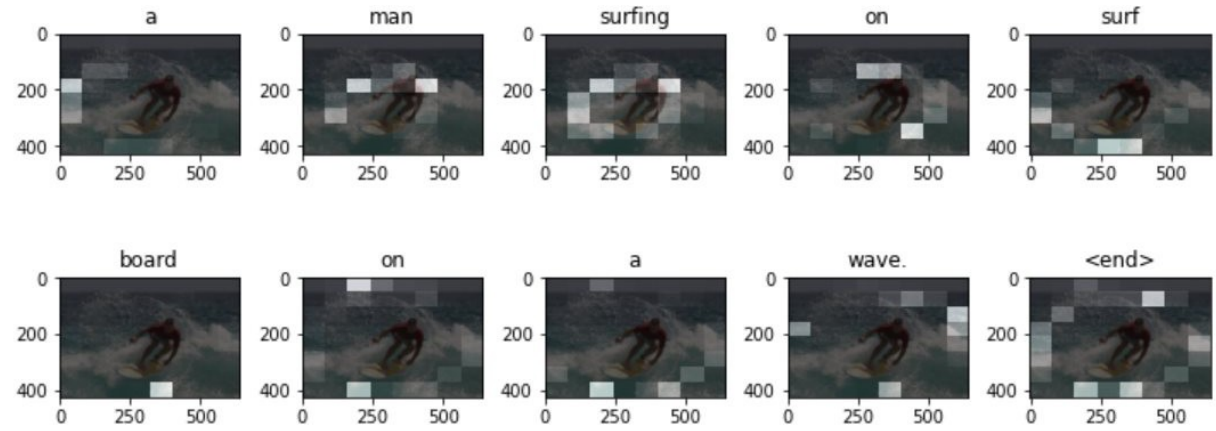
Task 3: Deep Learning

image captioning

The topic of the 3rd assignment of the course is Deep Learning. We will study a problem that combines Computer Vision and Natural Language Processing. Specifically, we will build a neural network for generating verbal descriptions from images (Image Captioning).

The starting point will be the official TensorFlow tutorial (and notebook) "Image captioning with visual attention". However, we will work on another dataset and try to improve the tutorial in several places.

Finally, there will be an optional possibility to submit predictions on control data without verbal descriptions to participate in a small in-class competition without any scoring significance.



Open the notebook in Colab so you can see all the cells. In the web version you need to "Toggle code" and "Toggle section".

Data set

The most widely used datasets in Image Captioning are Flickr8k, Flickr30k, and COCO. The TensorFlow example uses Flickr8k and Conceptual Captions. We will use "flickr30k-images-ecemod", a split of Flick30k specifically for our course.

The flickr30k-images-ecemod data is as follows:

- an "image_dir" folder with 31,783 images from Flickr
- a file "captions_new.csv" with 148.915 captions for the images of "image_dir" a file
- "train_files.csv" list of 21.000 images that make up the training set a file
- "test_files.csv" list of 4.524 images that make up the test set

Monitoring of data

flickr30k-images-ecemod has a similar organization to COCO. Each image has 5 captions made by different people via Amazon's Mechanical Turk service. An example:

Example image: _100007487.jpg

```
_100007487.jpg#0    A young child walks down a gravel path lined with a row of red outdoor chairs .
```

```
_100007487.jpg#1    A racetrack with red chairs stacked beside a fence with a walking .
```

```
_100007487.jpg#2    A child in a striped shirt walks by some red chairs.
```

```
_100007487.jpg#3    A child walking and leaving a trail behind them.
```

```
_100007487.jpg#4    A little kid is walking next to red banners.
```

Each caption has three fields, the name of the image file, the serial number of the caption and finally the caption itself.

Download the dataset

With the code below, we will download our dataset, replacing the "Choose a dataset" and "Download the dataset" sections. The previous cells with installations and imports need to be run.

