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

1. Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.
2. Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?
3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?
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?

```
# Install Kaggle library
!pip install -q kaggle

import os
os.environ['KAGGLE_USERNAME'] = "awful1996" # username from the json file
os.environ['KAGGLE_KEY'] = "5e55f76a1cc7a4772bd7803cba8fb2c1" # key from the json file
#!kaggle datasets download -d iarunava/happy-house-dataset # api copied from kaggle
!kaggle datasets download -d datamunge/sign-language-mnist
```

```
Downloading sign-language-mnist.zip to /content
 66% 41.0M/62.6M [00:00<00:00, 26.1MB/s]
100% 62.6M/62.6M [00:00<00:00, 67.8MB/s]
```

```
!ls
```

```
american_sign_language.PNG  sample_data              sign_mnist_test.csv
amer_sign2.png              sign-language-mnist.zip  sign_mnist_train
amer_sign3.png              sign_mnist_test          sign_mnist_train.csv
```

```
!unzip sign-language-mnist.zip
```

```
Archive:  sign-language-mnist.zip
  inflating: amer_sign2.png
  inflating: amer_sign3.png
  inflating: american_sign_language.PNG
  inflating: sign_mnist_test.csv
  inflating: sign_mnist_test/sign_mnist_test.csv
  inflating: sign_mnist_train.csv
  inflating: sign_mnist_train/sign_mnist_train.csv
```

```
!ls
```

```
 american_sign_language.PNG  sample_data              sign_mnist_test.csv
 amer_sign2.png              sign-language-mnist.zip  sign_mnist_train
 amer_sign3.png              sign_mnist_test          sign_mnist_train.csv
```

```
import pandas as pd
df = pd.read_csv('sign_mnist_train.csv')
print(df)

testDF = pd.read_csv('sign_mnist_test.csv')
print(testDF)
```

```
       label  pixel1  pixel2  pixel3  ...  pixel781  pixel782  pixel783  pixel784
0          3     107     118     127  ...       206       204       203       202
1          6     155     157     156  ...       175       103       135       149
2          2     187     188     188  ...       198       195       194       195
3          2     211     211     212  ...       225       222       229       163
4         13     164     167     170  ...       157       163       164       179
...      ...     ...     ...     ...  ...       ...       ...       ...       ...
27450     13     189     189     190  ...       234       200       222       225
27451     23     151     154     157  ...       195       195       195       194
27452     18     174     174     174  ...       203       202       200       200
27453     17     177     181     184  ...        47        64        87        93
27454     23     179     180     180  ...       197       205       209       215

[27455 rows x 785 columns]
      label  pixel1  pixel2  pixel3  ...  pixel781  pixel782  pixel783  pixel784
0         6     149     149     150  ...       106       112       120       107
1         5     126     128     131  ...       184       184       182       180
2        10      85      88      92  ...       226       225       224       222
3         0     203     205     207  ...       230       240       253       255
4         3     188     191     193  ...        49        46        46        53
...     ...     ...     ...     ...  ...       ...       ...       ...       ...
7167      1     135     119     108  ...       184       176       167       163
7168     12     157     159     161  ...       210       210       209       208
7169      2     190     191     190  ...       210       211       209       208
7170      4     201     205     208  ...        91        67        70        63
7171      2     173     174     173  ...       195       195       193       192

[7172 rows x 785 columns]
```

```
def getData(df):
  y = df["label"].values.reshape((df.shape[0], 1))
  x = df.drop(columns="label").values.reshape((df.shape[0], 28, 28, 1))
  print(y.shape, x.shape)
  return x, y

x, y = getData(df)
tX, tY = getData(testDF)
```

```
(27455, 1) (27455, 28, 28, 1)
(7172, 1) (7172, 28, 28, 1)
```

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=.25)
```

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

Была получена точность 93 процента

```
from tensorflow.keras import layers, models, Sequential

model = Sequential([
    layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
    layers.Conv2D(32, kernel_size=5, activation='relu'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(500, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(25, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
import tensorflow as tf
```

```
model.fit(x_train, y_train, validation_split=0.1, epochs=100, callbacks=[
```

```
model.fit(x_train, y_train, validation_split=0.1, epochs=100, callbacks=[
      tf.keras.callbacks.EarlyStopping(
        patience=2,
        restore_best_weights=True,
        monitor='accuracy'
      )
    ])
```

