Introduction to Deep Learning

Assignment 2: building MLPs, CNNs and generative models with TensorFlow

October 2022

Introduction

In this assignment you are going to use Keras/Tensorflow to build various deep neural network models for classification, regression and generative tasks on image datasets. The key objectives are:

- Learning how to use Keras and TensorFlow APIs.
- Gain practical experience by comparing various MLP and CNN architectures, getting some intuitions for tuning hyperparameters
- Apply this knowledge to develop a CNN for the "tell-the-time" problem.
- Train generative models on an image dataset of your choice, perform experiments on the learned latent spaces.

Tips:

1. *We suggest all members of a group to start with task 1 instead of splitting individually as this experience will prove useful for the remaining parts of the assignment.*
2. *If you do not have a machine with a dedicated NVidia graphics card, we suggest using Google Colab or DeepNote with GPU acceleration enabled to complete this assignment. Since training multiple models will take time, we would also advise you to start training models early.*

Task 1: Learn the basics of Keras API for TensorFlow.

Start with reading the section "Implementing MLPs with Keras" from *Chapter 10 of Geron's textbook (pages 292-325)*. Then install TensorFlow 2.0+ and experiment with the code included in this section. Additionally, study the official documentation `https://keras.io/` and get an idea of the numerous options offered by Keras (layers, loss functions, metrics, optimizers, activations, initializers, regularizers). Don't get overwhelmed with the number of options – you will frequently return to this site in the coming months. Your tasks:

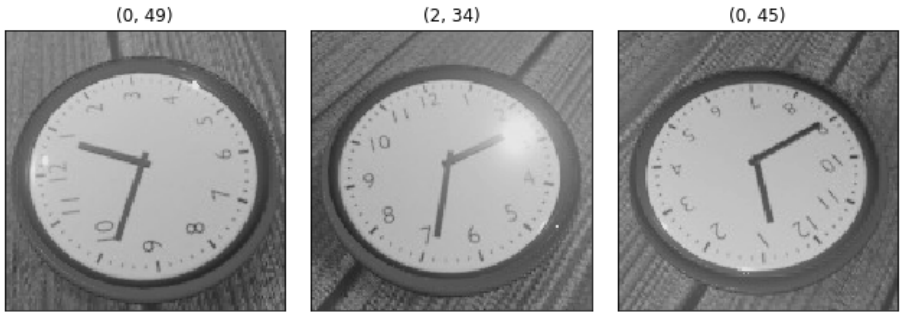
1. Check out this official repository with many examples of Keras implementations of various sorts of deep neural networks here. We recommend cloning this repository and try to get some of these examples running on your system (or Colab/DeepNote). In particular, experiment with *mnist_mlp.py* and *mnist_cnn.py* scripts which show you how to build simple neural networks for the MNIST dataset (useful for the next task).
2. Next, take the two well-known datasets: Fashion MNIST (introduced in Ch 10, p. 295) and CIFAR-10. The first dataset contains 2D (grayscale) images of size 28x28, split into 10 categories; 60,000 images for training and 10,000 for testing, while the latter contains 32x32x3 RGB images (50,000/10,000 train/test). Apply two reference networks on the fashion MNIST dataset: a MLP described in detail in *Ch. 10, pp. 297-307* and a CNN described in *Ch. 14,*

p. 447. Experiment with both networks, trying various options: initializations, activations, optimizers (and their hyperparameters), regularizations (L1, L2, Dropout, no Dropout). You may also experiment with changing the architecture of both networks: adding/removing layers, number of convolutional filters, their sizes, etc. After you have found the best performing hyperparameter sets, take the 3 best ones and train new models on the CIFAR-10 dataset, see whether your performance gains translate to a different dataset. Provide your thoughts on these results in the report.

The purpose of this task is NOT to get the highest accuracy (which is usually the most important performance measure). Instead you are supposed to gain some practical experience with tuning networks, running multiple experiments (with help of some scripting), and, very important, documenting your findings. In your report you have to provide a concise description of your experiments, results, conclusions. The quality of your work (code and report) is more important than the quantity or the accuracy you've achieved.

Task 2: Develop a "Tell-the-time" network.

We have produced a big collection of images of an analog clock along with labels corresponding to the time shown on these images. Your task it to develop and train a CNN that would tell the time as correctly as possible. Here is a random sample of 3 images from our collection with the corresponding time (hour, minutes) displayed on top of these images:



The dataset can be downloaded from here and it consists of 18000 grayscale images (18000x150x150) contained in 'images.npy'. The labels for each sample are represented by two integers (18000x2, 'labels.npy' file), that correspond to the hour and minute displayed by the clock. You can see that each image is taken from a different angle, rotation and they might contain light reflections from within the scene making this a non-trivial problem.

