте обучить ее на реальных данных (набор Google Street View). Что изменилось в модели?

Выберем пути к необходимым директориям и отобразим несколько примеров изображений

```
train_dir = 'train'
test_dir = 'test'
```

```
fig, ax = plt.subplots(1, 3, figsize=(16, 4))

for i, j in enumerate(np.random.randint(1, len(os.listdir(train_dir)) - 3, 3)):
    img = im.imread(train_dir+f'/{j}.png')
    ax[i].imshow(img)
```

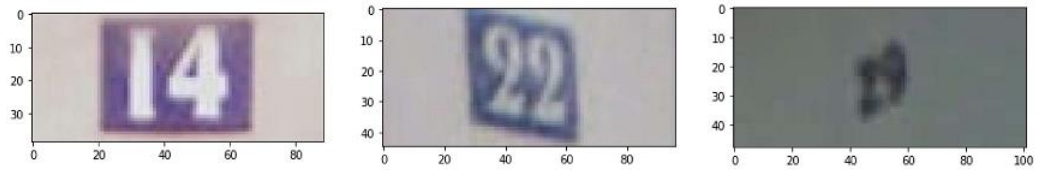


Рисунок 16 – Выбор путей к необходимым директориям и отображение примеров изображений

Далее необходимо из дополнительных файлов к датасету разметить изображений, чтобы на них остались лишь числа, то есть обрезать отображения, а также из метаданных разметить изображения. Напишем класс, который будет это реализовывать.

```
class DigitStructFile:
    def __init__(self, inf):
        self.inf = h5py.File(inf, 'r')
        self.digitStructName = self.inf['digitStruct']['name']
        self.digitStructBbox = self.inf['digitStruct']['bbox']

    def getName(self,n):
        return ''.join([chr(c[0]) for c in self.inf[self.digitStructName[n][0]].value])

    def bboxHelper(self,attr):
        if (len(attr) > 1):
            attr = [self.inf[attr.value[j].item()].value[0][0] for j in range(len(attr
))]
        else:
            attr = [attr.value[0][0]]
        return attr

    def getBbox(self,n):
        bbox = {}
        bb = self.digitStructBbox[n].item()
        bbox['height'] = self.bboxHelper(self.inf[bb]["height"])
        bbox['label'] = self.bboxHelper(self.inf[bb]["label"])
        bbox['left'] = self.bboxHelper(self.inf[bb]["left"])
        bbox['top'] = self.bboxHelper(self.inf[bb]["top"])
        bbox['width'] = self.bboxHelper(self.inf[bb]["width"])
        return bbox

    def getDigitStructure(self,n):
        s = self.getBbox(n)
        s['name']=self.getName(n)
        return s

    def getAllDigitStructure(self):
        return [self.getDigitStructure(i) for i in range(len(self.digitStructName))]

    def getAllDigitStructure_ByDigit(self):
        pictDat = self.getAllDigitStructure()
        result = []
        structCnt = 1
        for i in range(len(pictDat)):
            item = { 'filename' : pictDat[i]["name"] }
            figures = []
            for j in range(len(pictDat[i]['height'])):
                figure = {}
                figure['height'] = pictDat[i]['height'][j]
                figure['label']  = pictDat[i]['label'][j]
                figure['left']   = pictDat[i]['left'][j]
                figure['top']    = pictDat[i]['top'][j]
                figure['width']  = pictDat[i]['width'][j]
                figures.append(figure)
            structCnt = structCnt + 1
            item['boxes'] = figures
            result.append(item)
        return result
```

Рисунок 17 – Класс, обрабатывающий данные об изображениях

Применим реализованный класс и отобразим пример.

```
structure_file = DigitStructFile(train_dir+'/digitStruct.mat')
train_data = structure_file.getAllDigitStructure_ByDigit()
```

```
structure_file = DigitStructFile(test_dir+'/digitStruct.mat')
test_data = structure_file.getAllDigitStructure_ByDigit()
```

```
train_data[0]
```

