EE 046746 - Technion - Computer Vision

Homework 2 - Classifiers and Segmentation

Due Date: 21.12.2021

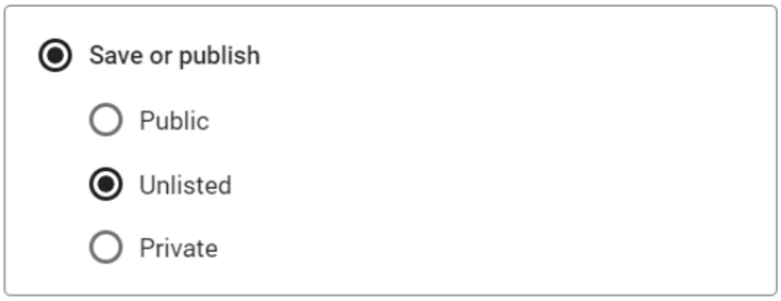
Submission Guidelines

READ THIS CAREFULLY

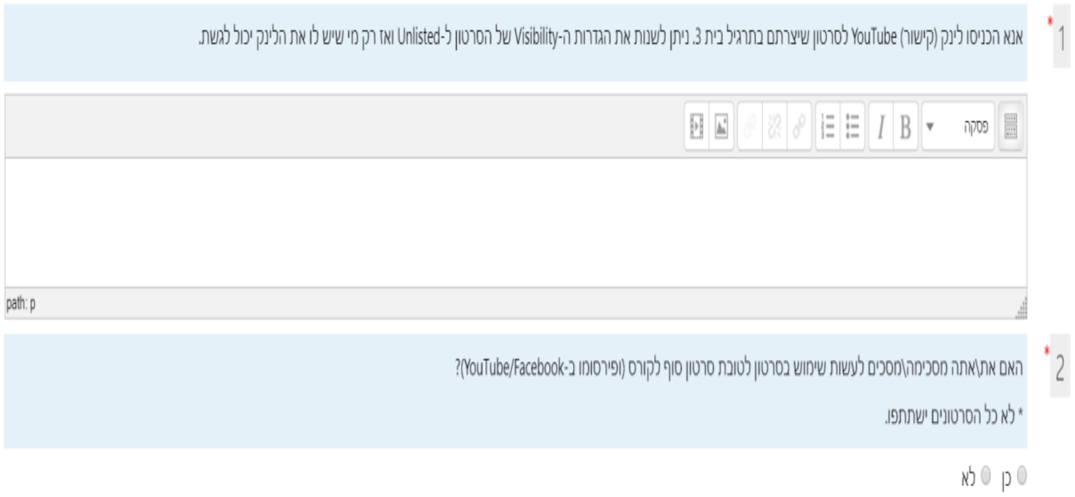
- Submission only in **pairs**.
- **No handwritten submissions**.
- You can choose your working environment:
  - You can work in a `Jupyter Notebook`, locally with Anaconda or online on Google Colab
    - **Important**: Colab also supports running code on GPU, so if you don't have one, Colab is the way to go. To enable GPU on Colab, in the menu: `Runtime` $\rightarrow$ `Change Runtime Type` $\rightarrow$ `GPU`.
  - You can work in a Python IDE such as PyCharm or Visual Studio Code.
    - Both also allow opening/editing Jupyter Notebooks.
- You should submit two **separated** files:
  - A compressed `.zip` file, with the name: `ee046746_hw2_id1_id2.zip` which contains:
    - A folder named `code` with all the code files inside (`.py` or `.ipynb` ONLY!), and all the files required for the code to run (your own images/videos).
      - **The code should run both on CPU and GPU without manual modifications**, require no special preparation and run on every computer.
    - A folder named `output` with all the output files that are not required to run the code. This includes videos and visualizations that are not included in your PDF report.
  - A report file (visualizations, discussing the results and answering the questions) in a `.pdf` format, with the name `ee046746_hw2_id1_id2.pdf`.
    - Be precise, we expect on point answers.
  - No other file-types (`.docx`, `.html`, ...) will be accepted.
- Submission on the course website (Moodle).

