FACULTY OF ELECTRONICS
COMPUTER ENGINEERING AND TELECOMMUNICATIONS DEPARTMENT

LABORATORY WORK #2

Development of Algorithms for Recognition

Student: Yury Shuba

Lecturer: **Gabriela Vdoviak**

Vilnius, 2025

Table of Contents

1 Goal

Prepare and train a model for object detection using the commercial tool SentiSight.ai

2 Objective

Familiarize with the process of creating an object detection model using SentiSight.ai. Students must find a dataset, annotate images with bounding boxes, train the object detection model, and perform predictions using REST API.

3 Requirments

- Select training data, annotate objects, and train the detection model.
    - Training data should consist of 3 – 5 object classes, with each class having 50 – 100 images (more can be used, be mindful of resources available on the SentiSight.ai platform).
- Provide a PDF report briefly describing the workflow, including examples of annotated training images and prediction results (using training and testing images). Additionally, include model training metrics.
- Utilize the SentiSight.ai REST API to create a graphical user interface for visualizing predictions.
    - There should be an option to upload an image to the system and return the image with annotated objects.

Grading criteria:

1. Data collection (3 points).
2. Image annotation (2 points).
3. Model training (0.5 point); Explanation of implemented solution (0.5 point).
4. Validation and model selection (0.5 point); Explanation of model selection based on training statistics interpretation (0.5 point).
5. Prediction using REST API (3 points).

4 Workflow

At first, we have to collect the dataset of unannotated images. I’ve found the following **animal_data** dataset as a basement for mine. It contained 15 classes with ~120 images for each. For the laboratory work 4 classes is enough, so I’ve chosen dolphin, zebra, bird and lion classes. Each class is represented by 30 unique images and their variations (mirrored, rotated, unlighted, compressed, all above combined). As we need to keep up with range (50 – 100 images), I’ve

written simple script, that would delete extensive variations, keeping as much uniqueness, as possible.

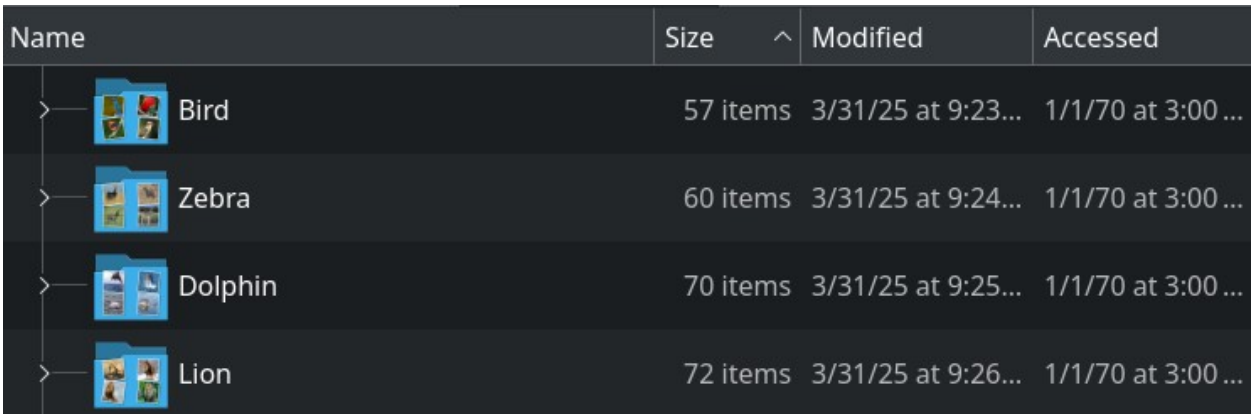


Figure 1: The resulted dataset after extensive image removal

Now we load each folder under respective automatically assigned label.

After the first part completed, we start image annotation. We manually set boxes for each picture and each animal on it. Several samples of the process are displayed below.

