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 (https://www.kaggle.com/datamunge/sign-language-mnist)](https://www.kaggle.com/datamunge/sign-language-mnist)

In [0]:
```python
# !pip install tensorflow==1.14.0
import pandas as pd
import tensorflow as tf
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models, Sequential
from sklearn.metrics import roc_auc_score
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePoo
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from tensorflow.keras.layers import InputLayer
from keras.layers import Concatenate
```

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

In [0]:
```python
df = pd.read_csv('/content/drive/My Drive/kaggle/sign_mnist_tra
print(df)

testDF = pd.read_csv('/content/drive/My Drive/kaggle/sign_mnist
print(testDF)
```

```
       label  pixel1  pixel2  pixel3  ...  pixel781  pixel782  pix
el783  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  pixe
l783  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]
```

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

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

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

In [0]:
```
trainX, testX, trainY, testY = train_test_split(x, y, test_size
```

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

In [0]:
```
model = Sequential([
    layers.Conv2D(64, kernel_size=3, activation='relu', input_s
    layers.Conv2D(32, kernel_size=5, activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(25, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crosse

model.fit(trainX, trainY, validation_data=(testX, testY), epoch

```

```
Train on 20591 samples, validate on 6864 samples
Epoch 1/15
20591/20591 [==============================] - 144s 7ms/sample - l
oss: 1.0571 - acc: 0.7136 - val_loss: 0.0424 - val_acc: 0.9850
Epoch 2/15
20591/20591 [==============================] - 145s 7ms/sample - l
oss: 0.0364 - acc: 0.9891 - val_loss: 0.0588 - val_acc: 0.9768
Epoch 3/15
20591/20591 [==============================] - 146s 7ms/sample - l
oss: 0.0436 - acc: 0.9884 - val_loss: 0.0113 - val_acc: 0.9969
Epoch 4/15
20591/20591 [==============================] - 146s 7ms/sample - l
```

```
oss: 0.0145 - acc: 0.9957 - val_loss: 7.3109e-04 - val_acc: 0.9999
Epoch 5/15
20591/20591 [==============================] - 147s 7ms/sample - l
oss: 8.1862e-05 - acc: 1.0000 - val_loss: 6.7947e-04 - val_acc: 0.
9999
Epoch 6/15
20591/20591 [==============================] - 145s 7ms/sample - l
oss: 3.6115e-05 - acc: 1.0000 - val_loss: 6.0458e-04 - val_acc: 0.
9999
Epoch 7/15
20591/20591 [==============================] - 144s 7ms/sample - l
oss: 2.1497e-05 - acc: 1.0000 - val_loss: 6.1721e-04 - val_acc: 0.
9999
Epoch 8/15
20591/20591 [==============================] - 144s 7ms/sample - l
oss: 1.3678e-05 - acc: 1.0000 - val_loss: 5.6933e-04 - val_acc: 0.
9999
Epoch 9/15
20591/20591 [==============================] - 142s 7ms/sample - l
oss: 9.1778e-06 - acc: 1.0000 - val_loss: 5.2476e-04 - val_acc: 0.
9999
Epoch 10/15
20591/20591 [==============================] - 142s 7ms/sample - l
oss: 6.2680e-06 - acc: 1.0000 - val_loss: 5.4921e-04 - val_acc: 0.
9999
Epoch 11/15
20591/20591 [==============================] - 141s 7ms/sample - l
oss: 4.3298e-06 - acc: 1.0000 - val_loss: 5.2253e-04 - val_acc: 0.
9999
Epoch 12/15
20591/20591 [==============================] - 141s 7ms/sample - l
oss: 3.0284e-06 - acc: 1.0000 - val_loss: 5.1974e-04 - val_acc: 0.
9999
Epoch 13/15
20591/20591 [==============================] - 141s 7ms/sample - l
oss: 2.1118e-06 - acc: 1.0000 - val_loss: 4.9742e-04 - val_acc: 0.
9999
Epoch 14/15
20591/20591 [==============================] - 141s 7ms/sample - l
oss: 1.4905e-06 - acc: 1.0000 - val_loss: 5.0703e-04 - val_acc: 0.
9999
Epoch 15/15
20591/20591 [==============================] - 142s 7ms/sample - l
oss: 1.0459e-06 - acc: 1.0000 - val_loss: 5.2003e-04 - val_acc: 0.
9999
```

Out[32]:
```
<tensorflow.python.keras.callbacks.History at 0x7f59071aa400>
```

In [0]:
```
test_loss, test_acc = model.evaluate(tX,  tY, verbose=2)

```

```
7172/7172 - 10s - loss: 0.9118 - acc: 0.8938
```

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

In [0]:
```
datagen = ImageDataGenerator(zoom_range=[0.5,1.0], brightness_r
train_generator = datagen.flow(trainX, trainY, batch_size=1)
test_generator = datagen.flow(testX, testY, batch_size=1)
```

