Homework 2, Urmas Pitsi
Computer Vision ITS8030, November 2020

Task description:
Detect portugese slugs (arion lusitanicus) by searching for their specific
features. It is necessary to distinguish the portugese slugs from endemic species living in Estonia: limax cinereoniger and limax maximus. Portugese slugs are sometimes also referred to as Spanish slugs. The features of Spanish slugs and the endemic species are described here:
https://www.keskkonnaamet.ee/en/node/4351.

Data:
I downloaded the data from the course repository. I selected randomly 40 images from both classes:
limax: https://gitlab.cs.ttu.ee/its8030-2020/lusitania_data/-/tree/master/limax
lusitania: https://gitlab.cs.ttu.ee/its8030-2020/lusitania_data/-/tree/master/lusitania

Source code: https://gitlab.cs.ttu.ee/urpits/its8030-2020/-/tree/master/hw2

Task 1: Preprocessing
Preprocessing for OpenCV.
No preprocessing was used. With OpenCV I used original images as inputs

Preprocessing for deep learning:
Resized all images to the size of: width=256 and heigth=256. The main reason for doing that is to reduce the memory footprint. The hypothesis is that by reducing image size to (256, 256) will still give us good results. At least we can iterate faster as our data set is physically smaller. All resized images are located in folder: /data256/all/
I prefixed all resized files' names with 'lusitania_' and 'limax_' accordingly in order to derive image class from file name.
Code for preprocessing is located in Jupyter Notebook: ImagePreprocess.ipynb.

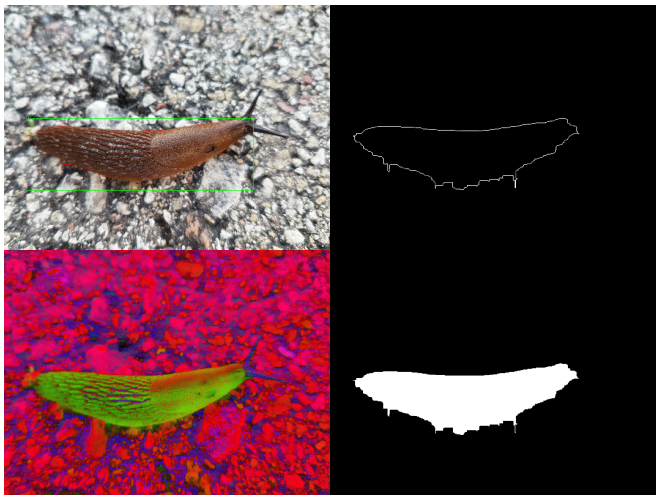
Task 2: Etalons
OpenCV feature and texture matching: chose randomly 2 images from Lusitania images as templates.
Key ideas to try:

- match by colour segmentation: extract segmentation mask and contour.
- match by texture: patch of tail, find matching keypoints
- match by texture: patch of head, find matching keypoints

Colour segmentation
For colour segmentation purposes I created simple tool in OpenCV that allows to segment an image based on its HSV (hue, saturation, value) representation (based on this wonderful tutorial: https://github.com/murtazahassan/Learn-OpenCV-in-3-hours).
Idea is that we fix hue, saturation and value ranges, then we threshold the image based on selected ranges. If object of interest is clearly separable by color then this solution provides excellent results very quickly and easily. In our case Lusitania slugs have pretty homogeneous colour and often very distinct compared to the background. However if the background colours contain patches that are very similar to slug colour then the method fails. Algorithm itself works in following steps: 1. HSV range thresholding, 2. blur, 3. dilate, 4. erode, 5. find contours, 6. draw rectangle around contour, fill in the countour to get object mask. By finetuning on etalon Lusitania image '20200723_210018.jpg' we obtain HSV range values (min,

max): hue (0,19), saturation (72,180), value (5,176). Using these fixed hsv parameters we can achieve pretty nice results.

Etalon Lusitania image ‘20200723_210018.jpg‘ with bounding box derived from HSV range thresholding



Bounding boxes on Lusitania images using fixed HSV segmentation

Initially I was pretty optimistic of creating some naive object detector based on colour segmentation. However due to time constraints I had to abandon the idea and move on to neural nets.