In the next cell we download the images of the dataset:

```
# Download image files
image_zip = tf.keras.utils.get_file('flickr30k-images-ecemod.zip',
                                    cache_subdir=os.path.abspath('.'),
                                    origin='https://spartacus.1337.cx/flickr-
mod/flickr30k-images-ecemod.zip',
                                    extract=True)
os.remove(image_zip)
```

and with the next one "captions_new.csv", "train_files.csv" and "test_files.csv":

```
# Download captions file
captions_file = tf.keras.utils.get_file('captions_new.csv',
                                        cache_subdir=os.path.abspath('.'),
                                        origin='https://spartacus.1337.cx/flickr-
mod/captions_new.csv',
                                        extract=False)

# Download train files list
train_files_list = tf.keras.utils.get_file('train_files.csv',
                                        cache_subdir=os.path.abspath('.'),
                                        origin='https://spartacus.1337.cx/flickr-
mod/train_files.csv',
                                        extract=False)

# Download test files list
test_files_list = tf.keras.utils.get_file('test_files.csv',
                                        cache_subdir=os.path.abspath('.'),
                                        origin='https://spartacus.1337.cx/flickr-
mod/test_files.csv',
                                        extract=False)
```

With the following code we organize the filenames and captions into lists and prepare the train and test sets for TensorFlow.

```
path="."
IMAGE_DIR="image_dir"
path = pathlib.Path(path)

captions = (path/captions_file).read_text().splitlines()
captions = (line.split('\t') for line in captions)
captions = ((fname.split('#')[0], caption) for (fname, caption) in captions)

cap_dict = collections.defaultdict(list)
for fname, cap in captions:
  cap_dict[fname].append(cap)

train_files = (path/train_files_list).read_text().splitlines()
train_captions = [(str(path/IMAGE_DIR/fname), cap_dict[fname]) for fname
in train_files]
```

```
test_files = (path/test_files_list).read_text().splitlines()
test_captions = [(str(path/IMAGE_DIR/fname), cap_dict[fname]) for fname
in test_files]

train_raw = tf.data.experimental.from_list(train_captions)
test_raw = tf.data.experimental.from_list(test_captions)
```

After this code, you can continue from the cell

```
train_raw.element_spec
```

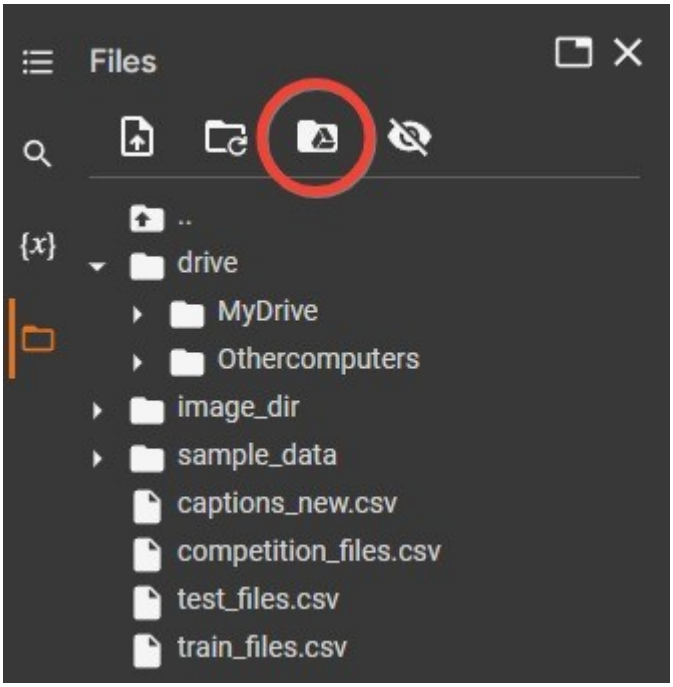
of the example, now working with our dataset.

Google Drive mount (optional)

Because downloading the images takes a few minutes, it would be a good idea to have them permanently in a Google Drive directory that you mount inside the notebook.

1. Create a new Gmail account to have space.
2. Download the images, unzip them and upload the "image_dir" folder to Google Drive.
3. Within the notebook, use the button on the left sidebar to enter the code you need to run to mount Google Drive. Drive will appear in the sidebar if you refresh. By hovering over the files and folders, a hamburger button appears that allows you to copy their path.

