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 №6

«Семантическое сегментирование с использованием U-Net»

Работу выполнил\_и

студент\_ка 3 курса

очного отделения

<ФИО>

Научный руководитель

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

<ФИО>

**Цель работы**

Подготовить систему семантического сегментирования изображений с использованием сети U-Net**Задачи**

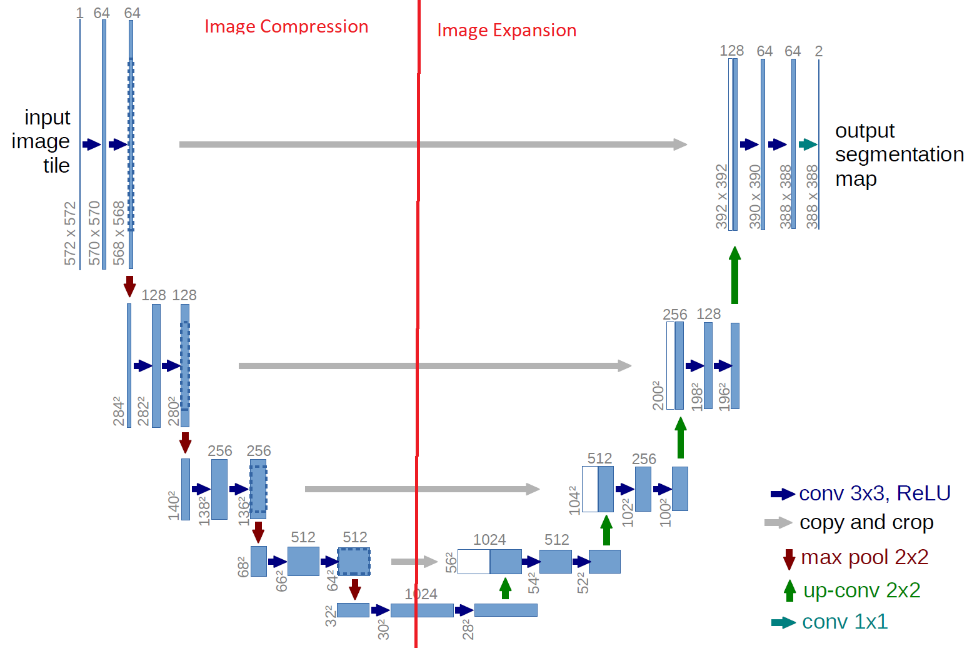
* Подготовить приложение для реализации сверточной нейронной сети U-Net (обучение сети и распознавание изображений) с графическим интерфейсом на Qt;
* Подготовить обучающую выборку по выбранной теме, содержащую не менее 50 размеченных изображений;
* Выполнить обучение сети и продемонстрировать работу сети на нескольких изображениях

**Ход работы**

Для реализации сети U-Net, были использованы библиотеки TensorFlow и Keras.

1. **Архитектура сети**

Структура сети выглядит следующим образом:

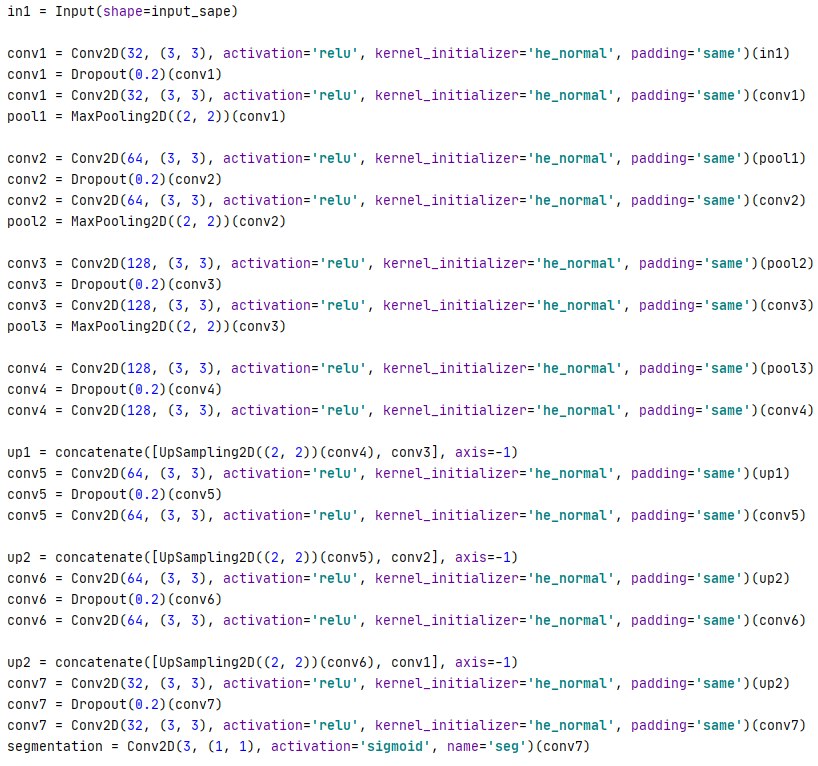


Архитектура сети выглядит как буква «U». В этой архитектуре есть две части: сжатие и расширение.

В разделе «Сжатие» есть несколько слоев свертки и max-pooling. Количество ядер или функциональных карт после каждого блока удваивается, чтобы архитектура могла изучать сложные структуры.

В разделе расширения используется похожий слой CNN и апсэмплинга. Количество блоков расширения такое же, как и количество блоков сжатия.

Сама же структура модели выглядит следующим образом:



Размер входного изображения может быть любым и задаётся через input\_shape. Также, данная структура модели поддерживает как Ч/Б, так и RGB изображение.