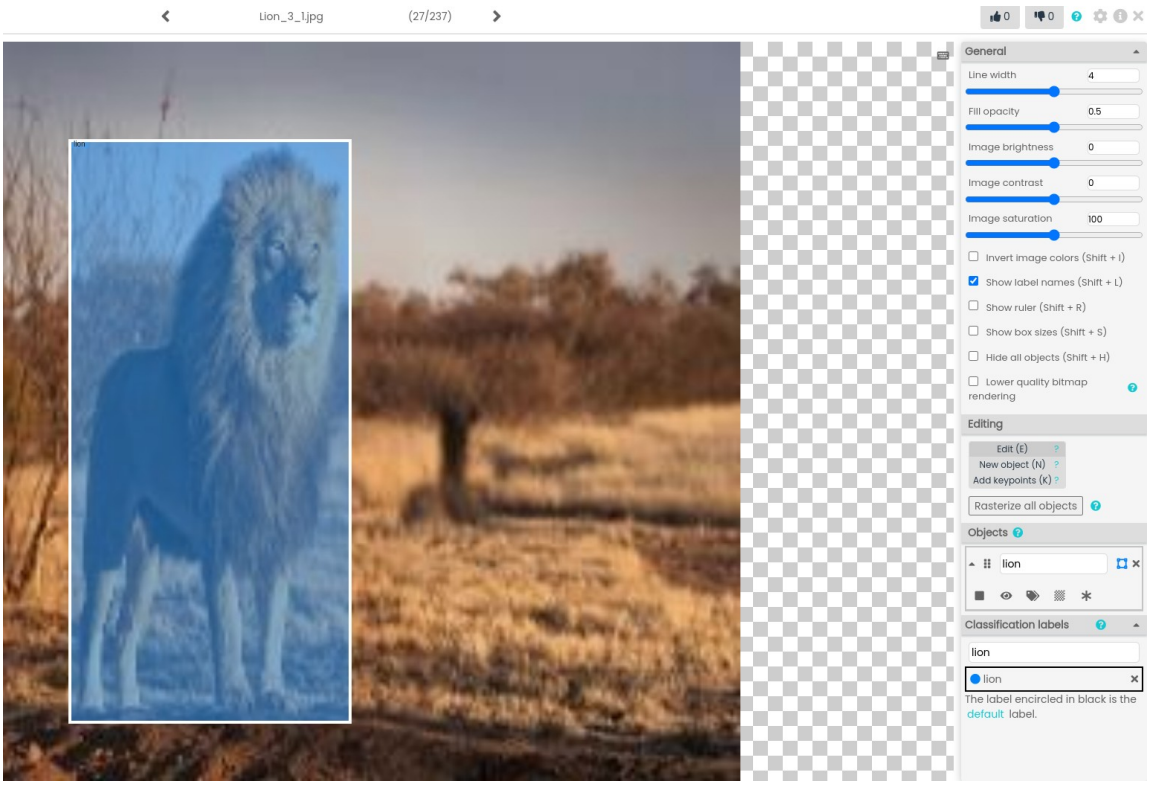


Figure 2: Image annotation process

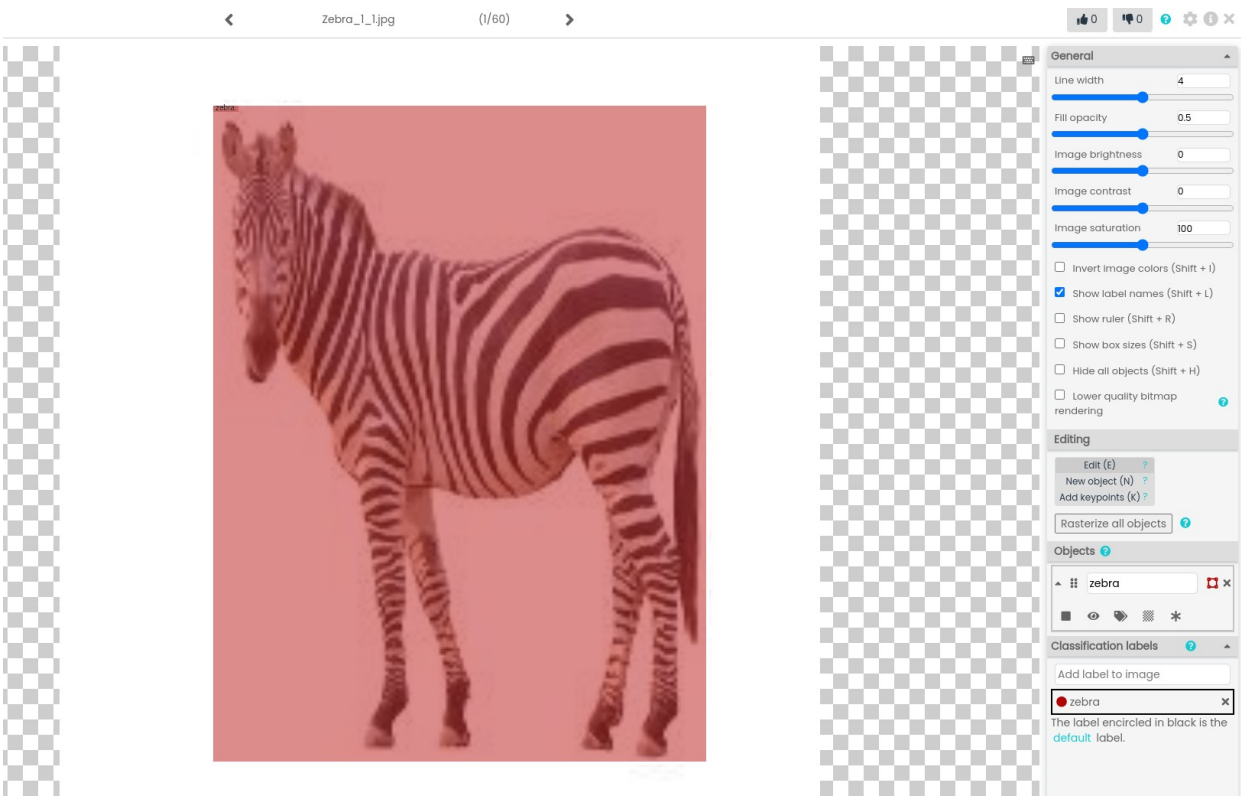


Figure 3: Zebra class image annotation sample

The resulted annotated dataset contains 237 images of 4 classes. Labeling time was 59 min 42 s with 14 seconds per image on average.

We start train process, set 60 minutes for training to avoid overfitting. We configure settings, so best-performing model would be chosen automatically instead of the latest one. After 15 minutes of training the learning curve appeared and was recorded.

