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]:

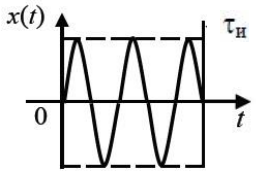


Рис. 1. Простой немодулированный радиоимпульс

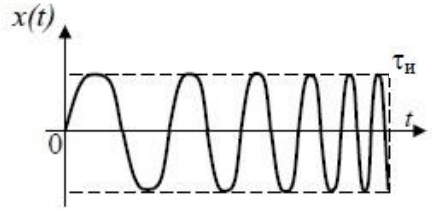


Рис. 2. Импульс с линейной частотной модуляцией (ЛЧМ)

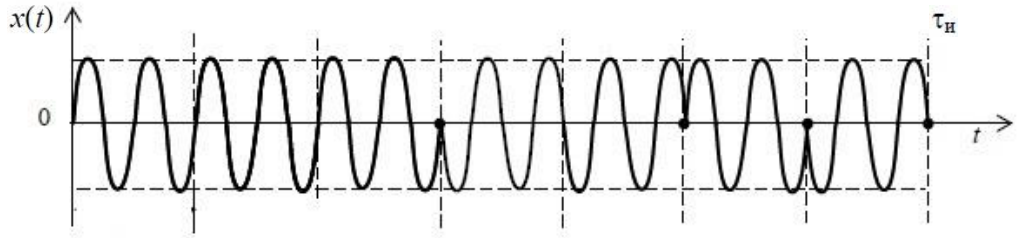


Рис. 3. Импульс с фазовой кодовой манипуляцией (ФКМ)

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

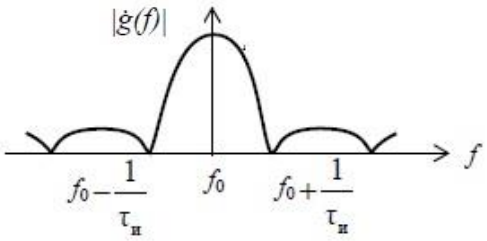


Рис. 4. Спектр простого немодулированного радиоимпульса

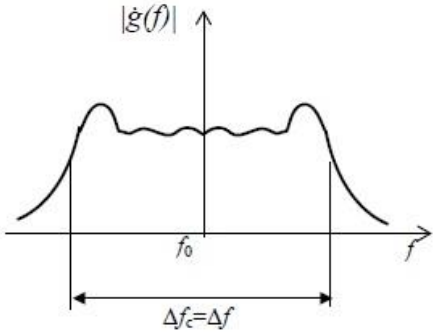


Рис. 5. Спектр ЛЧМ-сигнала

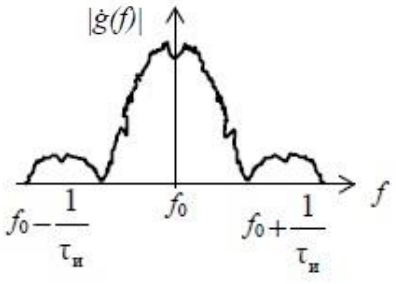


Рис. 6. Спектр ФКМ-сигнала

Решим задачу распознавания данных спектров. Для этого используем нейронные сети двух типов: сеть с несколькими полносвязными слоями и сверточную нейросеть [3].

Построим простую последовательную нейросеть следующего вида (рис.7):

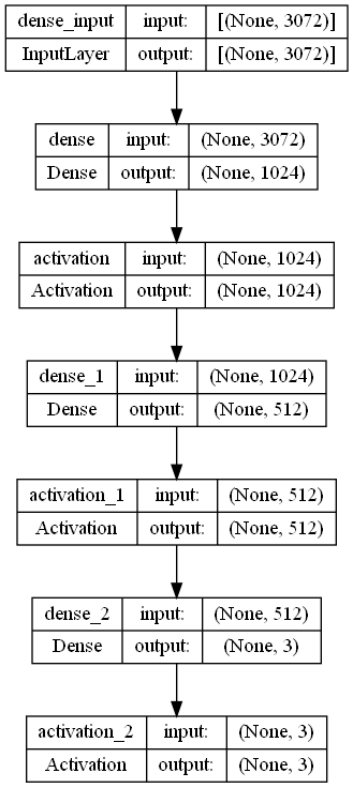


Рис. 7. Простая последовательная нейронная сеть

Моделирование будем проводить с использованием фреймворка Keras [3].

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

Рассмотрим результат работы простой последовательной нейросети, построенной на полносвязных слоях.

