Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем
Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №6
«Применение сверточных нейронных сетей (многоклассовая классификация)»

Магистрант:
гр. 956241
Шуба И.А.

Проверил:
Заливако С. С.

Минск, 2020

ХОД РАБОТЫ

Задание.

Данные: Набор данных для распознавания языка жестов, который состоит из изображений размерности 28x28 в оттенках серого (значение пикселя от 0 до 255). Каждое из изображений обозначает букву латинского алфавита, обозначенную с помощью жеста, как показано на рисунке ниже (рисунок цветной, а изображения в наборе данных в оттенках серого). Обучающая выборка включает в себя 27,455 изображений, а контрольная выборка содержит 7172 изображения. Данные в виде csv-файлов можно скачать на сайте Kaggle -> https://www.kaggle.com/datamunge/sign-language-mnist

Задание 1.

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

Задание 2.

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

Задание 3.

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

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?

Результат выполнения:

Задание 1. Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.

Объявим несколько констант и загрузим наборы данных.

```
TRAIN_DIR = '../sign-language-mnist/sign_mnist_train.csv'
TEST_DIR = '../sign-language-mnist/sign_mnist_test.csv'

ROWS = 28
COLS = 28
CHANNELS = 1

BATCH_SIZE=128
```

```
train = pd.read_csv(TRAIN_DIR)
test = pd.read_csv(TEST_DIR)
```

Рисунок 1 – Загрузка данных

Пример датафрейма.

```
train.head()
```

Out[5]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pi
0	3	107	118	127	134	139	143	146	150	153	...	207	
1	6	155	157	156	156	156	157	156	158	158	...	69	
2	2	187	188	188	187	187	186	187	188	187	...	202	
3	2	211	211	212	212	211	210	211	210	210	...	235	
4	13	164	167	170	172	176	179	180	184	185	...	92	

5 rows × 785 columns

Рисунок 2 – Пример датафрейма

```
train.shape
```

Out[7]:

```
(27455, 785)
```

```
test.shape
```

Out[8]:

```
(7172, 785)
```

Рисунок 3 – Размеры наборов данных

```
train_labels = train['label'].values
test_labels = test['label'].values
```

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

Рисунок 4 – Получение лейблов и удаление колонки label

```
unique_val = np.array(train_labels)
unique_classes = np.unique(unique_val)
NUM_CLASSES = len(unique_classes)
print(f'Unique classes: {NUM_CLASSES}')
```

```
Unique classes: 24
```

```
unique_classes
```

```
Out[11]:

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])
```

Рисунок 5 – Количество уникальных классов

```
sn.countplot(train_labels)
```

```
Out[12]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fc9b69b6310>
```

