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 [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import torch
import torch.nn as nn
import cv2
import matplotlib.pyplot as plt
import torchvision
from torch.utils.data import Dataset, DataLoader, ConcatDataset
from torchvision import transforms
import copy
import tqdm
from PIL import Image

%matplotlib inline
```

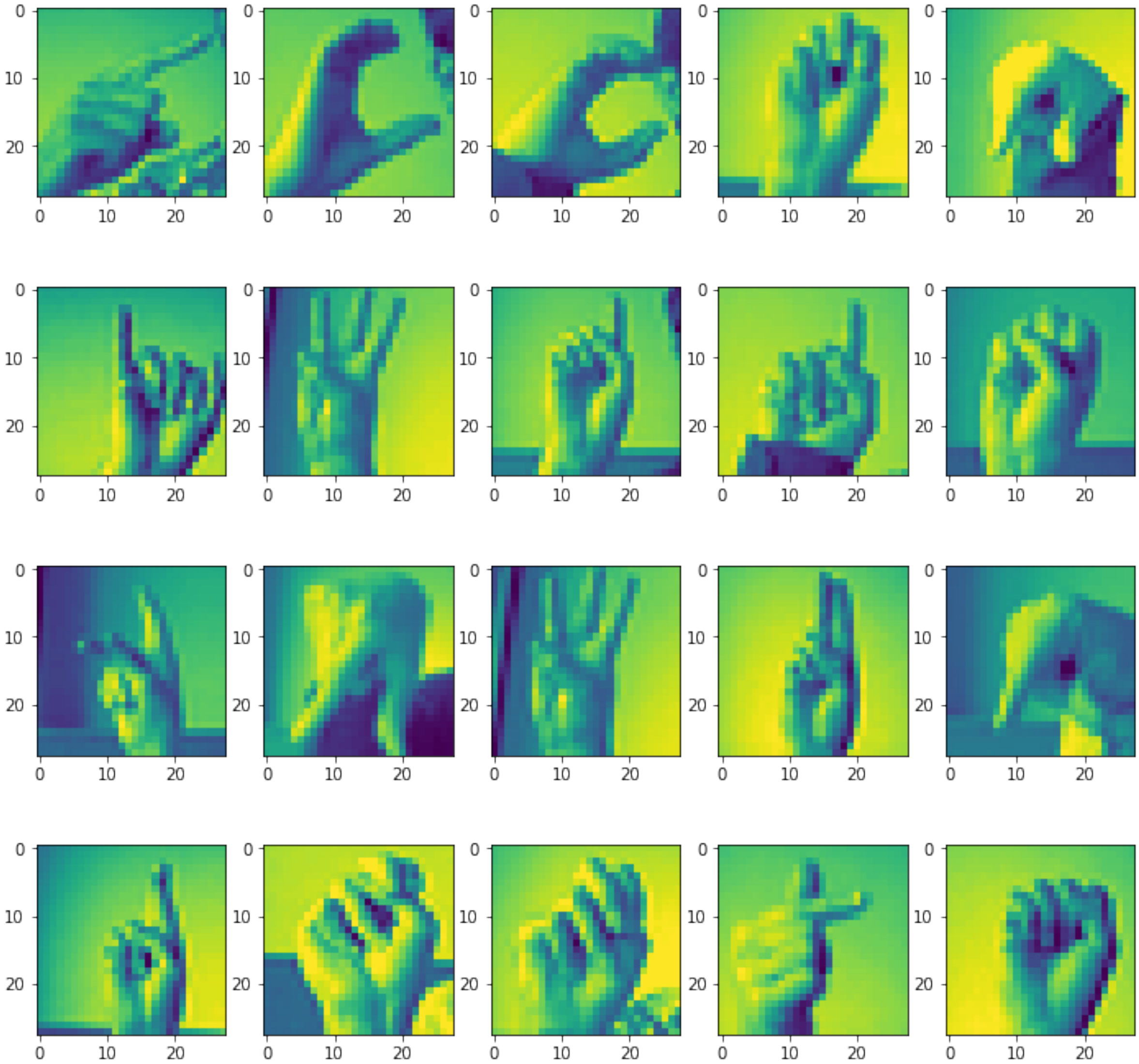
Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.

In [2]:

```
train_images = pd.read_csv('../input/sign-language-mnist/sign_mnist_train.csv'
)
test_images = pd.read_csv('../input/sign-language-mnist/sign_mnist_test.csv')
```

In [3]:

```
imgs = np.array(train_images.drop('label', axis=1).values)
plt.figure(figsize=(12,12))
for i in range(1,21):
    plt.subplot(4,5,i)
    plt.imshow(imgs[i].reshape(28,28))
```



In [4]:

```
class Dataset(Dataset):
    def __init__(self, data, transforms=None):
        self.data = data
        self.transforms = transforms

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        label = self.data.iloc[idx, 0]
        image = self.data.iloc[idx, 1:].values.reshape((28,28))

        if self.transforms:
            image = self.transforms(np.uint8(image))
        else:
            image = image.astype(np.float32)

        return image, label
```

In [14]:

```
from sklearn.model_selection import train_test_split

train_data, val_data = train_test_split(train_images, test_size=0.2)

batch_size = 32
num_workers = 4

transform = transforms.Compose([
    transforms.ToTensor()
])

train_dataset = Dataset(train_data, transforms=transform)
train_loader = DataLoader(train_dataset, num_workers=num_workers, batch_size=b
atch_size, shuffle=True)

val_dataset = Dataset(val_data, transforms=transform)
val_loader = DataLoader(val_dataset, num_workers=num_workers, batch_size=batch
_size, shuffle=False)

test_dataset = Dataset(test_images, transforms=transform)
test_loader = DataLoader(test_dataset, num_workers=num_workers, batch_size=bat
ch_size, shuffle=False)
```

Реализуйте глубокую нейронную сеть со сверточными слоями.

In [15]:

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()

        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, s
tride=1, padding=2)
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        self.cnn2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5,
stride=1, padding=2)
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.fc = nn.Linear(32 * 7 * 7, 25)

    def forward(self, x):
        out = self.cnn1(x)
        out = self.relu1(out)
        out = self.maxpool1(out)
        out = self.cnn2(out)
        out = self.relu2(out)
        out = self.maxpool2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)

        return out
```

In [16]:

```
def validate(model, val_loader, criterion):
    correct = 0
    total = 0
    total_loss = 0
    for images, labels in val_loader:
        images = images.cuda()
        labels = labels.cuda()

        outputs = model(images)
        loss = criterion(outputs, labels)

        total_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum()

    accuracy = correct.cpu().numpy() / total
    loss = total_loss / len(val_loader)
    return accuracy, loss

def train(model, train_loader, val_loader, num_epochs, optimizer, criterion, s
cheduler):
    train_acc = []
    val_acc = []
    for epoch in tqdm.tqdm(range(num_epochs)):
```

```python
        total_loss = 0
        correct = 0
        total = 0
        for i, (images, labels) in enumerate(train_loader):
            images = images.cuda()
            labels = labels.cuda()

            outputs = model(images)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()

            total_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum()

        train_loss = total_loss / len(train_loader)
        train_accuracy = correct.cpu().numpy() / total
        accuracy, loss = validate(model, val_loader, criterion)
        # Print Loss
        print(f'[Epoch {epoch+1}/{num_epochs}] Train Loss: {train_loss:.4f}, A
ccuracy: {train_accuracy:.3f}; Val Loss: {loss:.4f}, Accuracy: {accuracy:.3f}'
)
        val_acc.append(accuracy)
        train_acc.append(train_accuracy)

    return train_acc, val_acc
```