Out[13]:

```
{'filename': '1.png',
 'boxes': [{'height': 219.0,
   'label': 1.0,
   'left': 246.0,
   'top': 77.0,
   'width': 81.0},
  {'height': 219.0, 'label': 9.0, 'left': 323.0, 'top': 81.0, 'width': 96.
0}]}
```

Рисунок 18 – Применение реализованного класса и отображение примера

Напишем функцию для генерации датасета.

```
import PIL.Image as Image

def generate_dataset(data, folder):
    dataset = np.ndarray([len(data),RESOLUTION, RESOLUTION,1], dtype='float32')
    labels = np.ones([len(data), 6], dtype=int) * 10
    for i in np.arange(len(data)):
        filename = data[i]['filename']
        fullname = os.path.join(folder, filename)
        im = Image.open(fullname)
        boxes = data[i]['boxes']
        num_digit = len(boxes)
        labels[i,0] = num_digit
        top = np.ndarray([num_digit], dtype='float32')
        left = np.ndarray([num_digit], dtype='float32')
        height = np.ndarray([num_digit], dtype='float32')
        width = np.ndarray([num_digit], dtype='float32')
        for j in np.arange(num_digit):
            if j < 5:
                labels[i,j+1] = boxes[j]['label']
                if boxes[j]['label'] == 10:
                    labels[i,j+1] = 0
            top[j] = boxes[j]['top']
            left[j] = boxes[j]['left']
            height[j] = boxes[j]['height']
            width[j] = boxes[j]['width']

        im_top = np.amin(top)
        im_left = np.amin(left)
        im_height = np.amax(top) + height[np.argmax(top)] - im_top
        im_width = np.amax(left) + width[np.argmax(left)] - im_left

        im_top = np.floor(im_top - 0.1 * im_height)
        im_left = np.floor(im_left - 0.1 * im_width)
        im_bottom = np.amin([np.ceil(im_top + 1.2 * im_height), im.size[1]])
        im_right = np.amin([np.ceil(im_left + 1.2 * im_width), im.size[0]])

        im = im.crop((im_left, im_top, im_right, im_bottom)).resize([RESOLUTION,RESOLUT
ION], Image.ANTIALIAS)
        im = np.dot(np.array(im, dtype='float32'), [[0.2989],[0.5870],[0.1140]])
        mean = np.mean(im, dtype='float32')
        std = np.std(im, dtype='float32', ddof=1)
        if std < 1e-4: std = 1.
        im = (im - mean) / std
        dataset[i,:,:,:] = im[:,:,:]

    return dataset, labels
```

Рисунок 19 – Функция для генерации датасета

Сгенерируем размеченный датасет.

```
train_dataset, train_labels = generate_dataset(train_data, train_dir)
print(train_dataset.shape, train_labels.shape)

test_dataset, test_labels = generate_dataset(test_data, test_dir)
print(test_dataset.shape, test_labels.shape)
```

```
(33402, 28, 28, 1) (33402, 6)
(13068, 28, 28, 1) (13068, 6)
```

Рисунок 19 – Функция для генерации датасета

Далее полученный датасет необходимо отфильтровать от изображений, где встречаются последовательности больше 3 чисел. Для этого напишем несколько функций.

```
def find_incorrent_images(labels):
    list_of_incorrent_images = []
    for i in range(len(labels)):
        if labels[i][0] >= NUMBERS_GEN+1:
            list_of_incorrent_images.append(i)
    return list_of_incorrent_images
```

```
def delete_incorrent_images(data, labels):
    incorrent_images = find_incorrent_images(labels)
    data = np.delete(data, incorrent_images, axis=0)
    labels = np.delete(labels, incorrent_images, axis=0)
    labels = np.delete(labels,[0,4,5], axis=1)
    return data, labels
```

```
train_dataset, train_labels= delete_incorrent_images(train_dataset, train_labels)
```

```
test_dataset, test_labels = delete_incorrent_images(test_dataset, test_labels)
```