```
Epoch 1/100
580/580 [==============================] - 5s 9ms/step - loss: 3.0719 - accuracy: 0.1263 - val_loss: 1.9795 - val_accu
Epoch 2/100
580/580 [==============================] - 5s 8ms/step - loss: 1.1424 - accuracy: 0.6333 - val_loss: 0.3806 - val_accu
Epoch 3/100
580/580 [==============================] - 5s 8ms/step - loss: 0.3862 - accuracy: 0.8723 - val_loss: 0.1172 - val_accu
Epoch 4/100
580/580 [==============================] - 5s 8ms/step - loss: 0.2489 - accuracy: 0.9207 - val_loss: 0.0915 - val_accu
Epoch 5/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1934 - accuracy: 0.9377 - val_loss: 0.0396 - val_accu
Epoch 6/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1519 - accuracy: 0.9513 - val_loss: 0.0398 - val_accu
Epoch 7/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1315 - accuracy: 0.9608 - val_loss: 0.0433 - val_accu
Epoch 8/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1338 - accuracy: 0.9570 - val_loss: 0.0126 - val_accu
Epoch 9/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1150 - accuracy: 0.9637 - val_loss: 0.0115 - val_accu
Epoch 10/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1006 - accuracy: 0.9700 - val_loss: 0.0176 - val_accu
Epoch 11/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1325 - accuracy: 0.9607 - val_loss: 0.0493 - val_accu
Epoch 12/100
580/580 [==============================] - 5s 8ms/step - loss: 0.0789 - accuracy: 0.9765 - val_loss: 0.0059 - val_accu
Epoch 13/100
580/580 [==============================] - 5s 8ms/step - loss: 0.0865 - accuracy: 0.9752 - val_loss: 0.0184 - val_accu
Epoch 14/100
580/580 [==============================] - 5s 8ms/step - loss: 0.0679 - accuracy: 0.9799 - val_loss: 0.0073 - val_accu
Epoch 15/100
580/580 [==============================] - 5s 8ms/step - loss: 0.0779 - accuracy: 0.9773 - val_loss: 0.0155 - val_accu
Epoch 16/100
580/580 [==============================] - 5s 8ms/step - loss: 0.1046 - accuracy: 0.9718 - val_loss: 0.0025 - val_accu
<tensorflow.python.keras.callbacks.History at 0x7fb1a82672b0>
```

```
model.evaluate(tX,  tY)
```

```
225/225 [==============================] - 1s 4ms/step - loss: 0.2069 - accuracy: 0.9334
[0.20685824751853943, 0.9333519339561462]
```

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

После применение качество упало до 5 процентов

```
model = Sequential([
    layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
    layers.Conv2D(32, kernel_size=5, activation='relu'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Dropout(0.25),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(500, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(25, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(rotation_range=20, zoom_range=0.15,
  width_shift_range=0.2, height_shift_range=0.2, shear_range=0.15,
  horizontal_flip=True, fill_mode="nearest")

train_generator = datagen.flow(x_train, y_train, batch_size=32)
```

```
val_generator = datagen.flow(x_val, y_val, batch_size=32)
test_generator = datagen.flow(tX, tY, batch_size=32)


EPOCHS = 100
BS = 32

model.fit_generator(
  train_generator,
  epochs=100,
  validation_data=(x_val,y_val), steps_per_epoch=len(x_train) // 32)
```