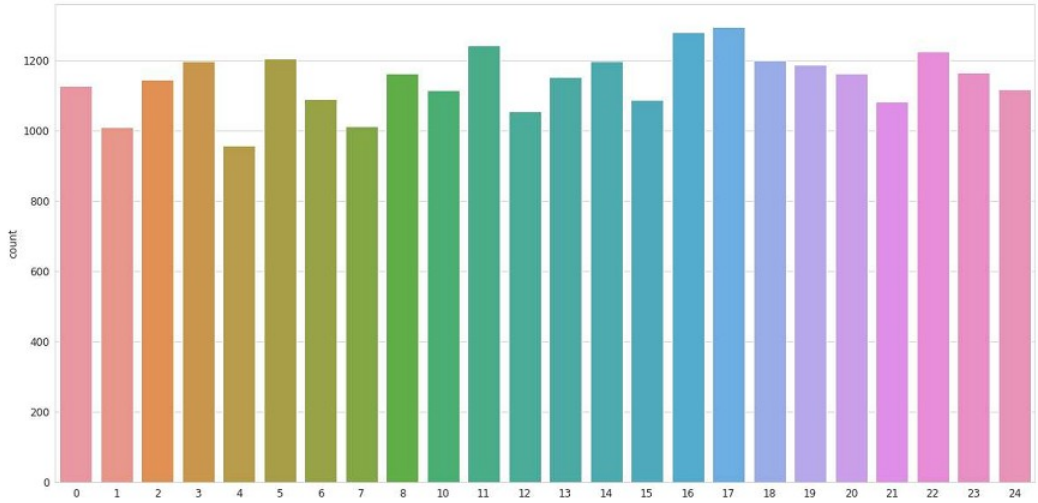


Рисунок 6 – Проверка сбалансированности классов

```
label_binrizer = LabelBinarizer()
labels_bin_train = label_binrizer.fit_transform(train_labels)
labels_bin_test = label_binrizer.fit_transform(test_labels)
labels_bin_train
```

Out[13]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0]])
```

```
images_train = train.values / 255
images_test = test.values / 255
```

Рисунок 7 – Бинаризация лейблов и нормализация изображений

```
plt.style.use('grayscale')
fig, axs = plt.subplots(1, 5, figsize=(15, 4), sharey=True)
for i in range(5):
        axs[i].imshow(images_train[i].reshape(28,28))
fig.suptitle('Grayscale images')
```

Out[19]:

```
Text(0.5, 0.98, 'Grayscale images')
```

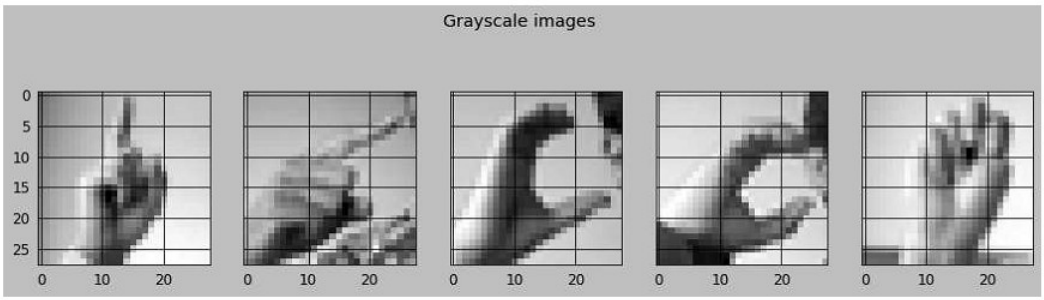


Рисунок 8 – Примеры изображений

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(images_train, labels_bin_train, test_
size = 0.3,
                                                  stratify = labels_bin_train, random_s
tate = 43)
```

```
x_train = x_train.reshape(x_train.shape[0], ROWS, COLS ,CHANNELS)
x_val = x_val.reshape(x_val.shape[0], ROWS, COLS ,CHANNELS)
x_test = images_test.reshape(images_test.shape[0], ROWS, COLS ,CHANNELS)
```

```
print('Train: ', x_train.shape, y_train.shape)
print('Validation: ', x_val.shape, y_val.shape)
print('Test: ', x_test.shape, y_test.shape)
```

```
Train:  (19218, 28, 28, 1) (19218, 24)
Validation:  (8237, 28, 28, 1) (8237, 24)
Test:  (7172, 28, 28, 1) (7172, 24)
```

Рисунок 9 – Разделение тренировочных данных и изменение формы вектора фич

Задание 2. Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?