Рисунок 20 – Фильтрация датасета

Теперь нужно разделить полученный датасет на обучающую, валидационную и контрольную выборки.

```
random.seed(43)

n_labels = 10
valid_index = []
train_index = []
for i in np.arange(n_labels):
    valid_index.extend(np.where(train_labels[:,1] == (i))[0][:400].tolist())
    train_index.extend(np.where(train_labels[:,1] == (i))[0][400:].tolist())

random.shuffle(valid_index)
random.shuffle(train_index)

valid_dataset = train_dataset[valid_index,:,:,:]
valid_labels = train_labels[valid_index,:]
valid_labels = [valid_labels.T[i] for i in range(NUMBERS_GEN)]
train_dataset_t = train_dataset[train_index,:,:,:]
train_labels_t = train_labels[train_index,:]
train_labels_t = [train_labels_t.T[i] for i in range(NUMBERS_GEN)]
test_labels = [test_labels.T[i] for i in range(NUMBERS_GEN)]

print('Train: ', train_dataset_t.shape, len(train_labels_t))
print('Val: ', valid_dataset.shape, len(valid_labels))
print('Test: ', test_dataset.shape, len(test_labels))
```

```
Train:  (22821, 28, 28, 1) 3
Val:  (4000, 28, 28, 1) 3
Test:  (12920, 28, 28, 1) 3
```

Рисунок 21 – Разделение датасета

Приведем несколько примеров полученного размеченного датасета.

```
fig, ax = plt.subplots(1, 3, figsize=(16, 4))

for i, j in enumerate(np.random.randint(0, len(train_dataset_t), 3)):
    ax[i].imshow(train_dataset_t[j].reshape(RESOLUTION,RESOLUTION), cmap='binary')
    ax[i].set_title(train_labels_t[j])
```

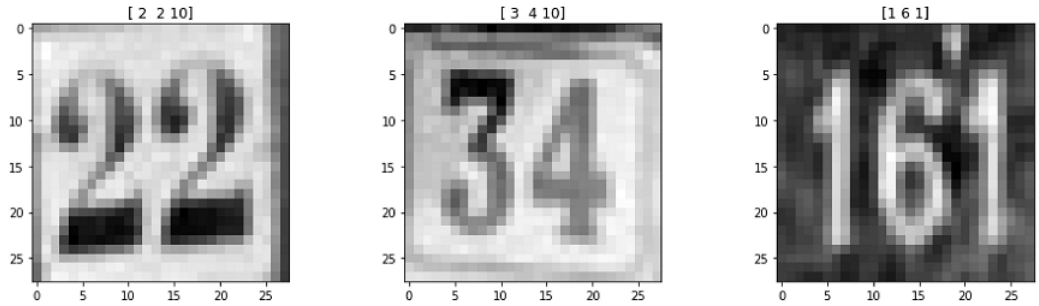


Рисунок 22 – Примеры размеченных изображений

Используя архитектуру глубокой нейронной сети (рисунок 7), обучим ее на полученном наборе данных.

```
history = model.fit(train_dataset_t, train_labels_t,
                    epochs=20,
                    batch_size=BATCH_SIZE,
                    validation_data=(valid_dataset, valid_labels),
                    verbose=1)
```

```
Train on 22821 samples, validate on 4000 samples
Epoch 1/20
22821/22821 [==============================] - 84s 4ms/sample - loss: 4.44
71 - dense_17_loss: 1.4832 - dense_18_loss: 1.9254 - dense_19_loss: 1.0332
- dense_17_acc: 0.4889 - dense_18_acc: 0.2910 - dense_19_acc: 0.6864 - val
_loss: 7.2134 - val_dense_17_loss: 1.6202 - val_dense_18_loss: 1.9735 - va
l_dense_19_loss: 3.6190 - val_dense_17_acc: 0.4325 - val_dense_18_acc: 0.3
045 - val_dense_19_acc: 0.0553
```

