# ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

# Програмні засоби розпізнавання об’єктів на зображеннях з використанням машинного навчання

Кваліфікаційна робота

Перший (бакалаврський) рівень

Автор:

Олексій ЧЕРНИХ,

студ. гр. КІУКІз-21-1

Керівник:

ас. каф. Анастасія ЛУЦЕНКО

## Мета і задачі роботи

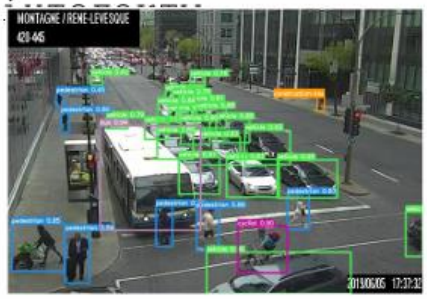
Мета: розробити програмний засіб для розпізнавання об'єктів на зображеннях за допомогою CNN, створеної на мові програмування Python.

Задачі:

- Проаналізувати існуючі рішення, виділити їх переваги та недоліки;
- Спроектувати та натренувати модель нейромережі CNN;
- Створити програму, що буде взаємодіяти з навченою моделлю та виводити результати розпізнавання;
- Узагальнити результати роботи програми та сформувати висновки щодо її ефективності.

# Актуальність теми

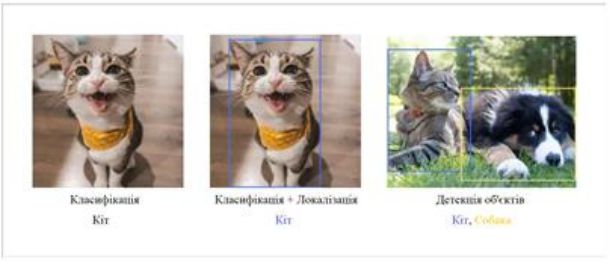
- Широке використання комп'ютерного зору в різних галузях: безпека, медицина, автономні транспортні засоби, промисловість, тощо;
- Стрімкий розвиток технологій штучного інтелекту.




3

# Задачі комп’ютерного зору

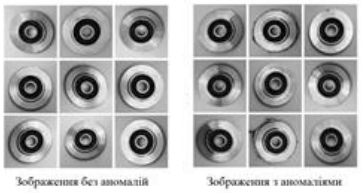
- Класифікація: визначення класу (категорії), до якої належить усе зображення;
- Локалізація: знаходження положення об'єкта на зображенні в конкретному місці;
- Детекція: поєднує в собі класифікацію та локалізацію та визначає декілька різних категорій одночасно.




4

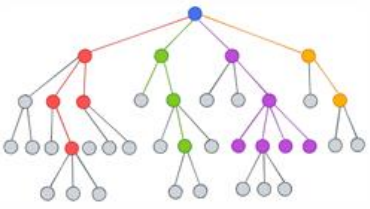
# Задачі комп'ютерного

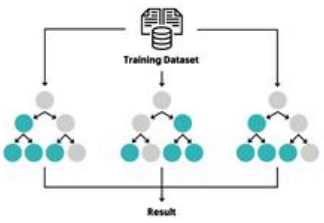
- Відстеження об'єктів (трекінг): визначення та постійне оновлення місцезнаходження об'єкту на відео у реальному часі;
- Виявлення аномалій: визначення у об'єкта відхилень від норми, наприклад, контроль якості продукту на виробництві;
- Розпізнавання облич: виявлення обличчя людини та ідентифікація особи, завдяки аналізу закономірностей в рисах обличчя.

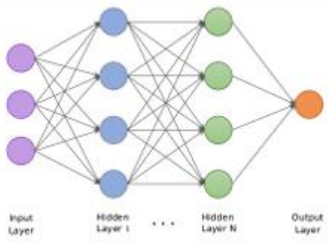



# Методи машинного навчання у розпізнаванні об'єктів

- Традиційні підходи: Decision Trees, Random Forest, SVM;
- Глибинні підходи: CNN, YOLO, ViT.

