Second homework Artificial Neural Networks and Deep Learning

In this homework we will tackle two different computer vision challenges: Image classification and image segmentation

Remark: This homework will require GPU compute power. In google colab, in 'runtime', 'change runtime type', you can turn on GPU or TPU support. Choose GPU.
First, you will classify the images of the CIFAR 100 dataset using convolutional networks. CIFAR100 is available as a Keras dataset: https://www.tensorflow.org/api_docs/python/tf/keras/datasets . This dataset contains 100 classes containing 600 images of each. Some of these models may take a long time to train. When your notbook session closes, all variables will be lost, including your models. Retraining every time you restart a session will be a pain, so save often! You can mount your Google Drive in colab and save your model weights:

```
from google.colab import drive
drive.mount('/content/drive')
```

and save and load you models using the functions:

```
model = ...  # Get model (Sequential, Functional Model, or Model subclass)
model.save('/content/drive/path/to/location')
from tensorflow import keras
model = keras.models.load_model('/content/drive/path/to/location')
```

Your final submission should include both code and a written report. You can choose to this separately, or make use of markdown cells in your notebook. If you go for the later, make sure your notebook is easy to follow. You can use different notebooks for the different task (eg. part_1_warmup.ipynb, part_1_scratch.ipynb) if you like. In any case, include relevant numbers and figures in your reports.

Part 1: Image classification

1.1 Warming up:

Use the following different approaches to train a classifier on the CIFAR 100 dataset and *discuss your results*. Feel free to re-use code from the book.

1. Use feature extraction with the VGG16 classifier (as in section 8.3.1). VGG16 is available in Keras https://www.tensorflow.org/api_docs/python/tf/keras/applications . Train a network with Dense layers to classify the images on the extracted features. Iterate at most 3 times on your dense network architecture choice (and shortly explain your choices).
tip: pre-trained networks expect the input images to be preprocessed in a certain way. You can use `tf.keras.applications.vgg16.preprocess_input` *to do this for vgg16 (this function expects the*

images to be in the range of 0-255, so if you rescaled your data already, you should reload the images here)

2. Fine-tune the upper block of the convolutional base of the VGG16 network, as in Listing 8.27 in the book. Add, of course, also a dense classification layer (you can use the same dense architecture from the previous excersise). Discuss the performance.
   Important remark: make sure to change your runtime type to GPU. With a GPU it should only take around 35 seconds per epoch.
   tip: The default learning rate for RMSprop will likely not give you good results. Try lowering the learning rate. (you can play around with the values a bit, but no need for a full hyper-parameter search) Also remember to use preprocessing

   - Does it make sense to use data augmentation for this model? Why? Does it make sense to use data augmentation with the previous method (method 2)?

3. Train the full VGG16 network, i.e., make all the convolutional blocks trainable. Compare the performance.
   tip: lower learning rate and larger batch size may help here as well (but again no need for a full search)

It is possible that after these experiments you did not get good performance out of the VGG models. VGG16 was intended for use with images of size 224x224. The architecture, receptive field and thus the learned features may not perform well on the much smaller CIFAR images. If you up sample the images by for example using `tf.keras.layers.Resizing` it should be easy to achieve much higher accuracies than before. (you can try this if you are curious, but it is not a task for this homework)

1.2 Create your own network from scratch:

Create and train a convnet from scratch and build as good a model as you can! You can take inspiration from excisting architectures, examples from the book etc. but write the code to define all the layers yourself.

Use some of the techniques you have learned in the book and iterate on your design choices to show you know how to build a classifier, you know what to do to prevent overfitting and underfitting, and you should use advanced features such as:

- Keras callbacks:
  - Early stopping and/or checkpointing
  - A learning rate schedule. For example reducing the learning rate when you're on a plateau (note that this also introduces extra parameters that can improve your model)
- Batch normalization
- Depth-wise separable convolutions
- Residual connections
- Model ensembling.
- Data augmentation

Some examples of things you could do and ask yourself:

- What happens if convolutions are replaced with depth-wise separable convolution?
- Does Data augmentation help to improve validation performance?
- Add residual connections. Does this help?
- The model is trained on 32x32 images. Can I finetune my model on larger images for more performance? (as in, load the trained weights of the small resolution network and freeze parts, then tune larger resolutions)
- Would my architecture handle more classes?
- After having iterated over several design choices, are their combined predictions better than a single model?

Make at least 3 iterations of your model. Be sure to discuss your results and comment on your design choices

Part 2: Image segmentation

For this part, your task will be to extract and segment objects from an image using the PASCAL VOC-2012 dataset. This dataset contains 20 different classes, however, we will only do foreground vs background segmentation to make the task easier (if you like a challenge, you can choose to segment the separate classes.)

You should start by downloading the data from the site or an alternative download. Explore the data a bit by looking at the description and plotting a few images. A separate notebook is also provided on BlackBoard to help you get the images and labels loaded and into a TensorFlow `tf.data.Dataset` object.

Build a segmentation model. The image segmentation example in your handbook is a good starting point. (you will want to lower the learning rate). Alternatively you could check out the modified U-Net architecture used in the TensorFlow image segmentation tutorial

Visualize your segmentation output and compare with the ground truth reference. Reflect on what metrics would be a good choice to evaluate a segmentation. For example, Dice score is commonly used in (medical) image segmentation. What other metrics can you use?

Replace the `Conv2DTranspose` used to upsample with a bilinear interpolation using `UpSampling2D` followed by a standard convolution. Is there any performance difference? Do you notice a visual difference in the segmentation masks produced?