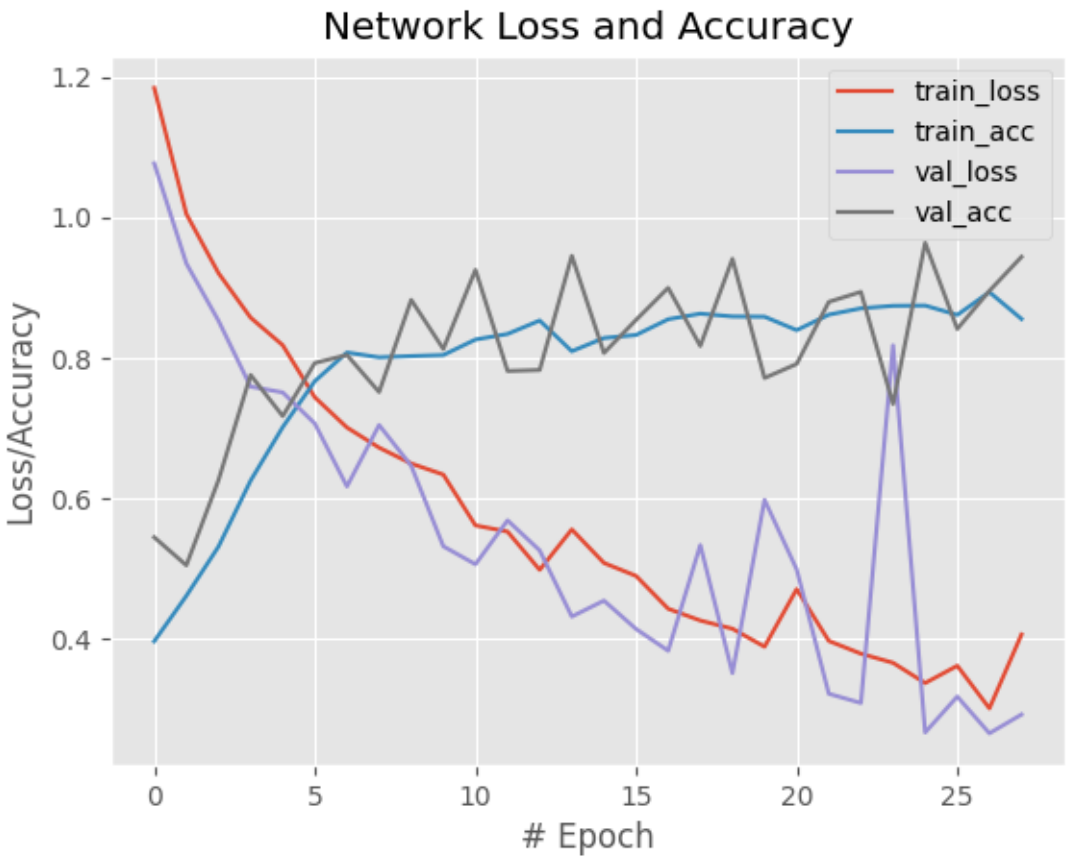
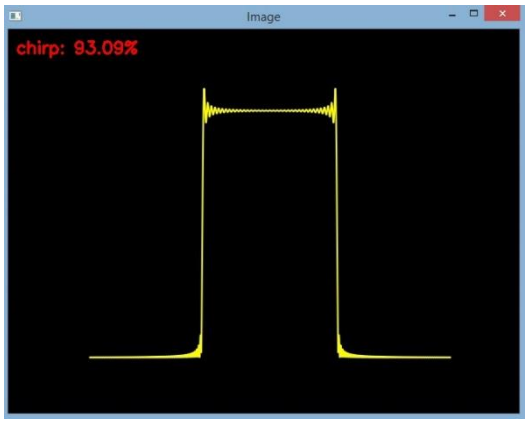


Рис.8. Графики функций потерь и точности при тренировке простой нейросети (оптимизатор SGD, размер батча – 32, количество эпох – 29)

	precision	recall	f1-score	support
chirp	0.88	0.98	0.93	228
m-seq	1.00	0.99	0.99	242
simple	0.97	0.86	0.91	231
accuracy			0.94	701
macro avg	0.95	0.94	0.94	701
weighted avg	0.95	0.94	0.94	701

Рис.9. Метрики классификации после тренировки простой нейросети



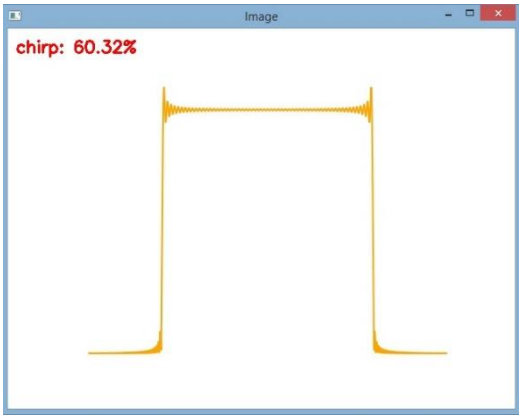
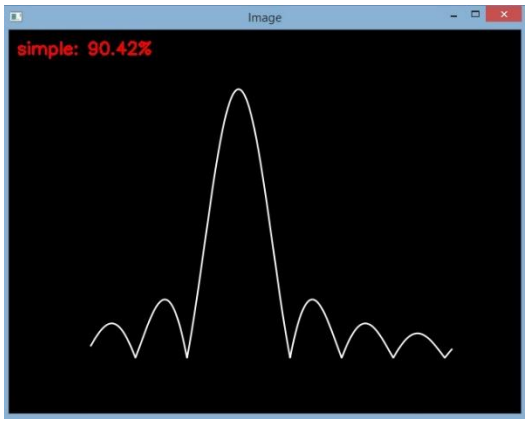


