Лабораторная работа №4. Реализация приложения по распознаванию номеров домов.

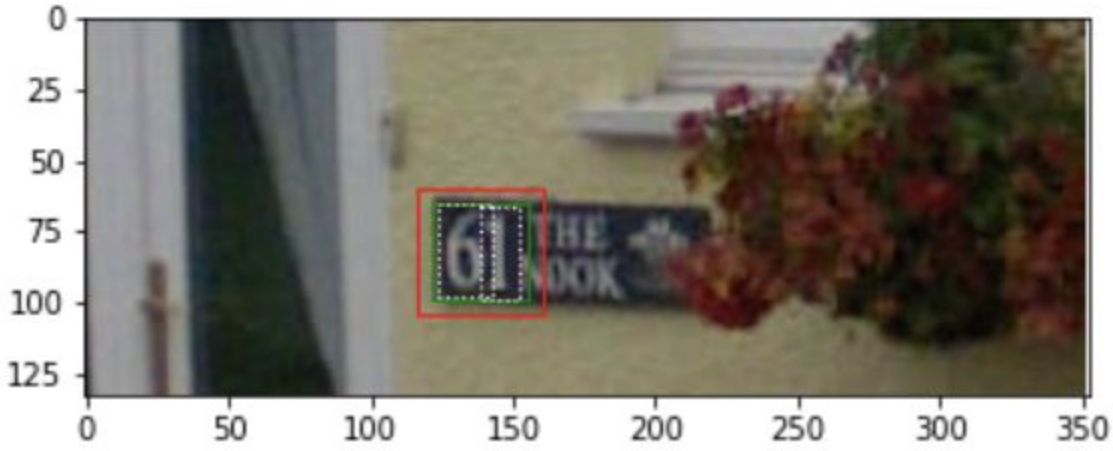
Данные: Набор изображений из Google Street View с изображениями номеров домов, содержащий 10 классов, соответствующих цифрам от 0 до 9.

- 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/

Ход работы:

Для начала подготовим данные для работы, возьмем исходный набор данных, считаем и распарсим .mat файлы и подготовим картинки, в исходном наборе представлены координаты прямоугольника в котором находятся сами номера, обрежем картинки немного большего размера чем исходные прямоугольники. Разобьем данные на тренировочный, валидационный и тестовый набор.

Пример:



Код подготовки данных:

```
def preprocess(image, bbox_left, bbox_top, bbox_width, bbox_height):
  cropped_left, cropped_top, cropped_width, cropped_height = (int(round(bbox_left - 0.15 * bbox_width)),
                                                              int(round(bbox_top - 0.15 * bbox_height)),

int(round(bbox_width * 1.3)),

int(round(bbox_height * 1.3)))
   image = image.crop([cropped_left, cropped_top, cropped_left + cropped_width, cropped_top + cropped_height])
 image = image.resize([64, 64])
 return image
```

Реализуем модель описанную в работе https://arxiv.org/abs/1312.6082

```
class Model(nn.Module):
   CHECKPOINT_FILENAME_PATTERN = 'model-{}.pth'

   __constants__ = [
                  '_hidden1', '_hidden2', '_hidden3',
                   '_hidden4', '_hidden5',
                   '_features', '_classifier',
                        '_digit_length', '_digit1', '_digit2', '_digit3', '_digit4', '_digit5']

   def __init__(self):
       super(Model, self).__init__()
```

```
        self._hidden1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, padding=2),
            nn.BatchNorm2d(num_features=16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            nn.Dropout(0.2)
        )
        self._hidden2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=48, kernel_size=5, padding=2),
            nn.BatchNorm2d(num_features=48),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
            nn.Dropout(0.2)
        )
        self._hidden3 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=64, kernel_size=5, padding=2),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            nn.Dropout(0.2)
        )
        self._hidden9 = nn.Sequential(
            nn.Linear(14400, 3072),
            nn.ReLU()
        )
        self._hidden10 = nn.Sequential(
            nn.Linear(3072, 3072),
            nn.ReLU()
        )

        self._digit_length = nn.Sequential(nn.Linear(3072, 7))
        self._digit1 = nn.Sequential(nn.Linear(3072, 11))
        self._digit2 = nn.Sequential(nn.Linear(3072, 11))
        self._digit3 = nn.Sequential(nn.Linear(3072, 11))
        self._digit4 = nn.Sequential(nn.Linear(3072, 11))
        self._digit5 = nn.Sequential(nn.Linear(3072, 11))

    def forward(self, x):
        x = self._hidden1(x)
        x = self._hidden2(x)
        x = self._hidden3(x)
        x = x.view(x.size(0), 14400)
```

```
        x = self._hidden4(x)
        x = self._hidden5(x)

        length_logits = self._digit_length(x)
        digit1_logits = self._digit1(x)
        digit2_logits = self._digit2(x)
        digit3_logits = self._digit3(x)
        digit4_logits = self._digit4(x)
        digit5_logits = self._digit5(x)

        return length_logits, digit1_logits, digit2_logits, digit3_logits,
digit4_logits, digit5_logits
```