1. **Входное и выходное изображение**

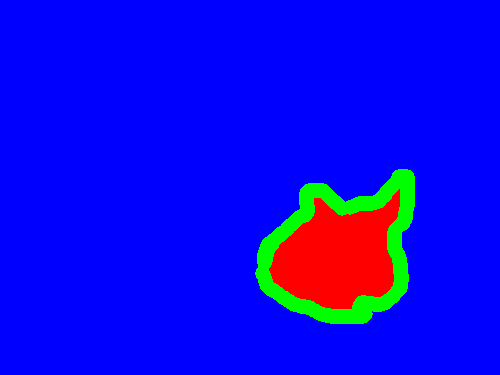
В данной работе, как входное (так и выходное) изображения имеют 3 слоя (RGB) и размер (120, 160, 3).

На вход подаётся обыкновенное RGB – изображение. Для примера – изображения животных:



Выходное же изображение – это маска, также имеющая 3 канала (RGB). Каждый канал представляется как отдельный класс для классификации.

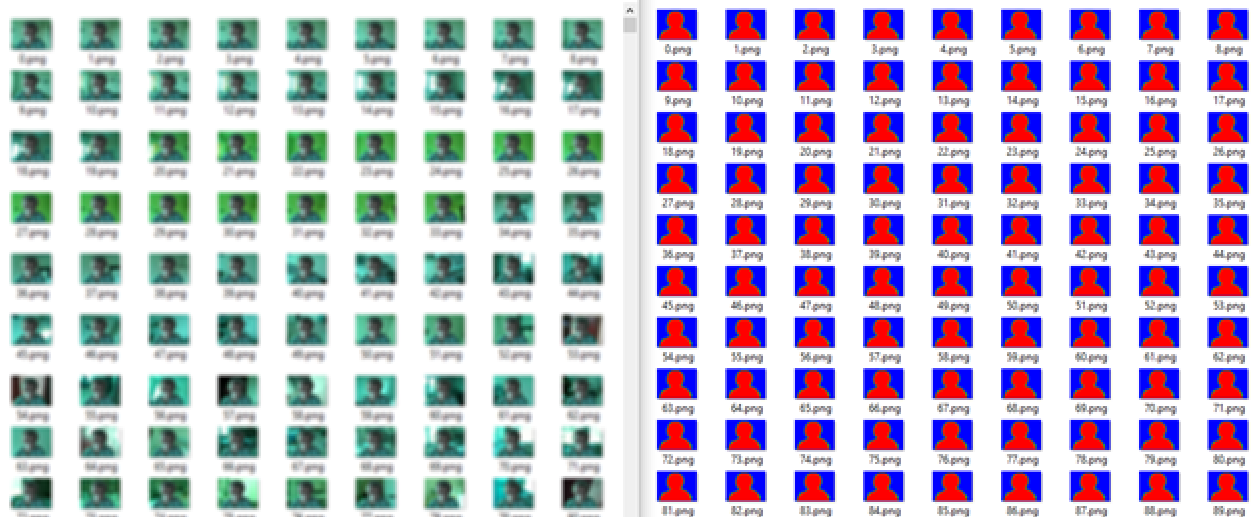
* Красный канал – объект интереса
* Зеленый канал – граница между объектом и фоном
* Синий канал – фон



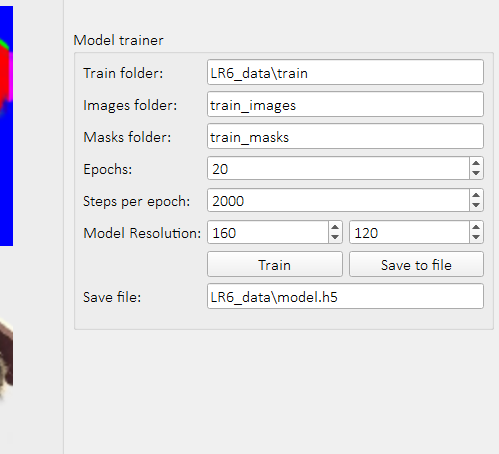
После обучения, нейросеть генерирует данную маску (производит сегментацию). По полученной маске можно, например создать «виртуальный» хромакей (вырезать объект интереса из изображения.)

1. **Обучение нейросети**

Для обучения данной нейросети использовались сделанные фотографии человека с масками на разных фона (5100шт.) – для создания «хромакея».



На форме имеется возможность выбора папок с изображениями и масками, а также количество эпох и шагов во время обучения

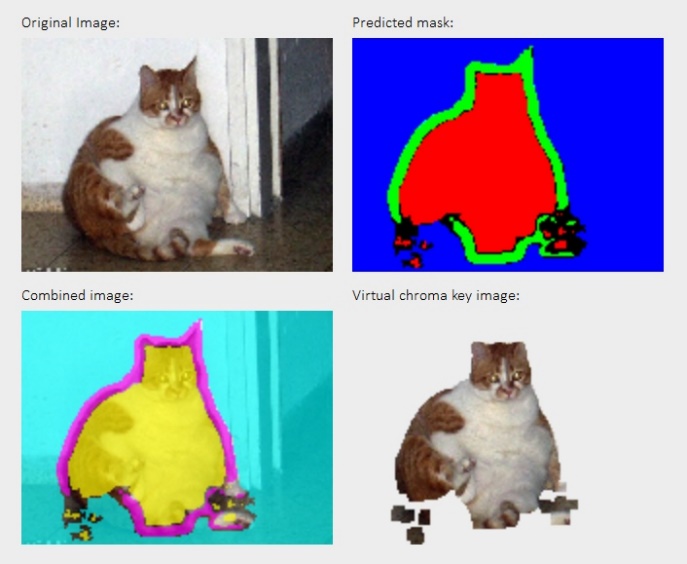


Для обучения используется генератор из библиотеки Keras, который, с определёнными аргументами берёт изображения из папок, преобразует их в требуемый диапазон и подаёт нейросети.