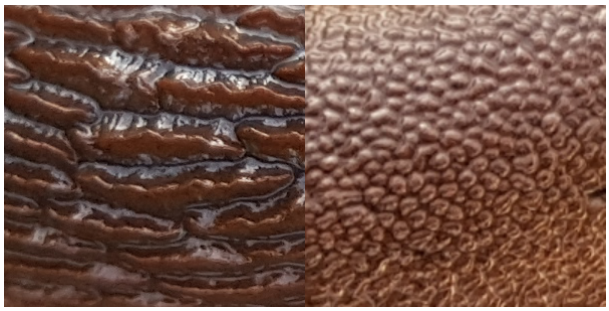
Code for experiments with colour segmentation is located in Jupyter Notebook: ColourSegmentation.ipynb.

Feature detectors, keypoints and template matching

Lusitania slugs seem to have 2 very distinctive textures: head texture and tail texture. Hypothesis was to take exemplary patches (templates) from etalon image and try to match these on other images. For etalon I took a nice example image of a Lusitania slug (‘lusitania_20200723_210018.jpg’). Cropped one rectangular patch from the head part and another patch from tail part. Next I tried feature detectors with keypoints matching and template matching with the same template patches.

Lusitania template patches, lusitania_20200723_210018.jpg:

tail head



Feature detectors and keypoint matching

I tried keypoint matching with 4 feature detectors: ORB, SIFT, FLANN, BRIEF. All, except BRIEF provided good keypoint matching with similar images. BRIEF somehow displayed weird results even on similar images.

ORB SIFT

BRIEF

FLANN

Additionally I tried Local Binary Patterns. I didn't have too much time to investigate that thoroughly enough, although it seems pretty interesting approach.
(https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html)

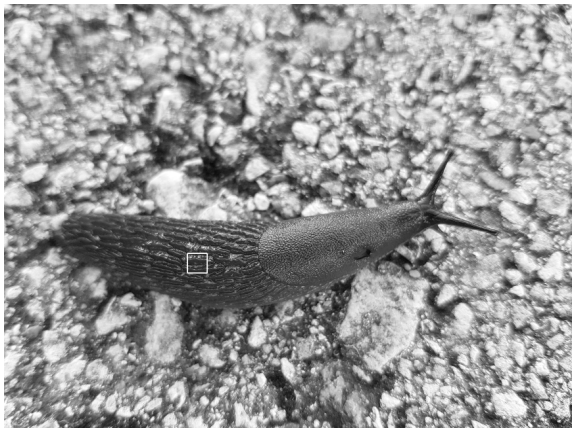
Code for experiments with feature detectors and keypoint matching is located in Jupyter Notebook: TextureMatching.ipynb.

Template matching.
Template matching worked really well on very similar images, but failed completely on not so similar images. Changes in lighting and viewing angles had drastic effect on the result.
Code for experiments with template matching is located in Jupyter Notebook: TemplateMatching.ipynb

Template matching:

on similar image

on original image



Task 3: Baseline
As our classes are perfectly balanced, then we hope that any baseline accuracy for binary classification should be above 0.5. OpenCV manual colour segmentation seems to work really good for some images and fails completely on others: in case when slug colors are not that different from background colors.

In any case manual colour segmentation doesn't generalize well. Although it seems pretty handy tool to know. Texture and features key point matching didn't provide good generalizable results. Maybe if more time is invested it could yield some meaningful results.
Therefore I tried to establish some meaningful baseline with deep learning approach (see task 4).

Task 4: Classification with deep learning
Approach:
Setup:
Environment: Google Colab notebooks. Here: ...???
Library: fastai2 (https://github.com/fastai/fastai)

Strategy of experiments
'Data' contains 80 images: 40 Lusitania and 40 Limax.
From the Data I take 1 to 4 image(s) from both classes and use them as 'TrainSet', all remaining images are kept as 'TestSet'. Expand TrainSet by a factor of 100 using image augmentations. Split TrainSet to 'train set' and 'valid set' with proportions 90%/10%. So that validation set is 10% of TrainSet. Usually it is a bad idea to have same underlying items in train and valid set, because it leads to drastic overfitting. In our case I used this approach to see whether it gives us a reasonable baseline and maybe it will even generalize to the TestSet.

For image classification I conducted 3 experiments with following setups.

Experiment 1
TrainSet={1 image of lusitania, 1 image of limax}, in total of 2 images: 'lusitania_20200723_210018.jpg', 'limax_1.jpg'
TrainSet expanding factor = 100, leading to number of images in TrainSet = 2 * 100 = 200.
Validation split pct = 0.1, leading to number of images in train set = 0.9 * 200 = 180; and number of images in valid set = 0.1 * 200 = 20.

