Group assignment 2:
In the name of deep learning

Computer Vision (H02A5a)

In this group assignment your team will delve into some deep learning applications for computer vision. The assignment will be delivered in the same groups from *Group assignment 1* and you start from the provided *template-notebook.ipynb* on Toledo. The notebook you submit for grading is the last notebook you submit in the Kaggle competition prior to the deadline on **Tuesday 24 May 23:59**. Closely follow these instructions for joining the competition, sharing your notebook with the TAs and making a valid notebook submission to the competition. A notebook submission not only produces a *submission.csv* file that is used to calculate your competition score, it also runs the entire notebook and saves its output as if it were a report. This way it becomes an all-in-one-place document for the TAs to review. As such, please make sure that your final submission notebook is self-contained and fully documented (e.g. provide strong arguments for the design choices that you make). Most likely, this notebook format is not appropriate to run all your experiments at submission time (e.g. the training of CNNs is a memory hungry and time consuming process; due to limited Kaggle resources). It can be a good idea to distribute your code otherwise and only summarize your findings, together with your final predictions, in the submission notebook. For example, you can substitute experiments with some text and figures that you have produced "offline" (e.g. learning curves and results on your internal validation set or even the test set for different architectures, pre-processing pipelines, etc). We advise you to go through this document entirely before you really start. It can be a good idea to then go through the template notebook and use this one as your first notebook submission to the competition. You can make use of the *Group assignment 2* forum/discussion board on Toledo if you have any questions. Good luck and have fun!

1 Overview

This assignment consists of *three main parts* for which we expect you to provide code and extensive documentation in the notebook:

- Image classification (Sect. 2)
- Semantic segmentation (Sect. 3)
- Adversarial attacks (Sect. 4)

In the first part, you will train an end-to-end neural network for image classification. In the second part, you will do the same for semantic segmentation. For these two tasks we expect you to put a significant effort into optimizing performance and as such competing with fellow students via the Kaggle competition. In the third part, you will try to find and exploit the weaknesses of your classification and/or segmentation network. For the latter there is no competition format, but we do expect you to put significant effort in achieving good performance on the self-posed goal for that part. Finally, we ask you to reflect and produce an overall discussion with links to the lectures and "real world" computer vision (Sect. 5). It is important to note that only a small part of the grade will reflect the actual performance of your networks. However, we do expect all things to work! In general, we will evaluate the correctness of your approach and your understanding of what you have done that you demonstrate in the descriptions and discussions in the final notebook.

1.1 Deep learning resources

If you did not yet explore this in *Group assignment 1* (Sect. 2), we recommend using the TensorFlow/Keras [Ten; Cho+15] library for building deep learning models. You can find a nice crash course here. The deep learning paradigm is extensive but most of it is out of the scope for this course. Therefore, it is not necessary to explore different optimizers (e.g. SGD, ADAM), regularization mechanisms (e.g. Lp norm, data augmentation), non-linearities (e.g. ReLU, sigmoid), architectures (e.g. U-Net, DeepMedic, ResNet), etc. The main goal of this assignment 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 itself. However, keep in mind that these things can lead to performance gains and as such may help you in the Kaggle competition! With a validated Kaggle account you can make use of GPU accelerated computing as well.

1.2 PASCAL VOC 2009

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). The specific subset of this dataset that will be used during this assignment is made available in the Kaggle competition. Ground truth image labels and segmentation masks are only available for the training set. It should be clear that multiple objects can be present in a single image, but that a single pixel can only belong to one object class. The classification task (Sect. 2) could therefore be interpreted as for each image: "What object classes are present in this image?" The segmentation task (Sect. 3) could be interpreted as for each image and for each pixel: "To what object class does this pixel belong?" Keep in mind that some of these inherent properties of the dataset (e.g. image sizes can be different, more than one object can be present) will influence your work further downstream (e.g. pre-processing, CNN architecture, loss function choice).