In [0]:
```
model.fit_generator(
  train_generator,
  epochs=15,
  validation_data=test_generator)
```

```
Epoch 1/15
20591/20591 [==============================] - 291s 14ms/step - lo
ss: 3.1809 - acc: 0.0447 - val_loss: 3.1770 - val_acc: 0.0455
Epoch 2/15
20591/20591 [==============================] - 283s 14ms/step - lo
ss: 3.2048 - acc: 0.0444 - val_loss: 3.1705 - val_acc: 0.0447
Epoch 3/15
20591/20591 [==============================] - 278s 13ms/step - lo
ss: 3.1752 - acc: 0.0470 - val_loss: 3.1752 - val_acc: 0.0398
Epoch 4/15
20591/20591 [==============================] - 282s 14ms/step - lo
ss: 3.1747 - acc: 0.0459 - val_loss: 3.1761 - val_acc: 0.0455
Epoch 5/15
20591/20591 [==============================] - 287s 14ms/step - lo
ss: 3.1806 - acc: 0.0462 - val_loss: 3.1747 - val_acc: 0.0446
Epoch 6/15
20591/20591 [==============================] - 282s 14ms/step - lo
ss: 3.2386 - acc: 0.0478 - val_loss: 3.1763 - val_acc: 0.0453
Epoch 7/15
20591/20591 [==============================] - 293s 14ms/step - lo
ss: 3.1835 - acc: 0.0448 - val_loss: 3.1741 - val_acc: 0.0424
Epoch 8/15
20591/20591 [==============================] - 279s 14ms/step - lo
ss: 3.1763 - acc: 0.0440 - val_loss: 3.1778 - val_acc: 0.0455
Epoch 9/15
20591/20591 [==============================] - 280s 14ms/step - lo
ss: 3.1751 - acc: 0.0440 - val_loss: 3.1771 - val_acc: 0.0455
Epoch 10/15
20591/20591 [==============================] - 283s 14ms/step - lo
ss: 3.1761 - acc: 0.0436 - val_loss: 3.1768 - val_acc: 0.0455
Epoch 11/15
20591/20591 [==============================] - 289s 14ms/step - lo
ss: 3.1866 - acc: 0.0460 - val_loss: 3.1766 - val_acc: 0.0503
Epoch 12/15
20591/20591 [==============================] - 287s 14ms/step - lo
ss: 3.1774 - acc: 0.0453 - val_loss: 3.1745 - val_acc: 0.0510
Epoch 13/15
20591/20591 [==============================] - 290s 14ms/step - lo
```

```
ss: 3.1773 - acc: 0.0464 - val_loss: 3.1788 - val_acc: 0.0514
Epoch 14/15
20591/20591 [==============================] - 282s 14ms/step - lo
ss: 3.1792 - acc: 0.0450 - val_loss: 3.1771 - val_acc: 0.0433
Epoch 15/15
20591/20591 [==============================] - 279s 14ms/step - lo
ss: 3.1765 - acc: 0.0456 - val_loss: 3.1749 - val_acc: 0.0440
```

Out[66]:
```
<tensorflow.python.keras.callbacks.History at 0x7f5906ab69e8>
```

In [0]:
```
test_loss, test_acc = model.evaluate(tX,  tY, verbose=2)
```

```
7172/7172 - 10s - loss: 3.2389 - acc: 0.0291
```

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 [92]:

```
from keras.applications import MobileNet

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)
model.summary()
```

```
/usr/local/lib/python3.6/dist-packages/keras_applications/mobilenet.py:207: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
  warnings.warn('`input_shape` is undefined or non-square, '

Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, None, None, 3)     0
_________________________________________________________________
conv1 (Conv2D)               (None, None, None, 32)    864
_________________________________________________________________
conv1_bn (BatchNormalization (None, None, None, 32)    128
_________________________________________________________________
conv1_relu (ReLU)            (None, None, None, 32)    0
```

In [0]:

```
for layer in model.layers:
    layer.trainable=False
```

In [0]:

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

newTrainX = transform(trainX)
newTestX = transform(testX)
```

In [90]:
```
print(newTrainX.shape, newTestX.shape)
```

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

In [94]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crosse

model.fit(newTrainX, trainY, validation_data=(newTestX, testY),
```

```
Train on 20591 samples, validate on 6864 samples
Epoch 1/15
20591/20591 [==============================] - 47s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 2/15
20591/20591 [==============================] - 44s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 3/15
20591/20591 [==============================] - 44s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 4/15
20591/20591 [==============================] - 44s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 5/15
20591/20591 [==============================] - 44s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 6/15
20591/20591 [==============================] - 44s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 7/15
20591/20591 [==============================] - 45s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 8/15
20591/20591 [==============================] - 45s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 9/15
20591/20591 [==============================] - 45s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 10/15
20591/20591 [==============================] - 45s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 11/15
20591/20591 [==============================] - 45s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 12/15
20591/20591 [==============================] - 46s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 13/15
20591/20591 [==============================] - 46s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 14/15
20591/20591 [==============================] - 46s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
Epoch 15/15
20591/20591 [==============================] - 46s 2ms/step - loss
: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
```

Out[94]:
```
<keras.callbacks.History at 0x7f9abbce00b8>
```