Experiment 2
TrainSet={4 images of lusitania, 4 image of limax}, in total of 8 images: 'lusitania_20200723_210018.jpg', 'lusitania_20200723_210055.jpg', 'lusitania_arion_lusitanicus_9.jpg', 'lusitania_arion_vulgaris_4.jpg', 'limax_1.jpg', 'limax_limax_cinereoniger_1.jpg', 'limax_limax_cinereoniger_2.jpg', 'limax_limax_maximus_10.jpg'
TrainSet expanding factor = 100, leading to number of images in TrainSet = 8 * 100 = 800.
Validation split pct = 0.1, leading to number of images in train set = 0.9 * 800 = 720; and number of images in valid set = 0.1 * 800 = 80.

Experiment 3
TrainSet={20 images of lusitania, 20 image of limax}, in total of 40 images.
TrainSet expanding factor = 1, leading to number of images in TrainSet = 1 * 40 = 40.
Validation split pct = 0.2, leading to number of images in train set = 0.8 * 40 = 32; and number of images in valid set = 0.2 * 40 = 8. In this case train and validation sets do not overlap.

Our strategy for experiments 1 and 2 is to heavily (perfectly) overfit to train and validation set and see how well it will perform on TestSet. Images in TestSet are completely unseen by our model. Hypothesis is that if this strategy provides any meaningful result (classification accuracy well above 0.5) then we can only improve on that later on by constructing a 'real' valid set: generate train and valid sets from

non-overlapping sources using image augmentation. Although Experiment 3 has non-overlapping train and valid sets, it is still limited. Ideally we should generate bigger fixed set of validation data from couple of examples. However due to technical/time reasons could not implement it in this work. Here, the Experiment 3 follows 'classical' training regime: validation set is just those 8 images that are obtained by random split before training.

Model architecture
For all experiments: resnet34, pretrained on ImageNet.

Image augmentation and additional preprocessing
Fastai augmentation utility function 'aug_transforms' (https://docs.fast.ai/vision.augment) with following parameters: aug_transforms(flip_vert=True, max_rotate=45, max_zoom=1.3, max_lighting=0.8, max_warp=0.4, p_affine=0.5, p_lighting=0.5)
Additionally: resize to 224, pad_mode='reflection', normalize with imagenet_stats.

Training details
Experiments 1,2: batch size=64. Experiment 3: batch size=16.
Finetuned pretrained resnet34 model with 5 epochs (total running time 5 seconds).