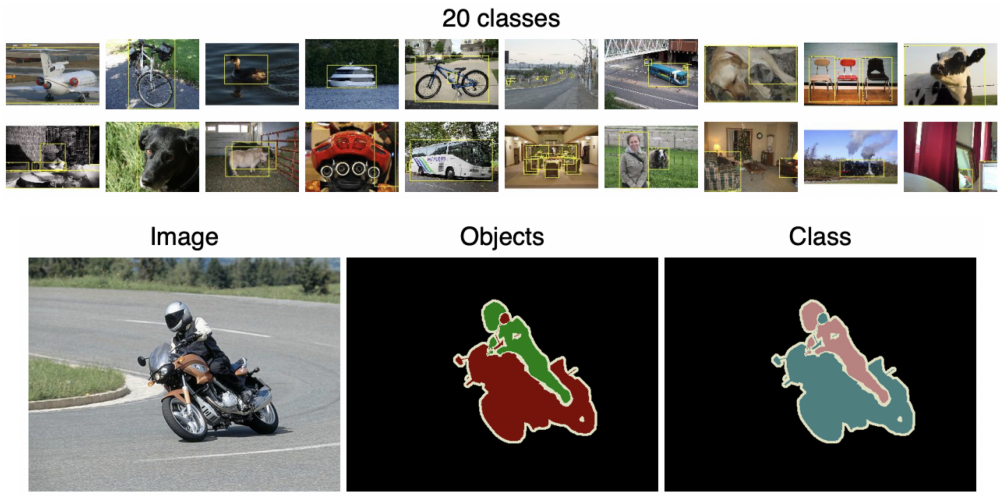

Figure 1: PASCAL VOC 2009 dataset [Eve+10] (source).

1.3 Your Kaggle submission

In order to compete in the Kaggle competition each team should come up with image labels and segmentation maps for the test set during Sect. 2 and Sect. 3. However, to fit Kaggle's internal scoring mechanism, a valid submission should output a CSV file for which a single metric is computed for each row. Therefore, we have provided a *submit_df_to_competition* function in the template notebook, that will convert your predictions to a valid *submission.csv* file. This file now has two rows per example (one for classification and one for segmentation) and the predictions for all the classes are now grouped into a single column. Both for classification and segmentation the predictions are running length encoded and for each we compute the Dice score. Optionally, you can have a look at this function in more detail. Just make sure to call this function once after you have completed Sect. 2 and Sect. 3.

2 Image classification

The goal here is simple: implement a classification CNN and train it to recognise all 20 classes (and/or background) using the training set and compete on the test set. You can choose your own architecture, but we advise you to stick to a network architecture that is known to work. We encourage you to experiment with two types of models: (i) a model where you train all the parameters from scratch; and (ii) a model where 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). When you choose to do so, make sure that none of these pre-trained models have been trained using images from a PASCAL VOC dataset!

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 more context and the learned *features* are/can be more complex (due to successive non-linearities and multiple convolutions per layer). In the final layer, the resulting features are classified, mostly using logistic regression. Elaborate on the design choices of your network architecture, learning parameters and loss function. Visualise the error during training (choose an appropriate metric in line with the competition). Visualize a few classified training/validation/test images. Pick a few good and a few bad cases and discuss these qualitatively, e.g. why are these mistakes happening and what would you try next? Put significant effort in the optimization of your classification CNN so that you obtain competitive performance in the competition! A few guiding questions could be:

- What layers do you use? What is the use of each layer?
- Do you preprocess and/or augment the data in any way?
- How do you establish that the training is converged? Do you split the training data?

3 Semantic segmentation

The goal here is to implement a segmentation CNN that labels every pixel in the image as belonging to one of the 20 classes (and/or background). Use the training set to train your CNN and compete on the test set. We again encourage you to experiment with two types of models: (i) a model that is trained from scratch; and (ii) a model that is trained 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.

A segmentation task differs from classification in that you want to label every single pixel in the image (so not just one label for the 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 more closely follow the encoder-decoder structure, thereby having some internal regularization, 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. Again, elaborate on the design choices of your network architecture, learning parameters and loss function. Visualise the error during training (choose an appropriate metric in line with the competition). Visualize a few segmented training/validation/test images. Pick a few good and a few bad cases and discuss these qualitatively, e.g. why are these mistakes happening and what would you

try next? Put significant effort in the optimization of your segmentation CNN so that you obtain competitive performance in the competition! A few additional guiding questions for segmentation could be:

- What is the difference with a classification setup?
- Do you use skip connections?
- Are there losses specifically useful for segmentation?

4 Adversarial attacks

As you will undoubtedly have noticed in the previous exercises, computer vision has seen substantial performance 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 AI 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.

For this part, your goal is to fool your classification and/or segmentation CNN, using an *adversarial attack*. More specifically, the goal is build a CNN to perturb test images in a way that (i) they look unperturbed to humans; but (ii) the CNN classifies/segments these images in line with the perturbations. As an example, consider in Figure 2 the case where you initially trained a classifier that recognizes animals, but now you want to perturb an unseen panda image, so that it still looks like a panda, but such that the CNN will recognize it as a gibbon. In this case, you can build an encoder-decoder type CNN that learns how to perturb panda images such that you can fool/attack your initial classifier. You are free in choosing your approach. Next, we will give some more introduction and hints for where you could start.