Google Drive mount



4. If "image_dir" is at the root of the Drive then you can modify the previous variables `train_captions` an `test_captions` as follows:
   d

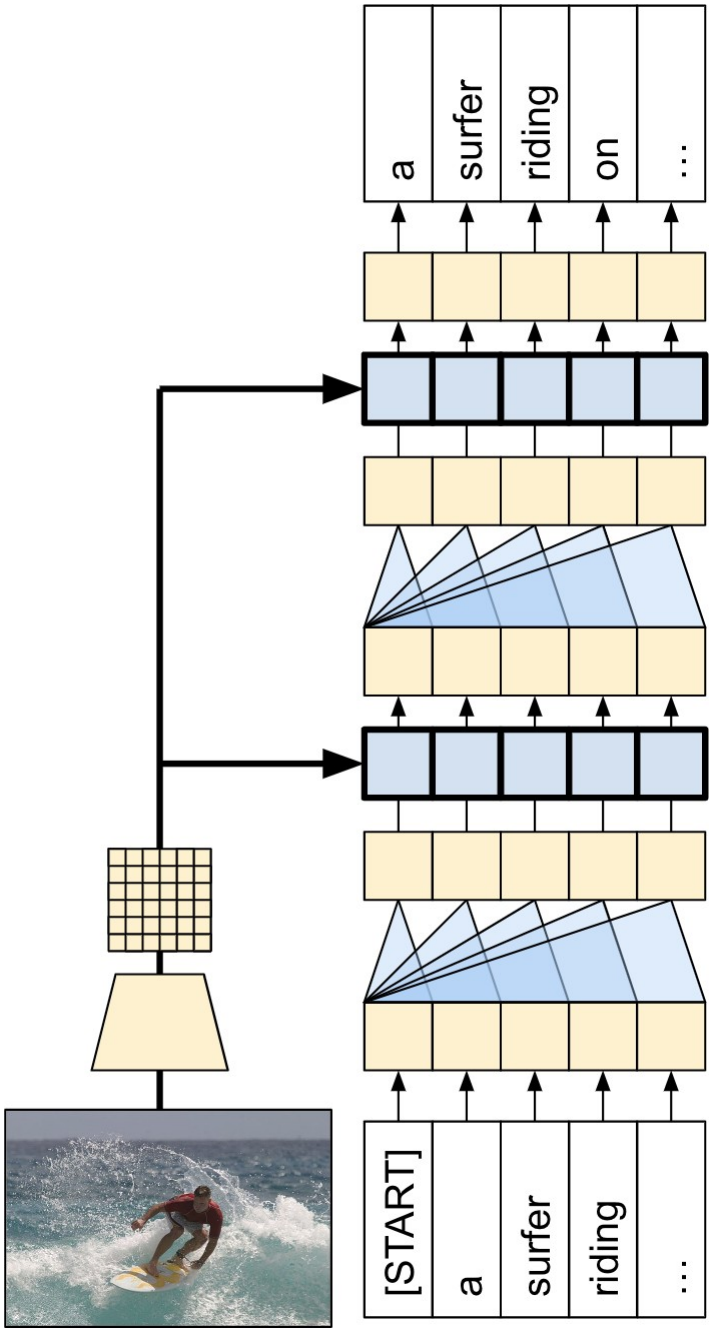
```
new_path = '/content/drive/MyDrive'
new_path = pathlib.Path(new_path)

train_captions = [(str(new_path/IMAGE_DIR/fname), cap_dict[fname]) for fname in
train_files]
test_captions = [(str(new_path/IMAGE_DIR/fname), cap_dict[fname]) for fname in
test_files]
```

Now it will be enough to have the Drive mounted to be able to read the images. You can do the same for the other files if you want.

The model

The model is based on the general architecture of transformers. A convolutional network is used as an encoder of the visual information and a series of transformer-decoder layers produce the verbal description. The transformer-decoder layers include attention layers.

Transformer-decoder architecture



To familiarize yourself with these architectures you can read the tutorials of TensorFlow Text generation, sequence-to-sequence, and Transformers.

You can then run the whole notebook to see what it does overall.

Assessment of the quality of captioning (BLEU)

The tutorial does not include any mention of the quality of the captioning produced. If we assume that each image has some real captions (references) and the neural generates its own caption (hypothesis) we will use the BLEU (Bilingual Evaluation Understudy) score, between hypothesis and references. Briefly, BLEU is a weighted average of the number of common unigrams, bigrams, trigrams, and fourgrams between hypothesis and references. The worst captioning receives 0 and the best 1. See a detailed example of the BLEU calculation here.

To generate a caption for an image, first load it with the `image = load_image(image_path)` and then call the `model.simple_gen(image)` method.