**def** train\_generator(self, batch\_size, train\_path, image\_folder, mask\_folder, aug\_dict, image\_color\_mode=**'grayscale'**,  
 mask\_color\_mode=**'grayscale'**, image\_save\_prefix=**'image'**, mask\_save\_prefix=**'mask'**,  
 save\_to\_dir=**None**, target\_size=(256, 256), seed=1):  
  
 image\_datagen = ImageDataGenerator(\*\*aug\_dict)  
 mask\_datagen = ImageDataGenerator(\*\*aug\_dict)  
 image\_generator = image\_datagen.flow\_from\_directory(  
 train\_path,  
 classes=[image\_folder],  
 class\_mode=**None**,  
 color\_mode=image\_color\_mode,  
 target\_size=target\_size,  
 batch\_size=batch\_size,  
 save\_to\_dir=save\_to\_dir,  
 save\_prefix=image\_save\_prefix,  
 seed=seed)  
 mask\_generator = mask\_datagen.flow\_from\_directory(  
 train\_path,  
 classes=[mask\_folder],  
 class\_mode=**None**,  
 color\_mode=mask\_color\_mode,  
 target\_size=target\_size,  
 batch\_size=batch\_size,  
 save\_to\_dir=save\_to\_dir,  
 save\_prefix=mask\_save\_prefix,  
 seed=seed)  
 train\_generator = zip(image\_generator, mask\_generator)  
 **for** (img, mask) **in** train\_generator:  
 **yield** img, mask

1. **Результат предсказания маски**

В результате работы нейросети, на выходе имеется предсказанная маска, по которой, средствами OpenCV происходит раскраска и вырезка необходимых сегментов изображения. Также, имеется возможность бинаризации каждого канала маски



**Вывод**

В ходе данной работы было создано приложение на языке Python с использованием библиотек Keras, TensorFlow и PyQT5, реализующее свёрточную нейронную сеть U-Net, способную обучаться на различных примерах. После обучения, нейросеть способна генерировать сегментирующие маски для изображения.

**Исходный код (основный скрипт)**

*"""*

*Licensed under the Unlicense License;*

*you may not use this file except in compliance with the License.*

*You may obtain a copy of the License at*

*https://unlicense.org*

*Unless required by applicable law or agreed to in writing, software*

*distributed under the License is distributed on an "AS IS" BASIS,*

*WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.*

*See the License for the specific language governing permissions and*

*limitations under the License.*

*"""***import** os

**import** random  
**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  
**import** tensorflow **as** tf  
**import** Model  
  
**import** gui\_6  
  
  
**class** LR6(QtWidgets.QMainWindow, gui\_6.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.setupUi(self)  
 self.btn\_load\_model.clicked.connect(self.load\_model)  
 self.btn\_single.clicked.connect(self.single)  
 self.btn\_camera\_start.clicked.connect(self.camera\_start)  
 self.btn\_camera\_pause.clicked.connect(self.camera\_pause)  
 self.btn\_camera\_stop.clicked.connect(self.camera\_stop)  
 self.btn\_train\_start.clicked.connect(self.train\_start)  
 self.btn\_train\_save.clicked.connect(self.train\_save)  
  
 self.camera\_running = **False** self.camera\_paused = **False** self.cv\_cap = **None** self.model = **None** self.image\_shape = **None  
  
 def** single(self):  
 random\_file = self.test\_image.text() + **'/'** + random.choice(os.listdir(self.test\_image.text()))  
 self.proceed\_frame(cv2.imread(random\_file))  
  
 **def** load\_model(self):  
 self.model = tf.keras.models.load\_model(self.model\_file.text())  
  
 **def** proceed\_frame(self, image):  
 image = cv2.resize(image, (self.model\_width.value(), self.model\_height.value()),   
 interpolation=cv2.INTER\_NEAREST)  
  
 mask = self.model.predict(np.array([image / 255]))[0]  
 mask = np.array(mask \* 255).astype(**'uint8'**)  
 mask\_b = mask[:, :, 2]  
 mask\_g = mask[:, :, 1]  
 mask\_r = mask[:, :, 0]  
 black = np.zeros((mask.shape[0], mask.shape[1]), np.uint8)  
 white = np.ones((mask.shape[0], mask.shape[1]), np.uint8) \* 255  
  
 **if** self.checkbox\_binarize.isChecked():  
 mask\_r[mask\_r >= 127] = 255  
 mask\_r[mask\_r < 127] = 0  
 mask\_g[mask\_g >= 127] = 255  
 mask\_g[mask\_g < 127] = 0  
 mask\_b[mask\_b >= 127] = 255  
 mask\_b[mask\_b < 127] = 0  
  
 result = cv2.cvtColor(image.copy(), cv2.COLOR\_BGR2BGRA) *# BGRA source image  
  
 # Create BGRA images of all 3 layers* object\_bgra = [black, white, white, np.array(mask\_r / 1.5).astype(**'uint8'**)]  
 object\_bgra = cv2.merge(object\_bgra, 4)  
 edge\_bgra = [white, black, white, np.array(mask\_g / 1.5).astype(**'uint8'**)]  
 edge\_bgra = cv2.merge(edge\_bgra, 4)  
 background\_bgra = [white, white, black, np.array(mask\_b / 1.5).astype(**'uint8'**)]  
 background\_bgra = cv2.merge(background\_bgra, 4)  
  
 *# Merge layers with source image* alpha = object\_bgra[:, :, 3] / 255.0  
 result[:, :, 0] = (1. - alpha) \* result[:, :, 0] + alpha \* object\_bgra[:, :, 0]  
 result[:, :, 1] = (1. - alpha) \* result[:, :, 1] + alpha \* object\_bgra[:, :, 1]  
 result[:, :, 2] = (1. - alpha) \* result[:, :, 2] + alpha \* object\_bgra[:, :, 2]  
 alpha = edge\_bgra[:, :, 3] / 255.0  
 result[:, :, 0] = (1. - alpha) \* result[:, :, 0] + alpha \* edge\_bgra[:, :, 0]  
 result[:, :, 1] = (1. - alpha) \* result[:, :, 1] + alpha \* edge\_bgra[:, :, 1]  
 result[:, :, 2] = (1. - alpha) \* result[:, :, 2] + alpha \* edge\_bgra[:, :, 2]  
 alpha = background\_bgra[:, :, 3] / 255.0  
 result[:, :, 0] = (1. - alpha) \* result[:, :, 0] + alpha \* background\_bgra[:, :, 0]  
 result[:, :, 1] = (1. - alpha) \* result[:, :, 1] + alpha \* background\_bgra[:, :, 1]  
 result[:, :, 2] = (1. - alpha) \* result[:, :, 2] + alpha \* background\_bgra[:, :, 2]  
  