Video Submission (YouTube - Optional)

- In this exercise you are going to produce a video.
- In addition to submiting this video in the output folder (MANDATORY), we are also giving you the option to upload it to YouTube and submit the link in a different section on the website (just below the original submission section).
  - This will benefit you when you explain your expertise in Computer Vision (e.g., in a job interview).
  - We also may make something nice with your works at the end of the course.
- If you don't want to make your video public, you can change its visibility to "Unlisted", and then it can only be accessed via link:


Save or publish [✓]
Public
[✓ Unlisted]
Private

- Finally, submit the link on the course website and answer if you are okay with making this video public:


1 * אנא הכניסו לינק (קישור) YouTube לסרטון שיצרתם בתרגיל בית 3. ניתן לשנות את הגדרות ה-Visibility של הסרטון ל-Unlisted ואז רק מי שיש לו את הלינק יכול לגשת.

2 * האם את\אתה מסכימה\מסכים לעשות שימוש בסרטון לטובת סרטון סוף לקורס (ופירסומו ב-YouTube/Facebook)?

* לא כל הסרטונים ישתתפו.

כן לא

Python Libraries

- `numpy`
- `matplotlib`
- `pytorch` (and `torchvision`)
- `opencv` (or `scikit-image`)
- `scikit-learn`
- Anything else you need (`PIL` , `os` , `pandas` , `csv` , `json` ,...)

Tasks

- In all tasks, you should document your process and results in a report file (which will be saved as `.pdf`).
- You can reference your code in the report file, but no need for actual code in this file, the code is submitted in a seprate folder as explained above.

Part 1 - Classic Vs. Deep Learning-based Semantic Segmentation

In this part you are going to compare classic methods for segmentation to deep learning-based methods.

1. Load the images in the `./data/frogs` and `./data/horses` folders and display them.
2. Pick 1 classic method for segmentation and 1 deep learning-based method and segment the given images. Display the results.
   - **Briefly** summarize each method you picked and discuss the advantages and disadvantages of each method. In your answer, relate to the results you received in Q1.
   - You can use a ready implementation from the internet or OpenCV, no need to implement it yourselves.
   - Note: the classic method **must not** use any neural network.
3. Pick 3 images (download from the internet or take them yourself) that satisfy the following, and dispaly them:
   - One image of a living being (human, animal,...).
   - One image of commonly-used object (car, chair, smartphone, glasses,...).
   - One image of not-so-commonly-used object (fire extinguisher, satellite,... **BE CREATIVE**).
4. Apply each method (one classic and one deep learning-based) on the 3 images. Display the results (mask and segmented image).
   - Which method performed better on each image? Describe your thoughts on why one method is better than the other.
   - For the classic method you can change parameters per-image, document them in the report.
   - You can add manual post-processing to get a mask if needed. If you do that, document in your report "how hard" you had to work in the post-processing stage, as it's an indication of the quality of the method.
5. As you probably have noticed, segmentation can be rough around the edges, i.e., the mask is not perfect and may be noisy around the edges. What can be done to fix or at least alleviate this problem? Your suggestions can be in pre-processing, inside the segmentation algorithm or in post-processing.

Part 2 - Analyzing a Pre-trained CNN

In this part you are going to analyze a (large) pre-trained model. Pre-trained models are quite popular these days, as big companies can train really large models on large datasets (something that personal users can't do as they lack the sufficient hardware). These pre-trained models can be used to fine-tune on other/small datasets or used as components in other tasks (like using a pre-trained classifier for object detection).

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of `[0, 1]` and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`.