Реализуем глубокую сверточную сеть.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(4,4), activation = 'relu', input_shape=(ROWS, COLS ,C
HANNELS), padding='same'))
model.add(Dropout(0.4))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (4, 4), activation = 'relu', padding='same' ))
model.add(Dropout(0.4))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(Dropout(0.4))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dense(NUM_CLASSES, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])
keras2ascii(model)
```

Рисунок 10 – Глубокая сверточная нейронная сеть

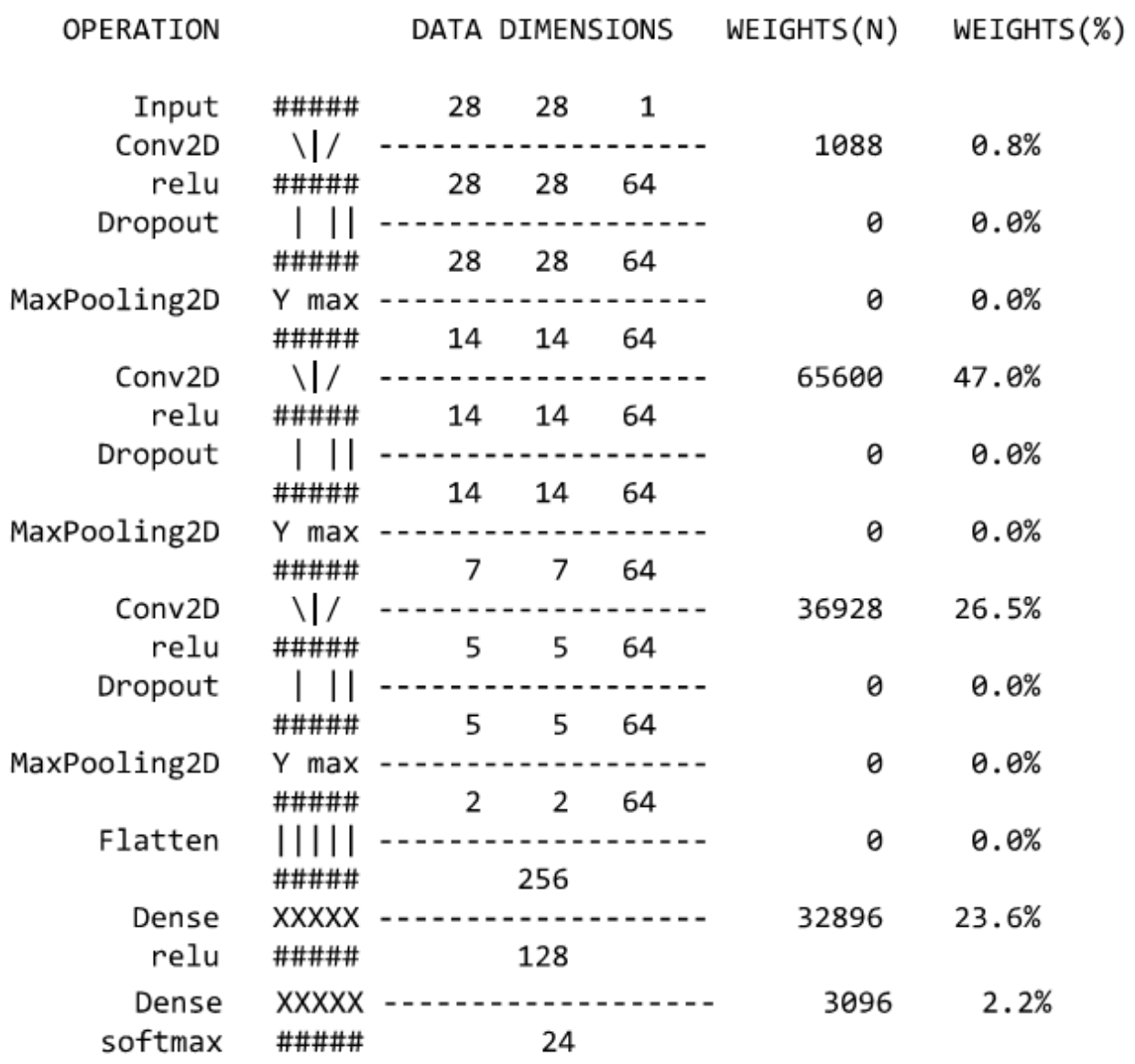
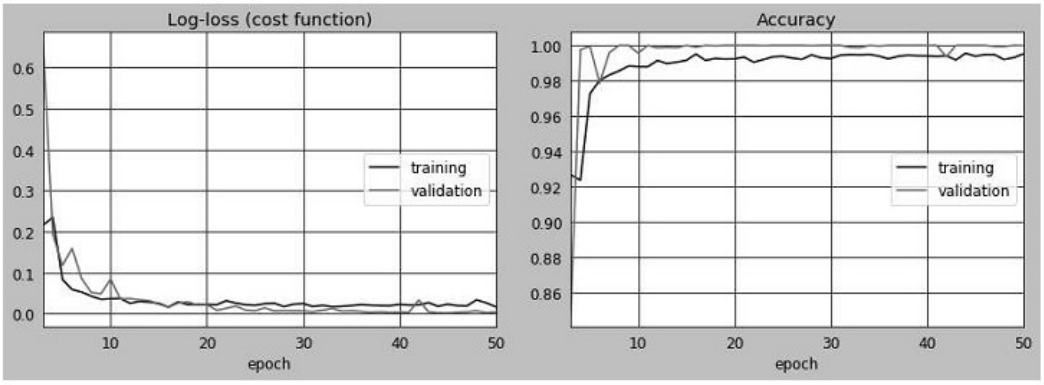


Рисунок 11 – Схема глубокой сверточной нейронной сети