 *# Create BGRA image with mask (virtual chromakey)* kernel = np.ones((5, 5), np.uint8)  
 b, g, r = cv2.split(image)  
 rgba = [b, g, r, cv2.dilate(mask\_r, kernel)]  
 chroma\_image = cv2.merge(rgba, 4)  
  
 self.main\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.cvtColor(cv2.resize(image, (320, 240)), cv2.COLOR\_BGR2RGB))))  
  
 self.mask\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(cv2.resize(mask, (320, 240)))))  
  
 self.combined\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.cvtColor(cv2.resize(result, (320, 240)), cv2.COLOR\_BGRA2RGBA))))  
  
 self.chroma\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.cvtColor(cv2.resize(chroma\_image, (320, 240)), cv2.COLOR\_BGRA2RGBA))))  
  
 **def** camera\_start(self):  
 **if not** self.camera\_paused:  
 self.camera\_running = **True** self.cv\_cap = cv2.VideoCapture(self.camera\_id.value(), cv2.CAP\_DSHOW)  
 thread = threading.Thread(target=self.camera\_process)  
 thread.start()  
 self.camera\_paused = **False** self.camera\_id.setEnabled(**False**)  
 self.btn\_camera\_start.setEnabled(**False**)  
 self.btn\_camera\_pause.setEnabled(**True**)  
  
 **def** camera\_pause(self):  
 self.camera\_paused = **True** self.btn\_camera\_start.setEnabled(**True**)  
 self.btn\_camera\_pause.setEnabled(**False**)  
  
 **def** camera\_process(self):  
 **while** self.camera\_running:  
 **if not** self.camera\_paused:  
 ret, img = self.cv\_cap.read()  
 *# img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)* **if** img **is not None**:  
 self.proceed\_frame(img)  
 self.cv\_cap.release()  
  
 **def** camera\_stop(self):  
 self.camera\_paused = **False** self.camera\_running = **False** self.camera\_id.setEnabled(**True**)  
 self.btn\_camera\_start.setEnabled(**True**)  
 self.btn\_camera\_pause.setEnabled(**False**)  
  
 **def** train\_start(self):  
 self.image\_shape = (self.cnn\_model\_height.value(), self.cnn\_model\_width.value(), 3)  
  
 thread = threading.Thread(target=self.training\_thread)  
 thread.start()  
  
 **def** training\_thread(self):  
 *# noinspection PyBroadException* **try**:  
 self.btn\_train\_start.setEnabled(**False**)  
 self.btn\_train\_save.setEnabled(**False**)  
  
 self.model = Model.unet(self.image\_shape)  
 data\_gen\_args = dict(rescale=1 / 255.0) *# rescale=1 / 255.0* self.model.fit(self.train\_generator(2, self.cnn\_train\_folder.text(),  
 self.cnn\_images\_folder.text(), self.cnn\_masks\_folder.text(),  
 data\_gen\_args, save\_to\_dir=**None**, image\_color\_mode=**'rgb'**,  
 mask\_color\_mode=**'rgb'**, target\_size=(self.image\_shape[0],  
 self.image\_shape[1])),  
 steps\_per\_epoch=self.cnn\_steps\_per\_epoch.value(), epochs=self.cnn\_epochs.value()) *# 24* self.btn\_train\_start.setEnabled(**True**)  
 self.btn\_train\_save.setEnabled(**True**)  
 **except**:  
 print(sys.exc\_info())  
  
 **def** train\_save(self):  
 self.model.save(self.cnn\_train\_save\_file.text())  
  
 **def** train\_generator(self, batch\_size, train\_path, image\_folder, mask\_folder, aug\_dict, image\_color\_mode=**'grayscale'**,  
 mask\_color\_mode=**'grayscale'**, image\_save\_prefix=**'image'**, mask\_save\_prefix=**'mask'**,  
 save\_to\_dir=**None**, target\_size=(256, 256), seed=1):  
  
 image\_datagen = ImageDataGenerator(\*\*aug\_dict)  
 mask\_datagen = ImageDataGenerator(\*\*aug\_dict)  
 image\_generator = image\_datagen.flow\_from\_directory(  
 train\_path,  
 classes=[image\_folder],  
 class\_mode=**None**,  
 color\_mode=image\_color\_mode,  
 target\_size=target\_size,  
 batch\_size=batch\_size,  
 save\_to\_dir=save\_to\_dir,  
 save\_prefix=image\_save\_prefix,  
 seed=seed)  
 mask\_generator = mask\_datagen.flow\_from\_directory(  
 train\_path,  
 classes=[mask\_folder],  
 class\_mode=**None**,  
 color\_mode=mask\_color\_mode,  
 target\_size=target\_size,  
 batch\_size=batch\_size,  
 save\_to\_dir=save\_to\_dir,  
 save\_prefix=mask\_save\_prefix,  
 seed=seed)  
 train\_generator = zip(image\_generator, mask\_generator)  
 **for** (img, mask) **in** train\_generator:  
 **yield** img, mask  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QtWidgets.QApplication(sys.argv)  
 app.setStyle(**"fusion"**)  
 window = LR6()  
 window.show()  
 app.exec\_()

**Исходный код (Model.py)**

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**from** keras.layers **import** \*  
**from** keras.models **import** \*  
  