Рис.10. Результаты распознавания ЛЧМ-сигналов



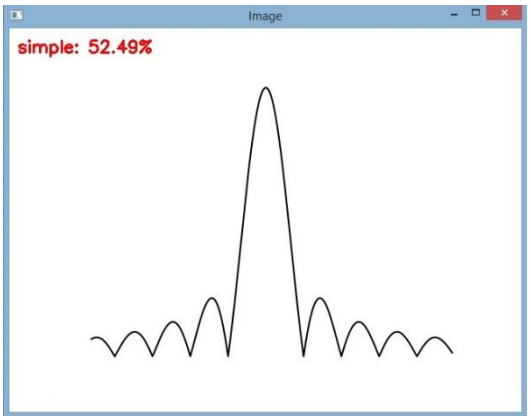
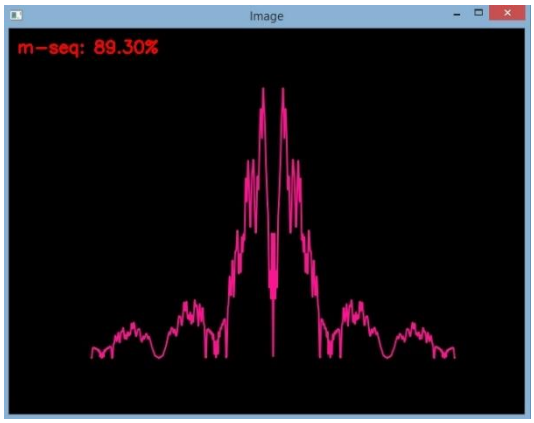


Рис. 11. Результаты распознавания простых немодулированных радиоимпульсов



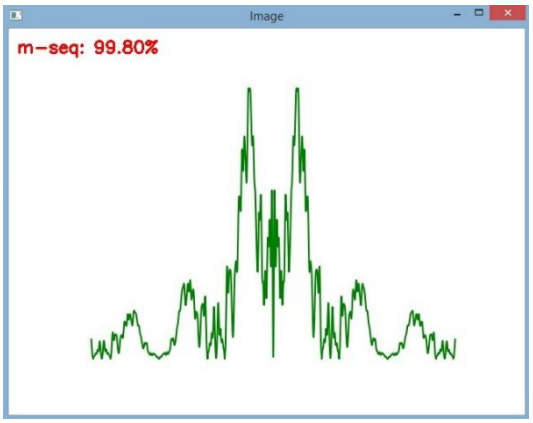
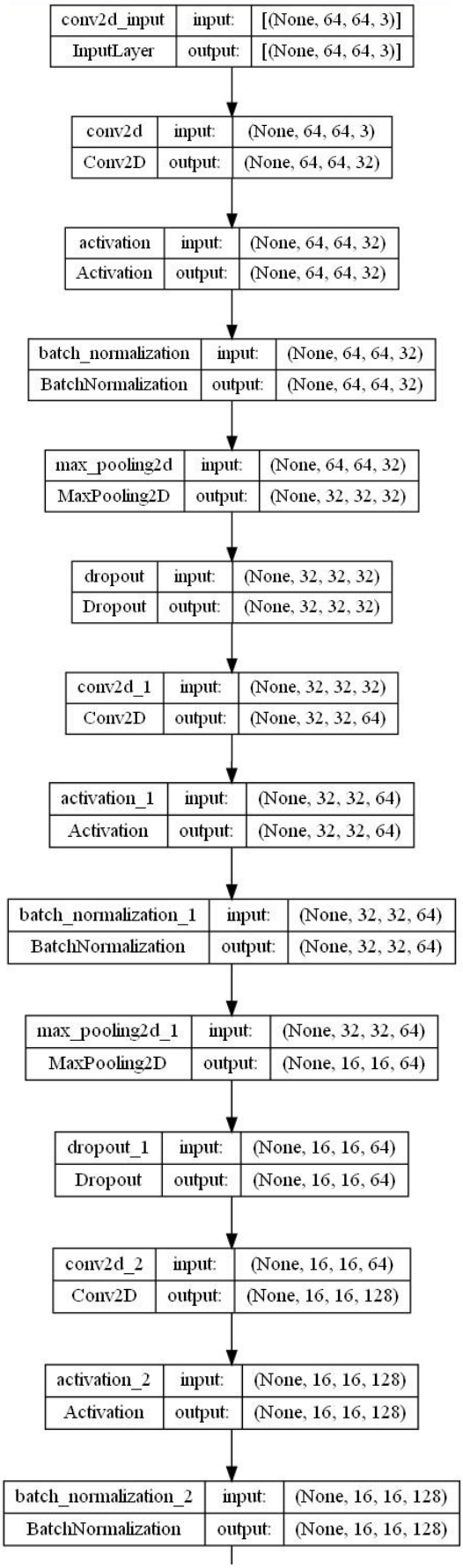


Рис. 12. Результаты распознавания ФКМ-сигналов

Из рисунков видно, что даже с помощью простой последовательной нейросети большинство спектров удалось распознать с 90%-ной точностью, и только два из них оценены с точностями 60% и 52%.

