РАСПОЗНАВАНИЕ СПЕКТРОВ ИМПУЛЬСНЫХ РАДИОЛОКАЦИОННЫХ СИГНАЛОВ
С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ

Сигналы радиолокационных станций (РЛС) имеют разную структуру и, соответственно, разные спектры [1, 2]. Возьмем для примера три широко применяемых в радиолокации типа импульсных сигналов [2]:

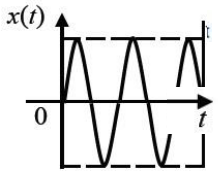


Рис. 1. Простой немодулированный радиоимпульс

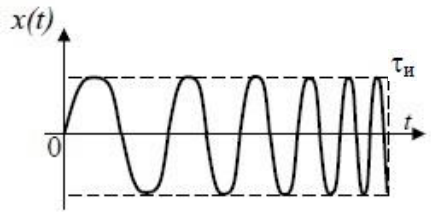


Рис. 2. Импульс с линейной частотной модуляцией (ЛЧМ)

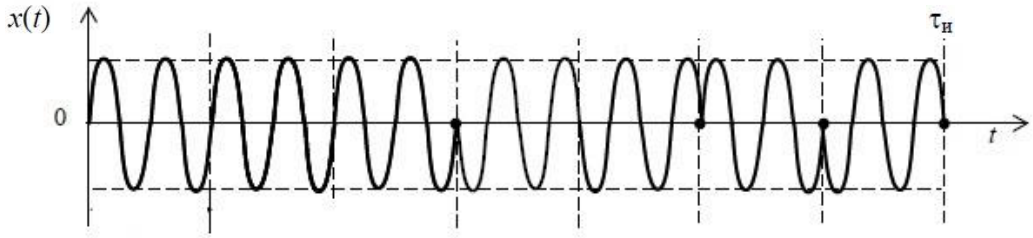


Рис. 3. Импульс с фазовой кодовой манипуляцией (ФКМ)

Спектры (модули амплитудных спектральных функций) перечисленных сигналов, как известно, имеют следующий вид [2]:

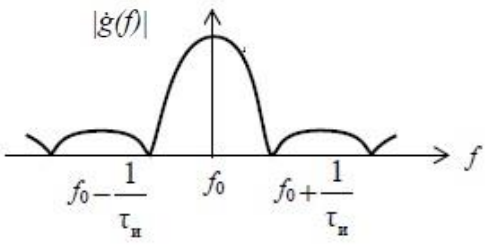


Рис. 4. Спектр простого немодулированного радиоимпульса

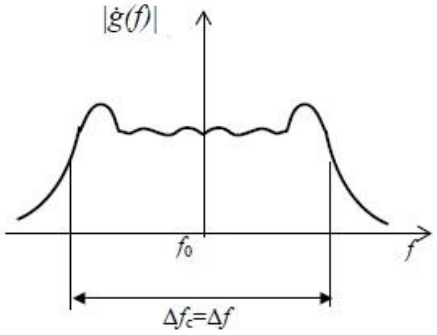


Рис. 5. Спектр ЛЧМ-сигнала

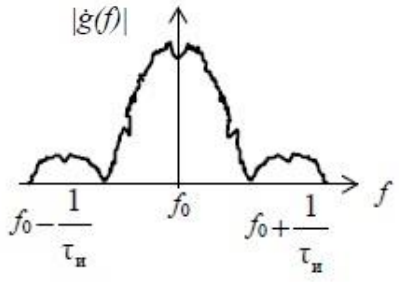


Рис. 6. Спектр ФКМ-сигнала

Решим задачу распознавания данных спектров. Для этого используем сверточную нейронную сеть (CNN). Как известно [3, 4, 5], сети такого типа наилучшим образом подходят для распознавания изображений.

Моделирование будем проводить с использованием фреймворка PyTorch [3].