  
**def** unet(input\_sape=(256, 256, 3)):  
 in1 = Input(shape=input\_sape)  
  
 conv1 = Conv2D(32, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(in1)  
 conv1 = Dropout(0.2)(conv1)  
 conv1 = Conv2D(32, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv1)  
 pool1 = MaxPooling2D((2, 2))(conv1)  
  
 conv2 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(pool1)  
 conv2 = Dropout(0.2)(conv2)  
 conv2 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv2)  
 pool2 = MaxPooling2D((2, 2))(conv2)  
  
 conv3 = Conv2D(128, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(pool2)  
 conv3 = Dropout(0.2)(conv3)  
 conv3 = Conv2D(128, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv3)  
 pool3 = MaxPooling2D((2, 2))(conv3)  
  
 conv4 = Conv2D(128, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(pool3)  
 conv4 = Dropout(0.2)(conv4)  
 conv4 = Conv2D(128, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv4)  
  
 up1 = concatenate([UpSampling2D((2, 2))(conv4), conv3], axis=-1)  
 conv5 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(up1)  
 conv5 = Dropout(0.2)(conv5)  
 conv5 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv5)  
  
 up2 = concatenate([UpSampling2D((2, 2))(conv5), conv2], axis=-1)  
 conv6 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(up2)  
 conv6 = Dropout(0.2)(conv6)  
 conv6 = Conv2D(64, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv6)  
  
 up2 = concatenate([UpSampling2D((2, 2))(conv6), conv1], axis=-1)  
 conv7 = Conv2D(32, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(up2)  
 conv7 = Dropout(0.2)(conv7)  
 conv7 = Conv2D(32, (3, 3), activation=**'relu'**, kernel\_initializer=**'he\_normal'**, padding=**'same'**)(conv7)  
 segmentation = Conv2D(3, (1, 1), activation=**'sigmoid'**, name=**'seg'**)(conv7)  
  
 model = Model(inputs=[in1], outputs=[segmentation])  
  
 losses = {**'seg'**: **'binary\_crossentropy'**}  
  
 metrics = {**'seg'**: [**'acc'**]}  
 model.compile(optimizer=**"adam"**, loss=losses, metrics=metrics)  
  
 **return** model

**Исходный код (gui\_6.py)**

*# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'gui\_6.ui'  
#  
# Created by: PyQt5 UI code generator 5.9  
#  
# WARNING! All changes made in this file will be lost!***from** PyQt5 **import** QtCore, QtGui, QtWidgets  
  