Для сравнения построим сверточную нейронную сеть (CNN). Как известно [3, 4, 5], для распознавания изображений CNN подходит гораздо больше, чем простая сеть с полносвязными слоями.

Спроектируем простую VGG-образную CNN [3, 4] по следующей схеме (рис.13):


conv2d_input | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 64, 64, 32)
activation | input: | (None, 64, 64, 32)
Activation | output: | (None, 64, 64, 32)
batch_normalization | input: | (None, 64, 64, 32)
BatchNormalization | output: | (None, 64, 64, 32)
max_pooling2d | input: | (None, 64, 64, 32)
MaxPooling2D | output: | (None, 32, 32, 32)
dropout | input: | (None, 32, 32, 32)
Dropout | output: | (None, 32, 32, 32)
conv2d_1 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 32, 32, 64)
activation_1 | input: | (None, 32, 32, 64)
Activation | output: | (None, 32, 32, 64)
batch_normalization_1 | input: | (None, 32, 32, 64)
BatchNormalization | output: | (None, 32, 32, 64)
max_pooling2d_1 | input: | (None, 32, 32, 64)
MaxPooling2D | output: | (None, 16, 16, 64)
dropout_1 | input: | (None, 16, 16, 64)
Dropout | output: | (None, 16, 16, 64)
conv2d_2 | input: | (None, 16, 16, 64)
Conv2D | output: | (None, 16, 16, 128)
activation_2 | input: | (None, 16, 16, 128)
Activation | output: | (None, 16, 16, 128)
batch_normalization_2 | input: | (None, 16, 16, 128)
BatchNormalization | output: | (None, 16, 16, 128)

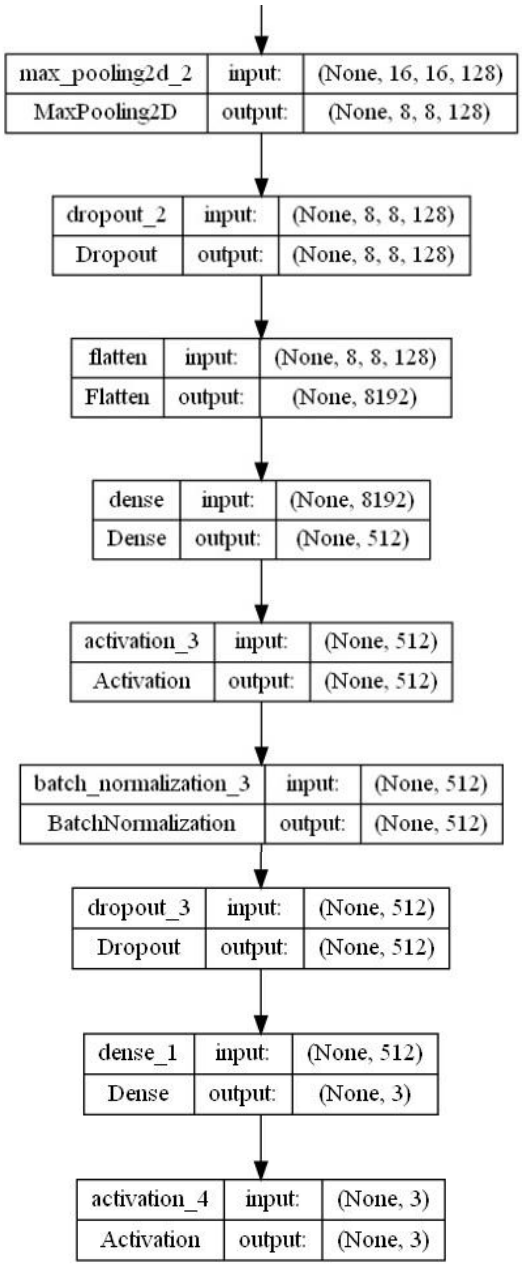


Рис. 13. Сверточная VGG-образная нейронная сеть

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 2 к данной статье.

Рассмотрим результат работы данной сверточной нейросети.

