Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе #6

«**Применение сверточных нейронных сетей (многоклассовая классификация)**

»

Карп Александр Игоревич

магистрант кафедры информатики

группа №858641

Минск 2019

**Исходный код: https://colab.research.google.com/drive/1v3OfbvS9ZVMBHw-wPwVnShHg3O3yfzAd**

**Задание 1.**

Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.

train = pd.read\_csv('sign\_mnist\_train.csv')

labels = train['label'].values

train.drop('label', axis=1, inplace=True)

images = train.values

images = np.array([np.reshape(i, (28, 28)) for i in images])

images = np.array([i.flatten() for i in images])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(images, labels, test\_size = 0.3, random\_state = 42)

**Задание 2.**

Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?

model = Sequential()

model.add(Conv2D(64, kernel\_size=(3,3), activation = 'relu', input\_shape=(28, 28 ,1) ))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Conv2D(64, kernel\_size = (3, 3), activation = 'relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Conv2D(64, kernel\_size = (3, 3), activation = 'relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Flatten())

model.add(Dense(128, activation = 'relu'))

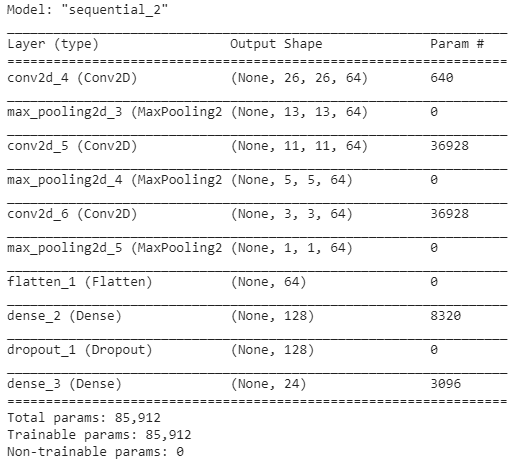
model.add(Dropout(0.20))

model.add(Dense(num\_classes, activation = 'softmax'))

model.compile(loss = categorical\_crossentropy, optimizer='adam',

              metrics=['accuracy'])

history = model.fit(datagen.flow(x\_train, y\_train, batch\_size=128), steps\_per\_epoch=len(x\_train) / 128, validation\_data = (x\_test, y\_test), epochs=epochs )



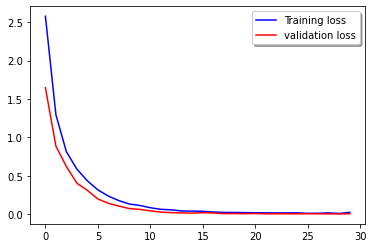


Рис 1. Значение функции потерь в зависимости от эпохи

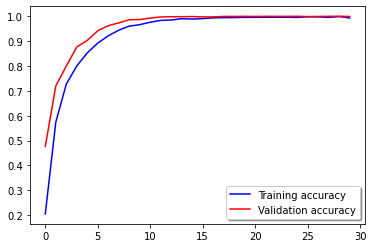


Рис 2. Зависимость точности от эпохи

Значения функции потерь стремятся к нулю с каждой эпохой, принимая минимальное значение равное 0.02. Точность как на тренировочном, так и на валидацинном сете практически единица.

Проверим точность на тестовом сете:

test = pd.read\_csv('sign\_mnist\_test.csv')

labels\_true = test['label'].values

test.drop('label', axis=1, inplace=True)

images = test.values

images = np.array([np.reshape(i, (28, 28)) for i in images])

images = np.array([i.flatten() for i in images])

images = images.reshape(images.shape[0], 28, 28, 1)/255

labels = label\_binarizer.fit\_transform(labels\_true)

score, acc = model.evaluate(images, labels)

acc

225/225 [==============================] - 1s 4ms/step - loss: 0.5570 - accuracy: 0.9024

0.9023982286453247

Точность составила 90%.