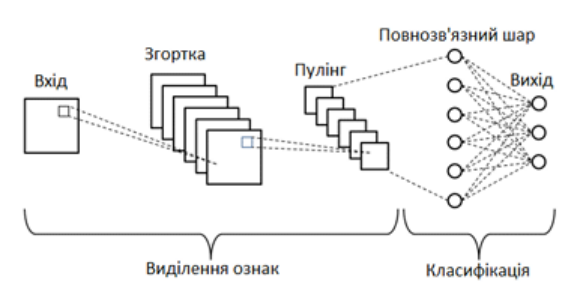




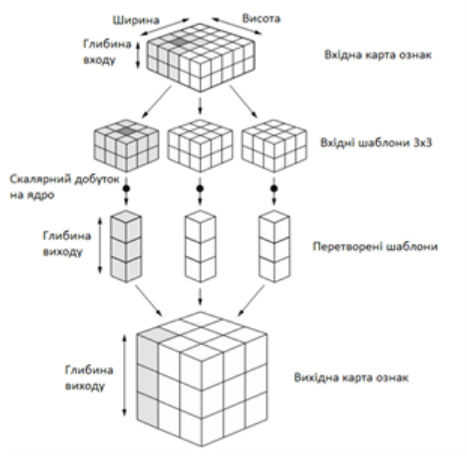
# Згорткові нейронні мережі (CNN)

- Основні шари:
- Згортка (Convolution);
- Субдискретизація (Pooling)
- Повнозв’язний шар (Fully Connected, FC)




7

## Шар згортки (Convolution)
## Шар об’єднання (Pooling)





8

# Програмні рішення

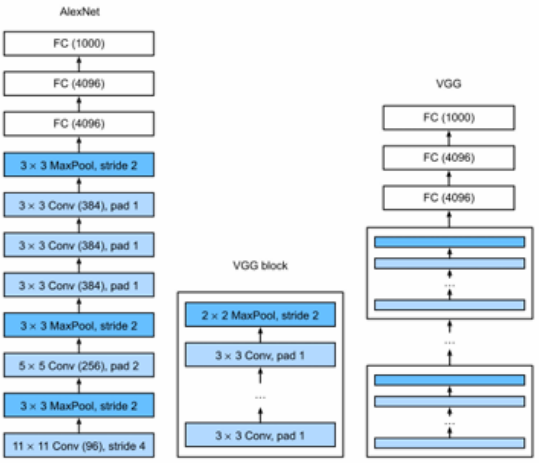





9

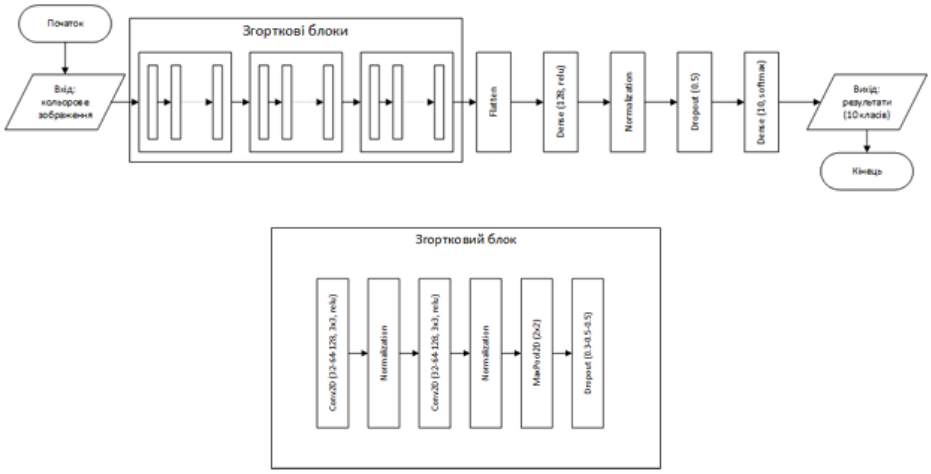
# Архітектурні існуючі рішення

- AlexNet;
- VGGNet;
- ResNet;
- MobileNet;
- Та інші.