Рисунок 23 – Обучение на реальных данных

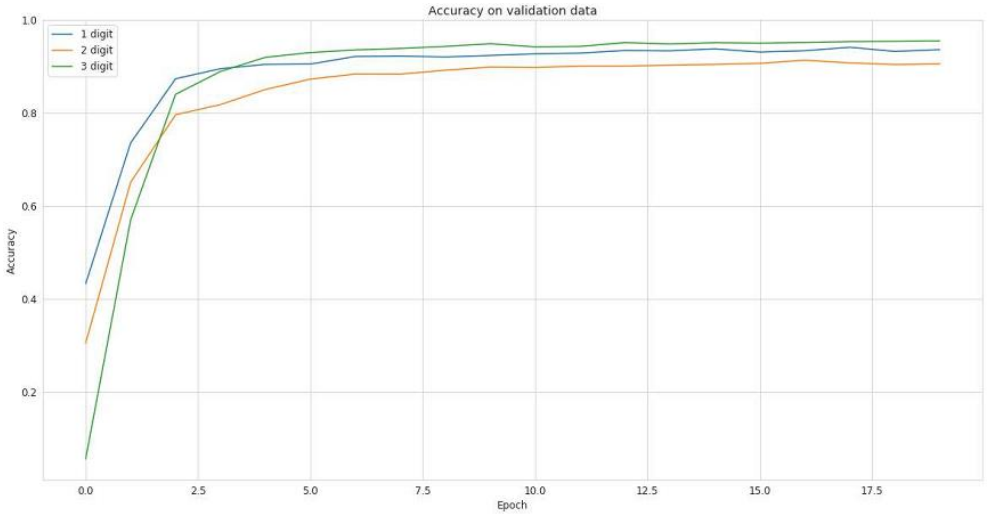


Рисунок 24 – График Accuracy от количества эпох
на валидационной выборке

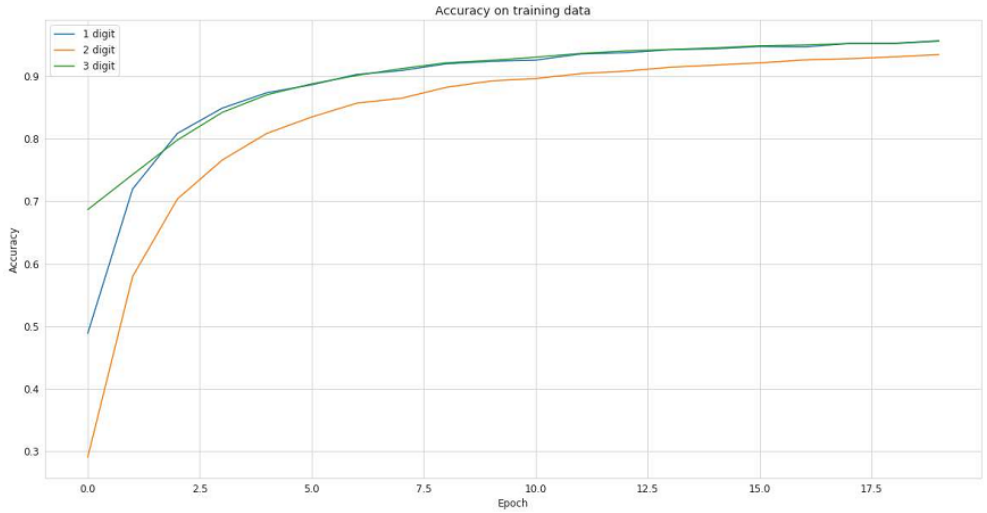


Рисунок 25 – График Accuracy от количества эпох на обучающей выборке

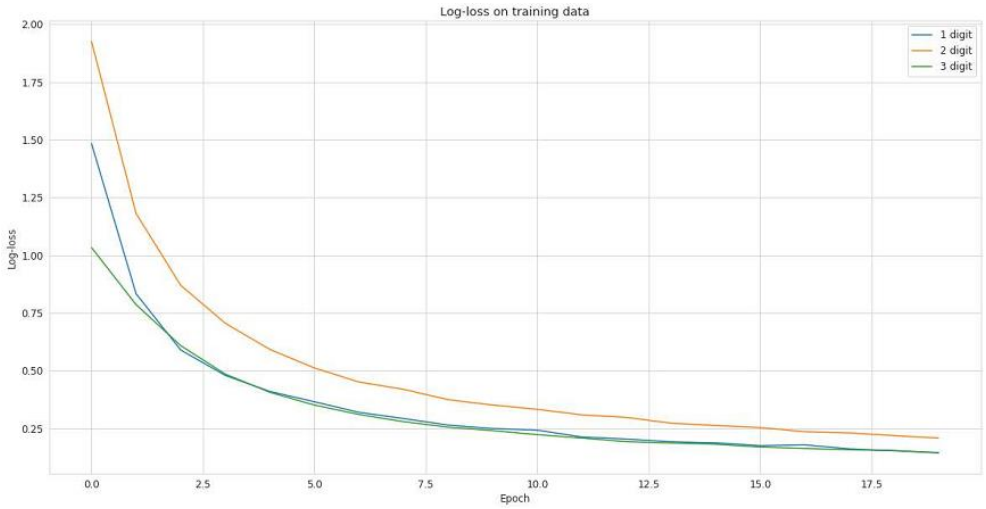


Рисунок 26 – График Log-loss от количества эпох на валидационной выборке