**class** Ui\_MainWindow(object):  
 **def** setupUi(self, MainWindow):  
 MainWindow.setObjectName(**"MainWindow"**)  
 MainWindow.resize(1209, 858)  
 font = QtGui.QFont()  
 font.setFamily(**"Calibri Light"**)  
 font.setPointSize(12)  
 MainWindow.setFont(font)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName(**"centralwidget"**)  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(10, 10, 521, 31))  
 font = QtGui.QFont()  
 font.setPointSize(22)  
 self.label.setFont(font)  
 self.label.setObjectName(**"label"**)  
 self.groupBox\_4 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_4.setGeometry(QtCore.QRect(10, 50, 751, 751))  
 self.groupBox\_4.setObjectName(**"groupBox\_4"**)  
 self.gridLayoutWidget\_4 = QtWidgets.QWidget(self.groupBox\_4)  
 self.gridLayoutWidget\_4.setGeometry(QtCore.QRect(340, 30, 401, 131))  
 self.gridLayoutWidget\_4.setObjectName(**"gridLayoutWidget\_4"**)  
 self.gridLayout\_4 = QtWidgets.QGridLayout(self.gridLayoutWidget\_4)  
 self.gridLayout\_4.setContentsMargins(0, 0, 0, 0)  
 self.gridLayout\_4.setObjectName(**"gridLayout\_4"**)  
 self.label\_19 = QtWidgets.QLabel(self.gridLayoutWidget\_4)  
 self.label\_19.setObjectName(**"label\_19"**)  
 self.gridLayout\_4.addWidget(self.label\_19, 0, 0, 1, 1)  
 self.label\_3 = QtWidgets.QLabel(self.gridLayoutWidget\_4)  
 self.label\_3.setObjectName(**"label\_3"**)  
 self.gridLayout\_4.addWidget(self.label\_3, 2, 0, 1, 1)  
 self.test\_image = QtWidgets.QLineEdit(self.gridLayoutWidget\_4)  
 self.test\_image.setObjectName(**"test\_image"**)  
 self.gridLayout\_4.addWidget(self.test\_image, 0, 1, 1, 1)  
 self.btn\_single = QtWidgets.QPushButton(self.gridLayoutWidget\_4)  
 self.btn\_single.setObjectName(**"btn\_single"**)  
 self.gridLayout\_4.addWidget(self.btn\_single, 1, 1, 1, 1)  
 self.gridLayout\_5 = QtWidgets.QGridLayout()  
 self.gridLayout\_5.setObjectName(**"gridLayout\_5"**)  
 self.btn\_camera\_stop = QtWidgets.QPushButton(self.gridLayoutWidget\_4)  
 self.btn\_camera\_stop.setObjectName(**"btn\_camera\_stop"**)  
 self.gridLayout\_5.addWidget(self.btn\_camera\_stop, 0, 3, 1, 1)  
 self.camera\_id = QtWidgets.QSpinBox(self.gridLayoutWidget\_4)  
 self.camera\_id.setObjectName(**"camera\_id"**)  
 self.gridLayout\_5.addWidget(self.camera\_id, 0, 0, 1, 1)  
 self.btn\_camera\_start = QtWidgets.QPushButton(self.gridLayoutWidget\_4)  
 self.btn\_camera\_start.setObjectName(**"btn\_camera\_start"**)  
 self.gridLayout\_5.addWidget(self.btn\_camera\_start, 0, 1, 1, 1)  
 self.btn\_camera\_pause = QtWidgets.QPushButton(self.gridLayoutWidget\_4)  
 self.btn\_camera\_pause.setEnabled(**False**)  
 self.btn\_camera\_pause.setObjectName(**"btn\_camera\_pause"**)  
 self.gridLayout\_5.addWidget(self.btn\_camera\_pause, 0, 2, 1, 1)  
 self.gridLayout\_4.addLayout(self.gridLayout\_5, 2, 1, 1, 1)  
 self.label\_8 = QtWidgets.QLabel(self.gridLayoutWidget\_4)  
 self.label\_8.setObjectName(**"label\_8"**)  
 self.gridLayout\_4.addWidget(self.label\_8, 3, 0, 1, 1)  
 self.checkbox\_binarize = QtWidgets.QCheckBox(self.gridLayoutWidget\_4)  
 self.checkbox\_binarize.setChecked(**True**)  
 self.checkbox\_binarize.setObjectName(**"checkbox\_binarize"**)  
 self.gridLayout\_4.addWidget(self.checkbox\_binarize, 3, 1, 1, 1)  
 self.gridLayoutWidget\_6 = QtWidgets.QWidget(self.groupBox\_4)  
 self.gridLayoutWidget\_6.setGeometry(QtCore.QRect(10, 30, 301, 131))  
 self.gridLayoutWidget\_6.setObjectName(**"gridLayoutWidget\_6"**)  
 self.gridLayout\_6 = QtWidgets.QGridLayout(self.gridLayoutWidget\_6)  
 self.gridLayout\_6.setContentsMargins(0, 0, 0, 0)  
 self.gridLayout\_6.setObjectName(**"gridLayout\_6"**)  
 self.label\_20 = QtWidgets.QLabel(self.gridLayoutWidget\_6)  
 self.label\_20.setObjectName(**"label\_20"**)  
 self.gridLayout\_6.addWidget(self.label\_20, 0, 0, 1, 1)  
 self.btn\_load\_model = QtWidgets.QPushButton(self.gridLayoutWidget\_6)  
 self.btn\_load\_model.setObjectName(**"btn\_load\_model"**)  
 self.gridLayout\_6.addWidget(self.btn\_load\_model, 2, 1, 1, 1)  
 self.model\_file = QtWidgets.QLineEdit(self.gridLayoutWidget\_6)  
 self.model\_file.setObjectName(**"model\_file"**)  
 self.gridLayout\_6.addWidget(self.model\_file, 0, 1, 1, 1)  
 self.label\_2 = QtWidgets.QLabel(self.gridLayoutWidget\_6)  
 self.label\_2.setObjectName(**"label\_2"**)  
 self.gridLayout\_6.addWidget(self.label\_2, 1, 0, 1, 1)  
 self.horizontalLayout = QtWidgets.QHBoxLayout()  
 self.horizontalLayout.setObjectName(**"horizontalLayout"**)  
 self.model\_width = QtWidgets.QSpinBox(self.gridLayoutWidget\_6)  
 self.model\_width.setMinimum(1)  
 self.model\_width.setMaximum(9999)  
 self.model\_width.setProperty(**"value"**, 160)  
 self.model\_width.setObjectName(**"model\_width"**)  
 self.horizontalLayout.addWidget(self.model\_width)  
 self.model\_height = QtWidgets.QSpinBox(self.gridLayoutWidget\_6)  
 self.model\_height.setMinimum(1)  
 self.model\_height.setMaximum(9999)  
 self.model\_height.setProperty(**"value"**, 120)  
 self.model\_height.setObjectName(**"model\_height"**)  
 self.horizontalLayout.addWidget(self.model\_height)  
 self.gridLayout\_6.addLayout(self.horizontalLayout, 1, 1, 1, 1)  
 self.verticalLayoutWidget = QtWidgets.QWidget(self.groupBox\_4)  
 self.verticalLayoutWidget.setGeometry(QtCore.QRect(40, 180, 322, 267))  
 self.verticalLayoutWidget.setObjectName(**"verticalLayoutWidget"**)  
 self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)  
 self.verticalLayout.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout.setObjectName(**"verticalLayout"**)  