The NLTK in nltk.translate.bleu_score provides the necessary functions for calculating BLEU scores:

- To be able to evaluate the captioning of an individual example, make a function that calculates the `sentence_bleu` between hypothesis and true captions (references) for an image.
- To evaluate the captioning of more images, e.g. part or all of the test set build a function that will calculate the `corpus_bleu` between all hypotheses and references of the images given to him. Note that the of `corpus_bleu` is not average `sentence_bleu` .

In all cases use the following parameters: `weights=(0.4, 0.3, 0.2,` and `smoothing_function=SmoothingFunction().method1` .

Before calculating, always remove the final dot (the last item in the list) from the references.

Improvements (and deliverables)

The first deliverable is the functions for the BLEU scores in the previous paragraph.

Then, once you get an idea of the performance of the default network (loss, accuracy, plots, training times, BLEU scores) we will test some ideas for network improvements (other deliverables).

Encoder

The example uses MobileNetV3Small for an encoder with learning transfer. In the Keras ready-made CNN models there are models that appear to have better benchmark performance than MobileNet (V2).

Keep the decoder's train process stable and see if you can get better performance using another encoder converter.

When you get to the encoder you can use the caching feature of the features contained in the notebook to read/save them to and from the Drive instead of generating them each time.

Text pre-processing

1. The captions have different lengths. Possibly too short and too long may not be useful in training. In the tutorial a maximum length of 50 words is used ad-hoc. You can filter the dataset so that you have a range of different lengths, neither too short nor too long (try a histogram). You should adjust the `max_length` accordingly.
2. Study the captions. Could the function `standardize` include other normalisation filters; Careful, we are not stemming here.

3. The tutorial decides ad hoc for a vocabulary of 5000 words. Try some different (smaller or larger) vocabulary sizes.

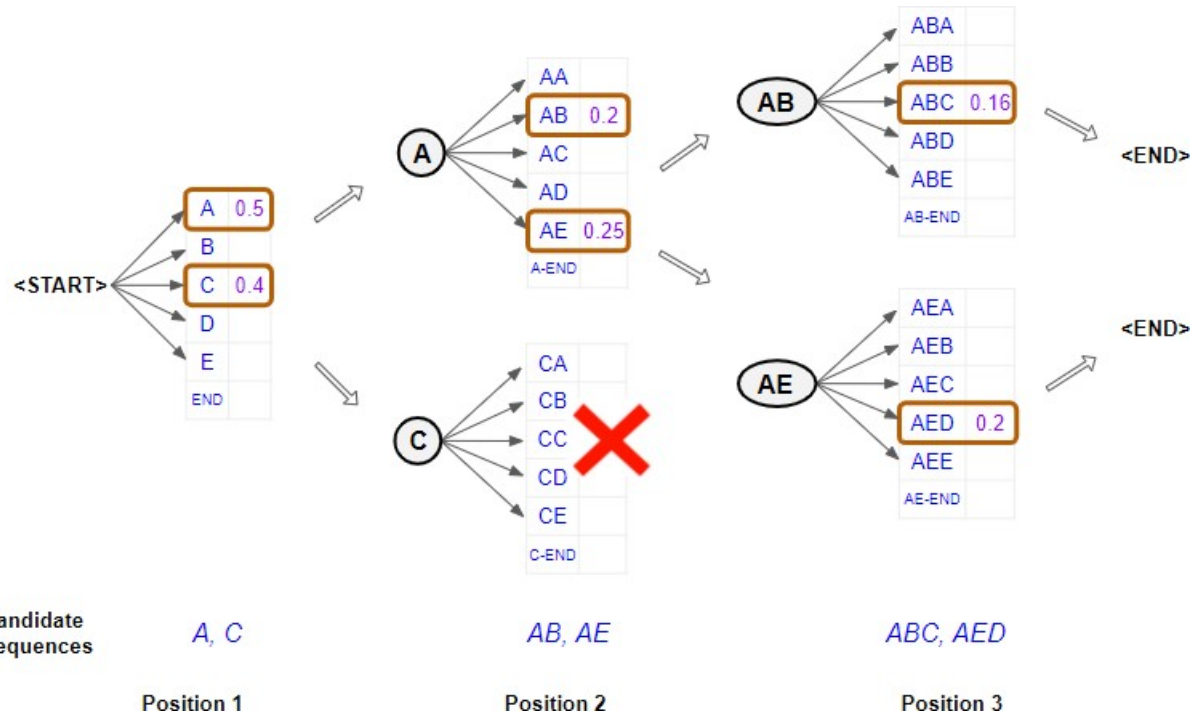
Embeddings

In the tutorial example the embeddings are learned during the training of the model. Instead we can use ready-made embeddings with learning transfer. See how the network performs for example with Gensim's glove-wiki embeddings of different dimensions (50, 100, 200, 300).