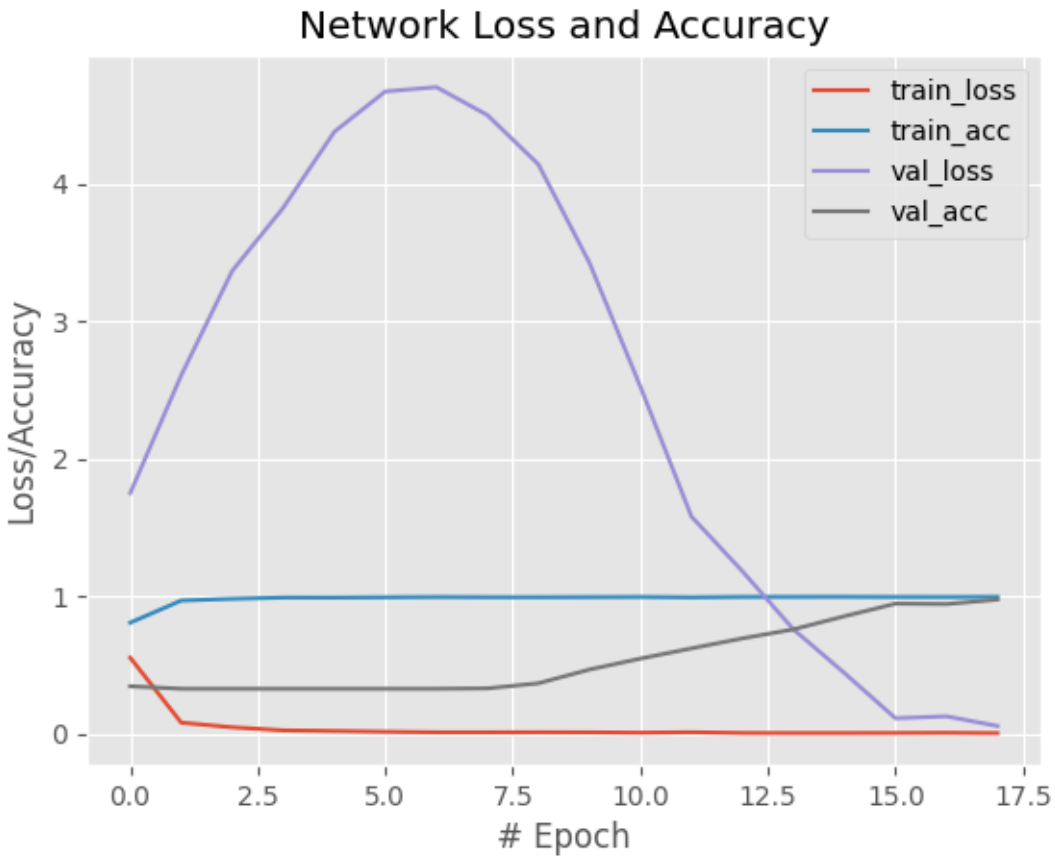
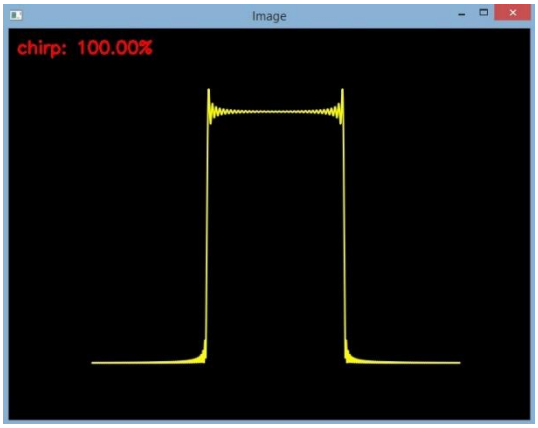


Рис.14. Графики функций потерь и точности при тренировке сверточной нейросети (оптимизатор SGD, размер батча – 64, количество эпох – 16

	precision	recall	f1-score	support
chirp	1.00	1.00	1.00	293
m-seq	1.00	0.99	0.99	290
simple	0.99	1.00	0.99	293
accuracy			1.00	876
macro avg	1.00	1.00	1.00	876
weighted avg	1.00	1.00	1.00	876

Рис.15. Метрики классификации после тренировки сверточной нейросети



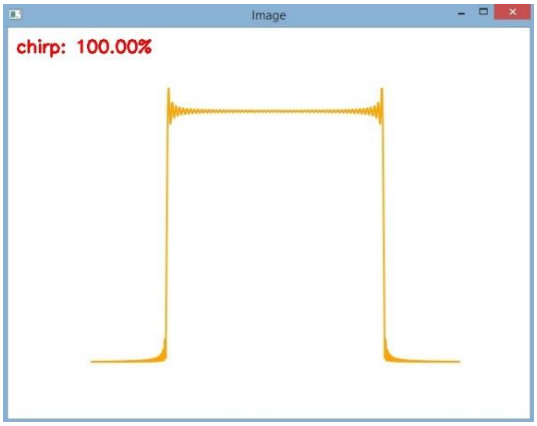
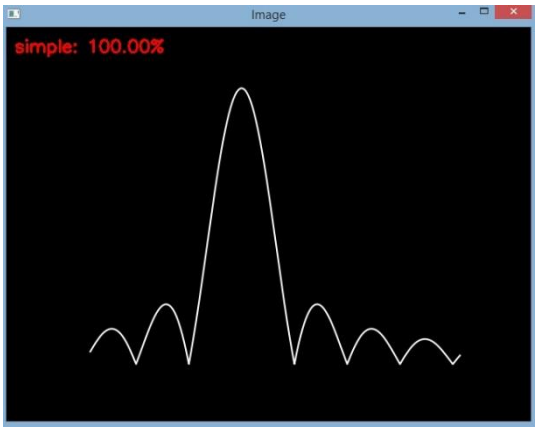