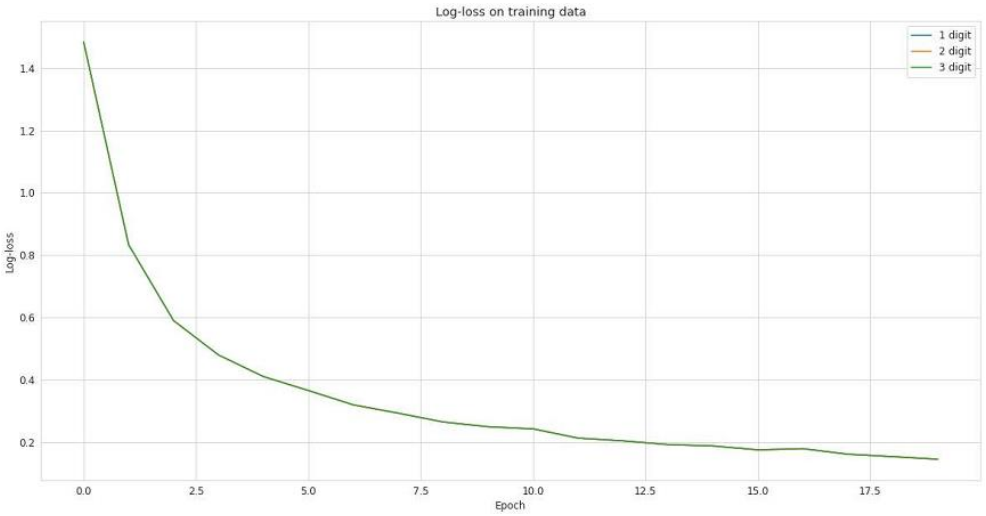


Рисунок 27 – График Log-loss от количества эпох на обучающей выборке

Как видно из графиков, модель не переобучена. Однако хорошо заметно, что 2 цифра в последовательности обучилась хуже всего, а чем свидетельствуют графики и точности и потерь, которые заметно отличаются от двух остальных чисел.