You can use the following transform to normalize:

`normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])`

Read more here

1. Load a pre-trained VGG16 with PyTorch using `torchvision.models.vgg16(pretrained=True, progress=True, **kwargs)` (read more here). Don't forget to use the model in evaluation mode (`model.eval()`).
2. Find 2 images of a dog/cat/bird/cow (or other *common* animal) on the internet (1 image per animal family, i.e. 2 total), and display them.
3. Pre-process the images to fit VGG16's architecture. What steps did you take?
4. Feed the images (forward pass) to the model. What are the outputs? To convert from class index to label, use the supplied `imagenet1000_clsidx_to_labels.txt` file
5. Choose one of the images, and segment the animal using a classic or deep segmentation method. Display the result.
6. Put the the animal in a different habitat, i.e., use the segmentation mask to place the animal on a different background. You can choose any background you want (which is not the animal's natural habitat). For example, put the bird on the moon (`./data/moon.jpg` for example) or the cow on a beach (`./data/beach.jpg` for example). Dispaly the result.
   - You should submit the final image in the `output` folder.
   - It's encouraged to make the homework checker laugh.
7. Feed the new image to the network, what is the output? is it different than section 4? Discuss the reasons for that to happen. What do you think would have happened if you would have only applied a transformation to the image (e.g. geometric transformations, such as rotation or scaling or color transformations)?
8. For the first 3 filters in the *first layer* of VGG16, plot the filters, and then plot their response (their output) for the chosen original image from section 2 and the image from section 6-7 (total of 2 input images). Explain what do you see.

- Consult `ee046746_appndx_visualizing_cnn_filters.ipynb` to refresh your memory.

9. How would applying a filter (e.g. gaussian blur) to the input image affect the responses you plotted (for the firstt 3 filters in the first layer of VGG16)?
10. For each image in the `./data/dogs` and `./data/cats` folders, extract and save their feature vectors (create a numpy array or a torch tensor that contains the features for all samples) from a fully-connected layer (such as `FC7`) of the VGG16 model. Which layer did you pick? What is the size of the feature space?
    - You need to write a function that does the feed forward manually until the desired layer. See the example in `ee046746_appndx_visualizing_cnn_filters.ipynb`.
11. Build a Support Vector Machine (SVM) classifier (hint: `sklearn.svm.LinearSVC`) to classify cats and dogs based on the features you extracted. Use the 20 images as train set, and choose 4 images (2 dogs, 2 cats) from the internet as test sets. You can choose a different classifer than SVM from the `scikit-learn` library, no need to explain how it works (but report the name of the algorithm you used). What are the results?

Part 3 - Jurrasic Fishbach

In this part you are going to apply segmentation on a video, and integrate with other elements.

1. Film a short video of yourself (you can use your phone for that), but without too much camera movement. You on the other hand, can move however you want (we expect you to). Convert the video to frames and resize the images for a reasonable not too high resolution (lower than 720p ~ 1280x720 pixles). You can use the function in `frame_video_convert.py` to help you. Display 2 frames in the report.
2. Segment yourself out of the video (frame-by-frame) using one of the methods (classic or deep). Display 2 frames in the report.
3. Pick one of the objects in the supplied videos (`./data/dancing_man_model.mp4` , `./data/dinosaur_model.mp4` , `./data/jet_model.mp4`), convert it to images and segement it out using one of the methods (classic or deep). Display 2 frames in the

report. You can choose another object from: https://pixabay.com/videos/search/green%20screen/.
   - Explain how you performed the sementation for this specific type of video (i.e., green-screen videos). Did you use a simple/classic method? Deep method? Combined both?
4. Put it all together - pick a background, put yourself and the segemented object on the background. Stich it frame-by-frame (don't make the video too long or it will take a lot of time, 10secs maximum). Display 2 frames of the result. Convert the frames back to video. You can use the function in `frame_video_convert.py` to help you.
   - Tip: To make it look good, you can resize the images, create a mapping from pixel locations in the original image to pixels locations in the new image.
   - You should submit the final video in the `output` folder (**MANDATORY**), and upload it to YouTube (**OPTIONAL**) as instructed above.
   - We expect some creative results, this can benefit you a lot when you want to demonstrate your Computer Vision abilities.

Credits

- Images from Imagenet
- Videos from Pixabay
  - Dinosaur video from Modern Media
- Icons from Icon8.com - https://icons8.com