# Deep Learning | Assignment 2

Aram Davtyan aram.davtyan@unibe.ch
Sebastian Stapf sebastian.stapf@unibe.ch

**Due date: May 26, 23:55**

## Assignment

In this assignment you are asked to solve the problem of Image Captioning. You are given an existing code base with the baseline model already implemented for you. The task consists of multiple iterations of improving the code in order to achieve better performance. The task is considered to be completed when both the code of the models and a Jupyter Notebook with the results are submitted to ILIAS. Here are the steps to guide you through the assignment:

1. Read the summary of the assignment in the Image Captioning section.
2. Look through the Working With the Code Base section for information on how to install all the necessary components (either you use Cluster (UBELIX), Google Colab, or CPU Only On a Local Machine) and start training (Running the Code).

   Please look through If Something is Wrong during installation and code running.
3. Check the Models [60 points] section to see what models you need to implement, train and evaluate. Train your models as described in the Running the Code section and evaluate them following the Evaluation section.

   Training a single model takes several hours. Do not leave everything to the last minute.
4. Compose a Jupyter Notebook with the report following the Report [40 points] section.
5. Prepare the submission as described in the Preparing Your Submission section.
6. Submit the prepared zip file to ILIAS as described below.

## Submission

Upload the zip file to ILIAS → Assignments | Solutions → Assignment 02 → Assignment 02. The submission should contain:

- models folder with the code of the models in files `model_*.py`

  You are not allowed to change the code in other files except for the `model_*.py` ones.
- configs folder with the corresponding model configurations in files `config_model_*.yaml`
- report with analysis of your models in the `report.ipynb` file
- other files utilized in the report in the `report_utils` folder

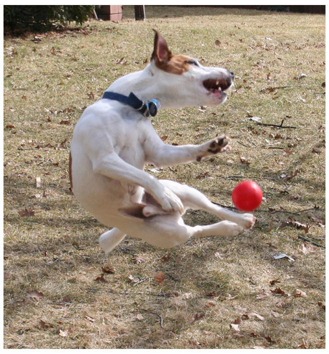
⚠ Notice, that you are not asked to submit the `.pth.tar` checkpoint files. However, please, keep them handy, as you may be asked to provide them later.

**Late submission:** upload to ILIAS → Assignments | Solutions → Assignment 02 → Assignment 02 late

## Image Captioning

The task is to develop a model that generates a short description (caption) for a given image.

GT: A snowboarder sits on a slope with skiers and boarders nearby .
GT: A snowboarder takes a rest on the mountainside .
GT: Snowboarders sitting in the snow while skiers take the hill .
GT: The snowboarder is sitting down .
GT: "Two skiers stand , two sit on slopes ."



GT: A dog jumps for a red ball on the grass .
GT: A white dog is jumping in the air attempting to catch a red ball .
GT: A white dog jumping to catch a red ball .
GT: A white dog with a blue collar plays with a red ball .
GT: The dog is playing with a red ball .

Fig. 1. Images and corresponding ground truth captions from the Flickr8k dataset.

You will go through all the steps of improving the model starting from an existing code base – a common path for deep learning engineers, both in industry and academia. You are provided with a baseline and need to implement several modifications that can potentially lead to better performance of the model.

**Dataset.** The model will be trained on Flickr8k, a dataset containing 8k images and corresponding captions (see Fig. 1). Each image in the dataset has 5 ground truth captions, allowing for the training of a more flexible model in terms of the diversity of the generated captions.

**Metric.** In order to measure the quality of a model in the case of multiple ground truth references, a common approach is to use the BLEU score.

What is the BLEU metric?

**Model.** You are asked to base your model on an encoder/decoder (generator) architecture. The model will be trained using Cross Entropy Loss for the next token prediction.

**Encoder.** Given an image $x \in \mathbb{R}^{C\times H\times W}$, where $C$ is the number of channels in the image ($C = 3$ in our case), and $H \times W$ is the resolution of the input, the encoder first encodes it to some latent representation

- $z \in \mathbb{R}^{d}$ for the vector case,
- $z \in \mathbb{R}^{d\times h\times w}$ for the feature map case.

**Decoder (Caption Generator).** This is followed by the decoder that maps the latent image representation $z$ to a sequence of tokens that can be translated into English words.