Sentence Generator (Beam Search)

In the example, the variable "`temperature`" is used to generate the sentences. If the temperature is 0.0, we have greedy decoding and the most likely token is selected at each step. If the temperature is 1.0 it does random sampling in terms of the probabilities (logits) of each token. If the temperature is much higher than 1.0 we end up with uniform random sampling.

Beam Search sentence generation



However, the literature reports that the Beam Search method produces significantly better results. Beam Search has a hyperparameter which is the width of the beam b. To choose the next word starts from the beginning of the sentence `[START]` and keeps the b to plural best (most likely) words for the next step. In this way they create b branches. For the next step it calculates the probabilities of the next words for all branches and keeps the b most likely cokes. In the end `[END]` we end up with b possible propositions and we pick the one with the best overall chances. For the latter, we could, for example, use S of the log of the probabilities divided by the length of each sentence.

You will find ready-made implementations for Beam Search on the web that you can customize, as well as standard values for b. You can modify the arguments of `simple_gen` to cover Beam Search.

Hyperparameters of the decoder

Study the effect of the hyperparameters `units` , `dropout_rate` , `num_layers` , and mod `num_heads`

Training times can be significant. Use some method of storing the model so that you can continue training.

Obviously we cannot do cross-validation of parameters and architectures. You use incremental training (relatively few epochs) and estimate network performance based on loss and time required, while examining the quality of captioning with fro `sentence_bleu` for selected images and especially the `corpus_bleu` for a piece of test m set.

Deliverable

The notebook with your best model as `corpus_bleu` to at least 1000 pictures of the test set. You can try on more images, ideally the whole test set, it's just a matter of time to produce the captions.

Use markdown to explain your options. You can mark down BLEU values and for intermediate options and present them in a table. Give examples of images with successful and less successful captioning.

Competition (optional)

Participation in the competition is optional and has no rating significance. However, because the ranking is based on `corpus_bleu` , participation and comparison with other submissions can show you how far you have come in improving your captions.

Contest images

For the competition we have selected 500 images for which you have no captions at all. Their names are included in "competition_files.csv". You can read them with the following code:

```
# Download competition files list
competition_files_list = tf.keras.utils.get_file('competition_files.csv',
                                                 cache_subdir=os.path.abspath('.'),
                                                 origin='https://spartacus.1337.cx/flickr-
mod/competition_files.csv',
                                                 extract=False)

path="."
IMAGE_DIR="image_dir"
path = pathlib.Path(path)

competition_files = (path/competition_files_list).read_text().splitlines()
competition_files = [str(path/IMAGE_DIR/fname) for fname in competition_files]
```

Don't forget to change the paths for the image files if you use Google Drive.

Creating a submission file

For all the images of "competition_files.csv" and in the order of the images (by appending them) create captions in the form of lists. If we say we had two images we would we stored in a variable `test_hypotheses` the captions as follows:

```
[['a', 'woman', 'floats', 'with', 'her', 'face', 'out', 'of', 'water', 'in',
'a', 'pool', 'with', 'another', 'woman', 'nearby', 'posing', 'for', 'the',
'camera'], ['a', 'black', 'and', 'white', 'dog', 'walks', 'along', 'a', 'sandy',
'beach']]
```