Рис. 16. Результаты распознавания ЛЧМ-сигналов



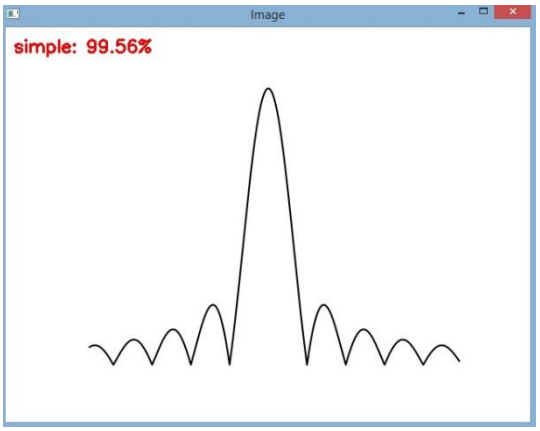
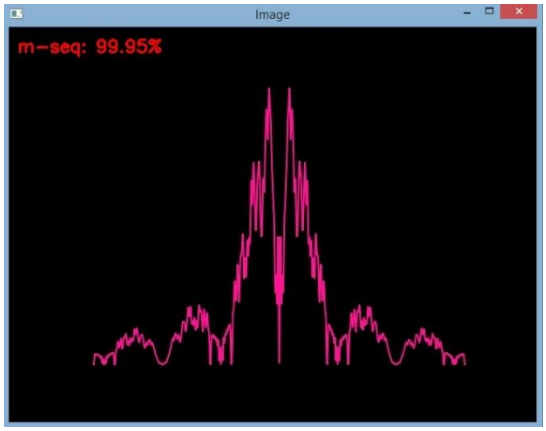


Рис. 17. Результаты распознавания простых немодулированных радиоимпульсов



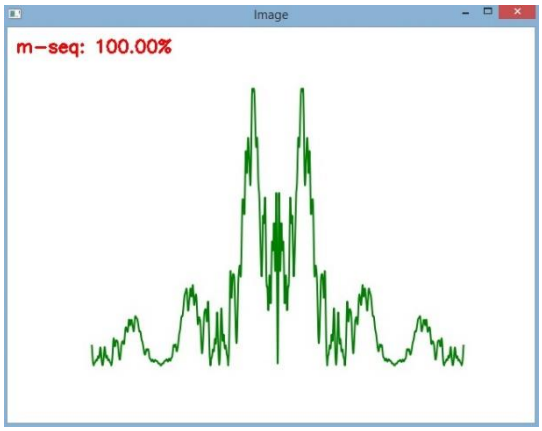


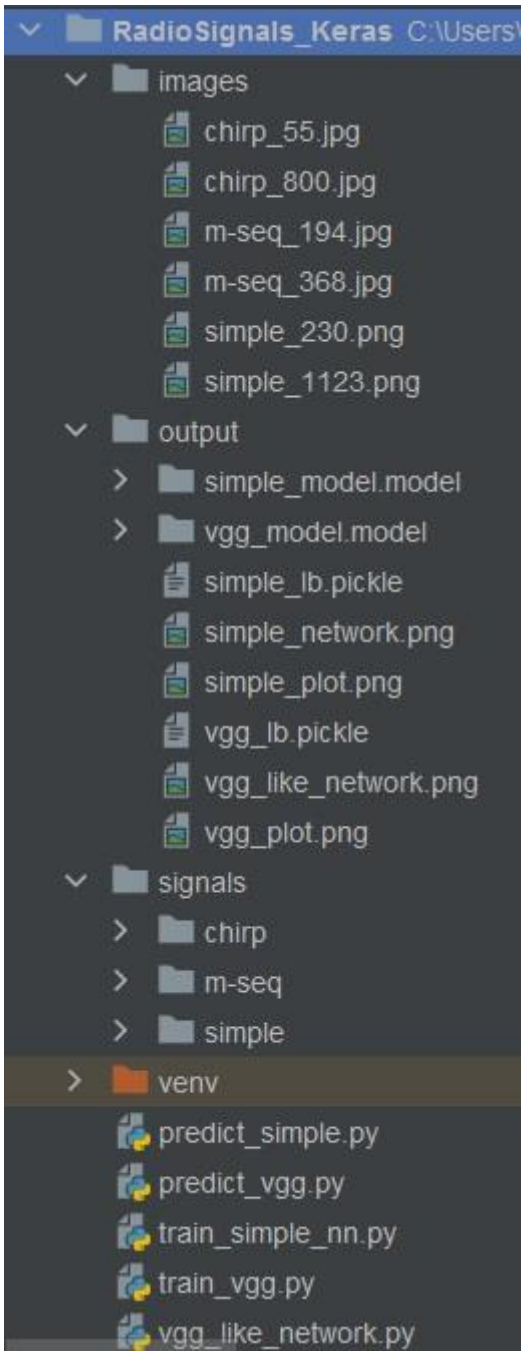
Рис. 18. Результаты распознавания ФКМ-сигналов

