Лабораторная работа №6. Применение сверточных нейронных сетей (многоклассовая классификация)

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.

In [13]:

```
import numpy as np
import pandas as pd
from tensorflow import keras

data_folder = '../data'
h, w, c = 28, 28, 1
batch_size = 256
epochs = 3
classes = 25
```

In [14]:

```
def load_data(name):
    df = pd.read_csv(data_folder + '/sign-language-mnist/' + name)
    y = df['label'].values.reshape(len(df['label'].values), 1)
    df.drop('label', axis = 1, inplace = True)
    X = np.array([np.reshape(i, (h, w, c)) for i in df.values])
    return X, y
```

In [15]:

```
X_train, y_train = load_data('sign_mnist_train.csv')
X_dev, y_dev = load_data('sign_mnist_test.csv')
```

In [16]:

```
data_generator = keras.preprocessing.image.ImageDataGenerator()

train_generator = data_generator.flow(X_train, keras.utils.to_categorical(y_train),
    batch_size=batch_size)

dev_generator = data_generator.flow(X_dev, keras.utils.to_categorical(y_dev),
    batch_size=batch_size)
```

Задание 2.

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

In [17]:

```
def model_factory():
    model = keras.models.Sequential([
        keras.layers.Conv2D(6, (5, 5), activation='relu', input_shape=(h, w, c)),
        keras.layers.MaxPooling2D(),

        keras.layers.Conv2D(16, (5, 5), activation='relu'),
        keras.layers.MaxPooling2D(),

        keras.layers.Flatten(),
```

```
        keras.layers.Dense(120, activation='relu'),

        keras.layers.Dense(84, activation='relu'),

        keras.layers.Dense(classes, activation='softmax')
    ])

    return model
```

In [18]:

```
def train(model):
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])

    model.fit(train_generator, validation_data=dev_generator, epochs=epochs)
```

In [19]:

```
model = model_factory()
train(model)
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 108 steps, validate for 29 steps
Epoch 1/3
108/108 [==============================] - 10s 95ms/step - loss: 3.3918 - accuracy: 0.3868 - val_l
oss: 1.3333 - val_accuracy: 0.5877
Epoch 2/3
108/108 [==============================] - 11s 103ms/step - loss: 0.2356 - accuracy: 0.9464 - val_
loss: 0.9759 - val_accuracy: 0.7267
Epoch 3/3
108/108 [==============================] - 9s 79ms/step - loss: 0.0322 - accuracy: 0.9981 - val_lo
ss: 0.9999 - val_accuracy: 0.7369
```

Задание 3.

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

In [20]:

```
data_generator_with_augmentation = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=1,
    shear_range=0.1,
    vertical_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

train_generator = data_generator_with_augmentation.flow(
    X_train, keras.utils.to_categorical(y_train), batch_size=batch_size)
```

In [21]:

```
model = model_factory()
train(model)
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
```

```
  ['...']
Train for 108 steps, validate for 29 steps
Epoch 1/3
108/108 [==============================] - 8s 76ms/step - loss: 4.3834 - accuracy: 0.1031 - val_lo
ss: 2.6543 - val_accuracy: 0.2069
Epoch 2/3
108/108 [==============================] - 8s 74ms/step - loss: 2.2234 - accuracy: 0.3276 - val_lo
ss: 1.6596 - val_accuracy: 0.4689
Epoch 3/3
108/108 [==============================] - 7s 67ms/step - loss: 1.6079 - accuracy: 0.4995 - val_lo
ss: 1.2455 - val_accuracy: 0.5993
```

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?

In [22]:

```
def pre_trained_model_factory():
    pre_trained_model = keras.applications.VGG16(input_shape=(h, w, c),
                                                 include_top=False,
                                                 weights='imagenet')
    pre_trained_model.trainable = False

    model = keras.models.Sequential([
        pre_trained_model,
        keras.layers.GlobalAveragePooling2D(),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.Dense(classes, activation='softmax')
    ])

    return model
```

In [23]:

```
h, w, c = 32, 32, 3
```

In [24]:

```
X_train = np.pad(np.concatenate([X_train, X_train, X_train], axis=-1),
                 ((0, 0), (2, 2), (2, 2), (0, 0)))
X_dev = np.pad(np.concatenate([X_dev, X_dev, X_dev], axis=-1),
               ((0, 0), (2, 2), (2, 2), (0, 0)))

train_generator = data_generator_with_augmentation.flow(
    X_train, keras.utils.to_categorical(y_train), batch_size=batch_size)

dev_generator = data_generator.flow(X_dev, keras.utils.to_categorical(y_dev),
    batch_size=batch_size)
```

In [25]:

```
model = pre_trained_model_factory()
train(model)
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 108 steps, validate for 29 steps
Epoch 1/3
108/108 [==============================] - 61s 567ms/step - loss: 2.3607 - accuracy: 0.5264 - val_
loss: 1.7137 - val_accuracy: 0.5142
Epoch 2/3
```

```
108/108 [==============================] - 63s 584ms/step - loss: 0.8854 - accuracy: 0.7172 - val_
loss: 1.5605 - val_accuracy: 0.5661
Epoch 3/3
108/108 [==============================] - 55s 508ms/step - loss: 0.6679 - accuracy: 0.7794 - val_
loss: 1.3707 - val_accuracy: 0.5998
```