To the Assignment ↑

## Working With the Code Base

### Google Colab

1. Copy the code and the Flickr8k dataset zip files to your Google Drive.
2. Copy `colab_run.ipynb` to your Google Drive.
3. Open `colab_run.ipynb` on your Drive and select the runtime type:

   ```
   Runtime --> Change runtime type --> select CPU or any available GPU --> save
   ```

4. Follow the cells (run the `colab_run.ipynb`).
5. See how to run the code for training in Running the Code and for evaluation in Evaluation.

### Cluster (UBELIX)

UBELIX is the University of Bern's SLURM-based high-performance computing cluster. SLURM is widely used in research and industry to manage and schedule computational workloads. Learning SLURM is essential for careers in research, data science, machine learning, and related fields.
**Useful Links:** First Steps Guide | Quickstart for Running Jobs

**Setup Instructions:**

1. **Access the UBELIX Cluster**
   Follow the instructions on Access Ubelix HPC to activate your UBELIX account. Once your account has been created, you can try to connect to a login node using: Replace `<user>` with your HPC account and make sure you are connected to the university network.

   ```
       ssh <user>@submit03.unibe.ch
   ```

   Login nodes `submit01`, `submit02`, `submit03`, and `submit04` are available.

2. **Copy assignment and dataset to the cluster** From your local computer run:

   ```
   scp assignment_2.zip <user>@submit03.unibe.ch:./
   scp flickr8k.zip <user>@submit03.unibe.ch:./
   ```

3. **Connect to your node**

   ```
   ssh <user>@submit03.unibe.ch
   ```

4. **Unzip the files**

   ```
   unzip assignment_2.zip
   unzip flickr8k.zip -d ./assignment_2
   ```

5. **Remove the zip files (optional)**

```
rm assignment_2.zip
rm flickr8k.zip
```

6. **Set up the Conda environment**

```
cd assignment_2/
module load Anaconda3
eval "$(conda shell.bash hook)"
conda env create -f environment.yaml
```

If GPU support is needed, this must be done with a GPU visible on the machine (e.g. on a SLURM cluster within a GPU interactive job).

7. **Running and submitting jobs**

To run jobs on UBELIX, you must first allocate a node. Find more information about the available resources at: UBELIX partitions and more information about running SLURM jobs at SLURM quickstart.

**Interactive Jobs:**

- CPU only:

```
srun --pty bash
```

- With GPU (e.g., 1 RTX 3090):

```
srun --partition=gpu --gpus-per-node=rtx3090:1 --pty bash
```

After allocation, you can proceed as normal:

```
module load Anaconda3
eval "$(conda shell.bash hook)"
conda activate dl_a2
python3 some_script.py
```

**Batch Jobs:**

For longer jobs where you do not need to stay logged in, use `sbatch`.
Example `your_job_script.sbatch` script:

```
#!/bin/bash
#SBATCH --job-name="Python example"           # name of the job
#SBATCH --time=02:00:00                       # maximum runtime (e.g. 2 hours)
#SBATCH --ntasks=1                            # number of tasks (processes)
#SBATCH --cpus-per-task=8                     # allocates 8 CPU cores per task
#SBATCH --mem-per-cpu=2G                      # 2 GB RAM per CPU core
#SBATCH --partition=gpu                       # GPU or CPU partition
#SBATCH --gpus-per-node=rtx3090:1             # which GPUs to allocate
```

```
# Commands to execute:
module load Anaconda3
eval "$(conda shell.bash hook)"
conda activate dl_a2
python3 some_script.py
```

Submit your job using:

```
sbatch your_job_script.sbatch
```

You can check the status of your job with:

```
squeue --me
```

8. **Download the English language tokenizer**
   With either an interactive job or sbatch you can run:

```
python -m spacy download en_core_web_sm
```

## CPU Only On a Local Machine

1. Put the code and the Flickr8k dataset zip files next to each other.
2. Unzip:

```
unzip assignment_2.zip; unzip flickr8k.zip -d ./assignment_2
```

3. Create a conda environment with the name `dl_a2` and install all packages:

```
cd assignment_2/
conda env create -f environment_cpu.yaml
```

4. Download the English language tokenizer:

```
conda activate dl_a2
python -m spacy download en_core_web_sm
```

## If Something is Wrong

1. Create a conda environment on your own and then activate it:

```
conda create -n dl_a2 python=3.11
conda activate dl_a2
```

2. Run the training code (see how to do this in Running the Code).
3. Install all missing libraries one by one.

To install the correct version of PyTorch, select your OS, package, and computing platform, then use the command provided.