Обучим модель используя подготовленный набор данных, для обучения используется SGD с адаптивным шагом обучения

```
model = Model()

transform = transforms.Compose([
    transforms.RandomCrop([54, 54]),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
train_loader = torch.utils.data.DataLoader(Dataset(path_to_train_lmdb_dir, transform),
                                           batch_size=batch_size, shuffle=True,
                                           num_workers=4, pin_memory=True)
evaluator = Evaluator(path_to_val_lmdb_dir)
optimizer = optim.SGD(model.parameters(), lr=initial_learning_rate, momentum=0.9,
weight_decay=0.0005)
scheduler = StepLR(optimizer, step_size=training_options['decay_steps'],
gamma=training_options['decay_rate'])

for batch_idx, (images, length_labels, digits_labels) in enumerate(train_loader):
    length_logits, digit1_logits, digit2_logits, digit3_logits, digit4_logits,
digit5_logits = model.train()(images)
    loss = calc_loss(length_logits, digit1_logits, digit2_logits, digit3_logits,
digit4_logits, digit5_logits, length_labels, digits_labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    scheduler.step()
```

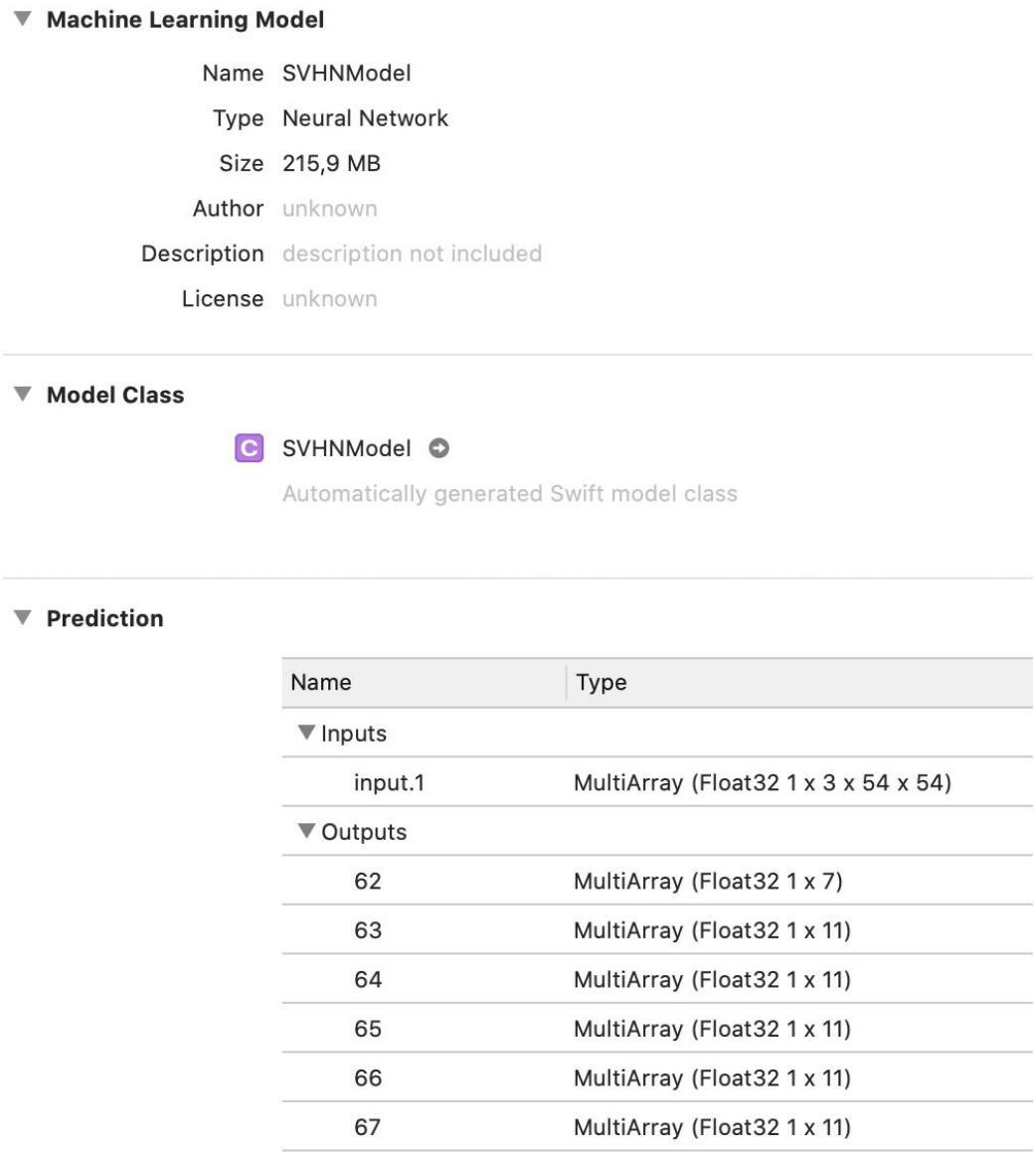
В результате обучения 5 эпох получили точность модели 77%.