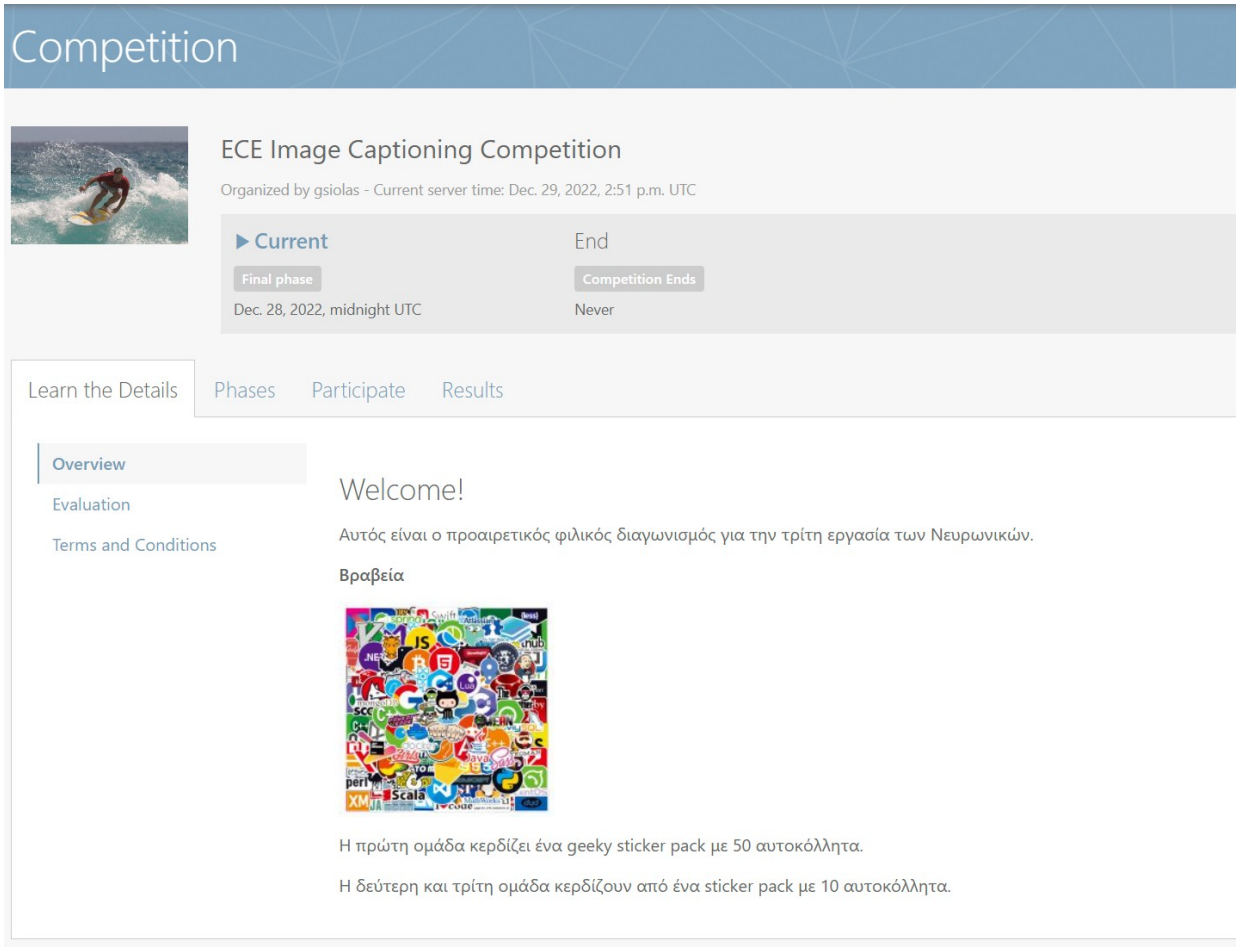
Calculate the 500 captions in the variable test_hypotheses and save it as JSON as follows:

```
import json

jsonString = json.dumps(test_hypotheses)
jsonFile = open("test_hypotheses.json", "w")
jsonFile.write(jsonString)
jsonFile.close()
```

You download the test_ypotheses.json file locally, rename it obligatorily to answer.txt and zip it to a zip file whose name does not matter.

Attached to the pronunciation you will find an example of a functional answer.txt.

The contest site on Codalab



The competition is at [this link to Codalab]. Codalab was used instead of Kaggle because Kaggle in the community tier does not allow custom metrics like BLEU. Codalab is also the official repository of COCO.

You will need to register. Note that if you go to the settings in your profile you can set a group name and group members with the usernames of all your Codalab members. Obviously the group name can be whatever you want, irrelevant to the number in the lab.

Submit a response and leaderboard

In order to upload a response, you will need to go to the "Participate" tab, "Submit/View Results" and click "Submit" to upload the submission file. You can refresh the status of the submission via the available "Refresh Status" button. When the submission is complete, and if it is valid, refresh the page via the browser to view your score. If your score is satisfactory, you can "Submit to leaderboard".

In the tab "Results" you will find the leaderboard of the submissions of all teams based on the `corpus_bleu`