```
Epoch 1/100
643/643 [==============================] - 13s 20ms/step - loss: 3.2701 - accuracy: 0.0173 - val_loss: 2.9611 - val_ac
Epoch 2/100
643/643 [==============================] - 12s 19ms/step - loss: 2.8287 - accuracy: 0.0168 - val_loss: 2.4577 - val_ac
Epoch 3/100
643/643 [==============================] - 12s 19ms/step - loss: 2.4743 - accuracy: 0.0523 - val_loss: 1.8554 - val_ac
Epoch 4/100
643/643 [==============================] - 12s 19ms/step - loss: 2.0677 - accuracy: 0.0498 - val_loss: 1.4050 - val_ac
Epoch 5/100
643/643 [==============================] - 12s 19ms/step - loss: 1.6249 - accuracy: 0.0453 - val_loss: 0.8157 - val_ac
Epoch 6/100
643/643 [==============================] - 12s 19ms/step - loss: 1.3345 - accuracy: 0.0450 - val_loss: 0.5770 - val_ac
Epoch 7/100
643/643 [==============================] - 12s 19ms/step - loss: 1.1157 - accuracy: 0.0407 - val_loss: 0.4387 - val_ac
Epoch 8/100
643/643 [==============================] - 12s 19ms/step - loss: 0.9863 - accuracy: 0.0430 - val_loss: 0.3492 - val_ac
Epoch 9/100
643/643 [==============================] - 13s 20ms/step - loss: 0.8957 - accuracy: 0.0443 - val_loss: 0.3064 - val_ac
Epoch 10/100
643/643 [==============================] - 12s 19ms/step - loss: 0.8245 - accuracy: 0.0413 - val_loss: 0.2737 - val_ac
Epoch 11/100
643/643 [==============================] - 12s 19ms/step - loss: 0.7717 - accuracy: 0.0398 - val_loss: 0.1802 - val_ac
Epoch 12/100
643/643 [==============================] - 12s 19ms/step - loss: 0.7431 - accuracy: 0.0396 - val_loss: 0.2054 - val_ac
Epoch 13/100
643/643 [==============================] - 13s 20ms/step - loss: 0.6836 - accuracy: 0.0444 - val_loss: 0.2196 - val_ac
Epoch 14/100
643/643 [==============================] - 13s 20ms/step - loss: 0.6731 - accuracy: 0.0407 - val_loss: 0.1774 - val_ac
Epoch 15/100
643/643 [==============================] - 12s 19ms/step - loss: 0.6326 - accuracy: 0.0411 - val_loss: 0.1585 - val_ac
Epoch 16/100
643/643 [==============================] - 12s 19ms/step - loss: 0.6043 - accuracy: 0.0422 - val_loss: 0.1750 - val_ac
Epoch 17/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5662 - accuracy: 0.0411 - val_loss: 0.1830 - val_ac
Epoch 18/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5747 - accuracy: 0.0390 - val_loss: 0.0813 - val_ac
Epoch 19/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5519 - accuracy: 0.0440 - val_loss: 0.0826 - val_ac
Epoch 20/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5429 - accuracy: 0.0421 - val_loss: 0.0864 - val_ac
Epoch 21/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5425 - accuracy: 0.0417 - val_loss: 0.0843 - val_ac
```

```
Epoch 22/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5047 - accuracy: 0.0417 - val_loss: 0.0566 - val_ac
Epoch 23/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5330 - accuracy: 0.0411 - val_loss: 0.0739 - val_ac
Epoch 24/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5250 - accuracy: 0.0412 - val_loss: 0.0601 - val_ac
Epoch 25/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5188 - accuracy: 0.0410 - val_loss: 0.0697 - val_ac
Epoch 26/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4970 - accuracy: 0.0429 - val_loss: 0.1668 - val_ac
Epoch 27/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5056 - accuracy: 0.0409 - val_loss: 0.0565 - val_ac
Epoch 28/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4992 - accuracy: 0.0406 - val_loss: 0.1064 - val_ac
Epoch 29/100
643/643 [==============================] - 12s 19ms/step - loss: 0.5036 - accuracy: 0.0407 - val_loss: 0.0907 - val_ac
Epoch 30/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4769 - accuracy: 0.0401 - val_loss: 0.0521 - val_ac
Epoch 31/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4577 - accuracy: 0.0436 - val_loss: 0.0539 - val_ac
Epoch 32/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4644 - accuracy: 0.0401 - val_loss: 0.0487 - val_ac
Epoch 33/100
643/643 [==============================] - 13s 19ms/step - loss: 0.4518 - accuracy: 0.0404 - val_loss: 0.0576 - val_ac
Epoch 34/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4576 - accuracy: 0.0413 - val_loss: 0.0413 - val_ac
Epoch 35/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4462 - accuracy: 0.0415 - val_loss: 0.0423 - val_ac
Epoch 36/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4730 - accuracy: 0.0418 - val_loss: 0.0680 - val_ac
Epoch 37/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4535 - accuracy: 0.0425 - val_loss: 0.0491 - val_ac
Epoch 38/100
643/643 [==============================] - 13s 20ms/step - loss: 0.4598 - accuracy: 0.0407 - val_loss: 0.0609 - val_ac
Epoch 39/100
643/643 [==============================] - 13s 19ms/step - loss: 0.4579 - accuracy: 0.0420 - val_loss: 0.0551 - val_ac
Epoch 40/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4430 - accuracy: 0.0412 - val_loss: 0.0730 - val_ac
Epoch 41/100
643/643 [==============================] - 12s 19ms/step - loss: 0.4553 - accuracy: 0.0429 - val_loss: 0.1224 - val_ac
Epoch 42/100
575/643 [=======================>....] - ETA: 1s - loss: 0.4364 - accuracy: 0.0422
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-50-e6d17237f5c8> in <module>()
     13   train_generator,
     14   epochs=100,
---> 15   validation_data=(x_val,y_val), steps_per_epoch=len(x_train) // 32)