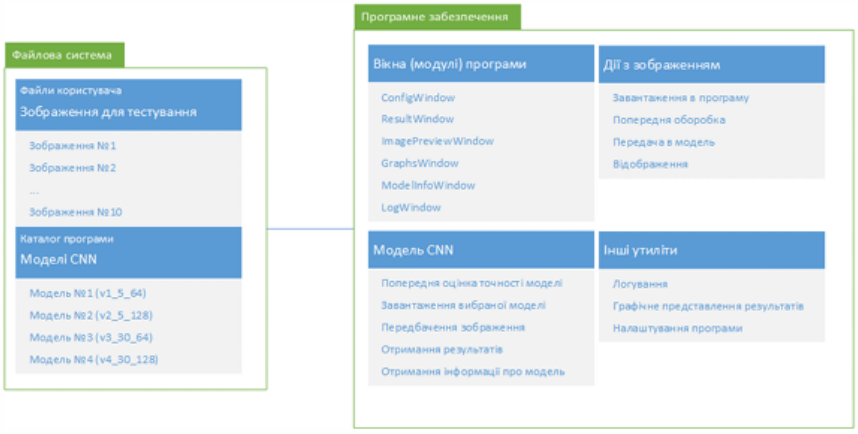

10

# Архітектура моделі нейромережі, що розробляється



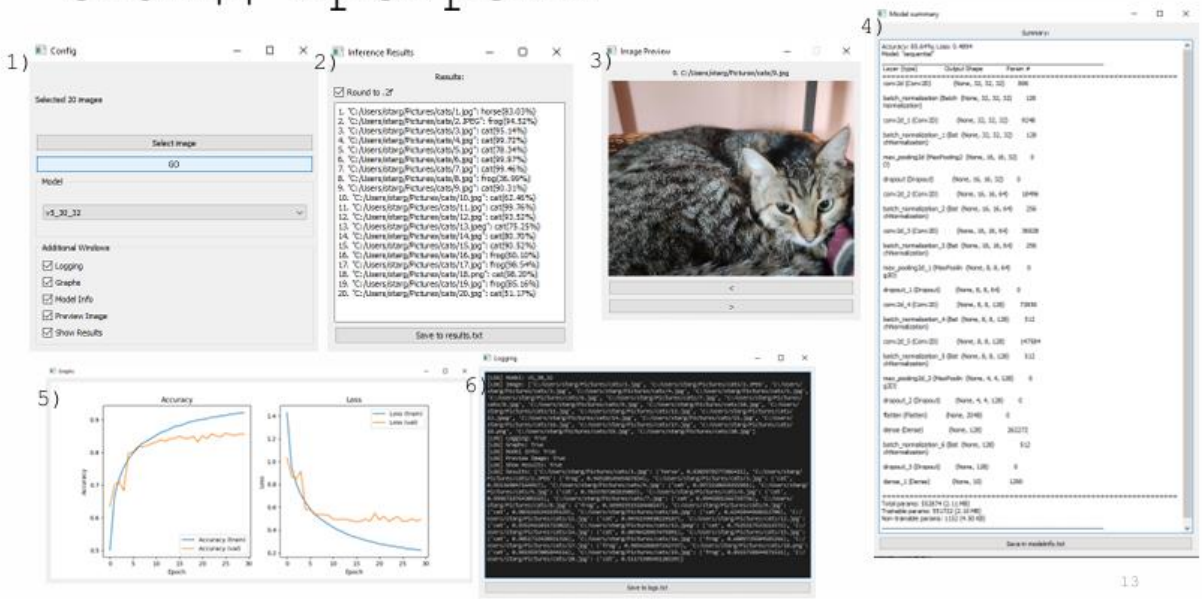

11

# Структура програми розпізнавання




12

# Огляд програми




13

# Результати розпізнавання

Таблиця 4.1 – Вірогідності розпізнавання зображень котів різних моделей

| Зображення, № | Вірогідності розпізнавання, % | |
|---|---|---|
| | V1_10_32 | V2_30_32 |
| 1 | 55.45 | 81.38 |
| 2 | 10.27 | 24.23 |
| 3 | 95.83 | 88.40 |
| 4 | 97.35 | 99.87 |
| 5 | 16.57 | 99.72 |
| 6 | 99.10 | 99.97 |
| 7 | 98.61 | 99.46 |
| 8 | 39.72 | 56.32 |
| … | … | … |
| 20 | 18.93 | 51.17 |
| Середнє знач. | 47.84 | 71.22 |

