Final project: In the name of deep learning

Computer Vision (H02A5a)

In the final project, you will delve into applying deep learning techniques to computer vision problems. You will again be randomly assigned to *groups of 4* and requested to evaluate the contribution of each of your team members. We have prepared a template notebook on Google Colab that should be submitted together with your trained model weights and a max 20 min narrated Powerpoint presentation file as one single ZIP-file to Toledo (one per group) by **Monday 1 June 23:59**. Make sure that your work is well documented and the notebook runs from start-to-end (Runtime → Run all) without errors, and without any intervention from our side (apart from configuring the path to your model weights).

1 Overview

First, we introduce the data that you will use for this project and the toolkit you can use to work on the following sections.
The project consists of **two assignments** for which we expect you to provide code and extensive documentation in the notebook. In the first assignment, you will train an end-to-end neural network for classification (task 1) and segmentation (task 2) purposes. For each task, you will train one network from scratch and finetune a second network from pre-trained weights. In the second assignment, you will try to find and exploit the weaknesses of the classification network that you built previously. You will build an encoder-decoder type CNN that learns how to transform input images such that you can fool your classifier. It is important to note that you are not graded on the actual performance of your networks (although we do expect it to work), but we will evaluate the correctness of your approach and your understanding of what you have done that you demonstrate in the description.

2 Getting started

2.1 Data

For this project you will be using the PASCAL VOC-2009 dataset [Eve+10]. This dataset consists of colour images of various scenes with different object classes (e.g. animal: *bird, cat, ...*; vehicle: *aeroplane, bicycle, ...*), totalling 20 classes (Figure 1). The dataset has been used as a benchmark for classification, detection and segmentation challenges and for comparing results of individual research (more details can be found here). A cell is provided in the Google Colab notebook to download this dataset. The PASCAL VOC dataset already

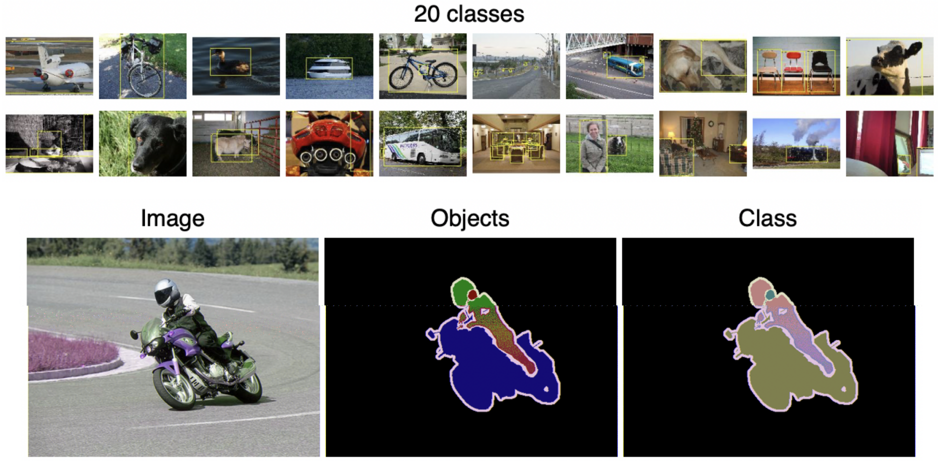

Figure 1: PASCAL VOC-2009 dataset [Eve+10] (source).

contains splits into train, val and test[1] so we recommend that you just use these. You will notice that these splits are different per task (classification vs segmentation) with different totals per task (more details on the train-val splits can be found here).

2.2 Deep learning

We build further on the previous assignment and advise you to use Keras [Cho+15] (now built into TensorFlow 2.0+) as the library for building your (deep) neural networks. The deep learning paradigm is extensive but most of it is out of the scope for this course. Therefore, unless explicitly requested, we do not expect, although we don't discourage, you to explore different optimizers (e.g. SGD, ADAM), regularization mechanisms (e.g. weight decay, L2, data augmentation), non-linearities (e.g. ReLU, sigmoid), architectures (e.g. U-Net, DeepMedic, ResNet), etc. **The main goal of this project is to gain a fundamental understanding of how deep learning can be used in computer vision applications and not to explore all the intricacies of deep learning.**

3 Supervised, end-to-end learning

In this first assignment, you will train a neural network for **two tasks** in a supervised learning setting (all labels are provided with the dataset discussed in Section 2.1).