 self.label\_7 = QtWidgets.QLabel(self.verticalLayoutWidget)  
 self.label\_7.setObjectName(**"label\_7"**)  
 self.verticalLayout.addWidget(self.label\_7)  
 self.main\_image = QtWidgets.QLabel(self.verticalLayoutWidget)  
 self.main\_image.setMinimumSize(QtCore.QSize(320, 240))  
 self.main\_image.setMaximumSize(QtCore.QSize(320, 240))  
 self.main\_image.setText(**""**)  
 self.main\_image.setObjectName(**"main\_image"**)  
 self.verticalLayout.addWidget(self.main\_image)  
 self.verticalLayoutWidget\_2 = QtWidgets.QWidget(self.groupBox\_4)  
 self.verticalLayoutWidget\_2.setGeometry(QtCore.QRect(380, 180, 322, 267))  
 self.verticalLayoutWidget\_2.setObjectName(**"verticalLayoutWidget\_2"**)  
 self.verticalLayout\_2 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget\_2)  
 self.verticalLayout\_2.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout\_2.setObjectName(**"verticalLayout\_2"**)  
 self.label\_9 = QtWidgets.QLabel(self.verticalLayoutWidget\_2)  
 self.label\_9.setObjectName(**"label\_9"**)  
 self.verticalLayout\_2.addWidget(self.label\_9)  
 self.mask\_image = QtWidgets.QLabel(self.verticalLayoutWidget\_2)  
 self.mask\_image.setMinimumSize(QtCore.QSize(320, 240))  
 self.mask\_image.setMaximumSize(QtCore.QSize(320, 240))  
 self.mask\_image.setText(**""**)  
 self.mask\_image.setObjectName(**"mask\_image"**)  
 self.verticalLayout\_2.addWidget(self.mask\_image)  
 self.verticalLayoutWidget\_3 = QtWidgets.QWidget(self.groupBox\_4)  
 self.verticalLayoutWidget\_3.setGeometry(QtCore.QRect(380, 460, 322, 267))  
 self.verticalLayoutWidget\_3.setObjectName(**"verticalLayoutWidget\_3"**)  
 self.verticalLayout\_3 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget\_3)  
 self.verticalLayout\_3.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout\_3.setObjectName(**"verticalLayout\_3"**)  
 self.label\_10 = QtWidgets.QLabel(self.verticalLayoutWidget\_3)  
 self.label\_10.setObjectName(**"label\_10"**)  
 self.verticalLayout\_3.addWidget(self.label\_10)  
 self.chroma\_image = QtWidgets.QLabel(self.verticalLayoutWidget\_3)  
 self.chroma\_image.setMinimumSize(QtCore.QSize(320, 240))  
 self.chroma\_image.setMaximumSize(QtCore.QSize(320, 240))  
 self.chroma\_image.setText(**""**)  
 self.chroma\_image.setObjectName(**"chroma\_image"**)  
 self.verticalLayout\_3.addWidget(self.chroma\_image)  
 self.verticalLayoutWidget\_4 = QtWidgets.QWidget(self.groupBox\_4)  
 self.verticalLayoutWidget\_4.setGeometry(QtCore.QRect(40, 460, 322, 267))  
 self.verticalLayoutWidget\_4.setObjectName(**"verticalLayoutWidget\_4"**)  
 self.verticalLayout\_4 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget\_4)  
 self.verticalLayout\_4.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout\_4.setObjectName(**"verticalLayout\_4"**)  
 self.label\_11 = QtWidgets.QLabel(self.verticalLayoutWidget\_4)  
 self.label\_11.setObjectName(**"label\_11"**)  
 self.verticalLayout\_4.addWidget(self.label\_11)  
 self.combined\_image = QtWidgets.QLabel(self.verticalLayoutWidget\_4)  
 self.combined\_image.setMinimumSize(QtCore.QSize(320, 240))  
 self.combined\_image.setMaximumSize(QtCore.QSize(320, 240))  
 self.combined\_image.setText(**""**)  
 self.combined\_image.setObjectName(**"combined\_image"**)  
 self.verticalLayout\_4.addWidget(self.combined\_image)  
 self.groupBox\_3 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_3.setGeometry(QtCore.QRect(770, 280, 421, 301))  
 self.groupBox\_3.setObjectName(**"groupBox\_3"**)  
 self.gridLayoutWidget\_3 = QtWidgets.QWidget(self.groupBox\_3)  
 self.gridLayoutWidget\_3.setGeometry(QtCore.QRect(10, 30, 401, 250))  
 self.gridLayoutWidget\_3.setObjectName(**"gridLayoutWidget\_3"**)  
 self.gridLayout\_3 = QtWidgets.QGridLayout(self.gridLayoutWidget\_3)  
 self.gridLayout\_3.setContentsMargins(0, 0, 0, 0)  
 self.gridLayout\_3.setObjectName(**"gridLayout\_3"**)  
 self.cnn\_masks\_folder = QtWidgets.QLineEdit(self.gridLayoutWidget\_3)  
 self.cnn\_masks\_folder.setObjectName(**"cnn\_masks\_folder"**)  
 self.gridLayout\_3.addWidget(self.cnn\_masks\_folder, 2, 1, 1, 1)  
 self.cnn\_train\_save\_file = QtWidgets.QLineEdit(self.gridLayoutWidget\_3)  
 self.cnn\_train\_save\_file.setObjectName(**"cnn\_train\_save\_file"**)  
 self.gridLayout\_3.addWidget(self.cnn\_train\_save\_file, 9, 1, 1, 1)  
 self.horizontalLayout\_2 = QtWidgets.QHBoxLayout()  
 self.horizontalLayout\_2.setObjectName(**"horizontalLayout\_2"**)  
 self.btn\_train\_start = QtWidgets.QPushButton(self.gridLayoutWidget\_3)  
 self.btn\_train\_start.setObjectName(**"btn\_train\_start"**)  
 self.horizontalLayout\_2.addWidget(self.btn\_train\_start)  
 self.btn\_train\_save = QtWidgets.QPushButton(self.gridLayoutWidget\_3)  
 self.btn\_train\_save.setObjectName(**"btn\_train\_save"**)  
 self.horizontalLayout\_2.addWidget(self.btn\_train\_save)  
 self.gridLayout\_3.addLayout(self.horizontalLayout\_2, 8, 1, 1, 1)  
 self.label\_21 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_21.setObjectName(**"label\_21"**)  
 self.gridLayout\_3.addWidget(self.label\_21, 2, 0, 1, 1)  
 self.label\_4 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_4.setObjectName(**"label\_4"**)  
 self.gridLayout\_3.addWidget(self.label\_4, 6, 0, 1, 1)  
 self.horizontalLayout\_4 = QtWidgets.QHBoxLayout()  
 self.horizontalLayout\_4.setObjectName(**"horizontalLayout\_4"**)  
 self.cnn\_model\_width = QtWidgets.QSpinBox(self.gridLayoutWidget\_3)  
 self.cnn\_model\_width.setMinimum(1)  
 self.cnn\_model\_width.setMaximum(9999)  
 self.cnn\_model\_width.setProperty(**"value"**, 160)  