Дообучим модель используя дополнительные данные из исходного набора. В результате обучения 5 эпох получили точность модели 84%.

Для интеграции сделаем iOS приложение, которое по фото будет давать предикшен номера на картинке. Для интеграции модели будем использовать CoreML.
Для начала нужно сконвертировать полученную модель в формат CoreML.
Код экспорта:

```
def export(path_to_checkpoint_file, path_to_input_image):
    # Step 0 - (b) Create model or Load from dist
    model = Model()
    model.restore(path_to_checkpoint_file)

    with torch.no_grad():

        transform = transforms.Compose([
            transforms.Resize([64, 64]),
            transforms.CenterCrop([54, 54]),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ])

        image = Image.open(path_to_input_image)
        image = image.convert('RGB')
        image = transform(image)
        images = image.unsqueeze(dim=0)

        outputs = model.eval()(images)

        # Step 1 - PyTorch to ONNX model
        torch.onnx.export(model, images, './model.onnx', example_outputs=outputs)

        # Step 2 - ONNX to CoreML model
        mlmodel = convert(model='./model.onnx', minimum_ios_deployment_target='13')
        # Save converted CoreML model
        mlmodel.save('SVNHModel.mlmodel')
```

Добавляем модель в проект iOS приложения.



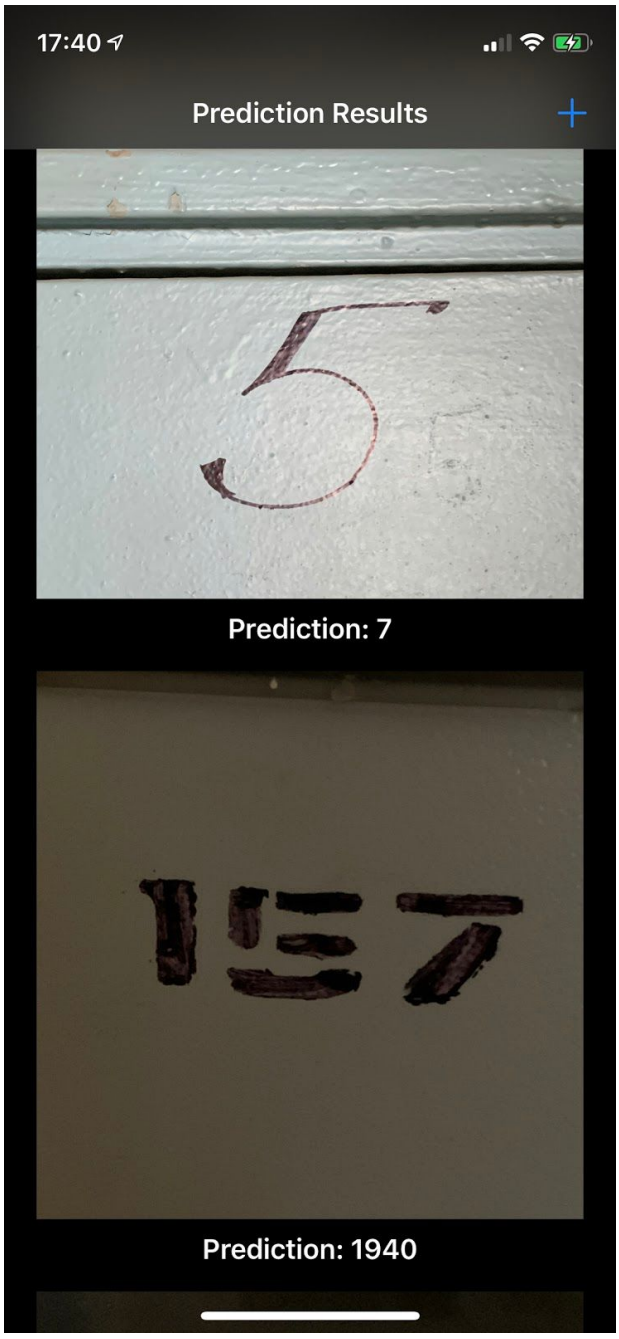
Далее нужно сделать предобработку изображений, чтобы они были в формате, соответствующему входу модели.