3.1 Task 1: Classification

You can think of a convolutional neural network (CNN) built for classification as an operation that takes a NxN input and processes this into a 1x1 output (the class label). Due to the convolutional and layer-wise processing, the latent variables in deeper layers get to see

[1] The test data is available as a separate download, but it should be enough to demonstrate your performance on the val data.

more context and the learned *features* are/can be more complex (due to the successive non-linearity's). In the final layer, the resulting features are classified.

Implement a classification CNN and train it to recognise all 20 classes using the training set. You can choose your own architecture, but we advise you to stick to a network architecture that is known to work. Create **two classification models**: in model 1 you train all the parameters from scratch and in model 2 you transfer weights from a different model and apply fine-tuning (you can use a pre-trained network from the Keras library or create one yourself). Make sure that none of these pre-trained models have been trained using images from the test and validation set.
Elaborate on the design choices of your network and the loss function. A few guiding questions (but feel free to discuss more if you believe this is important): what layers did you use?, what is the use of each layer?, did you pre-process/augment the data in any way? Visualise the error on the training and validation set during training and report the classification performance on the test set (choose an appropriate metric for this task). How did you establish that the training converged? Once the network is trained export the weights and hand them in together with the assignment.

3.2 Task 2: Semantic segmentation

A segmentation task differs from classification in that you want to label every single pixel in the image (not just one label for entire image). Imagine that you are classifying the central pixel of the image (a 1x1 output) based on its surroundings (a NxN input). In order to label every pixel in the image, we can reformulate one NxN image as being N^2 pixel-wise examples, each having a different pixel in the center that gets classified based on a surrounding window [Cir+12]. Later, more parallel implementations closely follow the encoder-decoder structure, by using convolutional layers to encode and transposed convolution (a.k.a. deconvolution) layers to decode directly into pixel-wise predictions [LSD15]. The loss function which you should have used for the classification task is then applied to and averaged across *all* the pixels. From this, one of the most popular networks for image segmentation was derived by adding so-called *skip connections* from encoding layers to their corresponding decoding layers at the same scale [RFB15] to enhance learning in earlier layers and improve the reconstruction of feature maps at a higher resolution from feature maps at a lower resolution.

Implement a segmentation CNN that extracts the objects from an image. Create again two segmentation models: model 1 trained from scratch and model 2 using transfer learning. You can design the architecture as you see fit but we recommend again to start from known segmentation architectures that have been reported in literature.
Elaborate on the design choices of your network and loss function. A few guiding questions (but feel free to discuss more as you see fit): explain the difference with a classification setup, did you use skip connections?, did you pre-process/augment the data in any way? Visualize the training and validation learning curves (error over epoch). How did you establish that the training converged? Report the performance of your model on the test set (choose an appropriate metric for this task, e.g. Dice score is used a lot in medical image segmentation). Visualize a few segmented test images that your network outputs (pick a few good and a

few bad cases and discuss these qualitative results, e.g. why are these mistakes happening and what would you try next if you had more time?). Once the network is trained export the weights and hand them in together with the assignment.

4 Computer vision: a solved problem?

As you will undoubtedly have noticed in the previous exercises, computer vision has seen substantial average accuracy improvements in the last decade, even outperforming humans in certain tasks. However, computer vision is still far away from being a solved problem. One important limitation is that the state-of-the-art today, unlike humans, does not have prior knowledge about its environment, it can only learn from what is present in the training data. By using a lot of data in the training phase, it is assumed that the model will be able to generalise outside of the training set and we attempt to validate this using an independent test set. However, it is still very likely that our dataset is biased in one way or another, so even though your model might show good performance in the lab environment it might fail when being released "in the wild". Look up "Tesla autopilot fails" on YouTube to see some examples.