 self.cnn\_model\_width.setObjectName(**"cnn\_model\_width"**)  
 self.horizontalLayout\_4.addWidget(self.cnn\_model\_width)  
 self.cnn\_model\_height = QtWidgets.QSpinBox(self.gridLayoutWidget\_3)  
 self.cnn\_model\_height.setMinimum(1)  
 self.cnn\_model\_height.setMaximum(9999)  
 self.cnn\_model\_height.setProperty(**"value"**, 120)  
 self.cnn\_model\_height.setObjectName(**"cnn\_model\_height"**)  
 self.horizontalLayout\_4.addWidget(self.cnn\_model\_height)  
 self.gridLayout\_3.addLayout(self.horizontalLayout\_4, 6, 1, 1, 1)  
 self.label\_5 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_5.setObjectName(**"label\_5"**)  
 self.gridLayout\_3.addWidget(self.label\_5, 4, 0, 1, 1)  
 self.label\_16 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_16.setObjectName(**"label\_16"**)  
 self.gridLayout\_3.addWidget(self.label\_16, 9, 0, 1, 1)  
 self.label\_15 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_15.setObjectName(**"label\_15"**)  
 self.gridLayout\_3.addWidget(self.label\_15, 1, 0, 1, 1)  
 self.cnn\_images\_folder = QtWidgets.QLineEdit(self.gridLayoutWidget\_3)  
 self.cnn\_images\_folder.setObjectName(**"cnn\_images\_folder"**)  
 self.gridLayout\_3.addWidget(self.cnn\_images\_folder, 1, 1, 1, 1)  
 self.cnn\_epochs = QtWidgets.QSpinBox(self.gridLayoutWidget\_3)  
 self.cnn\_epochs.setMinimum(1)  
 self.cnn\_epochs.setMaximum(9999)  
 self.cnn\_epochs.setProperty(**"value"**, 20)  
 self.cnn\_epochs.setObjectName(**"cnn\_epochs"**)  
 self.gridLayout\_3.addWidget(self.cnn\_epochs, 4, 1, 1, 1)  
 self.cnn\_train\_folder = QtWidgets.QLineEdit(self.gridLayoutWidget\_3)  
 self.cnn\_train\_folder.setObjectName(**"cnn\_train\_folder"**)  
 self.gridLayout\_3.addWidget(self.cnn\_train\_folder, 0, 1, 1, 1)  
 self.label\_17 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_17.setObjectName(**"label\_17"**)  
 self.gridLayout\_3.addWidget(self.label\_17, 0, 0, 1, 1)  
 self.label\_6 = QtWidgets.QLabel(self.gridLayoutWidget\_3)  
 self.label\_6.setObjectName(**"label\_6"**)  
 self.gridLayout\_3.addWidget(self.label\_6, 5, 0, 1, 1)  
 self.cnn\_steps\_per\_epoch = QtWidgets.QSpinBox(self.gridLayoutWidget\_3)  
 self.cnn\_steps\_per\_epoch.setMinimum(1)  
 self.cnn\_steps\_per\_epoch.setMaximum(9999)  
 self.cnn\_steps\_per\_epoch.setProperty(**"value"**, 2000)  
 self.cnn\_steps\_per\_epoch.setObjectName(**"cnn\_steps\_per\_epoch"**)  
 self.gridLayout\_3.addWidget(self.cnn\_steps\_per\_epoch, 5, 1, 1, 1)  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(MainWindow)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 1209, 21))  
 self.menubar.setObjectName(**"menubar"**)  
 MainWindow.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName(**"statusbar"**)  
 MainWindow.setStatusBar(self.statusbar)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 **def** retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate(**"MainWindow"**, **"LR6 U-Net"**))  
 self.label.setText(\_translate(**"MainWindow"**, **"U-Net CNN"**))  
 self.groupBox\_4.setTitle(\_translate(**"MainWindow"**, **"U-Net"**))  
 self.label\_19.setText(\_translate(**"MainWindow"**, **"Single image Folder:"**))  
 self.label\_3.setText(\_translate(**"MainWindow"**, **"Camera:"**))  
 self.test\_image.setText(\_translate(**"MainWindow"**, **"LR6\_data\\test"**))  
 self.btn\_single.setText(\_translate(**"MainWindow"**, **"Proceed random image"**))  
 self.btn\_camera\_stop.setText(\_translate(**"MainWindow"**, **"Stop"**))  
 self.btn\_camera\_start.setText(\_translate(**"MainWindow"**, **"Start"**))  
 self.btn\_camera\_pause.setText(\_translate(**"MainWindow"**, **"Pause"**))  
 self.label\_8.setText(\_translate(**"MainWindow"**, **"Binarize mask:"**))  
 self.checkbox\_binarize.setText(\_translate(**"MainWindow"**, **"Enable binarization"**))  
 self.label\_20.setText(\_translate(**"MainWindow"**, **"Model file:"**))  
 self.btn\_load\_model.setText(\_translate(**"MainWindow"**, **"Load model"**))  
 self.model\_file.setText(\_translate(**"MainWindow"**, **"LR6\_data\\model.h5"**))  
 self.label\_2.setText(\_translate(**"MainWindow"**, **"Model Resolution:"**))  
 self.label\_7.setText(\_translate(**"MainWindow"**, **"Original Image:"**))  
 self.label\_9.setText(\_translate(**"MainWindow"**, **"Predicted mask:"**))  
 self.label\_10.setText(\_translate(**"MainWindow"**, **"Virtual chroma key image:"**))  
 self.label\_11.setText(\_translate(**"MainWindow"**, **"Combined image:"**))  
 self.groupBox\_3.setTitle(\_translate(**"MainWindow"**, **"Model trainer"**))  
 self.cnn\_masks\_folder.setText(\_translate(**"MainWindow"**, **"train\_masks"**))  
 self.cnn\_train\_save\_file.setText(\_translate(**"MainWindow"**, **"LR6\_data\\model.h5"**))  
 self.btn\_train\_start.setText(\_translate(**"MainWindow"**, **"Train"**))  
 self.btn\_train\_save.setText(\_translate(**"MainWindow"**, **"Save to file"**))  
 self.label\_21.setText(\_translate(**"MainWindow"**, **"Masks folder:"**))  
 self.label\_4.setText(\_translate(**"MainWindow"**, **"Model Resolution:"**))  
 self.label\_5.setText(\_translate(**"MainWindow"**, **"Epochs:"**))  
 self.label\_16.setText(\_translate(**"MainWindow"**, **"Save file:"**))  
 self.label\_15.setText(\_translate(**"MainWindow"**, **"Images folder:"**))  
 self.cnn\_images\_folder.setText(\_translate(**"MainWindow"**, **"train\_images"**))  
 self.cnn\_train\_folder.setText(\_translate(**"MainWindow"**, **"LR6\_data\\train"**))  
 self.label\_17.setText(\_translate(**"MainWindow"**, **"Train folder:"**))  
 self.label\_6.setText(\_translate(**"MainWindow"**, **"Steps per epoch:"**))