Results

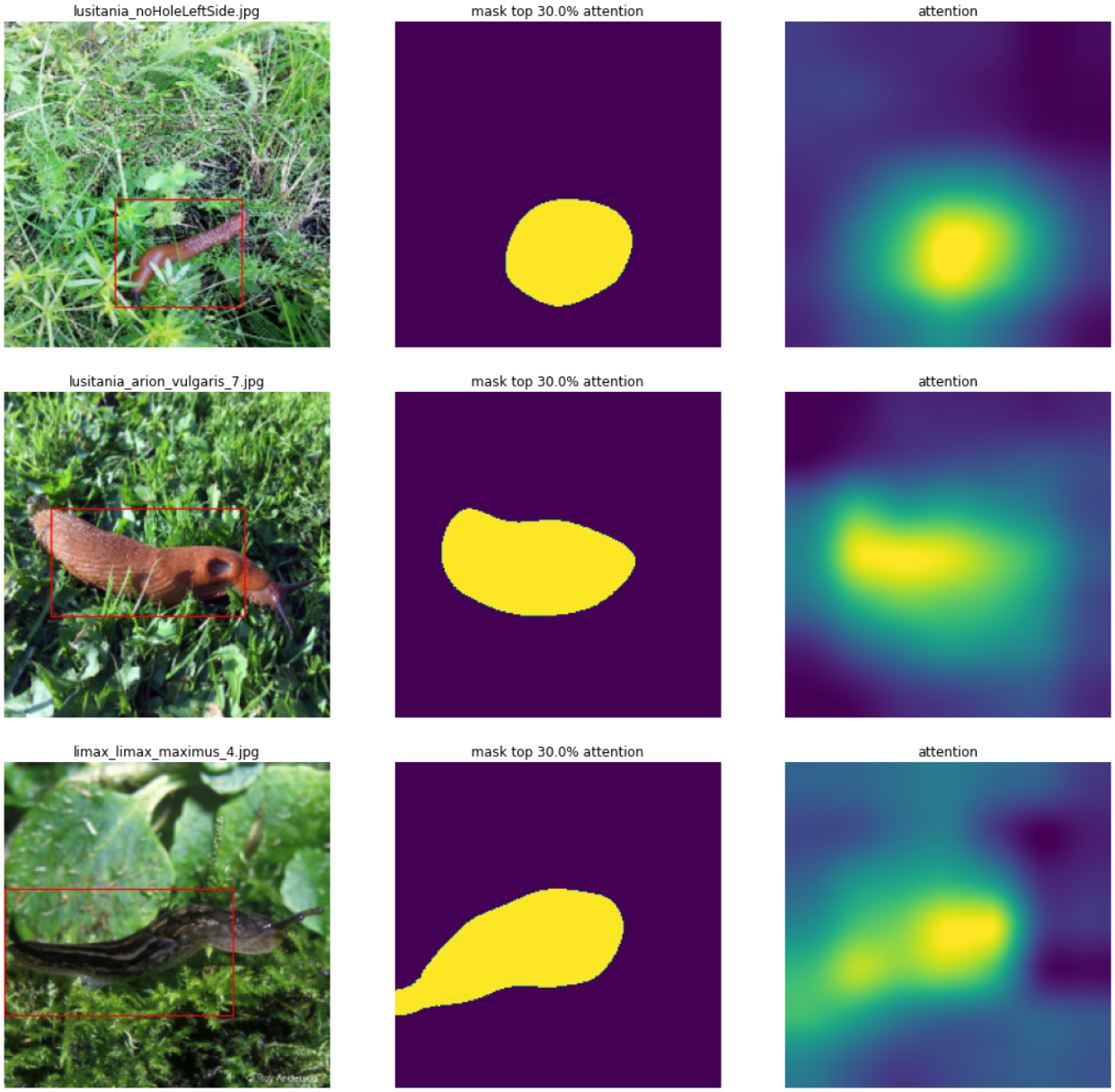
Image classification results:

	Num images per class	train / test images	Test accuracy	Avg Precision	Precision limax/lusitania	Recall limax/lusitania
Experiment 1	1	200 / 78	78.2%-86.8%	0.94-0.97	0.9 / 0.75	0.69 / 0.92
Experiment 2	4	800 / 72	87.5%-91.7%	0.95-0.97	0.89 / 0.86	0.86 / 0.89
Experiment 3	20	40 / 40	82.5%-92.5%	0.95-0.98	0.89 / 0.86	0.85 / 0.9

Object detection: locating slugs in the picture

Visualizing attention heatmaps and constructing slug detection bounding boxes from that.
Based on this paper: 'Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization' (https://arxiv.org/abs/1610.02391) and fastai's implementation (https://github.com/fastai/fastai/blob/master/dev_nbs/course/lesson6-pets-more.ipynb). I extended the idea of neural networks 'visual attention' to extracting a bounding box that could represent the object under attention.
This is yet purely experimental thought, however it seems to work pretty good. We need to specify manually the threshold value which creates the mask for extracting area with higher attention. I used 30% from max value as threshold, as it seemed to produce reasonable results while testing.

Attention heatmaps: yellow areas have higher values.



Detecting bounding boxes from scratch

In order to detect slugs we need ground truth bounding boxes. For this purposes I created simple bounding box annotator tool using Jupyter Notebook and OpenCV: BoundingBoxAnnotator.ipynb
I annotated manually the same 8 (4+4) images that were used in Experiment 2. And ran training experiment with same setup as Experiment 2. Only this time I included bounding box annotations. I was not able to use pretrained model in this case, so I trained resnet18, resnet34 and resnet50 from scratch. Best results gave
resnet50: 50 epochs + 5 epochs finetuning, validation avg IOU=0.87. Total running time 10 min.
Observing the results visually we can see that they are pretty good, not perfect of course. I did not have enough time to properly implement the usage of mAP metric for bounding boxes.

Predicted bounding boxes on TestSet sample

Discussion
Maybe use some preprocessing for OpenCV: resizing, some blurring etc. Usually the original images have quite large resolution, maybe less information works better for manual features?
Implement better metric for bounding boxes (mAP), also find better loss function if possible. Maybe converges much better. Also with more time could find a way to use pretrained models in bounding box detection pipeline.
All relevant code is provided in: https://gitlab.cs.ttu.ee/urpits/its8030-2020/-/tree/master/hw2