Спроектируем простую VGG-образную CNN [3, 4] по следующей схеме (рис.7):

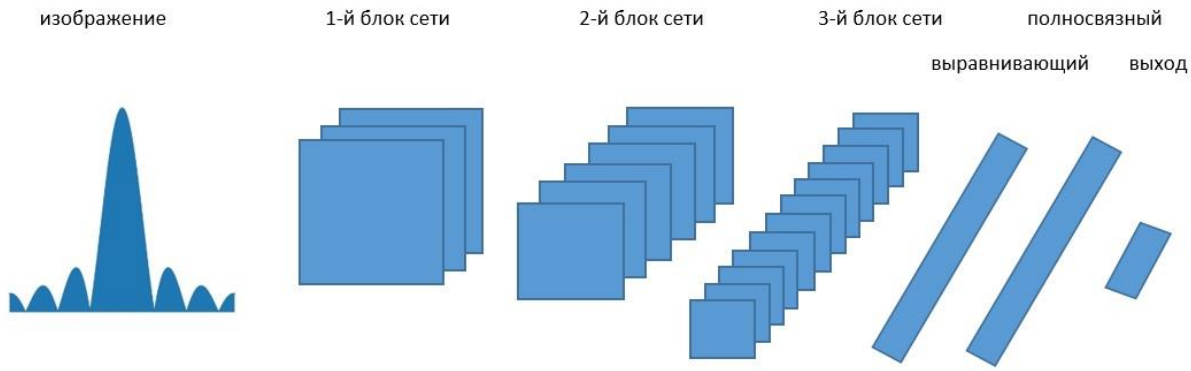


Рис. 7. Сверточная VGG-образная нейронная сеть

Каждый блок сети состоит последовательно из сверточного слоя, слоев активации, пакетной нормализации, а также подвыборочного и прореживающего слоев. Завершают сеть выравнивающий, полносвязный и выходной слои.

Первый блок содержит 32 элемента свертки с ядром 3x3, второй – 64, последний– 128. Используемые здесь функции активации - “relu”, подвыборочные слои – “maxpooling (2x2)”, прореживание – 25%. После выравнивающего слоя использован 512-элементный полносвязный слой с последующим 50%-м прореживанием. На выходе имеем трехэлементный полносвязный слой с последующей активацией “softmax”.

Используемый нами код для тренировки сети и распознавания ею контрольных изображений смотрите в Приложении 1 к данной статье.

Рассмотрим результат работы данной сверточной нейросети.