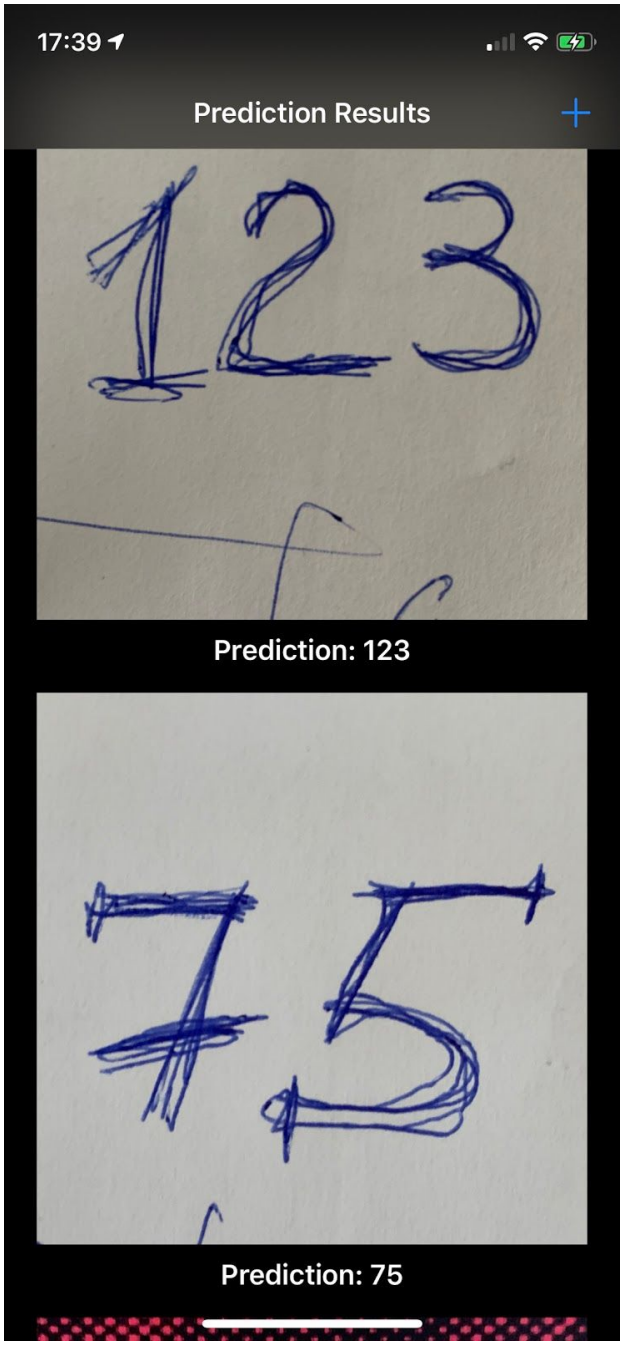
```
func predict(image: UIImage) -> Prediction? {
    guard let input = image
        .resizedImage(for: CGSize(width: 54, height: 54))?
        .pixelData()?
        .channelBasedOrder()
        .normalized() else {

            return nil
    }

    return predict(input: input)
}
```

Аналогично нужно преобразовать выход модели для получения читаемого результата.