Проверим точность модели на контрольной выборке.

```
model.evaluate(test_dataset, test_labels)
```

```
12920/12920 [==============================] - 26s 2ms/sample - loss: 3.83
62 - dense_17_loss: 0.5359 - dense_18_loss: 3.1361 - dense_19_loss: 0.1637
- dense_17_acc: 0.8515 - dense_18_acc: 0.7302 - dense_19_acc: 0.9599

Out[357]:

[3.836177135916317,
 0.53587496,
 3.1361353,
 0.16365573,
 0.8514706,
 0.73018575,
 0.9599071]
```

Рисунок 28 – Результаты обучения на контрольной выборке

Из рисунка 28 следует, что числа в последовательности обучились достаточно хорошо, что точность составляет: 0,85, 0,73, 0,95. Как предполагалось, точность 2 цифры заметно ниже. Отсюда следует, что следует, что можно расширить выборку для обучения или поднастроить модель.

Приведем примеры работы полученной модели. Напишем функцию для отображения.

```
def check_model(x_test_gen, model):
    fig = plt.figure(figsize=(12, 10))
    for i in range(1, 10):
        ax = fig.add_subplot(3, 3, i)
        test_image = random.choice(test_dataset)
        prediction = model.predict(test_image.reshape(1,RESOLUTION,RESOLUTION,1))
        ax.set_title(f"Prediction: {[np.argmax(i) for i in prediction]}")
        ax.set_axis_off()
        plt.imshow(test_image.reshape(RESOLUTION, RESOLUTION), cmap='binary')
```