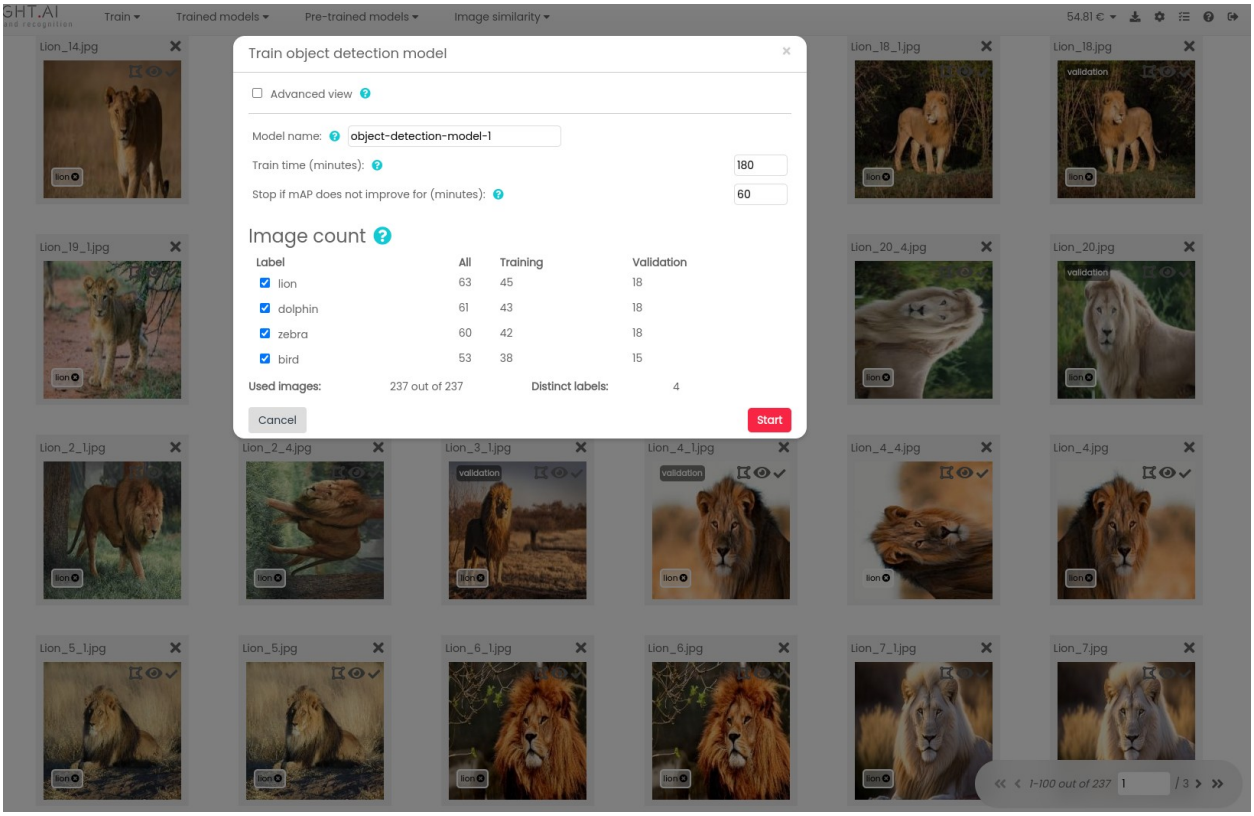


Figure 4: Train process settings

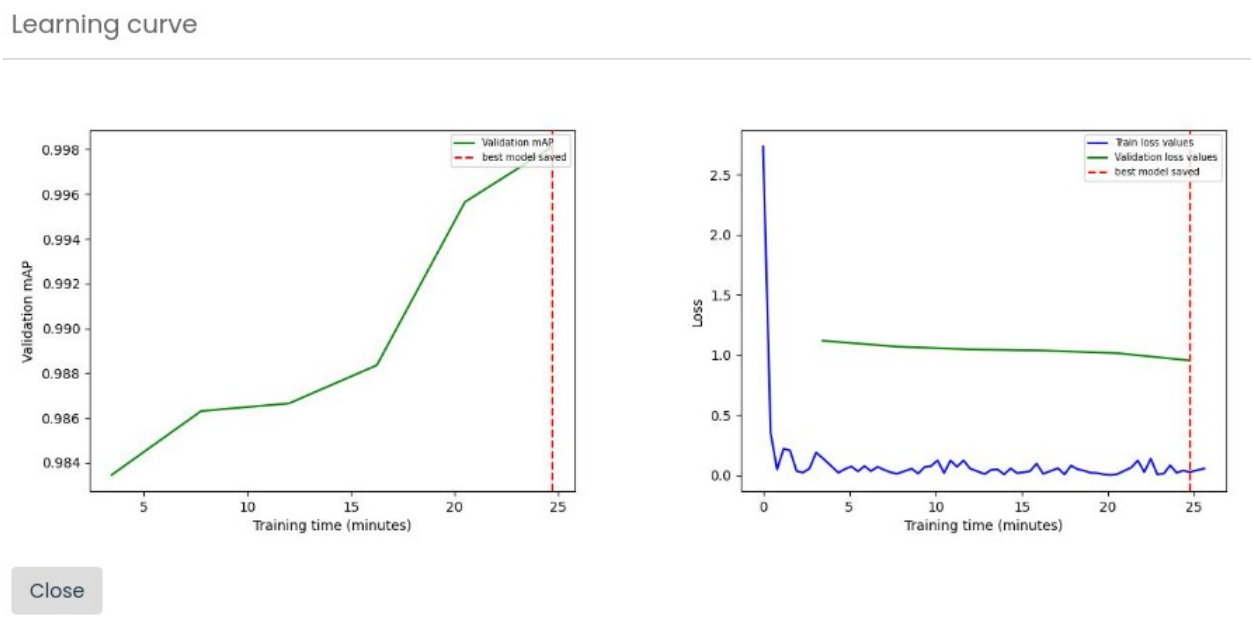
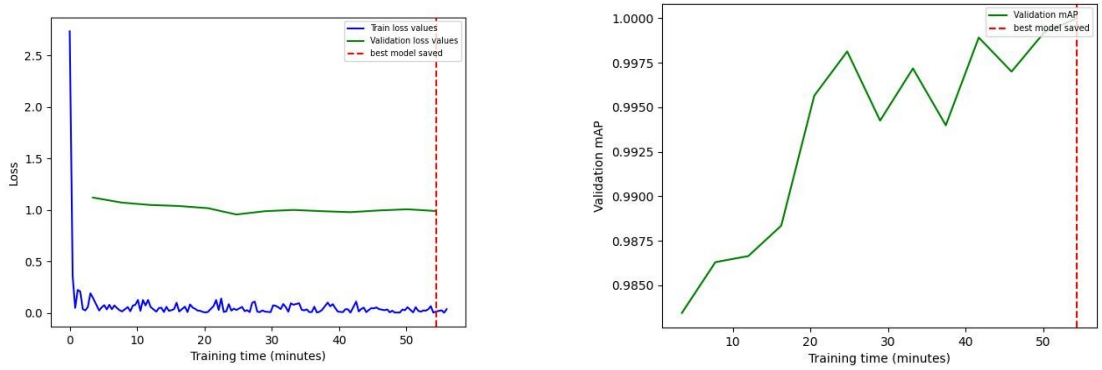


Figure 5: Learning curve view at 30 minutes mark

Final learning curves after training finished are displayed below:



Despite our effort to avoid it, overfitting is clearly visible. It may be explained by low quality of dataset (not enough of images) or too distinct classes chosen, so the patterns are immediately visible. The best model capture occurred at 56 minutes mark.

Train | Validation

Show predictions | Download predictions

Label count

Label	Images	Objects
zebra	18	18
lion	18	18
dolphin	18	18
bird	15	15

Total images: 69
Total object labels: 69
Distinct labels: 4

Intersection over union 50 % Set

Score thresholds

Use optimized thresholds Reset

All classes Set all

lion 67.95 %
dolphin 71.01 %
zebra 67.55 %
bird 69.75 %

Global statistics

Metric name	Value
Precision*	94.6%
Recall*	100.0%
F1*	97.1%
mAP	99.8%

Global statistics averaged

Per class statistics

Class	Precision*	Recall*	F1*	AP
lion	100.0%	100.0%	100.0%	100.0%
dolphin	90.0%	100.0%	94.7%	99.1%
zebra	100.0%	100.0%	100.0%	100.0%
bird	88.2%	100.0%	93.8%	100.0%