```
history = model.fit(x_train, y_train,
                    epochs=50,
                    batch_size=BATCH_SIZE,
                    validation_data=(x_val, y_val),
                    callbacks=[PlotLossesKeras()],
                    verbose=2)
```




```
Log-loss (cost function):
training   (min:    0.016, max:    2.090, cur:    0.017)
validation (min:    0.002, max:    1.423, cur:    0.004)

Accuracy:
training   (min:    0.354, max:    0.996, cur:    0.995)
validation (min:    0.646, max:    1.000, cur:    1.000)
```

Рисунок 12 – Результаты обучения

Как видно из графиков, точность на валидационных данных приблизительно такая же, как на тренировочных. Максимальная точность на валидационных данных составила 1, а минимальный log-loss – 0,002.

```
y_pred = model.predict_classes(x_test)
evaluation1 = model.evaluate(x_test, y_test)
acc1 = accuracy_score(test_labels, y_pred)
f1_1 = f1_score(test_labels, y_pred, average='weighted')
print(evaluation1)
```

```
7172/7172 [==============================] - 2s 290us/step
[0.09921373493744272, 0.9714166201896264]
```

Рисунок 13 – Результаты модели на контрольной выборке

Таким образом, точность модели составила 0,97, а log-loss 0.099.

Задание 3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

Применим дополнение данных путем генерации изображений с некоторыми преобразованиями.

```
train_datagen = ImageDataGenerator(rotation_range=10,
                                   width_shift_range=0.05,
                                   height_shift_range=0.05,
                                   shear_range=0.1,
                                   zoom_range=0.075)

validation_datagen = image.ImageDataGenerator()
```

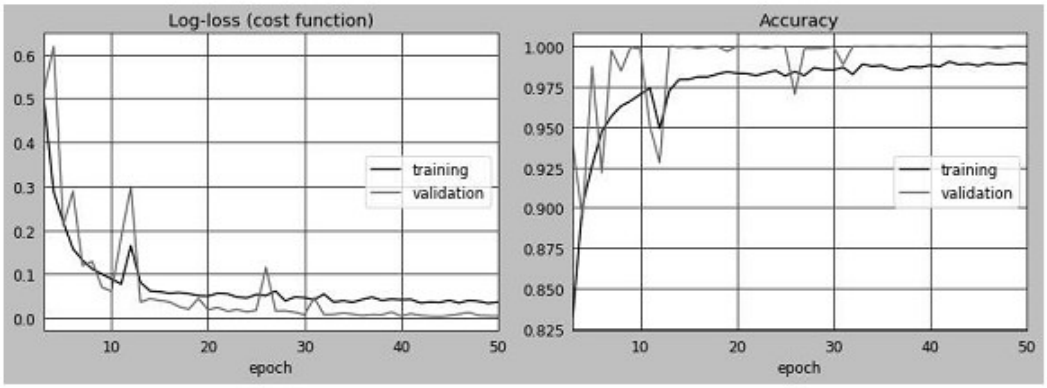
```
train_generator = train_datagen.flow(
    x_train,
    y_train,
    batch_size=BATCH_SIZE)

validation_generator = validation_datagen.flow(
    x_val,
    y_val,
    batch_size=BATCH_SIZE)
```

Рисунок 14 – Генерация изображений с некоторыми преобразованиями