**Задание 3.**

Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

Добавим конфигурацию для класса ImageDataGenerator

datagen = ImageDataGenerator(

    rotation\_range=15,

    shear\_range=0.1,

    zoom\_range=0.2,

    horizontal\_flip=True,

width\_shift\_range=0.1,

height\_shift\_range=0.1)

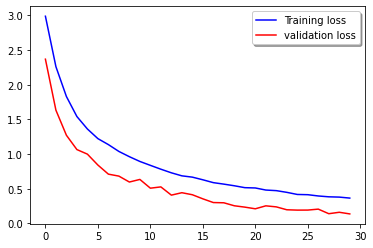


Рис 1. Значение функции потерь в зависимости от эпохи

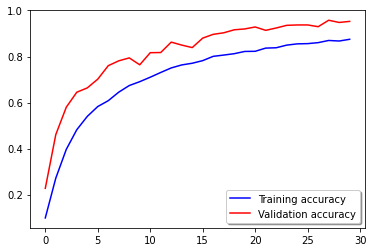


Рис 2. Зависимость точности от эпохи

На графиках видно стабильное уменьшение функции потерь и увеличение точности как на тренировночных, так и на валидационных данных

Полученная точность: 94%, аугментирование данных позволило увеличить точность с 90% до 94%.

**Задание 4.**

Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Можно ли было обойтись без него?

Какой максимальный результат удалось получить на контрольной выборке?

Воспользумся преобученной сетью VGG16

Здесь возникла проблема с ее конфигурацией – минимальная размерность входных данных для этой сети равна (32,32), также изображение должно быть трех-канальным

Воспользуемся функцией resize из библиотеки opencv и приобразуем изображения (28,28) в (32,32)

Для получения трехканального изображения просто копируем значение пикселя для трех измерений

Импортируем структуру VGG модели, отключим обучение на уже обученных слоях, также добавим два нижних полносвязных слоя, которые и будут обучаться для конкретной задачи

base\_model=VGG16(weights='imagenet',include\_top=False, input\_shape=(32,32,3)) #imports the mobilenet model and discards the last 1000 neuron layer.

x=base\_model.output

x=GlobalAveragePooling2D()(x)

x=Dense(128,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.

x = Dropout(0.2)(x)

x=Dense(64,activation='relu')(x) #dense layer 3

x = Dropout(0.2)(x)

preds=Dense(24,activation='softmax')(x) #final layer with softmax activationcallbacks=callbacks

model=Model(inputs=base\_model.input,outputs=preds)

for layer in model.layers[:20]:

    layer.trainable=False

for layer in model.layers[20:]:

    layer.trainable=True

model.compile(loss = categorical\_crossentropy, optimizer='adam',

              metrics=['accuracy'])

history = model.fit(datagen.flow(x\_train, y\_train, batch\_size=128), steps\_per\_epoch=len(x\_train) / 128, validation\_data = (x\_test, y\_test), epochs=24 )

Рассмотрим структуру модели:

Model: "model\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_3 (InputLayer) [(None, 32, 32, 3)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv1 (Conv2D) (None, 32, 32, 64) 1792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv2 (Conv2D) (None, 32, 32, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_pool (MaxPooling2D) (None, 16, 16, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv1 (Conv2D) (None, 16, 16, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv2 (Conv2D) (None, 16, 16, 128) 147584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_pool (MaxPooling2D) (None, 8, 8, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv1 (Conv2D) (None, 8, 8, 256) 295168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv2 (Conv2D) (None, 8, 8, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv3 (Conv2D) (None, 8, 8, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_pool (MaxPooling2D) (None, 4, 4, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv1 (Conv2D) (None, 4, 4, 512) 1180160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 4, 4, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 4, 4, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 2, 2, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 1, 1, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_5 (Flatten) (None, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_16 (Dense) (None, 128) 65664

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_17 (Dense) (None, 64) 8256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_18 (Dense) (None, 24) 1560

=================================================================

Total params: 14,790,168

Trainable params: 75,480

Non-trainable params: 14,714,688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Обучим модель:

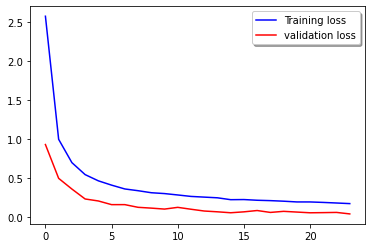


Рис 1. Значение функции потерь в зависимости от эпохи

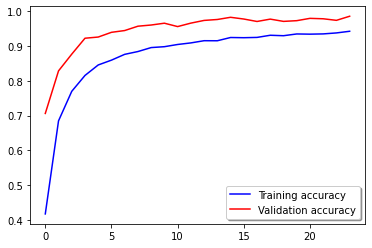


Рис 2. Зависимость точности от эпохи

Проверим точность обученной сети:

225/225 [==============================] - 4s 18ms/step - loss: 0.2009 - accuracy: 0.9304

0.9304238557815552