The seemingly unavoidable problem of bias has resulted in some skeptics saying that machine learning technologies (like deep learning) are not ready to be adopted in our daily lives. We (the TA's) do not follow this line of thinking but we do encourage you (the future A.I. experts) to evaluate what you read and develop with a critical mindset and to be aware of the limitations. The interested student is referred to this nice tutorial session by J.Z. Kolter and A. Madry.

The **goal of this second assignment** is two-fold: we want

1. to teach you about the limitations of contemporary computer vision techniques through adversarial examples
2. you to experiment with some different types of architectures i.e. encoder-decoder CNNs.

4.1 Adversarial Examples

In an adversarial attack, an adversary (attacker) adds noise (perturbation) to an image in an attempt to fool the system under attack. A famous example of such adversarial input that was generated using the fast gradient sign method is shown in Figure 2. The adversary could be a model that was trained/configured by a hacker or it might simply be a simulation of the real world input variability that you use to gain a better understanding of the limitations of your model or to improve robustness. The system under attack could be a cloud API, a spam filter or an automated vehicle. We distinguish two levels of knowledge/access that the adversary can have over the system under attack: white-box and black-box. In the case of a white-box attack, the attacker has full knowledge and access to the model under attack and its weights (e.g. open source implementation, your own implementation, ...). In a black-box attack, the adversary only has knowledge of the input of the model under attack and the related output (e.g. an API, automated vehicle with encrypted source code, ...). In this assignment we will focus on the white-box attack, which is generally considered to be the simplest since you have full access to the full model under attack (and thus its gradients!).

For this assignment, you will implement and train an encoder-decoder model of choice f_{θ_a} (e.g. normal, denoising or variational auto-encoder) that, given an image X, can generate a perturbation $\delta = f_{\theta_a}(X)$. The generated perturbation is then added to the image and fed into classifier h_{θ_c} from assignment 3.1. The goal of your adversary is to generate the perturbation in such a way that the classifier h_{θ_c} assigns a class to the perturbed image $y^* = h_{\theta_c}(X + \delta)$ that is different from the ground truth y. An overview of the adversarial model is shown in Figure 3.
You have to train two adversaries, one using the set that you used to train the final version of the classifier and one you used to test the classifier. Generate your own deceptive labels that are different from the corresponding ground truth $y_{dec} \neq y$. For this assignment you only have to be able to make the classifier misclassify one class (it doesn't matter what the new output is but if you're feeling motivated try convincing the classifier that a picture of a cow is a bird, or the bus is a cat, ...). You can choose any loss function L that minimises the distance between the deceptive labels y_{dec} (the ones you come up with) and the predicted label y^*. Since you don't want to retrain the original classifier, it is important to freeze the weights θ_c of the classifier so that they do not change during the training phase of the adversary! Further note that the generated perturbation has to be constrained such that the perturbed images still look the same to our naked eye (if not you are just generating new images). It is common to use a regularisation term R which represents the perturbation norm e.g. l_2 (euclidean distance) or l_∞ (maximum norm).

Provide a comparative analysis where you study the difference between the perturbations that were generated using the two different training sets, do you notice a difference in overall magnitude? Do you notice a difference in how well the adversary can fool the model under attack? Try to explain your observations. Furthermore, explain your overall approach, design choices etc in your own words (feel free to use math notation, explain objective, labels, network, ...). Is this a realistic attack setup? Can it be used in a "real world" scenario? If so, how/when? Was the adversary successful? Are the images still recognisable for a human observer? Does this mean that your classifier h_{θ_c} from assignment 3.1 is now unreliable? What could we do to increase the robustness of a model under attack in general? Visualise some of your most successful cases (original image, perturbation, perturbed images).

5 Discussion

Finally, take some time to reflect on what you have learned in the two assignments of this final project. We don't expect that you do a full literature study on these tasks but think critically about what you did and the results that you achieved. Did your classification and segmentation networks perform well? Has your adversary reached its goal? What would you do differently if you had some more time? Where would you spend your time on to improve your results? What are the limitations of your models? Are they ready to be deployed in a 'real world' setting? Feel free to add any other critical reflections you might think of.