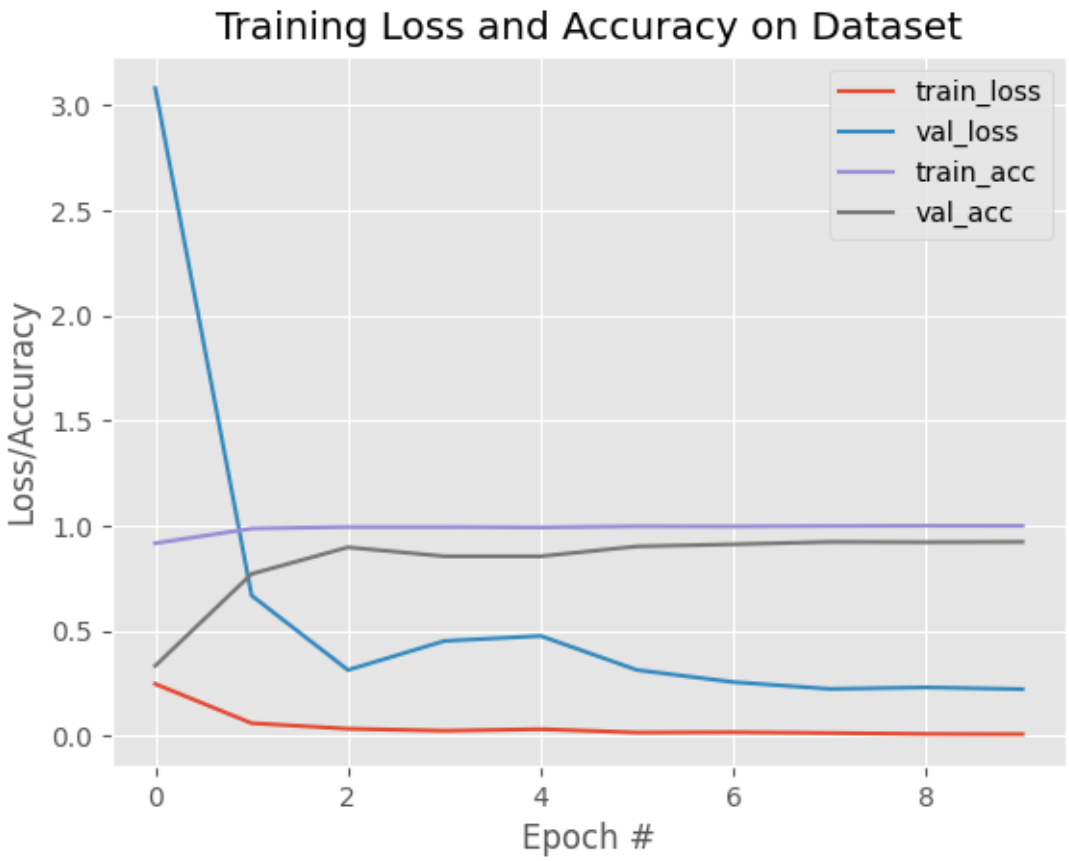


Рис.8. Графики функций потерь и точности при тренировке сверточной нейросети (оптимизатор SGD, размер батча – 64, количество эпох – 10