14

# ДОДАТОК Б
# КОД ПРОГРАМИ

## Б.1 Сценарій train.py

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras import utils
import pickle
import numpy as np


(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.cifar10.load_data()

y_train = y_train.flatten()
y_test = y_test.flatten()

input_shape = (32, 32, 3)

x_train=x_train.reshape(x_train.shape[0], x_train.shape[1],
x_train.shape[2], 3)
x_train=x_train / 255.0
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1],
x_test.shape[2], 3)
x_test=x_test / 255.0

y_train = tf.one_hot(y_train.astype(np.int32), depth=10)
y_test = tf.one_hot(y_test.astype(np.int32), depth=10)

model_version = 6
batch_size = 32
num_classes = 10
epochs = 1

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same',
input_shape=x_train.shape[1:], activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
```

```
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(num_classes, activation='softmax'),
])

model.compile(optimizer='adam',
loss=tf.keras.losses.categorical_crossentropy,
metrics=['accuracy'])

history = model.fit(x_train, y_train, batch_size=batch_size,
                    epochs=epochs, validation_data=(x_test,
y_test))

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f"\nTest Accuracy: {test_acc:.4f}")
print(f"\nTest Loss: {test_loss:.4f}")

model_name = f"v{model_version}_{epochs}_{batch_size}"

model.save(f"{model_name}.keras")
print(f"\nModel saved: '{model_name}.keras'")

with open(f"training_history_{model_name}.pkl", 'wb') as f:
    pickle.dump(history.history, f)

print(f"\nHistory saved: 'training_history_{model_name}.pkl'")
```

## Б.2 Сценарій main.py

```
import sys
from qt_imports import *
from config_window import ConfigWindow
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ConfigWindow()
    window.show()
    sys.exit(app.exec_())
```

Б.3 Сценарій qt_imports.py

```
from PyQt5.QtWidgets import (
    QApplication,
    QMainWindow,
    QWidget,
    QPushButton,
    QLabel,
    QVBoxLayout,
    QHBoxLayout,
    QLineEdit,
    QTextEdit,
    QMessageBox,
    QSpinBox,
    QProgressBar,
    QFileDialog,
    QCheckBox,
    QComboBox,
    QGroupBox,
)

from PyQt5.QtCore import Qt, QTimer, QThread, pyqtSignal
from PyQt5.QtGui import QFont, QPixmap

import sys
```

Б.1 Сценарій config_window.py

```
import sys
import os

os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

import contextlib
import io
import pickle

from qt_imports import *

from results_window import ResultsWindow
```

```
from image_preview_window import PreviewWindow
from graphs_window import GraphsWindow
from logs_window import LogsWindow
from model_info_window import ModelInfoWindow

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from tensorflow.keras import utils
from tensorflow.keras.losses import CategoricalCrossentropy
import numpy as np

def load_model(model_name='v1_5_ep'):
    return tf.keras.models.load_model(f"{model_name}.keras")

class ConfigWindow(QMainWindow):
    results_window = None
    preview_window = None
    graphs_window = None
    logs_window = None
    model_info_window = None

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Config")
        self.setGeometry(100, 100, 400, 400)

        self.selected_image_path = None
        self.model = None
        self.class_names = [
            "airplane", "automobile", "bird", "cat", "deer",
            "dog", "frog", "horse", "ship", "truck"
        ]
        self.init_ui()

    def init_ui(self):
        main_layout = QVBoxLayout()

        self.img_label = QLabel("[Image not selected]")
        pick_img_btn = QPushButton("Select image")
        pick_img_btn.clicked.connect(self.pick_image)

        self.start_btn = QPushButton("GO")
        self.start_btn.clicked.connect(self.start_prediction)

        model_group = QGroupBox("Model")
        self.model_combo = QComboBox()
        self.model_combo.addItems([
            "v1_10_64", "v1_10_128", "v1_30_64",
            "v1_30_128", "v1_30_64_old", "v2_50_32",
            "v2_10_32", "v3_10_32", "v3_50_32",
            "v4_30_32", "v5_10_32", "v5_30_32"])
        model_layout = QVBoxLayout()
```

