МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (МАИ)**

**Институт № 7 «Робототехнические и интеллектуальные системы»**

**Кафедра 704 «Информационно-управляющие комплексы»**

**Отчёт по лабораторной работе**

по дисциплине: «Математическое моделирование процессов функционирования интегрированных систем летательных аппаратов»

**Нейронные сети**

Вариант 11

Работу выполнил студент группы № 7О-508с:

Шайхутдинов М.Р.

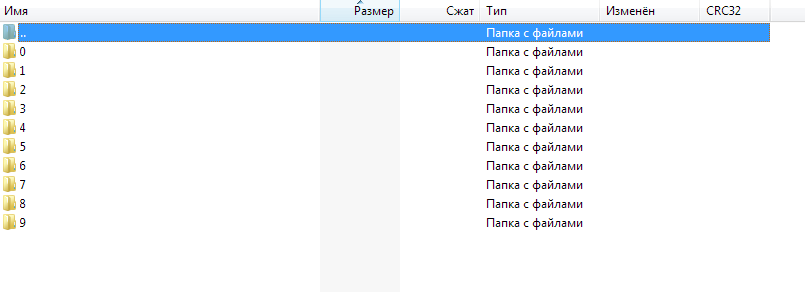
Работу принял аспирант кафедры 704:

Глушанкова.В.И.

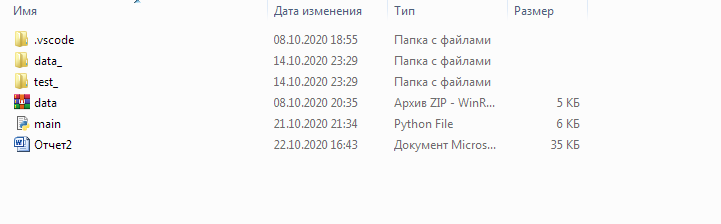
Москва, 2020

**1.Исходные данные**

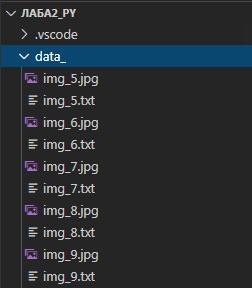
Был скачен архив с цифрами.



Для удобства для каждого файла генерировался анотационный файл с указанием принадлежности к тому или иному файлу.

Далее полученные файлы мы распределяли на обучающую выборку (data\_) и тестовую выборку (test\_).

Название класса содержится в файле с расширением (.txt) , он же – анотационный.



**2. Модель**

# Нейронная сеть с её архитектурой

class MLPNet(nn.Module):

    def \_\_init\_\_(self):

        super(MLPNet, self).\_\_init\_\_()

        self.fc1 = nn.Linear(100, 100)

        self.fc2 = nn.Linear(100, 50)

        self.fc4 = nn.Linear(50, 10)

    def forward(self, x):

        # x=torch.FloatTensor(x)

        x=x.view(-1,100)

        x = torch.sigmoid(self.fc1(x))

        x = torch.sigmoid(self.fc2(x))

        x = self.fc4(x)

        return x

    def name(self):

        return "MLP"

1. У модели 100 входов и 10 выходов.
2. Функция активация – сигмоида.
3. 3 слоя.

**3. Маркировка данных (выход)**

#определяем значение метки под каждый класс

label\_mark = {

    0: [[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

    1: [[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]],

    2: [[0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]],

    3: [[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]],

    4: [[0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]],

    5: [[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]],

    6: [[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]],

    7: [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]],

    8: [[0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]],

    9: [[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]],

}

**4.Метод нахождения частной производной и определения ошибки.**

#Подбираем метод оптимизации

optimizer = optim.Adamax(net.parameters())#Метод оптимизации

criterion = nn.MSELoss()# Mean Squared Error - критерий оптимальности

**5. Предобработка данных**

Исходные данные имели размерность 28х28, но так как по условию задачи сказано было брать 10х10, то соответственно сжимали картинку.