* – these statistics measures depend on the selected score threshold

Figure 6: Validation post-train statistics

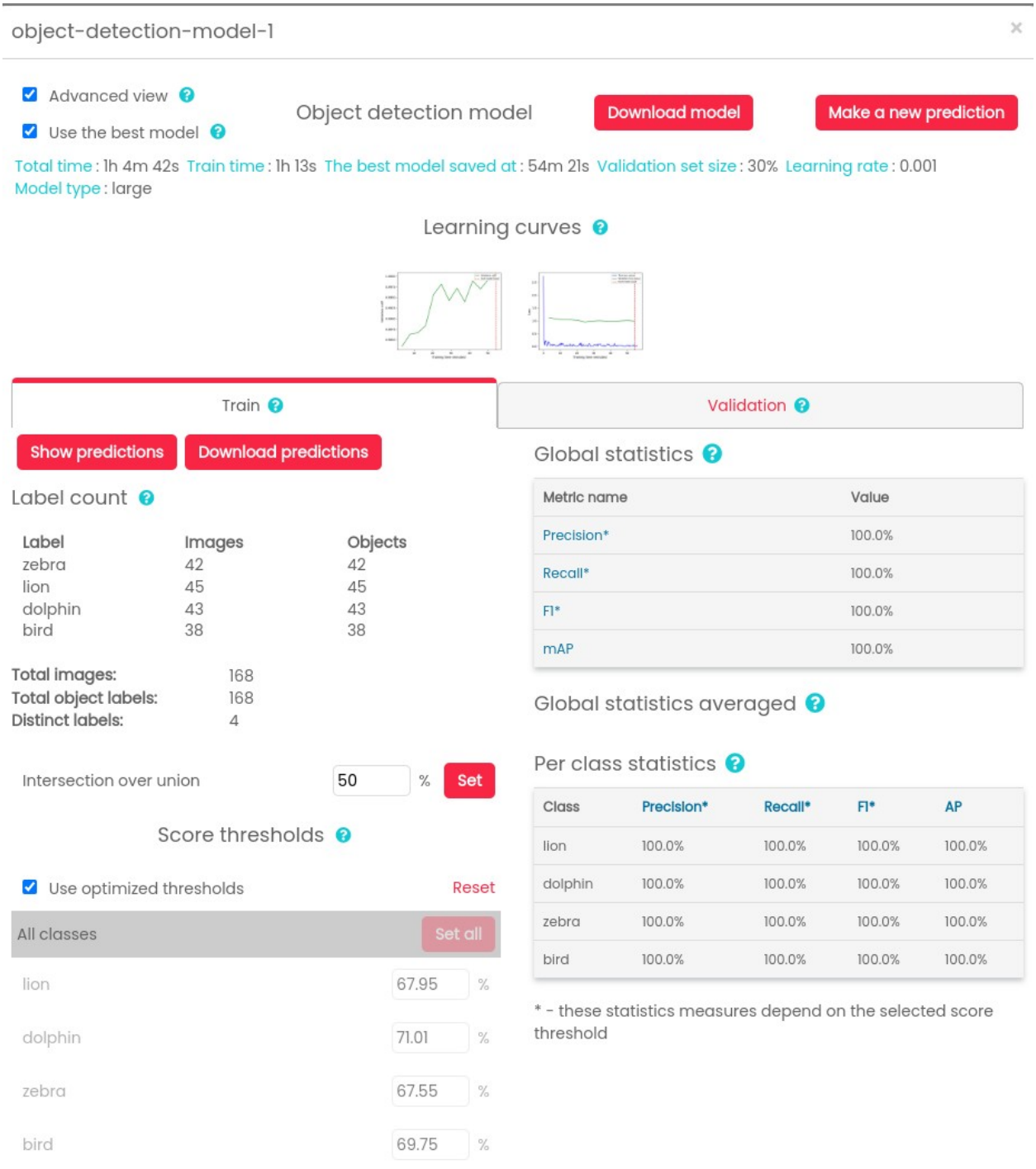


Figure 7: Train statistics with advanced view

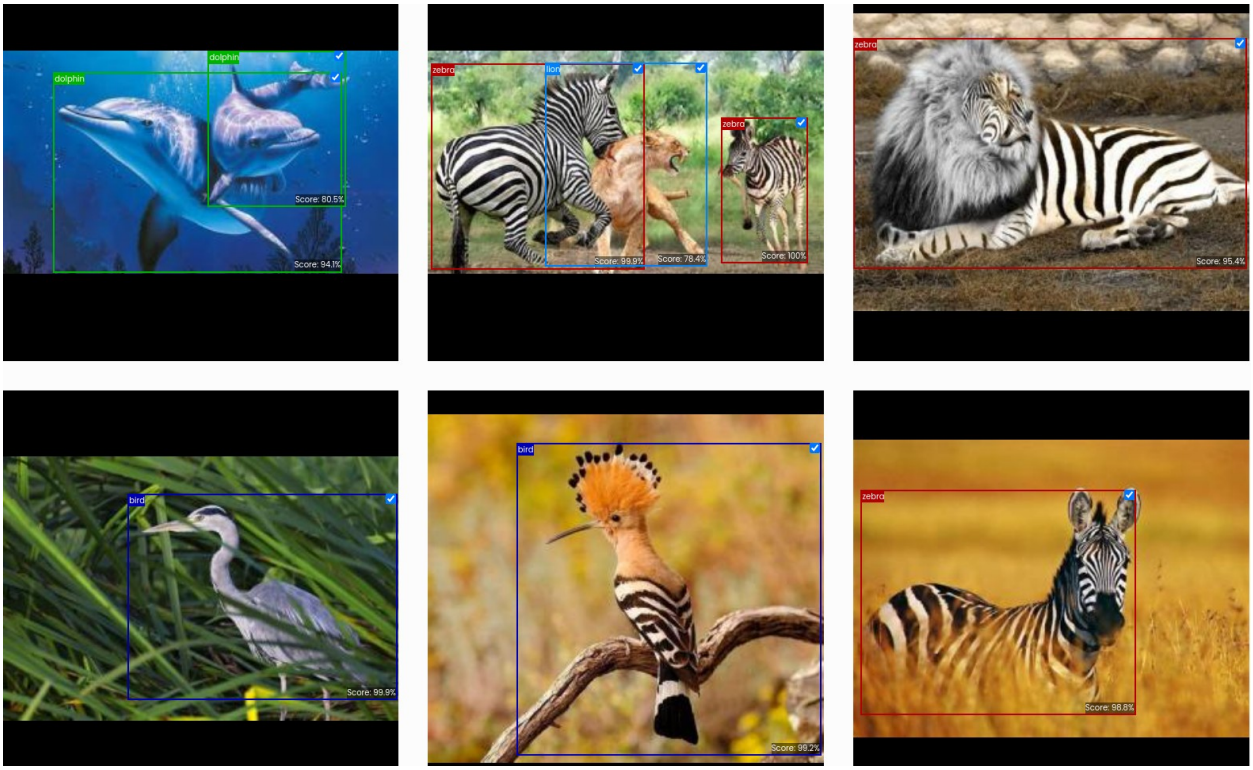


Figure 8: The result of new predictions made on test data

To assure quality of the model, 20 new images from the internet under copyleft licenses were used for test. And with exclusion of the provocative zebra-like lion, the rest of test images were properly processed.

For proper access to the model from API, the primitive GUI was designed with the ability to make the predictions on loaded images.