Рисунок 29 – Функция для отображения результата обучения модели

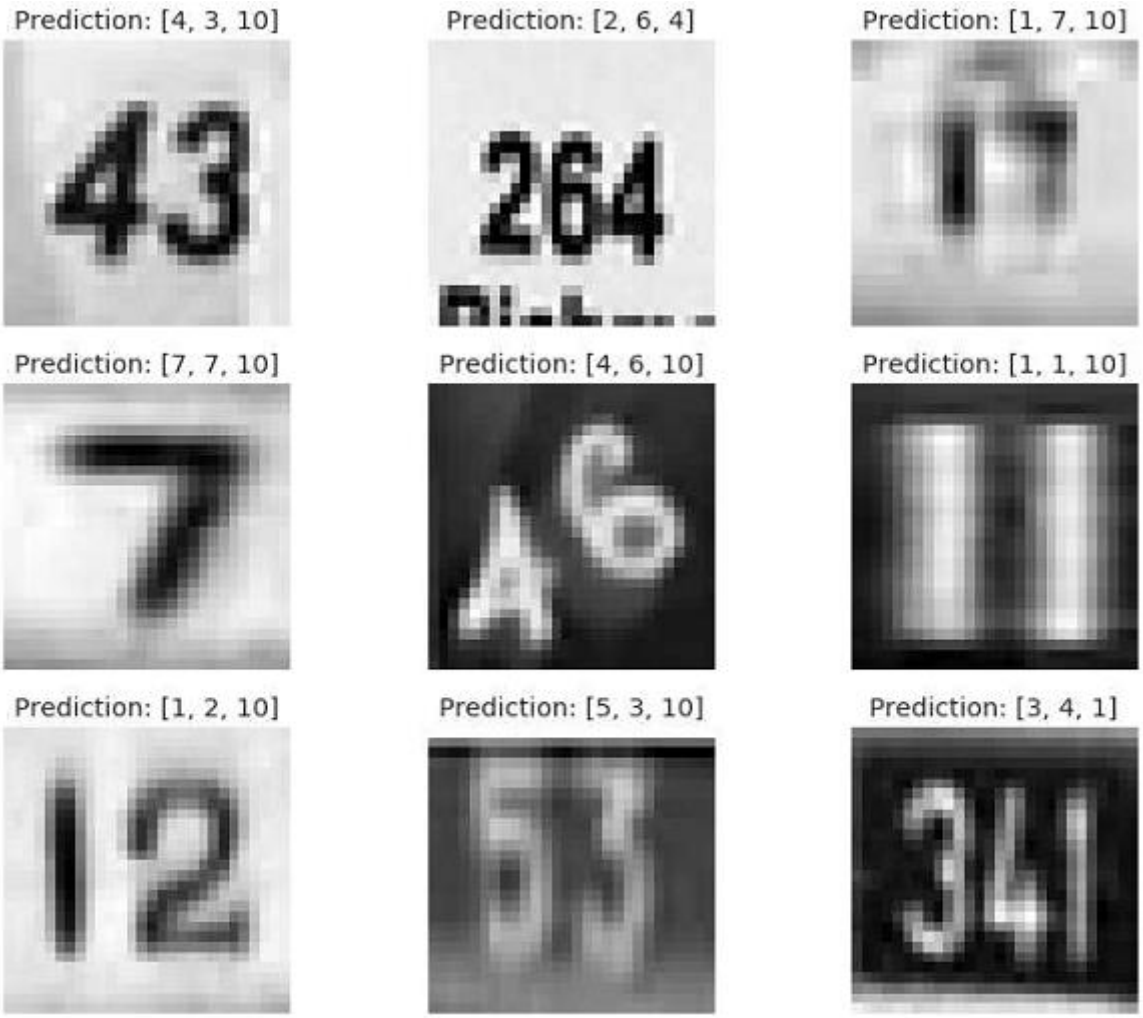


Рисунок 30 – Примеры изображений с их предсказаниями

Задание 3. Сделайте множество снимков изображений номеров домов с помощью смартфона на ОС Android. Также можно использовать библиотеки OpenCV, Simple CV или Pygame для обработки изображений с общедоступных камер видеонаблюдения (например, https://www.earthcam.com/). Пример использования библиотеки TensorFlow на смартфоне можете воспользоваться демонстрационным приложением от Google
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android

С помощью ОС Android было сделано порядка 12 снимков различных табличек с номерами домов. Напишем функцию и отобразим все изображения.

```python
def show_images(prefix=''):
    fig = plt.figure(figsize=(15, 15))
    for number in range(1, 13):
        ax = fig.add_subplot(3, 4, number)
        image = Image.open(f'Images/{prefix}{number}.jpg')
        ax.set_axis_off()
        plt.imshow(image)
```

```python
show_images()
```

Рисунок 31 – Примеры изображений сделанных на мобильный телефон

Обрежем изображения и изменим их размер на стандартный для d[jlf сети 28x28.

Рисунок 32 – Примеры обрезанных изображений сделанных на мобильный телефон

Далее напишем функцию для построения прогноза на обрезанных изображениях.

```
def show_prediction():
    fig = plt.figure(figsize=(15, 15))
    for number in range(1, 13):
        ax = fig.add_subplot(4, 3, number)
        ax.set_axis_off()
        image = Image.open(f'Images/crop_{number}.jpg')
        plt.imshow(image)
        image = np.dot(np.array(image, dtype='float32'), [[0.2989],[0.5870],[0.1140]])
        image /= 255
        prediction = model.predict(image.reshape(1,RESOLUTION,RESOLUTION,1))
        ax.set_title([np.argmax(i) for i in prediction])
```

Рисунок 33 – Функция для построения прогноза на обрезанных изображениях

Результат выполнения функции:

Рисунок 33 – Примеры прогноза модели на обрезанных изображениях, сделанных на мобильный телефон

Вывод:

В ходе выполнения лабораторной работы был изучен датасет MNIST, а также набор Google Street View. Была реализована глубокая нейронная сеть, описанная в статье под заданием. На этой сети обучались склеенные данные с набора MNIST в последовательность из 3 чисел (так как в основном встречаются максимум трехзначные числа), общая точность модели составила 0,9923, 0,9896, 0,9925 для каждой из чисел последовательности соответственно. А также набор данных Google Street View, точность модели была 0,85, 0,73, 0,95, что хуже, чем разработанный синтетический датасет. Также были проверены реальные изображения на полученной модели, и стоит отметить, что результат достаточно неплохой. Из 12 изображений в 4, и

то, в отдельных числах последовательности, были допущены ошибки моделью.