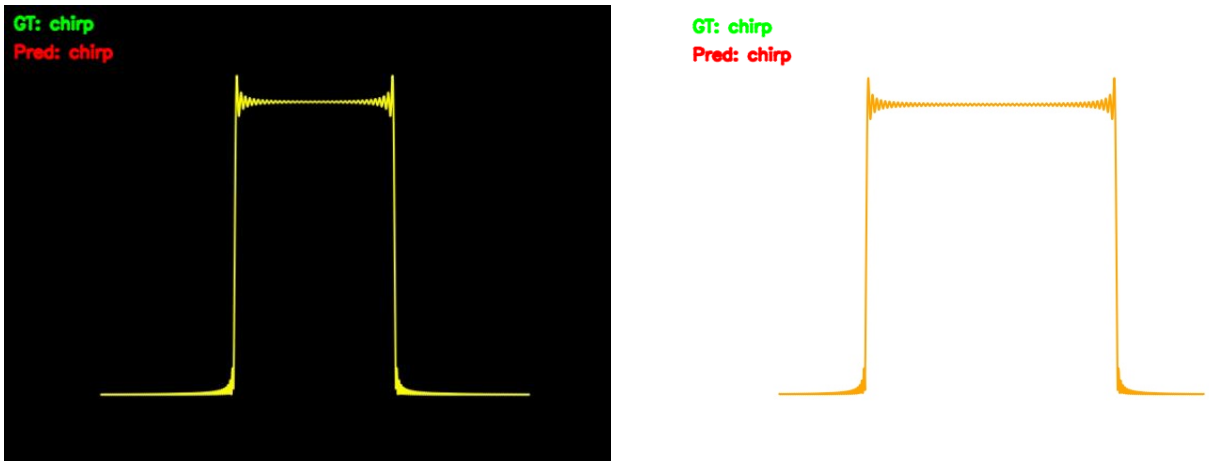


Рис. 9. Результаты распознавания ЛЧМ-сигналов

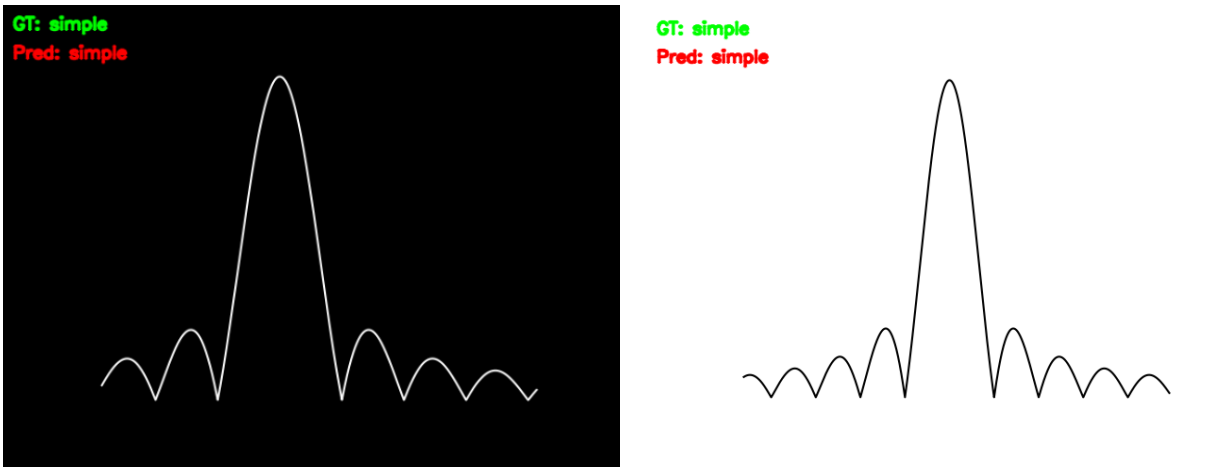


Рис. 10. Результаты распознавания простых немодулированных радиоимпульсов

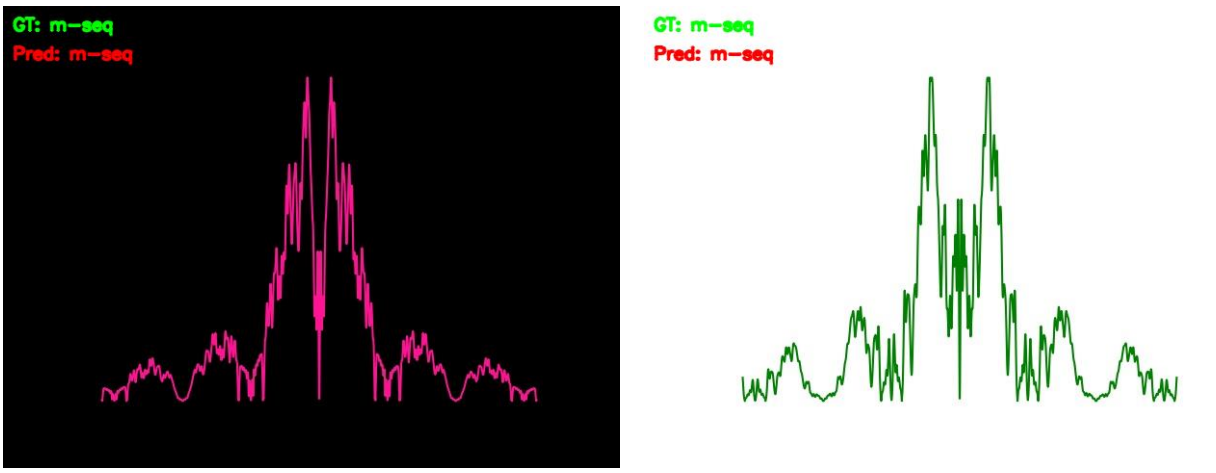


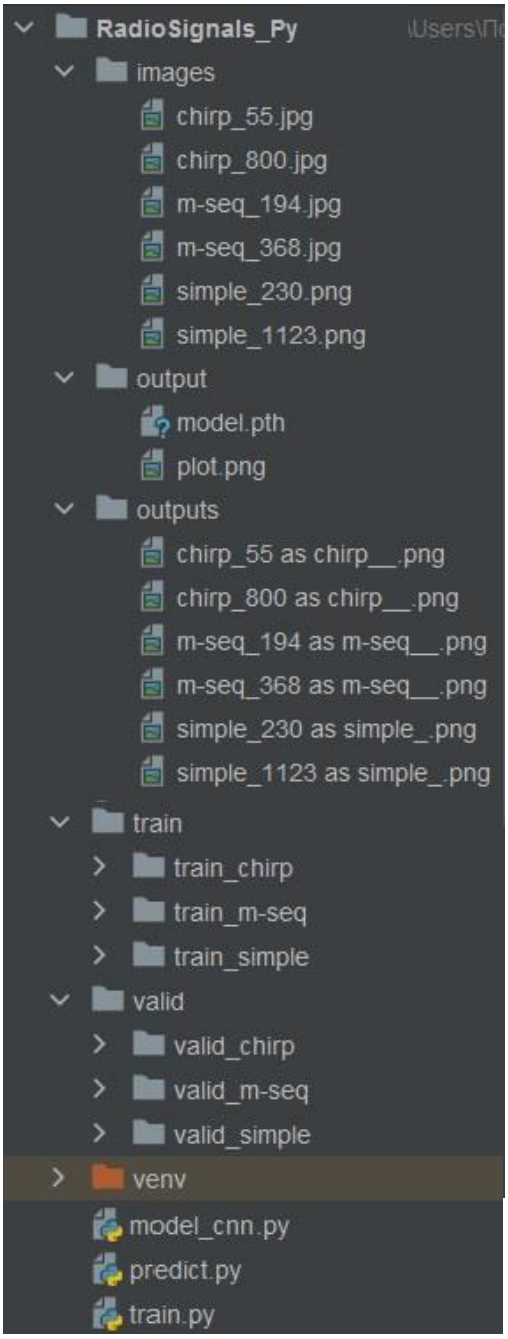
Рис. 11. Результаты распознавания ФКМ-сигналов

С помощью сверточной нейронной сети мы получили точные результаты. Все спектры распознаются правильно, в соответствии со своими классами.

Таким образом, можно сделать следующий вывод: для распознавания спектров радиолокационных сигналов целесообразно использовать достаточно несложную сверточную нейронную сеть.

ПРИЛОЖЕНИЕ 1

Структура проекта (папки/файлы):



```
|
|---images
|    |---chirp_55.jpg (контрольное изображение спектра ЛЧМ-импульса)
|    |---chirp_800.jpg (контрольное изображение спектра ЛЧМ-импульса)
|    I---m-seq_194.jpg (контрольное изображение спектра ФКМ-импульса)
|    I---m-seq_368.jpg (контрольное изображение спектра ФКМ-импульса)
|    |---simple_230.jpg (контрольное изображение спектра немодулированного импульса)
|    |---simple_1123.jpg (контрольное изображение спектра немодулированного импульса)
|
|---output
|    |---model.pth (сериализованная модель сверточной сети)
|    |---plot.png (график потерь/точности распознавания для простой сети)
|
|---outputs (результаты распознавания контрольных изображений)
|
|---train (тренировочный датасет)
|    |---train_chirp (спектры ЛЧМ-импульсов, 700 изображений)
|    |---train_m-seq (спектры ФКМ-импульсов, 700 изображений)
|    I---train_simple (спектры простых немодулированных импульсов, 700 изображений)
|
|---valid (проверочный датасет)
|    |---train_chirp (спектры ЛЧМ-импульсов, 234 изображения)
|    |---train_m-seq (спектры ФКМ-импульсов, 234 изображения)
|    I---train_simple (спектры простых немодулированных импульсов, 234 изображения)
|
|---model_cnn.py (модуль построения нейросети)
|---predict.py (модуль оценки верности распознавания изображений)
|---train.py (модуль тренировки сверточной сети)
```