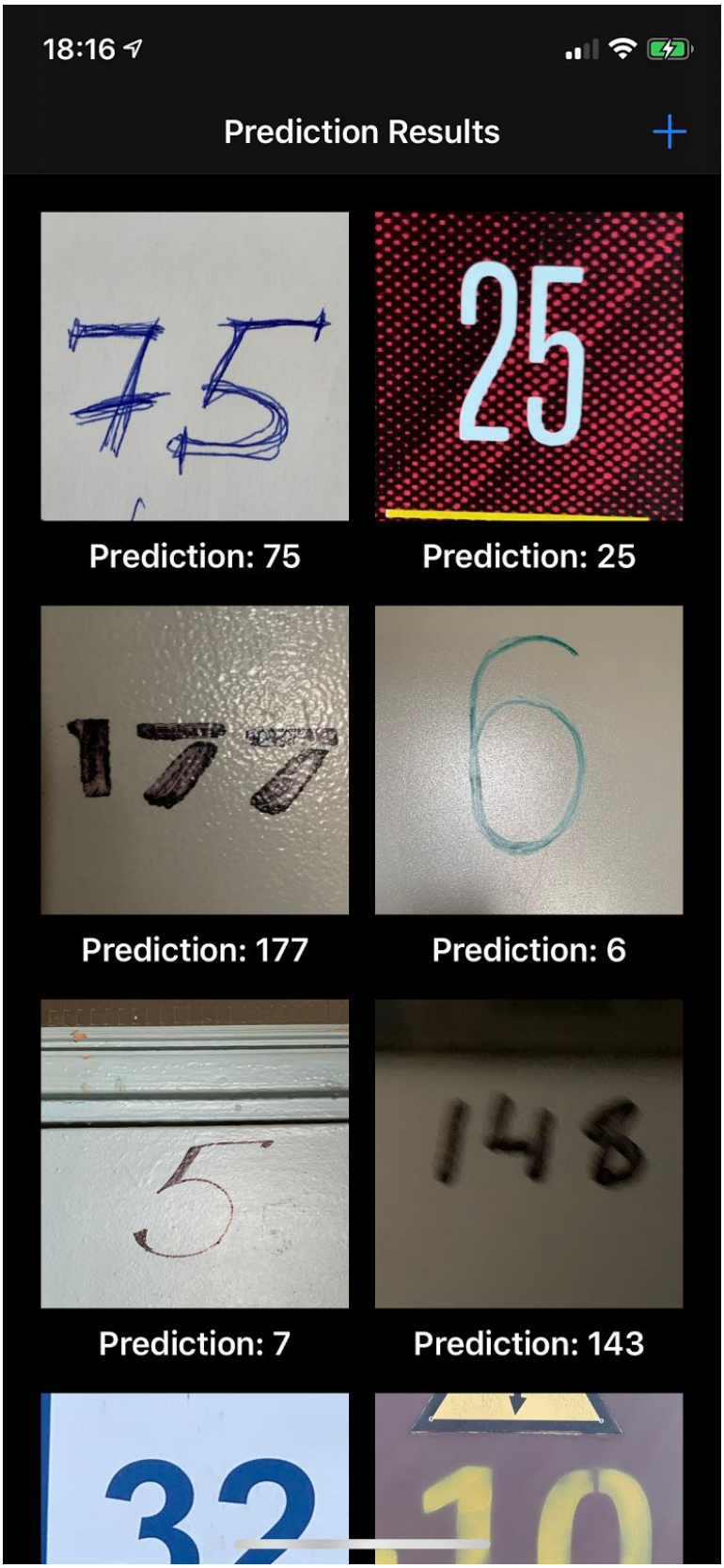
```
func predict(input: [Float32]) -> Prediction? {
    do {
        let input = SVHNModelInput(input_1: try MLMultiArray(input, shape: [1, 3, 54, 54]))
        let result = try prediction(input: input)
        if let length = result._62.array(of: Float32.self).maxIndex(),
            let digit1 = result._63.array(of: Float32.self).maxIndex(),
            let digit2 = result._64.array(of: Float32.self).maxIndex(),
            let digit3 = result._65.array(of: Float32.self).maxIndex(),
            let digit4 = result._66.array(of: Float32.self).maxIndex(),
            let digit5 = result._67.array(of: Float32.self).maxIndex() {

            let digits = [digit1, digit2, digit3, digit4, digit5]
            print("Prediction: Length = \(length)")
            print("Prediction: Result = \(digits[0..<length])")
            return Prediction(length: length, digits: digits)
        }
    } catch {
        print(error)
    }
    return nil
}
```

Далее реализуем функционал приложения по получению изображение с камеры и отображения результата.
Получили и удачные варианты работы и неудачные.


17:40
Prediction Results
Prediction: 7
Prediction: 1940


17:39
Prediction Results
Prediction: 123
Prediction: 75

Финальный вид приложения:



Видео работы приложения можно посмотреть по ссылкам:
https://github.com/Stunba/MachineLearning2/raw/master/lab4/RPReplay_Final1587127222.MP4
https://github.com/Stunba/MachineLearning2/raw/master/lab4/RPReplay_Final1587136554.MP4

Вывод:
В данной работе была построена и обучена модель, которая может распознавать последовательность цифр на изображении, реализовано мобильное приложение, которое использует построенную модель для распознавания номеров по снимку с камеры телефона.