```
train_steps = len(x_train)/BATCH_SIZE
validation_steps = len(x_val)/BATCH_SIZE

history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_steps,
    epochs=50,
    validation_data=validation_generator,
    validation_steps=validation_steps,
    callbacks=[PlotLossesKeras()],
    verbose=2)
```




```
Log-loss (cost function):
training   (min:    0.034, max:    2.168, cur:    0.036)
validation (min:    0.004, max:    1.952, cur:    0.005)

Accuracy:
training   (min:    0.331, max:    0.991, cur:    0.989)
validation (min:    0.308, max:    1.000, cur:    1.000)
```

Рисунок 15 – Результаты обучения

Как видно из графиков, точность и log-loss на валидационных данных приблизительно такая же как и на, чем на тренировочных. Максимальная точность на валидационных данных составила 1, а минимальный log-loss – 0,004.

```
y_pred = model.predict_classes(x_test)
evaluation = model.evaluate(x_test, y_test)
acc1 = accuracy_score(test_labels, y_pred)
f1_1 = f1_score(test_labels, y_pred, average='weighted')
print(evaluation1)
```

```
7172/7172 [==============================] - 4s 522us/step
[0.019458772333980863, 0.9963747908533185]
```

Рисунок 16 – Результаты модели на контрольной выборке

Таким образом, точность модели составила 0,99, а log-loss 0,019.

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?

Чтобы применить передаточное обучение, необходимо подстроить форму данных, подаваемых на вход предобученной сети.

```
x_train_t = np.stack([x_train.reshape(x_train.shape[0],28,28)]*3, axis=3).reshape(x_tra
in.shape[0],28,28,3)
x_val_t = np.stack([x_val.reshape(x_val.shape[0],28,28)]*3, axis=3).reshape(x_val.shape
[0],28,28,3)
x_test_t = np.stack([x_test.reshape(x_test.shape[0],28,28)]*3, axis=3).reshape(x_test.s
hape[0],28,28,3)
x_train_t.shape, x_val_t.shape, x_test_t.shape
```

```
Out[31]:

((19218, 28, 28, 3), (8237, 28, 28, 3), (7172, 28, 28, 3))
```

```
from keras.preprocessing.image import img_to_array, array_to_img
x_train_tt = np.asarray([img_to_array(array_to_img(im, scale=True).resize((48,48))) for
im in x_train_t])/225
x_val_tt = np.asarray([img_to_array(array_to_img(im, scale=True).resize((48,48))) for i
m in x_val_t])/225
x_test_tt = np.asarray([img_to_array(array_to_img(im, scale=True).resize((48,48))) for
im in x_test_t])/225
x_train_tt.shape, x_val_tt.shape, x_test_tt.shape
```

```
Out[32]:

((19218, 48, 48, 3), (8237, 48, 48, 3), (7172, 48, 48, 3))
```

Рисунок 17 –Настройка входных данных под вход предобученной сети VGG16

Поэксперементируем с сетью VGG16.

```
base_model = VGG16(include_top = False, weights = 'imagenet', pooling = 'avg', input_sh
ape=(48, 48, 3))
```

```
model = Sequential()
model.add(base_model)

model.add(Dense(NUM_CLASSES, activation = 'softmax'))

model.layers[0].trainable = False
model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])
model.summary()
keras2ascii(model)
```

Рисунок 18 –Нейронная сеть с предобученной сетью VGG16