In an adversarial attack, an adversary (attacker) adds noise (perturbation) to an image in an attempt to fool the system under attack. Figure 2 is a famous example of such adversarial input that was generated using the fast gradient sign method. 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

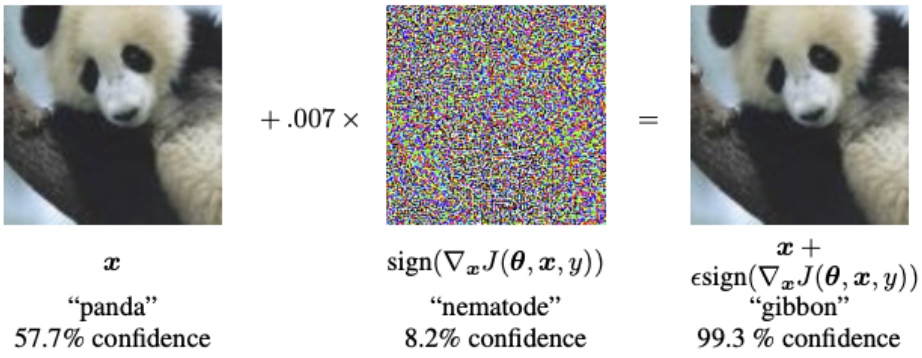

Figure 2: Adversarial example (from [GSS15]).

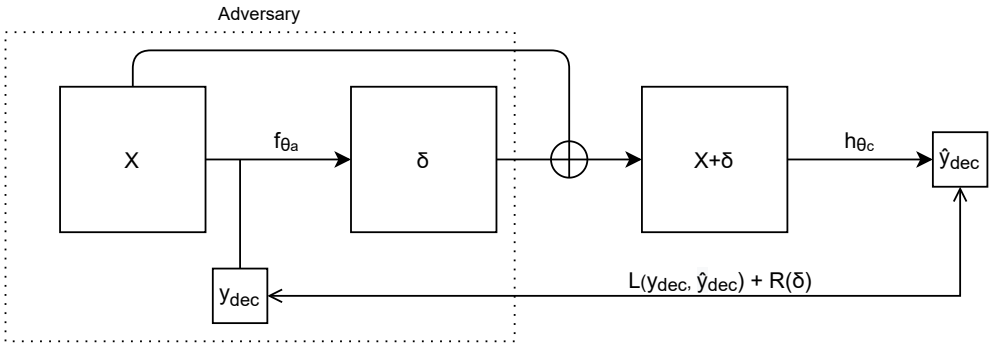

Figure 3: Adversarial model diagram.

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 advise you to 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!). One possibility is to 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 and a desired deceptive label y_{dec} (in case of a segmentation this could be the desired deceptive map Y_{dec}), can generate a perturbation $\delta = f_{\theta_a}(X, y_{dec})$. The generated perturbation is then added to the image and fed into the CNN h_{θ_c} from Sect. 2 or Sect. 3. The goal of your adversary is to generate the perturbation in such a way that the CNN h_{θ_c} assigns a class/label map to the perturbed image $\hat{y}_{dec} = h_{\theta_c}(X + \delta)$ that is different from the ground truth y. An overview of this type of adversarial model is shown in Figure 3. If you look back at the goal for this part in the previous paragraph, it should be clear that the regularization term R and the loss function L will take care of (i) and (ii), respectively.

While you can choose your setup, there are some things that are fixed and should be made very clear in the report:

- Define your task: e.g. we are going to perturb our images such that our classification CNN says there are aeroplanes in it, while there are not; we are going to perturb our images such that our segmentation CNN will segment chairs in it, while there aren't any.
- 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 $\hat{y}_{dec}$. Since you don't want to retrain the original CNN, it is important to freeze its weights θ_c such that they do not change during the training phase of the adversary. Also, it is common to use a regularisation term R which represents the perturbation norm, e.g. l_2 (euclidean distance) or l_∞ (maximum norm), in order to come up with perturbations that are invisible for the naked eye.
- At least give some visual examples of what you are doing, e.g. the test image, the perturbation, the perturbed test image.
- Reflect on your setup/approach: Is this a realistic attack? 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 initial CNN h_{θ_c} is now unreliable? What could we do to increase the robustness of a model under attack in general?

5 Discussion

Finally, take some time to reflect on what you have learned during this assignment. Reflect and produce an overall discussion with links to the lectures and "real world" computer vision. 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.

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.

[Ten] Google Tensorflow. *Tensorflow*. URL: www.tensorflow.org (visited on 05/22/2016).