```
        model_layout.addWidget(self.model_combo)
        model_group.setLayout(model_layout)

        # Additional windows (checkboxes)
        self.logs_cb = QCheckBox("Logging")
        self.graphs_cb = QCheckBox("Graphs")
        self.model_info_cb = QCheckBox("Model Info")
        self.preview_cb = QCheckBox("Preview Image")
        self.results_cb = QCheckBox("Show Results")

        # Grouping checkboxes
        windows_group = QGroupBox("Additional Windows")
        cb_layout = QVBoxLayout()
        for cb in [self.logs_cb, self.graphs_cb,
self.model_info_cb, self.preview_cb, self.results_cb]:
            cb_layout.addWidget(cb)
        windows_group.setLayout(cb_layout)

        # Layouts
        main_layout.addWidget(self.img_label)
        main_layout.addWidget(pick_img_btn)
        main_layout.addWidget(self.start_btn)
        main_layout.addWidget(model_group)
        main_layout.addWidget(windows_group)

        central_widget = QWidget()
        central_widget.setLayout(main_layout)
        self.setCentralWidget(central_widget)

    def pick_image(self):
        file_dialog = QFileDialog()
        path, _ = file_dialog.getOpenFileNames(self, "Select
Images", "", "Images (*.png *.jpg *.jpeg *.bmp)")
        if path:
            print(f"Selected {len(path)} images:\n")
            for i in path:
                print(i)
            self.selected_image_path = path
            self.img_label.setText(f"Selected {len(path)}
images")

    def start_prediction(self):
        model_name = self.model_combo.currentText()
        logging = self.logs_cb.isChecked()
        graphs = self.graphs_cb.isChecked()
        model_info = self.model_info_cb.isChecked()
        preview = self.preview_cb.isChecked()
        results = self.results_cb.isChecked()

        print("-- Start Arguments --")
        print("Model:", model_name)
        print("Image:", self.selected_image_path)
        print("Logging:", logging)
```

```
        print("Graphs:", graphs)
        print("Model Info:", model_info)
        print("Preview Image:", preview)
        print("Show Results:", results)

        self.model = load_model(model_name)

        (_, _), (x_test, y_test) =
tf.keras.datasets.cifar10.load_data()
        y_test = y_test.flatten()
        x_test = x_test.reshape(x_test.shape[0],
x_test.shape[1], x_test.shape[2], 3)
        x_test = x_test / 255.0
        y_test = tf.one_hot(y_test.astype(np.int32), depth=10)

        loss, acc = self.model.evaluate(x_test, y_test,
verbose=2)
        model_test_text = f"Accuracy: {acc * 100:.2f}%; Loss:
{loss:.4f}"

        model_summary = self.get_model_summary_text()

        try:
            with open(f'training_history_{model_name}.pkl',
'rb') as f:
                loaded_history = pickle.load(f)

        except Exception as e:
            print(f"Error while trying to open file: {e}")


        model_results =
self.classify_image(self.selected_image_path)
        print(model_results)
        logs = [model_name, self.selected_image_path, logging,
graphs, model_info, preview, results, model_results]

        if results:
            self.handle_show_results(model_results)

        if preview:
            self.handle_show_preview(self.selected_image_path)

        if graphs:
            self.handle_show_graphs(loaded_history)

        if logging:
            self.handle_show_logs(logs)

        if model_info:
            self.handle_show_model_info(model_test_text,
model_summary)
```

```
    def get_model_summary_text(self):
        stream = io.StringIO()
        with contextlib.redirect_stdout(stream):
            self.model.summary()
        return stream.getvalue()

    def handle_show_results(self, prediction_result):
        self.results_window = ResultsWindow(prediction_result)
        self.results_window.show()

    def handle_show_preview(self, image_path):
        self.preview_window = PreviewWindow(image_path)
        self.preview_window.show()

    def handle_show_graphs(self, history):
        self.graphs_window = GraphsWindow(history)
        self.graphs_window.show()