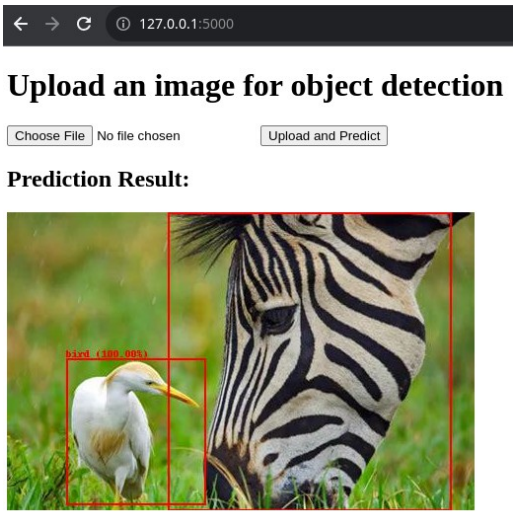


Figure 9: GUI view in browser

The code for application is in the **Appendix** part. Flusk framework was used for web user interface, while pillow library was used to draw box according to obtained coordinates.

```
(base) user@user-21cx:~/Documents/VGTU_sem_5/AI/LAB2$ python main.py
 * Serving Flask app 'main'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
erver instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (inotify)
 * Debugger is active!
 * Debugger PIN: 827-701-284
127.0.0.1 - - [[redacted]/2025 13:26:36] "GET /favicon.ico HTTP/1.1" 404 -
[
  {
    "label": "bird",
    "score": 100.0,
    "x0": 129,
    "y0": 30,
    "x1": 235,
    "y1": 152
  }
]
127.0.0.1 - - [[redacted]/2025 13:26:41] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [[redacted]/2025 13:26:41] "GET /uploads/annotated_21.jpg HTTP/1.1" 200 -
127.0.0.1 - - [[redacted]/2025 13:27:24] "POST / HTTP/1.1" 500 -
[
  {
    "label": "bird",
    "score": 100.0,
    "x0": 60,
    "y0": 148,
    "x1": 201,
    "y1": 296
  },
  {
    "label": "zebra",
    "score": 99.9,
    "x0": 163,
    "y0": 0,
    "x1": 450,
    "y1": 302
  }
]
127.0.0.1 - - [[redacted]/2025 13:27:25] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [[redacted]/2025 13:27:25] "GET /uploads/annotated_19.jpg HTTP/1.1" 200 -
```

Figure 10: Console view of API application

The respective number of successful API requests to the model is visible on SentiSight.ai project page after click on balance page.

5 Conclusions

To sum up, we've practiced in creation of commercially applicable classification model, touching all steps from data collection and prepossession up to API integration into user interface. We've obtain skills of visual models manipulation with SentiSight.ai platform functionality, as well as to receive and process predictions from REST API.

6 Appendix, GUI API python code

The original highlights and colors are kept to ensure clear code review.

```
import os

from flask import Flask, render_template, request, send_from_directory
import requests
import json
from PIL import Image, ImageDraw
import io

app = Flask(__name__)

# Configure these variables with your SentiSight.ai details
API_TOKEN = "some api token I won't share in public"
PROJECT_ID = "my project id"
MODEL_NAME = "object-detection-model-1"

UPLOAD_FOLDER = 'images'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
if request.method == 'POST':
if 'file' not in request.files:
return render_template('index.html', message='No file part')
file = request.files['file']
if file.filename == '':
return render_template('index.html', message='No selected file')
if file and allowed_file(file.filename):
filename = file.filename
file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
file.save(file_path)

# Make prediction using SentiSight.ai API
url = f"https://platform.sentisight.ai/api/predict/{PROJECT_ID}/{MODEL_NAME}/"
headers = {
"X-Auth-token": API_TOKEN,
"Content-Type": "application/octet-stream"
}
```

```
with open(file_path, 'rb') as image_file:
response = requests.post(url, headers=headers, data=image_file)

if response.status_code == 200:
prediction = json.loads(response.text)

# Annotate the image
image = Image.open(file_path)
draw = ImageDraw.Draw(image)
print(json.dumps(prediction, indent=2))
for obj in prediction:
draw.rectangle([obj['x0'], obj['y0'], obj['x1'], obj['y1']], outline="red", width=2)
label_text = f"{obj['label']} ({obj['score']:.2f}%)"
draw.text((obj['x0'], obj['y0'] - 10), label_text, fill="red")

# Save the annotated image
annotated_filename = f"annotated_{filename}"
annotated_path = os.path.join(app.config['UPLOAD_FOLDER'], annotated_filename)
image.save(annotated_path)

return render_template('index.html', filename=annotated_filename)
else:
return render_template('index.html', message='Error in prediction')
return render_template('index.html')

@app.route('/uploads/<filename>')
def uploaded_file(filename):
return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

if __name__ == '__main__':
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.run(debug=True)
```