Код класса, определяющего структуру простой VGG-образной CNN

```
from torch.nn import Module
from torch.nn import Conv2d
from torch.nn import ReLU
from torch.nn import BatchNorm2d
from torch.nn import MaxPool2d
from torch.nn import BatchNorm1d
from torch.nn import Dropout
from torch.nn import Linear
from torch.nn import LogSoftmax
from torch import flatten


class Net(Module):
    def __init__(self, numChannels, classes):
        super(Net, self).__init__()
        self.conv1 = Conv2d(in_channels=numChannels, out_channels=32,
kernel_size=(3, 3))
        self.relu1 = ReLU()
        self.bn1 = BatchNorm2d(32)
        self.maxpool1 = MaxPool2d(kernel_size=(2, 2))
        self.dropout1 = Dropout(0.25)

        self.conv2 = Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3))
        self.relu2 = ReLU()
        self.bn2 = BatchNorm2d(64)
        self.maxpool2 = MaxPool2d(kernel_size=(2, 2))
        self.dropout2 = Dropout(0.25)
```

```
        self.conv3 = Conv2d(in_channels=64, out_channels=128, kernel_size=(3,
3))
        self.relu3 = ReLU()
        self.bn3 = BatchNorm2d(128)
        self.maxpool3 = MaxPool2d(kernel_size=(2, 2))
        self.dropout3 = Dropout(0.25)

        self.fc1 = Linear(in_features=6144, out_features=500)
        self.relu4 = ReLU()
        self.bn4 = BatchNorm1d(500)
        self.dropout4 = Dropout(0.5)

        self.fc2 = Linear(in_features=500, out_features=classes)
        self.logSoftmax = LogSoftmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.bn1(x)
        x = self.maxpool1(x)
        x = self.dropout1(x)

        x = self.conv2(x)
        x = self.relu2(x)
        x = self.bn2(x)
        x = self.maxpool2(x)
        x = self.dropout2(x)

        x = self.conv3(x)
        x = self.relu3(x)
        x = self.bn3(x)
        x = self.maxpool3(x)
        x = self.dropout3(x)

        x = flatten(x, 1)
        x = self.fc1(x)
        x = self.relu4(x)
        x = self.bn4(x)
        x = self.dropout4(x)

        x = self.fc2(x)
        output = self.logSoftmax(x)

        return output
```