1. The problem of correctly telling the time can be formulated either as a multi-class classification problem (for example, with 12x60 classes) or a regression problem (for example, predicting the number of minutes after 12 o'clock). Therefore, your goal is to come up with different representations for the labels of your data and adapting the output layer of your neural network and see how it impacts the training time and performance. No matter which architecture and loss function you will use, when reporting results also provide "common sense" accuracy: the absolute value of the time difference between the predicted and the actual time (e.g., the "common sense" difference between "predicted" 11:55 and the "target" 0:05 is just 10 minutes and not 11 hours and 50 minutes!). Minimizing this "common sense" error measure is the main objective of this assignment! Notice that it is a common situation in Machine Learning: we often train models using one error measure (e.g., cross-entropy) while the actual error measure that we are interested in is different, e.g., the accuracy (the percentage of correctly classified cases). Here are some ideas that you should experiment with when building your models:

    - **Regression** - try to build a network that predicts the time using a single output node in the following format: ["03 : 00" $\rightarrow y = 3.0$]; ["05 : 30" $\rightarrow y = 5.5$]. What kind of problems does such a representation have?

- **Classification** - treat this as a n-class classification problem. We suggest starting out with a smaller number of categories e.g. grouping all the samples that are between $[3:00 - 3:30]$ into a single category (24 categories in total), and try to train a well performing model. Once you have found a working architecture, increase the number of categories by using smaller intervals for grouping samples. Can you train a network using all 720 different labels? What problems does such a label representation have?
- **Multi-head models**: your neural network can have multiple output heads e.g. one head for predicting hours and another head for predicting minutes. These heads could also potentially have different losses and targets (e.g. multiclass for hours and regression for minutes). Implement such a model and see how the performance of such a model compares to the previous models. How would you explain the differences?

  Note: more information on how to build multi-head models can be found at Ch 10, p. 308-310 of the textbook.
- **Label transformation (optional)**: think how the labels could be reformulated by using periodic functions (e.g. sine and cosine functions to represent the angles on the unit circle). How would you adapt your neural network to these kinds of labels? How many output nodes would be sufficient and what activation function would be perfect for this task? Note: you can achieve the lowest 'common sense error' by using this approach.

2. Use the knowledge gained by working with other datasets in the previous parts of this assignment to optimize your final models and decrease the error of telling the time as much as possible (common sense error of below 10 minutes is achievable using relatively simple CNN architectures). You should also compare the different ways of representing your labels and different neural network output layer combinations. You should use a 80:20% ratio for the train/test sets respectively. Document your experiments and findings in the report.

Task 3: Generative Models

The goal of this task is to learn training and using generative models for image-related tasks. You are provided with a Jupyter notebook that illustrates how you would build, train and use Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). You can download the dataset that was used in the notebook here. Your main goal in this task will be to find your own image dataset, preprocess it and retrain these networks, and use the models to generate novel content.

State of the art generative models can create an endless stream of photorealistic images of whatever data they are trained on (e.g. *StyleGAN2* - image below, now text-conditioned diffusion models like DALL-E, Midjourney, etc.). However, such enormous neural nets are very expensive and difficult to train, therefore it is better to start out small and understand the working principles behind simpler generative architectures.

Figure 1: *Output of StyleGAN2.*

Your tasks are as follows:

1. Go through the provided notebook and try to understand how the generative models are built using Keras. To start with you are provided with a convolutional autoencoder (CAE) architecture which contains the two main building blocks - the encoder and decoder. The Variational autoencoder (VAE) is built in a very similar way, but includes a sampling layer that gives this model its generative capabilities. The generator and discriminator in the GAN model are also build using these similar building blocks, however they are arranged differently and use a very different loss function.

2. Generally, training generative models is a tricky process, especially when your dataset includes a very diverse set of samples. That is why datasets of faces are often used to showcase the power of generative models as they are usually very uniform (people facing the camera at a similar distance and angle). However there are definitely many other interesting and viable datasets, use cases and applications. Your task is to find a new image dataset of your choice online and use the code provided in the notebook to retrain your generative models (both VAE and GAN). You should provide a link to the data that you used in the report along with its description.

3. To properly train the models you will probably need to rescale your data to be close to the size of the example dataset (64x64x3). Since all of the neural networks in the notebook are build using functions *build_conv_net()* and *build_deconv_net()* you can also adjust the convolutional architectures if you desire - both to more closely match the scaling of your data and reduce/increase the complexity of your models.

4. Once your models are trained, their latent spaces capture the distribution of the data that they were trained on. You can use randomly generated vectors as input to the decoder components of these models to generate novel images (the decoder in case of VAE, generator in case of GAN). Each vector represents a point in this latent space and since it is continuous, you can generate various interesting visualizations. Your task is to create one of these visualizations by linearly interpolating between two random points in the latent space and seeing how the output of your network changes.

5. In your report provide a detailed description of the working principles of these generative models and emphasize the differences between them. You should also include the modified notebook with your visualisations when submitting the assignment.

Deliverables: your submission should consist a single *.zip* file containing your report in *pdf* format (at most 10 pages) and neatly organized code (Jupyter notebooks) so that we could reproduce your results. You can find more information about report writing on Brightspace → Assignments section.

Deadline: 25/11/22, 23:30.