    def handle_show_logs(self, logs):
        self.logs_window = LogsWindow()
        self.logs_window.show()
        self.logs_window.log(f"Model: {logs[0]}")
        self.logs_window.log(f"Image: {logs[1]}")
        self.logs_window.log(f"Logging: {logs[2]}")
        self.logs_window.log(f"Graphs: {logs[3]}")
        self.logs_window.log(f"Model Info: {logs[4]}")
        self.logs_window.log(f"Preview Image: {logs[5]}")
        self.logs_window.log(f"Show Results: {logs[6]}")
        self.logs_window.log(f"Results: {logs[7]}")

    def handle_show_model_info(self, model_test_text,
model_summary):
        self.model_info_window =
ModelInfoWindow(model_test_text, model_summary)
        self.model_info_window.show()

    def classify_image(self, file_paths, true_label_index=3):
        imgs = []
        for path in file_paths:
            img = image.load_img(path, target_size=(32, 32))
            img_array = image.img_to_array(img) / 255.0
            imgs.append(img_array)

        predictions = self.model.predict(np.array(imgs))

        res_dict = dict()
        res_sum = 0.0
        cnt = 0
        for path, pred in zip(file_paths, predictions):
            class_idx = np.argmax(pred)
            print(f"cat: {float(pred[true_label_index]) *
100:.2f}")
            res_sum += float(pred[true_label_index]) * 100
```

```
            cnt += 1
            probability = float(np.max(pred))
            class_name = self.class_names[class_idx]
            res_dict[path] = (class_name, probability)

        print(res_sum / cnt)
        print(res_dict)
        return res_dict
```

## Б.1 Сценарій results_window.py

```
from qt_imports import *

class ResultsWindow(QWidget):
    def __init__(self, prediction_result, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Inference Results")
        self.resize(300, 300)

        layout = QVBoxLayout()

        self.label = QLabel("Results:")
        self.label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.label)

        self.checkboxfull = QCheckBox("Round to .2f")
        self.checkboxfull.setChecked(True)

self.checkboxfull.stateChanged.connect(self.handle_state_change)
        layout.addWidget(self.checkboxfull)

        self.result_box = QTextEdit()
        self.result_box.setReadOnly(True)

        self.result_text_rounded = ""
        self.result_text_full = ""

        for i, (key, value) in
enumerate(prediction_result.items(), 1):
            self.result_text_full += f'{i}. "{key}":
{value[0]}({value[1] * 100}%)\n'

        for i, (key, value) in
enumerate(prediction_result.items(), 1):
            self.result_text_rounded += f'{i}. "{key}":
{value[0]}({value[1] * 100:.2f}%)\n'
        self.result_box.setText(self.result_text_rounded)

        layout.addWidget(self.result_box)
```

```
        self.save_button = QPushButton("Save to results.txt")
        self.save_button.clicked.connect(self.save_to_file)
        layout.addWidget(self.save_button)

        self.setLayout(layout)

    def handle_state_change(self, state):
        if state == 2:
            self.result_box.setText(self.result_text_rounded)
        elif state == 0:
            self.result_box.setText(self.result_text_full)
        else:
            pass

    def save_to_file(self):
        try:
            with open("results.txt", "w", encoding="utf-8") as
f:
                f.write(self.result_box.toPlainText())
        except Exception as e:
            print(f"Error while saving file: {e}")
```

## Б.1 Сценарій image_preview_window.py

```
from qt_imports import *
import os

class PreviewWindow(QWidget):
    def __init__(self, image_path, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Image Preview")
        self.setFixedSize(400, 400)

        self.image_path = image_path
        self.image_index = 0

        layout = QVBoxLayout()

        self.path_label = QLabel()
        self.path_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.path_label)

        self.image_label = QLabel()
        self.image_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.image_label)

        self.button_prev = QPushButton("<")
        self.button_next = QPushButton(">")
        self.button_prev.clicked.connect(self.prev_image)
        self.button_next.clicked.connect(self.next_image)
```