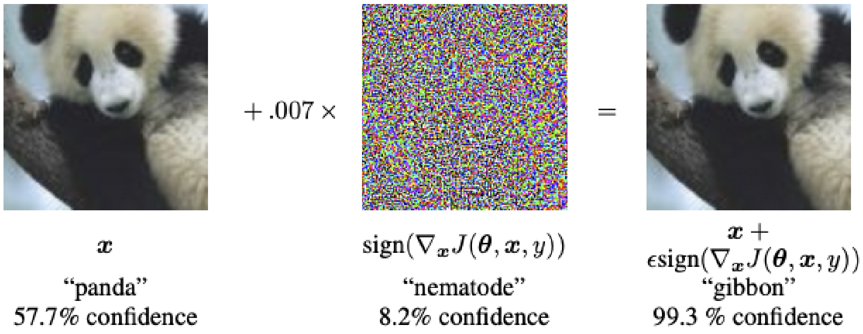
+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2: Adversarial example (from [GSS15]).

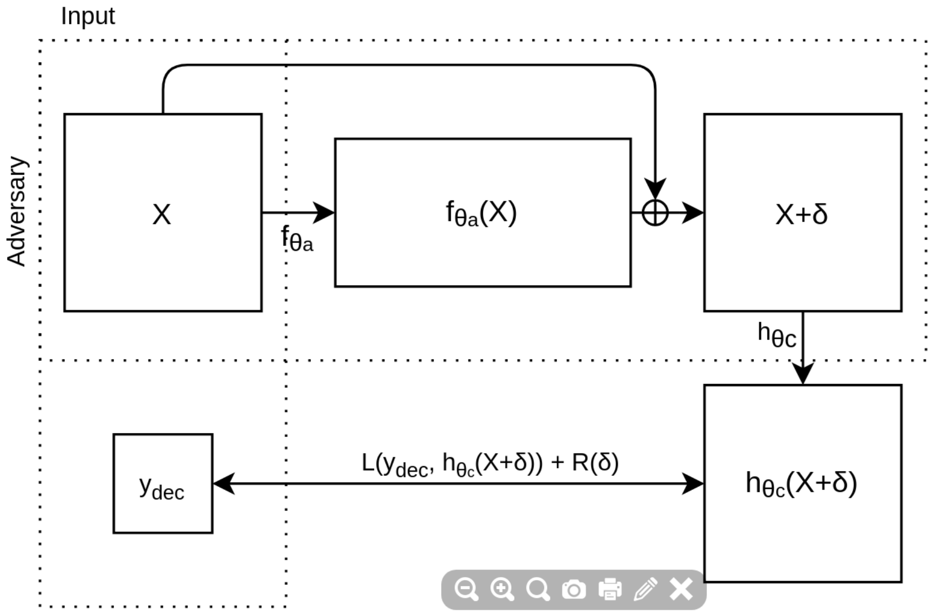

Figure 3: Adversarial model diagram.

6 Submission

Export your Google Colab notebook as a .ipynb file and the weights of each model as a .h5 file and compress everything together with your narrated powerpoint presentation of max 20 min in **one ZIP-file** and submit it (one per group) to Toledo by **Monday 1 June 23:59**.

Make sure that your notebook is self-contained and running smoothly. Test this out yourselves first, from someone else's Google account! It's important that you walk us through all steps of your code and explain in detail the techniques that you apply (why and how). Please make use of the Final Project forum/discussion board on Toledo if you have any questions.

References

[Eve+10] Mark Everingham et al. “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[Cir+12] Dc Ciresan et al. “Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images”. In: *Nips* (2012), pp. 1–9. URL: https://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf.

[Cho+15] François Chollet et al. *Keras*. 2015.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *arXiv:1412.6572 [cs, stat]* (Mar. 2015). arXiv: 1412.6572. URL: http://arxiv.org/abs/1412.6572 (visited on 03/12/2020).

[LSD15] J Long, E Shelhamer, and T Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3431–3440. ISSN: 1063-6919. DOI: 10.1109/CVPR.2015.7298965. arXiv: 1411.4038.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Miccai* (2015), pp. 234–241. ISSN: 16113349. DOI: 10.1007/978-3-319-24574-4_28. arXiv: 1505.04597.