Код для тренировки простой VGG-образной CNN

```
# USAGE

# python train.py --model output/model.pth --plot output/plot.png

import matplotlib
matplotlib.use("Agg")

from model_cnn import Net
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.optim import SGD
from torch import nn
import matplotlib.pyplot as plt
```

```
import numpy as np
import argparse
import torch
import time

ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", type=str, required=True,
                help="path to output trained model")
ap.add_argument("-p", "--plot", type=str, required=True,
                help="path to output loss/accuracy plot")
args = vars(ap.parse_args())

INIT_LR = 1e-2
BATCH_SIZE = 64
EPOCHS = 10

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

train_transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

valid_transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

train_dataset = datasets.ImageFolder(
    root='train',
    transform=train_transform
)

valid_dataset = datasets.ImageFolder(
    root='valid',
    transform=valid_transform
)

train_loader = DataLoader(
    train_dataset, batch_size=BATCH_SIZE, shuffle=True,
    num_workers=0, pin_memory=True
)

valid_loader = DataLoader(
    valid_dataset, batch_size=BATCH_SIZE, shuffle=False,
    num_workers=0, pin_memory=True
)

trainSteps = len(train_loader.dataset) // BATCH_SIZE
valSteps = len(valid_loader.dataset) // BATCH_SIZE

print("[INFO] initializing the network model...")
model = Net(numChannels=3, classes=3)

opt = SGD(model.parameters(), lr=INIT_LR)
lossFn = nn.NLLLoss()
```

```
H = {
    "train_loss": [],
    "train_acc": [],
    "val_loss": [],
    "val_acc": []
}

print("[INFO] training the network...")
startTime = time.time()

for e in range(0, EPOCHS):
    model.train()

    totalTrainLoss = 0
    totalValLoss = 0

    trainCorrect = 0
    valCorrect = 0

    for (x, y) in train_loader:
        (x, y) = (x.to(device), y.to(device))

        pred = model(x)
        loss = lossFn(pred, y)

        opt.zero_grad()
        loss.backward()
        opt.step()

        totalTrainLoss += loss
        trainCorrect += (pred.argmax(1) == y).type(
            torch.float).sum().item()

    with torch.no_grad():
        model.eval()

        for (x, y) in valid_loader:
            (x, y) = (x.to(device), y.to(device))

            pred = model(x)
            totalValLoss += lossFn(pred, y)

            valCorrect += (pred.argmax(1) == y).type(
                torch.float).sum().item()

    avgTrainLoss = totalTrainLoss / trainSteps
    avgValLoss = totalValLoss / valSteps

    trainCorrect = trainCorrect / len(train_loader.dataset)
    valCorrect = valCorrect / len(valid_loader.dataset)

    H["train_loss"].append(avgTrainLoss.cpu().detach().numpy())
    H["train_acc"].append(trainCorrect)
    H["val_loss"].append(avgValLoss.cpu().detach().numpy())
    H["val_acc"].append(valCorrect)

    print("[INFO] EPOCH: {}/{}".format(e + 1, EPOCHS))
    print("Train loss: {:.6f}, Train accuracy: {:.4f}".format(
        avgTrainLoss, trainCorrect))
    print("Val loss: {:.6f}, Val accuracy: {:.4f}\n".format(
        avgValLoss, valCorrect))

endTime = time.time()
```

```
print("[INFO] total time taken to train the model: {:.2f}s".format(
    endTime - startTime))

plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["val_loss"], label="val_loss")
plt.plot(H["train_acc"], label="train_acc")
plt.plot(H["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(args["plot"])

torch.save(model, args["model"])
```

Код для оценки верности распознавания изображений

```
# USAGE

# python predict.py --image images/chirp_55.jpg --model output\model.pth

import torchvision.transforms as transforms
import torch
import argparse
import cv2
import os

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True)
ap.add_argument('-m', '--model', required=True)
args = vars(ap.parse_args())

device = ('cuda' if torch.cuda.is_available() else 'cpu')

labels = ['chirp', 'm-seq', 'simple']

model = torch.load(args["model"]).to(device)
model.eval()

transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

image = cv2.imread(args['image'])

gt_class = args['image'].split('/')[-1][:6]

orig_image = image.copy()

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = transform(image)

image = torch.unsqueeze(image, 0)
with torch.no_grad():
    outputs = model(image.to(device))
```

```
output_label = torch.topk(outputs, 1)
pred_class = labels[int(output_label.indices)]
cv2.putText(orig_image,
            f"GT: {gt_class}",
            (10, 25),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.6, (0, 255, 0), 2, cv2.LINE_AA
            )
cv2.putText(orig_image,
            f"Pred: {pred_class}",
            (10, 55),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.6, (0, 0, 255), 2, cv2.LINE_AA
            )
print(f"GT: {gt_class}, pred: {pred_class}")
cv2.imwrite(f"outputs/{args['image'].split('/')[-1].split('.')[0]} as
{gt_class}_.png", orig_image)
cv2.imshow('Result', orig_image)
cv2.waitKey(0)
```

ЛИТЕРАТУРА

1. Л. Е. Варакин. Системы связи с шумоподобными сигналами. - М.: Радио и связь, 1985
2. Радиолокационные системы. / Под ред. В. П. Бердышева. – Красноярск, СФУ, 2012
3. Eli Stevens et al. Deep Learning with PyTorch. – Manning, 2020
4. Adrian Rosebrock. Deep Learning for Computer Vision with Python. – PyImageSearch, 2017
5. Ян Гудфеллоу и др. Глубокое обучение. – М.: ДМК Пресс, 2018