Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Нейронные сети в задачах технического зрения и управления

Отчёт по лабораторной работе №5

«Реализовать свёрточную сеть с использованием Keras»

Работу выполнил\_и

студент\_ка 3 курса

очного отделения

<ФИО>

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<ФИО>

**Цель работы**

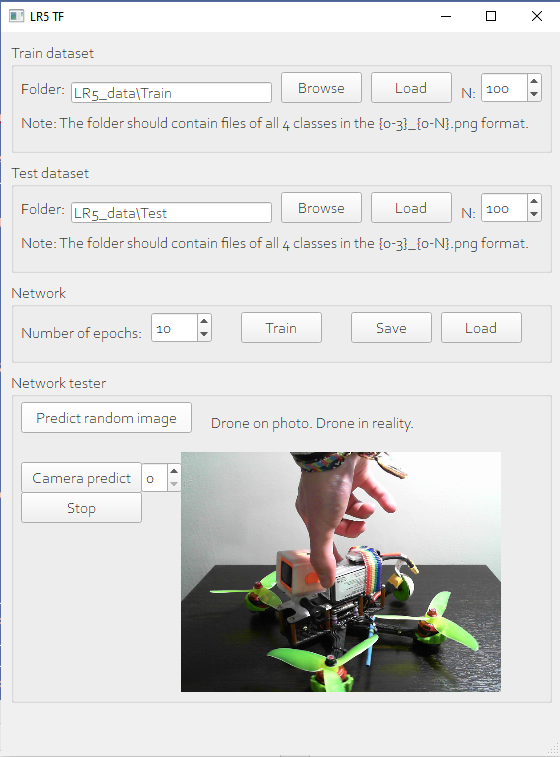
Подготовить систему распознавания изображений с использованием

языка Python и фреймворка Keras

**Задачи**

* Подготовить приложение для реализации сверточной нейронной сети SmallVGGNet (обучение сети и распознавание изображений) с графическим интерфейсом на Qt;
* Подготовить обучающую выборку по выбранной теме, содержащую не менее 3 классов со 100 изображениями каждого класса;
* Выполнить обучение сети и продемонстрировать работу сети на нескольких изображениях разных классов.

**Ход работы**



Для обучения и проверки используются изображения, сгенерированные в 4-ой лабораторной работе.

На форме имеются две группы элементов: Train dataset и Test dataset. Из выбранной папки, при помощи ImageDataGenerator производится загрузка и генерация обучающей и тестовой выборки.

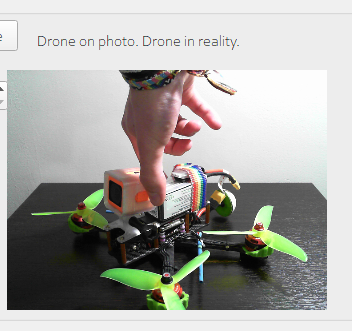
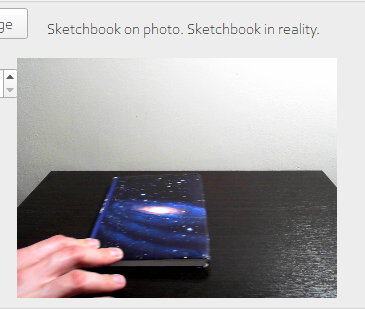
datagen = ImageDataGenerator(rescale=1 / 255.0)  
self.train\_data = datagen.flow\_from\_directory(  
 self.line\_train\_folder.text(),  
 target\_size=INPUT\_SHAPE[:2],  
 batch\_size=self.spin\_train\_n.value(),  
 class\_mode=**'categorical'**,  
 shuffle=**True**)

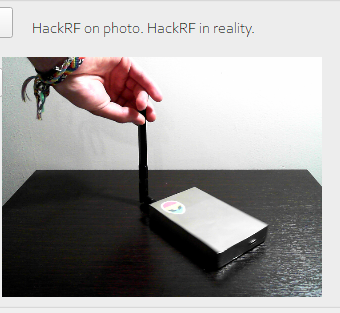
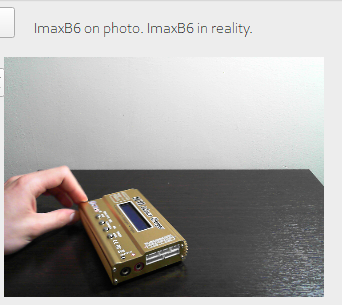
Далее, при нажатии кнопки Train производится компиляция модели, имеющей следующие слои:

self.model = tf.keras.models.Sequential([  
 layers.Conv2D(64, (6, 6), padding=**'same'**, activation=KERNEL\_ACTIVATION, input\_shape=INPUT\_SHAPE),  
 layers.Conv2D(64, (6, 6), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.MaxPooling2D((2, 2)),  
 *#* layers.Conv2D(32, (2, 2), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.Conv2D(32, (2, 2), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.MaxPooling2D((2, 2)),  
 *#* layers.Flatten(),  
 layers.Dense(128, activation=MLP\_ACTIVATION),  
 layers.Dense(self.train\_data.num\_classes, activation=**'softmax'**)  
])

Затем происходит обучение сети с заданным количеством эпох. Имеется возможность сохранения и загрузки модели в файл.

Точность предсказания 90-100%:

**Вывод**

В ходе данной работы было создано приложение на языке Python с использованием библиотек Keras, TensorFlow и PyQT5, реализующее свёрточную нейронную сеть с возможностью обучения и, способную классифицировать объекты с высокой точностью.

**Исходный код**

**import** os  
**import** threading  
  
**import** numpy **as** np  
**import** cv2  
**import** sys  
**from** PyQt5 **import** QtWidgets  
**from** PyQt5.QtGui **import** QPixmap  
**import** qimage2ndarray  
**from** keras\_preprocessing.image **import** ImageDataGenerator  
**from** tensorflow.keras **import** layers  
**import** random  
**from** tkinter **import** filedialog  
**import** tensorflow **as** tf  
**import** keras  
  
**import** gui\_5  
  
INPUT\_SHAPE = (64, 64, 3)  
PREDICT\_SHAPE = (-1, 64, 64, 3)  
KERNEL\_ACTIVATION = **'relu'**MLP\_ACTIVATION = **'relu'**OPTIMIZER = **'adam'  
  
  
class** LR5(QtWidgets.QMainWindow, gui\_5.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.btn\_train\_browse.clicked.connect(self.train\_browse)  
 self.btn\_train\_load.clicked.connect(self.train\_load)  
 self.btn\_test\_browse.clicked.connect(self.test\_browse)  
 self.btn\_test\_load.clicked.connect(self.test\_load)  
 self.btn\_tfn\_train.clicked.connect(self.tfn\_train)  
 self.btn\_tfn\_save.clicked.connect(self.tfn\_save)  
 self.btn\_tfn\_load.clicked.connect(self.tfn\_load)  
 self.btn\_tfn\_predict.clicked.connect(self.tfn\_predict)  
 self.btn\_tfn\_predict\_camera.clicked.connect(self.predict\_camera)  
 self.btn\_tfn\_predict\_camera\_stop.clicked.connect(self.predict\_camera\_stop)  
  
 self.train\_data = **None** self.test\_data = **None** self.model = **None** self.camera\_running = **False** self.cv\_cap = **None  
  
 def** train\_browse(self):  
 file\_path = filedialog.askdirectory()  
 **if** file\_path **is not None**:  
 self.line\_train\_folder.setText(file\_path)  
  
 **def** train\_load(self):  
 datagen = ImageDataGenerator(rescale=1 / 255.0)  
 self.train\_data = datagen.flow\_from\_directory(  
 self.line\_train\_folder.text(),  
 target\_size=INPUT\_SHAPE[:2],  
 batch\_size=self.spin\_train\_n.value(),  
 class\_mode=**'categorical'**,  
 shuffle=**True** )  
  
 **def** test\_browse(self):  
 file\_path = filedialog.askdirectory()  
 **if** file\_path **is not None**:  
 self.line\_test\_folder.setText(file\_path)  
  
 **def** test\_load(self):  
 datagen = ImageDataGenerator(rescale=1 / 255.0)  
 self.test\_data = datagen.flow\_from\_directory(  
 self.line\_test\_folder.text(),  
 target\_size=INPUT\_SHAPE[:2],  
 batch\_size=self.spin\_test\_n.value(),  
 class\_mode=**'categorical'** )  
  