## Running the Code

To train a model you need to run the `train.py` file with proper command line arguments:

```
python train.py
```

| cmd arg | description | default |
|---|---|---|
| -d, --device-id | cuda device id (assigned GPU number) | |
| --experiment-name | unique experiment name | |
| --num-epochs | number of epochs | |
| --num-workers | number of data loading workers | 8 |
| --seed | seed for reproducibility | 0 |
| --no-log | disable logs | |
| --resume | resume training | |

For example, this command will launch the training of the model_1 on GPU 0 for 2 epochs

```
python train.py --device-id=0 --config-file-path=./configs/config_model_1.yaml \
--experiment-name=model_1 --num-epochs=2
```

Each training will create

- a folder named `<experiment name>` in the `checkpoints` directory where you can find the `config.yaml` file and **the latest** model checkpoint `model.pth.tar` for this experiment,
- a folder named `<experiment name>` in the `logs` directory where you can find wandb logs for the corresponding experiment.

  ⚠ You need to sign up on the Weights & Biases platform to use the wandb logger!

To disable the wandb logging, you may add the `--no-log` argument:

```
python train.py --device-id=0 --config-file-path=./configs/config_model_1.yaml \
--experiment-name=model_1 --num-epochs=2 --no-log
```

You can resume the training from the saved model weights by adding `--resume`:

```
python train.py --device-id=0 --config-file-path=./configs/config_model_1.yaml \
--experiment-name=model_1 --num-epochs=2 --resume
```

## Evaluation

Once the model is trained, it can be evaluated on the entire validation set:

```
python evaluate.py -d 0 --checkpoint-path=./checkpoints/model_1/model.pth.tar
```

As a result, a file named `bleu_scores.txt` is created in the `<experiment name>` folder in the `checkpoints` directory. This file contains the BLEU scores that you need to include in the report.

## Preparing Your Submission

1. Make sure you have the code and the configs for your **best** models in files `model_*.py` and `config_model_*.yaml` respectively.

2. Create a zip file:

```
cd assignment_2
zip -r submission.zip models/model_*.py configs/config_model_*.yaml \
report.ipynb report_utils
```

3. Download the zip file from the cluster to your local machine:

```
scp <user>@cluster.inf.unibe.ch:./assignment_2/submission.zip ./
```

## Models [60 points]

In this part of the assignment, you need to implement and train the models in the `model_*.py` files. Write clean and readable code, do not avoid commenting your steps. You can use built-in PyTorch methods and classes in your implementation.

Each model is supposed to be an improvement on the previous one.
So, it is better to go step by step from Model 1 to Model 6.

For this assignment training your models for more than 100 epochs is NOT recommended.

### Model 1 – Baseline [3 points]

The baseline consists of a simple convolutional image encoder followed by 2 RNN layers stacked vertically one on top of another. The RNN decodes the image code into a sequence of word tokens by treating the image code as the first input token (similar to a `<SOS>` token). The hidden state of the RNN is initialized with zeros. For more details refer to the code.

**To Do** The first task is to get familiar with the code base by training the baseline and calculating its BLEU scores (evaluating it on the validation set).

### Model 2 [6 points]

Although the baseline provides a simple and logical encoder/decoder architecture, it is often difficult to train such a system end-to-end. At the beginning of the training the image encoder does not extract useful information from the input image and the RNN has to hallucinate the output almost from nothing. Moreover, the training signal from the loss has to pass through all recurrent decoding iterations to reach the encoder. This complicates the training of the encoder and hence the whole model.

To overcome this issue, following the good practices of representation and transfer learning, we propose to substitute the image encoder with a pre-trained network. Although this network can be further fine-tuned, this is not necessary, and it can be frozen for simplicity. As the image backbone, we suggest using DINOv2, as it is proven to have learned a great image representation.

**To Do** Substitute the image backbone with DINOv2. Use the final `<CLS>` token from the DINOv2's image representation as your image encoding. Freeze the backbone and train the decoder from scratch.