На это раз с помощью сверточной нейронной сети мы получили гораздо более точные результаты. Все спектры распознаются практически со 100%-ной точностью.

Таким образом, можно сделать следующий вывод: для распознавания спектров радиолокационных сигналов целесообразно использовать достаточно несложную сверточную нейронную сеть. Данная сеть дает более точные результаты по сравнению с простой последовательной нейросетью.

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
    |---simple_model.model (сериализованная модель простой сети)
    |---vgg_model.model (сериализованная модель сверточной сети)
    |---simple_lb.pickle (сериализованные метки изображений для простой сети)
    |---simple_network.png (структура простой сети)
    |---simple_plot.png (график потерь/точности распознавания для простой сети)
    |---vgg_lb.pickle (сериализованные метки изображений для сверточной сети)
    |---vgg_like_network.png (структура сверточной сети)
    I---vgg_plot.png (график потерь/точности распознавания для сверточной сети)
|
|---signals
|   |---simple (спектры простых немодулированных импульсов, 1168 изображений)
|   |---chirp (спектры ЛЧМ-импульсов, 1168 изображений)
|   I---m-seq (спектры ФКМ-импульсов, 1168 изображений)
|
|---predict_simple.py (модуль оценки точности распознавания простой нейросети)
|---predict_vgg.py (модуль оценки точности распознавания сверточной сети)
|---train_simple_nn.py (модуль тренировки простой нейросети)
|---train_vgg.py (модуль тренировки сверточной сети)
I---vgg_like_network.py (модуль построения сверточной сети)
```

Код для тренировки простой последовательной нейросети

```
# USAGE

# python train_simple_nn.py --dataset signals --model
output/simple_model.model --label-bin output/simple_lb.pickle --plot
output/simple_plot.png

import matplotlib
matplotlib.use("Agg")

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import plot_model
from imutils import paths
import argparse
import matplotlib.pyplot as plt
import numpy as np
import random
import pickle
import os
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True)
ap.add_argument("-m", "--model", required=True)
ap.add_argument("-l", "--label-bin", required=True)
ap.add_argument("-p", "--plot", required=True)
```

```
args = vars(ap.parse_args())

print("[INFO] loading images...")
data = []
labels = []

imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (32, 32)).flatten()
    data.append(image)
    label = imagePath.split(os.path.sep)[-2]
    labels.append(label)

data = np.array(data, dtype='float') / 255.0
labels = np.array(labels)

(trainX, testX, trainY, testY) = (train_test_split(data, labels,
test_size=0.25, random_state=42))

lb = LabelBinarizer()
trainY = lb. fit_transform(trainY)
testY = lb.transform(testY)

model = Sequential()
model.add(Dense(1024, input_shape=(3072,)))
model.add(Activation("relu"))
model.add(Dense(512))
model.add(Activation("relu"))
model.add(Dense(len(lb.classes_)))
model.add(Activation("softmax"))

plot_model(model, to_file="output\\simple_network.png", show_shapes=True)

INIT_LR = 0.01
EPOCHS = 29
BS = 32

print("[INFO training network...")
opt = SGD(learning_rate=INIT_LR)
model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

H = model.fit(trainX, trainY, batch_size=BS, validation_data=(testX, testY),
epochs=EPOCHS)

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=BS)
print(classification_report(testY.argmax(axis=1),
                            predictions.argmax(axis=1),
                            target_names=lb.classes_))

N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.plot(N, H.history["val_accuracy"], label="val_acc")
plt.title("Loss and Accuracy")
plt.xlabel("Epoch #")
```

```
plt.ylabel("Loss/Acc")
plt.legend()
plt.savefig(args["plot"])

print("[INFO] serializing network and label binarizer...")
model.save(args["model"])
f = open(args["label_bin"], "wb")
f.write(pickle.dumps(lb))
f.close()
```

Код для оценки точности распознавания изображений