```
        layout.addWidget(self.button_prev)
        layout.addWidget(self.button_next)

        self.setLayout(layout)
        self.set_image(image_path, self.image_index)

    def prev_image(self):
        if self.image_index > 0:
            self.image_index = self.image_index - 1

        self.set_image(self.image_path, self.image_index)

    def next_image(self):
        if self.image_index < len(self.image_path)-1:
            self.image_index = self.image_index + 1

        self.set_image(self.image_path, self.image_index)

    def set_image(self, image_path, image_index):
        if image_path and
os.path.exists(image_path[image_index]):
            pixmap = QPixmap(image_path[image_index])
            scaled_pixmap = pixmap.scaled(self.size(),
Qt.KeepAspectRatio, Qt.SmoothTransformation)
            self.image_label.setPixmap(scaled_pixmap)
            self.path_label.setText(f"{image_index+1}.
{image_path[image_index]}")
        else:
            self.image_label.setText("Image not found 404")
```

## Б.1 Сценарій graphs_window.py

```
from qt_imports import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
as FigureCanvas
from matplotlib.figure import Figure

class GraphsWindow(QWidget):
    def __init__(self, training_history=None, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Graphs")
        self.resize(600, 400)

        self.figure = Figure()
        self.canvas = FigureCanvas(self.figure)

        layout = QVBoxLayout()
        layout.addWidget(self.canvas)
        self.setLayout(layout)
```

```
        if training_history:
            self.plot_history(training_history)

    def plot_history(self, history):
        self.figure.clear()
        ax1 = self.figure.add_subplot(1, 2, 1)
        ax2 = self.figure.add_subplot(1, 2, 2)

        ax1.plot(history['accuracy'], label='Accuracy (train)')
        if 'val_accuracy' in history:
            ax1.plot(history['val_accuracy'], label='Accuracy
(val)')
        ax1.set_title('Accuracy')
        ax1.set_xlabel('Epoch')
        ax1.set_ylabel('Accuracy')
        ax1.legend()

        ax2.plot(history['loss'], label='Loss (train)')
        if 'val_loss' in history:
            ax2.plot(history['val_loss'], label='Loss (val)')
        ax2.set_title('Loss')
        ax2.set_xlabel('Epoch')
        ax2.set_ylabel('Loss')
        ax2.legend()

        self.canvas.draw()
```

## Б.1 Сценарій logs_window.py

```
from qt_imports import *

class LogsWindow(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Logging")
        self.resize(600, 300)

        self.text_edit = QTextEdit()
        self.text_edit.setReadOnly(True)
        self.text_edit.setStyleSheet("background-color: #1e1e1e;
color: #d4d4d4; font-family: Consolas;")

        layout = QVBoxLayout()
        layout.addWidget(self.text_edit)

        self.save_button = QPushButton("Save to logs.txt")
        self.save_button.clicked.connect(self.save_to_file)
        layout.addWidget(self.save_button)

        self.setLayout(layout)
```

```
    def log(self, message: str):
        self.text_edit.append(f"[LOG] {message}")

self.text_edit.verticalScrollBar().setValue(self.text_edit.verti
calScrollBar().maximum())

    def log_warning(self, message: str):
        self.text_edit.append(f"[WARNING] {message}")

    def log_error(self, message: str):
        self.text_edit.append(f"[ERROR] {message}")

    def clear_logs(self):
        self.text_edit.clear()

    def save_to_file(self):
        try:
            with open("logs.txt", "w", encoding="utf-8") as f:
                f.write(self.text_edit.toPlainText())
        except Exception as e:
            print(f"Error while saving file: {e}")
```

## Б.1 Сценарій modelinfo_window.py

```
from qt_imports import *

class ModelInfoWindow(QWidget):
    def __init__(self, model_test_text, model_summary,
parent=None):
        super().__init__(parent)
        self.setWindowTitle("Model summary")
        self.resize(300, 300)

        layout = QVBoxLayout()

        self.label = QLabel("Summary:")
        self.label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.label)

        self.summary_box = QTextEdit()
        self.summary_box.setReadOnly(True)

        self.summary_box.setText(model_test_text + '\n' +
model_summary)
        layout.addWidget(self.summary_box)

        self.save_button = QPushButton("Save in modelinfo.txt")
        self.save_button.clicked.connect(self.save_to_file)
        layout.addWidget(self.save_button)
```

```
        self.setLayout(layout)

    def save_to_file(self):
        try:
            with open("modelinfo.txt", "w", encoding="utf-8") as
f:
                f.write(self.summary_box.toPlainText())
        except Exception as e:
            print(f"Error while saving file: {e}")
```