In [17]:

```
from torch.optim import lr_scheduler

model = CNNModel().cuda()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.002, amsgrad=True)
scheduler = lr_scheduler.CosineAnnealingLR(optimizer, 30)
train_acc, val_acc = train(model, train_loader, val_loader, 5, optimizer, criterion, scheduler)
```

```
 20%|██        | 1/5 [00:14<00:56, 14.22s/it]

[Epoch 1/5] Train Loss: 1.1997, Accuracy: 0.652; Val Loss: 0.3159,
Accuracy: 0.916

 40%|████      | 2/5 [00:28<00:42, 14.31s/it]

[Epoch 2/5] Train Loss: 0.1578, Accuracy: 0.958; Val Loss: 0.0532,
Accuracy: 0.993

 60%|██████    | 3/5 [00:43<00:29, 14.55s/it]

[Epoch 3/5] Train Loss: 0.0331, Accuracy: 0.996; Val Loss: 0.0163,
Accuracy: 1.000

 80%|████████  | 4/5 [00:58<00:14, 14.66s/it]

[Epoch 4/5] Train Loss: 0.0110, Accuracy: 1.000; Val Loss: 0.0066,
Accuracy: 1.000

100%|██████████| 5/5 [01:13<00:00, 14.63s/it]

[Epoch 5/5] Train Loss: 0.0051, Accuracy: 1.000; Val Loss: 0.0045,
Accuracy: 1.000
```

In [18]:

```
accuracy, loss = validate(model, test_loader, criterion)
print(f'Test Loss: {loss:.4f}, Accuracy: {accuracy:.3f}')
```

```
Test Loss: 0.3553, Accuracy: 0.898
```

Примените дополнение данных (data augmentation).

In [19]:

```
data_transform = transforms.Compose([
    transforms.ToPILImage(mode=None),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomPerspective(distortion_scale=0.5, p=0.5, interpolation=3)
,
    transforms.ToTensor(),
])

aug_dataset = Dataset(train_data, transforms = data_transform)
aug_loader = DataLoader(aug_dataset, batch_size = 32, shuffle=True, num_worker
s=4)
```

In [20]:

```
train_acc, val_acc = train(model, aug_loader, val_loader, 5, optimizer, criter
ion, scheduler)
```

```
 20%|██        | 1/5 [00:19<01:17, 19.49s/it]

[Epoch 1/5] Train Loss: 2.4795, Accuracy: 0.387; Val Loss: 0.8227,
Accuracy: 0.805

 40%|████      | 2/5 [00:40<00:59, 19.84s/it]

[Epoch 2/5] Train Loss: 1.4786, Accuracy: 0.560; Val Loss: 0.5980,
Accuracy: 0.842

 60%|██████    | 3/5 [00:59<00:39, 19.74s/it]

[Epoch 3/5] Train Loss: 1.2590, Accuracy: 0.620; Val Loss: 0.4918,
Accuracy: 0.882

 80%|████████  | 4/5 [01:19<00:19, 19.66s/it]

[Epoch 4/5] Train Loss: 1.1090, Accuracy: 0.660; Val Loss: 0.4215,
Accuracy: 0.885

100%|██████████| 5/5 [01:40<00:00, 20.00s/it]

[Epoch 5/5] Train Loss: 1.0083, Accuracy: 0.690; Val Loss: 0.3231,
Accuracy: 0.925
```

In [21]:

```
accuracy, loss = validate(model, test_loader, criterion)
print(f'After augmentation Test Loss: {loss:.4f}, Accuracy: {accuracy:.3f}')
```

```
After augmentation Test Loss: 0.5894, Accuracy: 0.817
```

Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение.

In [27]:

```
from torchvision import models

model = models.resnet18(pretrained=True)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 25)
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3,
                        bias=False)
model = model.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.002, amsgrad=True)
scheduler = lr_scheduler.CosineAnnealingLR(optimizer, 30)
```

In [28]:

```
ds = ConcatDataset([train_dataset, aug_dataset])
resnet_loader = DataLoader(ds, batch_size = 32, shuffle=True, num_workers=4)
```

In [ ]:

```
train_acc, val_acc = train(model, resnet_loader, val_loader, 5, optimizer, cri
terion, scheduler)
```

```
 20%|██        | 1/5 [00:58<03:55, 58.94s/it]

[Epoch 1/5] Train Loss: 1.0287, Accuracy: 0.674; Val Loss: 0.1829,
Accuracy: 0.950
```

In [ ]:

```
accuracy, loss = validate(model, test_loader, criterion)
print(f'ResNet18 Test Loss: {loss:.4f}, Accuracy: {accuracy:.3f}')
```

Вывод:

В данной лабораторной мы использовали сверточную нейронную сеть для реализации многоклассового классификатора на наборе данных языка жестов. Использовали аугментацию данных и предаточное обучение.