 **def** tfn\_train(self):  
 *# Create model* self.model = tf.keras.models.Sequential([  
 layers.Conv2D(64, (6, 6), padding=**'same'**, activation=KERNEL\_ACTIVATION, input\_shape=INPUT\_SHAPE),  
 layers.Conv2D(64, (6, 6), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.MaxPooling2D((2, 2)),  
 *#* layers.Conv2D(32, (2, 2), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.Conv2D(32, (2, 2), padding=**'same'**, activation=KERNEL\_ACTIVATION),  
 layers.MaxPooling2D((2, 2)),  
 *#* layers.Flatten(),  
 layers.Dense(128, activation=MLP\_ACTIVATION),  
 layers.Dense(self.train\_data.num\_classes, activation=**'softmax'**)  
 ])  
  
 *# Compile model* self.model.compile(loss=**'categorical\_crossentropy'**,  
 optimizer=OPTIMIZER,  
 metrics=[**'accuracy'**])  
  
 *# Train model* self.model.fit(self.train\_data,  
 epochs=self.spin\_tfn\_epochs.value(),  
 validation\_data=self.test\_data)  
  
 print(**'Training done.'**)  
  
 loss, accuracy = self.model.evaluate(self.test\_data, verbose=2)  
 print(**'Accuracy: '** + str(accuracy))  
  
 **def** tfn\_save(self):  
 self.model.save(**'LR5\_data/model.h5'**)  
  
 **def** tfn\_load(self):  
 self.model = keras.models.load\_model(**'LR5\_data/model.h5'**)  
  
 **def** tfn\_predict(self):  
 class\_names = []  
 **for** folder, dirs, files **in** os.walk(self.line\_test\_folder.text()):  
 **for** directory **in** dirs:  
 class\_names.append(directory)  
  
 random\_class = random.randrange(0, self.test\_data.num\_classes)  
 test\_path = self.line\_test\_folder.text() + **'/'** + class\_names[random\_class]  
 random\_image = random.randrange(0, len(os.listdir(test\_path)))  
  
 img\_array = cv2.cvtColor(cv2.imread(  
 os.path.join(test\_path, os.listdir(test\_path)[random\_image])), cv2.COLOR\_BGR2RGB)  
  
 new\_array = cv2.resize(img\_array, INPUT\_SHAPE[:2])  
  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.resize(img\_array, (320, 240), interpolation=cv2.INTER\_NEAREST))))  
  
 new\_array = np.expand\_dims(new\_array, axis=0)  
 new\_array = np.array(new\_array).reshape(PREDICT\_SHAPE)  
 new\_array = new\_array / 255.0  
  
 result\_1 = self.model.predict([new\_array])  
 result = int(np.argmax(result\_1[0]))  
  
 print(class\_names[result] + **' / '** + class\_names[random\_class])  
 self.label\_tfn\_result.setText(class\_names[result] + **' on photo. '** + class\_names[random\_class] + **' in reality.'**)  
  
 **def** predict\_camera(self):  
 self.camera\_running = **True** self.cv\_cap = cv2.VideoCapture(self.camera\_id.value(), cv2.CAP\_DSHOW)  
 thread = threading.Thread(target=self.camera\_prediction)  
 thread.start()  
  
 **def** predict\_camera\_stop(self):  
 self.camera\_running = **False  
  
 def** camera\_prediction(self):  
 class\_names = []  
 **for** folder, dirs, files **in** os.walk(self.line\_test\_folder.text()):  
 **for** directory **in** dirs:  
 class\_names.append(directory)  
  
 **while** self.camera\_running:  
 ret, img = self.cv\_cap.read()  
 img\_array = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
  
 new\_array = cv2.resize(img\_array, INPUT\_SHAPE[:2])  
  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.resize(img\_array, (320, 240), interpolation=cv2.INTER\_NEAREST))))  
  
 new\_array = np.expand\_dims(new\_array, axis=0)  
 new\_array = np.array(new\_array).reshape(PREDICT\_SHAPE)  
 new\_array = new\_array / 255.0  
  
 result\_1 = self.model.predict([new\_array])  
 result = int(np.argmax(result\_1[0]))  
  
 self.label\_tfn\_result.setText(class\_names[result] + **' on photo.'**)  
 self.cv\_cap.release()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QtWidgets.QApplication(sys.argv)  
 app.setStyle(**'fusion'**)  
 window = LR5()  
 window.show()  
 app.exec\_()