X\_train = []

Y\_train = []

images\_dir = 'data\_'

image\_dim = 10

images = []

jpg\_filepaths = glob.glob(os.path.join(images\_dir, '\*.jpg'))

anotaions = glob.glob(os.path.join(images\_dir, '\*.txt'))

X = []

#Считываем файл с данными

for filepath in jpg\_filepaths:

    image = Image.open(filepath).resize((image\_dim, image\_dim));

    # нормализация картинки

    images.append( (np.asarray(image)) );

# Считываем анотационный файл

for filepath in anotaions:

    myfile = open(filepath, "rb")

    for line in myfile:

        mini\_label=int(line)

        Y\_train.append(label\_mark[mini\_label])

images\_dir = 'test\_'

image\_dim = 10

test\_jpg\_filepaths = glob.glob(os.path.join(images\_dir, '\*.jpg'))

test\_anotaions = glob.glob(os.path.join(images\_dir, '\*.txt'))

X\_test=[]

Y\_test=[]

#Считываем файл с данными

for filepath in test\_jpg\_filepaths:

    image = Image.open(filepath).resize((image\_dim, image\_dim));

    # нормализация картинки

    X\_test.append( (np.asarray(image)) );

# Считываем анотационный файл

for filepath in test\_anotaions:

    myfile = open(filepath, "rb")

    for line in myfile:

        mini\_label=int(line)

        Y\_test.append(label\_mark[mini\_label])

X\_Train=images

Y\_Train=np.asarray(Y\_train)

Y\_test=np.asarray(Y\_test)

trainloader = zip(X\_Train, Y\_Train)

test\_loader = zip(X\_test,Y\_test)

**6. Обучение**

running\_loss = 0.0

net.train()

loss\_history = []

acc\_history = []

def train(epoch):

    net.train()

    for batch\_id, (data, label) in enumerate(trainloader, 0):

        data = Variable(torch.from\_numpy(data.astype(np.float32)))

        target = Variable(torch.from\_numpy(label.astype(np.float32)))

        optimizer.zero\_grad()

        preds = net(data)

        #определяем ошибку между Истинным значением и Результатов выдачи нейронной сети

        loss = criterion(preds, target)

        #обратное распространение ошибки

        loss.backward()

        loss\_history.append(loss.data)

        #увеличиваем шаг

        optimizer.step()

        if batch\_id % 100 == 0:

            print(loss.data)

**7. Тестирование**

#определяем класс победитель

def pred\_\_\_(l):

  m = max(l[0])

  return l[0].index(m)

def test(epoch):

    net.eval()

    test\_loss = 0

    correct = 0

    for data, target in test\_loader:

        data = Variable(torch.from\_numpy(data.astype(np.float32)), volatile=True)

        target = Variable(torch.from\_numpy(target.astype(np.float32)))

        output = net(data)

        test\_loss += criterion(output, target).data

        pred\_\_=pred\_\_\_(output.tolist())

        real=pred\_\_\_(target.tolist())

        if (pred\_\_==real):

            correct =correct+1

        #Дополним функционал

        print("Предсказанное число:",pred\_\_)

        print("Реальное число:",real)

    test\_loss = test\_loss

    test\_loss /= len(X\_test)

    accuracy = 100. \* correct / len(X\_test)

    acc\_history.append(accuracy)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

        test\_loss, correct, len(X\_test),

        accuracy))

**7. Основная программа.**

for epoch in range(0, 1):

    print("Epoch %d" % epoch)

    train(epoch)

    test(epoch)

**8. Результаты.**

**Обучающая выборка порядков – 600 данных, тестовая – 21 данных.**

**Epoch 0**

**tensor(0.1161)**

**tensor(0.0889)**

**tensor(0.0724)**

**tensor(0.0683)**

**tensor(0.0548)**

**tensor(0.0913)**

**Предсказанное число: 1**

**Реальное число: 1**

**Предсказанное число: 0**

**Реальное число: 0**

**Предсказанное число: 1**

**Реальное число: 1**

**Предсказанное число: 4**

**Реальное число: 4**

**Предсказанное число: 0**

**Реальное число: 0**

**Предсказанное число: 2**

**Реальное число: 7**

**Предсказанное число: 6**

**Реальное число: 6**

**Предсказанное число: 9**

**Реальное число: 5**

**Предсказанное число: 8**

**Реальное число: 8**

**Предсказанное число: 6**

**Реальное число: 6**

**Предсказанное число: 6**

**Реальное число: 6**

**Предсказанное число: 3**

**Реальное число: 8**

**Предсказанное число: 9**

**Реальное число: 8**

**Предсказанное число: 8**

**Реальное число: 8**

**Предсказанное число: 3**

**Реальное число: 8**

**Предсказанное число: 9**

**Реальное число: 5**

**Предсказанное число: 9**

**Реальное число: 8**

**Предсказанное число: 9**

**Реальное число: 7**

**Предсказанное число: 6**

**Реальное число: 6**

**Предсказанное число: 8**

**Реальное число: 8**

**Предсказанное число: 9**

**Реальное число: 8**

**Test set: Average loss: 0.0768, Accuracy: 12/21 (57%)**