Notice that thanks to the modular implementation, to solve this task, it is sufficient to change only the encoder's code.

Explore the official DINOv2's GitHub to see how to load and use the pre-trained models. Tutorial 10 may also help you with this.

To the ☰ Assignment ↑

## Model 3 [6 points]

We know that DINOv2 provides a great signal. So, in order to improve the model, we need to modify the decoder. A simple RNN seems not a great choice. What about one of the gated recurrent counterparts discussed in the lectures and the tutorials?

**To Do** Given the encoder from Model 2, replace the RNN in the decoder with a gated recurrency (LSTM, GRU, *etc.*). Train the model and try different number of layers.

## Model 4 [12 points]

So far the decoder was conditioned on the output of the encoder by feeding it as the first input token to the RNN. This complicates the task for the RNN, as it has to distinguish the cases when the input is a word token and when the input is an image encoding. Are there other ways to condition the decoder on the encoder's output?

**To Do** Propose a different scheme for conditioning the decoder on the image encoding and train the model.

💡 You will be asked to justify your idea in the report.

## Model 5 [15 points]

Who are we trying to fool by training an RNN for language modeling in 2025?

**To Do** Replace the RNN with a Transformer and train the model. Be careful with how you condition the transformer on the image encoding.

💡 You may use the TransformerEncoderLayer class to build the decoder network.

## Model 6 [18 points]

Along with the `<CLS>` token, DINOv2 also produces spatial tokens that capture more fine-grained and local information in the image. These local features may be a better fit for our model in contrast to the global `<CLS>` token. However, now instead of one encoding per image, we have a sequence of image codes, from which we need to decode the caption. Does this remind you of anything from the lectures/tutorials? What if we could attend to those tokens during the decoding procedure?

**To Do** Replace the `<CLS>` token of DINOv2 with the spatial features. Condition on those tokens with the Cross-Attention procedure. Train the model.

💡 You may use the TransformerDecoderLayer or the MultiheadAttention classes to build the decoder network.

---

For the reference, to know how well you are doing, by this time the TAs could train a model that achieves the following BLEU scores:

```
BLEU_1: 0.67 | BLEU_2: 0.50 | BLEU_3: 0.36 | BLEU_4: 0.26
```

Can you do better?

To the Assignment ↑

## Report [40 points]

For the report you need to

- create the `report.ipynb` Jupyter Notebook in the code directory,
- put all the files necessary for the report in the `report_utils` folder.

⚠ Make sure that a TA can run your Jupyter Notebook in a single step by clicking `Run All`!

### Loss Curves [5 points]

**[3 points]** Plot training and validation loss curves for all the models.

- You can download loss values in `.csv` file from Weights & Biases.
- Choose experiments that are reasonable to show.
  Your plots should not be cluttered.

**[2 points]** Describe the training process and compare the curves of all models.

### BLEU Scores [5 points]

**[3 points]** Create a table with the BLEU scores for all the models using markdown.

- As a result of running `evaluation.py`, you will find a file named `bleu_scores.txt` in the `<experiment name>` folder in the `checkpoints` directory.
- Markdown Guide

**[2 points]** Compare the results of all the models. Do you see a correlation between the scores and the train/validation loss curves?

### Captions Generation [10 points]

In the folder `test_examples` in the `flickr8k` directory you can find 5 test images and their ground truth captions.

**[7 points]** Display those images, their ground truth captions and captions generated by all 6 models (choose the best model for each of model 1, . . . , model 6).

**[3 points]** Compare the generated and the ground truth captions of all models. Do you see a correlation between the BLEU scores and the generation quality?

- You can look through the code (*e.g.*, `train.py`) to see how to create the Model class instance and to load its weights.

### Model 4 [5 points]

Justify your idea of a different scheme for conditioning the decoder on the image encoding. What have you tried? What works and what does not?

To the Assignment ↑

## Model 5 [5 points]

Describe how you conditioned the transformer on the image encoding. What is the key difference between transformers and RNNs that gives you a performance boost?

## Model 6 [10 points]

In the `test_examples` folder in the `flickr8k` directory you can find 5 test images.

Visualize the attention maps of the generated tokens (words) to the image tokens for all test images to show which pixels the generator pays attention to when predicting the next token.

For all test examples display the original image with the generated caption and 3 attention maps with their corresponding tokens.