 10 frames 
/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
model.evaluate(tX,  tY)
```

```
225/225 [==============================] - 1s 5ms/step - loss: 0.0989 - accuracy: 0.0506
[0.09888710826635361, 0.05061349645256996]
```

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?

При обучении точность была около 4-х процентов, поэтому дообучивать не было смысла, скорее всего это из-за того что ImageNet не содержит фотографий жестов и не обучался изначально на этих данных, поэтому не стоит использовать передаточное обучение на этом наборе данных

```
from keras.applications import MobileNet
from keras.layers import GlobalAveragePooling2D, Dense
from keras.models import Model
```

```
base_model=MobileNet(weights='imagenet',include_top=False)

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x=Dense(1024,activation='relu')(x)
x=Dense(512,activation='relu')(x)
preds=Dense(25,activation='softmax')(x)
model=Model(inputs=base_model.input,outputs=preds)


print(len(model.layers))
#for layer in model.layers:
#    layer.trainable=False

# or if we want to set the first 20 layers of the network to be non-trainable
for layer in model.layers[:20]:
    layer.trainable=False
for layer in model.layers[20:]:
    layer.trainable=True
```

```
92
```

```
import numpy as np
def transform(dataset):
  newDataset = list()
  for x in dataset:
    x = np.repeat(x, 3, 2)
    newDataset.append(x)
  return np.array(newDataset)

newTrainX = transform(x_train)
newValX = transform(x_val)

print(newTrainX.shape, newValX.shape)
```

```
(20591, 28, 28, 3) (6864, 28, 28, 3)
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(newTrainX, y_train, validation_data=(newValX, y_val), epochs=15)
```

```
Train on 20591 samples, validate on 6864 samples
Epoch 1/15
20591/20591 [==============================] - 37s 2ms/step - loss: 3.2080 - accuracy: 0.0454 - val_loss: 3.1988 - va
Epoch 2/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1947 - accuracy: 0.0476 - val_loss: 3.1898 - va
Epoch 3/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1882 - accuracy: 0.0476 - val_loss: 3.1848 - va
Epoch 4/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1845 - accuracy: 0.0476 - val_loss: 3.1819 - va
Epoch 5/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1821 - accuracy: 0.0476 - val_loss: 3.1800 - va
Epoch 6/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1805 - accuracy: 0.0476 - val_loss: 3.1786 - va
Epoch 7/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1794 - accuracy: 0.0476 - val_loss: 3.1777 - va
Epoch 8/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1786 - accuracy: 0.0476 - val_loss: 3.1770 - va
Epoch 9/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1780 - accuracy: 0.0476 - val_loss: 3.1765 - va
Epoch 10/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1776 - accuracy: 0.0476 - val_loss: 3.1761 - va
Epoch 11/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1772 - accuracy: 0.0476 - val_loss: 3.1759 - va
Epoch 12/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1770 - accuracy: 0.0476 - val_loss: 3.1756 - va
Epoch 13/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1768 - accuracy: 0.0476 - val_loss: 3.1754 - va
Epoch 14/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1766 - accuracy: 0.0476 - val_loss: 3.1753 - va
Epoch 15/15
20591/20591 [==============================] - 32s 2ms/step - loss: 3.1765 - accuracy: 0.0476 - val_loss: 3.1752 - va
<keras.callbacks.callbacks.History at 0x7fb19ec2a978>
```