```
train_generator = train_datagen.flow(
    x_train_tt,
    y_train,
    batch_size=BATCH_SIZE)

validation_generator = validation_datagen.flow(
    x_val_tt,
    y_val,
    batch_size=BATCH_SIZE)
```

Рисунок 19 – Генерация изображений с некоторыми преобразованиями

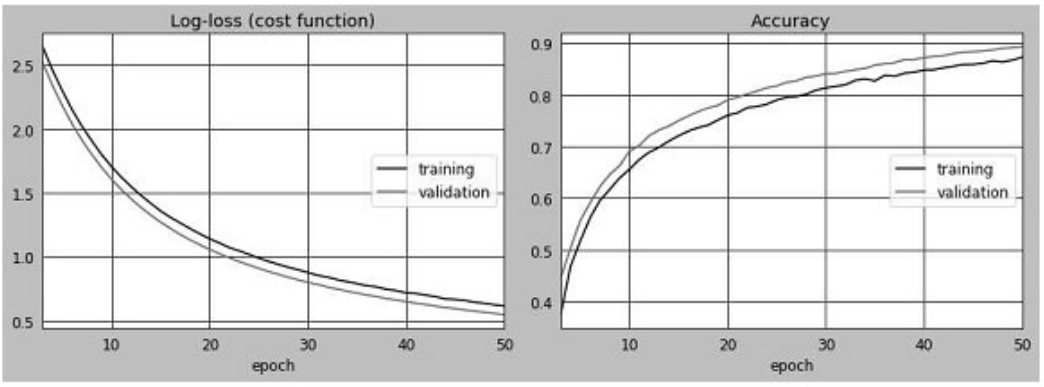
```
OPERATION            DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

    Input   #####     48   48    3
    Model   ?????  -------------------  14714688     99.9%
            #####          512
    Dense   XXXXX  -------------------     12312      0.1%
  softmax   #####          24
```

Рисунок 20 – Схема нейронной сети с предобученной сетью VGG16

```
train_steps = len(x_train)/BATCH_SIZE
validation_steps = len(x_val)/BATCH_SIZE

history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_steps,
    epochs=50,
    validation_data=validation_generator,
    validation_steps=validation_steps,
    callbacks=[PlotLossesKeras()],
    verbose=2)
```




```
Log-loss (cost function):
training   (min:    0.614, max:    3.151, cur:    0.614)
validation (min:    0.546, max:    2.964, cur:    0.546)

Accuracy:
training   (min:    0.083, max:    0.873, cur:    0.873)
validation (min:    0.129, max:    0.893, cur:    0.893)
```

Рисунок 21 – Результаты обучения

Как видно из графиков, точность и log-loss на валидационных данных приблизительно такая же, как на тренировочных. Максимальная точность на валидационных данных составила 0,893, а минимальный log-loss – 0,546.

```
y_pred = model.predict_classes(x_test_tt)
evaluation1 = model.evaluate(x_test_tt, y_test)
acc1 = accuracy_score(test_labels, y_pred)
f1_1 = f1_score(test_labels, y_pred, average='weighted')
print(evaluation1)
```

```
7172/7172 [==============================] - 111s 16ms/step
[0.7324652449750927, 0.8073061907417736]
```

Рисунок 22 – Результаты модели на контрольной выборке

Таким образом, точность модели составила 0,8, а log-loss 0.73 при обучении на 50 эпохах. Из графиков точности и log-loss заметим, что при увеличении количества эпох скор будет увеличиваться. При обучении с 50 эпохами, общая точность модели осталась достаточно низкой, однако виден положительный тренд – при увеличении количества эпох при обучении, точность будет повышаться, однако достаточно медленно, нежели с

применением реализованных выше архитектур. Качество классификатора заметно ниже, нежели обученной в прошлом задании, поэтому в данной задаче можно было бы обойтись без предобученной модели.

Вывод:

В ходе выполнения лабораторной работы был изучен датасет с жестами, были реализованы сверточные сети с разными архитектурами, максимальный скор, который удалось получить, составил 0,99, а минимальный log-loss – 0,019. Причем в модели не наблюдалось явного переобучения, так как графики точности на тренировочных и валидационных данных практически повторяли друг друга с ростом числа эпох. Было реализовано передаточное обучение. В качестве предобученной модели была использована одна из популярных сетей VGG16. Однако точность реализованной была хуже, чем созданной ранее, так точность составила 0,8.