```
# USAGE

# python predict_simple.py --image images/chirp_55.jpg --model
output/simple_model.model --label-bin output/simple_lb.pickle --width 32 --
height 32

from tensorflow.keras.models import load_model
import argparse
import pickle
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True)
ap.add_argument("-m", "--model", required=True)
ap.add_argument("-l", "--label-bin", required=True)
ap.add_argument("-w", "--width", type=int, default=28)
ap.add_argument("-e", "--height", type=int, default=28)
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
output = image.copy()
image = cv2.resize(image, (args["width"], args["height"]))
image = image.astype('float') / 255.0

image = image.flatten()
image = image.reshape((1, image.shape[0]))

model = load_model(args["model"])
lb = pickle.loads(open(args["label_bin"], "rb").read())

preds = model.predict(image)
print(preds)

i = preds.argmax(axis=1)[0]
label = lb.classes_[i]

text = "{}: {:.2f}%".format(label, preds[0][i]*100)
cv2.putText(output, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
255), 2)

cv2.imshow("Image", output)
cv2.waitKey(0)
```

ПРИЛОЖЕНИЕ 2

Код класса, определяющего структуру простой VGG-образной CNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras import backend as K


class SmallVGGNet:
    @staticmethod
    def build(width, height, depth, classes):
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1
        if K.image_data_format() == "channel_first":
            inputShape = (depth, height, width)
            chanDim = 1

        model.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.add(Dense(512))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dropout(0.5))
```

```
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        return model
```

Код для тренировки простой VGG-образной CNN

```
# USAGE

# python train_vgg.py --dataset signals --model output/vgg_model.model -# -
label-bin output/vgg_lb.pickle --plot output/vgg_plot.png

import matplotlib
matplotlib.use("Agg")

from mymodel.smallvggnet import SmallVGGNet
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import plot_model
from imutils import paths
import argparse
import matplotlib.pyplot as plt
import numpy as np
import random
import pickle
import os
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True)
ap.add_argument("-m", "--model", required=True)
ap.add_argument("-l", "--label-bin", required=True)
ap.add_argument("-p", "--plot", required=True)
args = vars(ap.parse_args())

print("[INFO] loading images...")
data = []
labels = []

imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

for imagePath in imagePaths:
   image = cv2.imread(imagePath)
   image = cv2.resize(image, (64, 64))
   data.append(image)
   label = imagePath.split(os.path.sep)[-2]
   labels.append(label)

data = np.array(data, dtype='float') / 255.0
labels = np.array(labels)

(trainX, testX, trainY, testY) = (train_test_split(data, labels, test_size=0.25, random_state=42))
```

```
lb = LabelBinarizer()
trainY = lb. fit_transform(trainY)
testY = lb.transform(testY)

model = SmallVGGNet.build(width=64, height=64, depth=3, classes=len(lb.classes_))
plot_model(model, to_file="output\\vgg_like_network.png", show_shapes=True)

INIT_LR = 0.01
EPOCHS = 14
BS = 64

print("[INFO training network...")
opt = SGD(learning_rate=INIT_LR, decay=INIT_LR/EPOCHS)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])

H = model.fit(aug.flow(trainX, trainY, batch_size=BS),
          validation_data=(testX, testY),
          steps_per_epoch=len(trainX)//BS,
          epochs=EPOCHS)

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=BS)
print(classification_report(testY.argmax(axis=1),
                    predictions.argmax(axis=1),
                    target_names=lb.classes_))

N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.plot(N, H.history["val_accuracy"], label="val_acc")
plt.title("Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Acc")
plt.legend()
plt.savefig(args["plot"])

print("[INFO] serializing network and label binarizer...")
model.save(args["model"])
f = open(args["label_bin"], "wb")
f.write(pickle.dumps(lb))
f.close()
```

Код для оценки точности распознавания изображений

```
# USAGE

# python predict.py --image images/chirp_55.jpg --model
output/vgg_model.model --label-bin output/vgg_lb.pickle --width 64 --height
64

from tensorflow.keras.models import load_model
```

```
import argparse
import pickle
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True)
ap.add_argument("-m", "--model", required=True)
ap.add_argument("-l", "--label-bin", required=True)
ap.add_argument("-w", "--width", type=int, default=28)
ap.add_argument("-e", "--height", type=int, default=28)
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
output = image.copy()
image = cv2.resize(image, (args["width"], args["height"]))
image = image.astype('float') / 255.0

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

model = load_model(args["model"])
lb = pickle.loads(open(args["label_bin"], "rb").read())

preds = model.predict(image)
print(preds)

i = preds.argmax(axis=1)[0]
label = lb.classes_[i]

text = "{}: {:.2f}%".format(label, preds[0][i]*100)
cv2.putText(output, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
255), 2)

cv2.imshow("Image", output)
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. Франсуа Шолле. Глубокое обучение на Python. – СПб.: Питер, 2018
4. Adrian Rosebrock. Deep Learning for Computer Vision with Python. – PyImageSearch, 2017
5. Ян Гудфеллоу и др. Глубокое обучение. – М.: ДМК Пресс, 2018