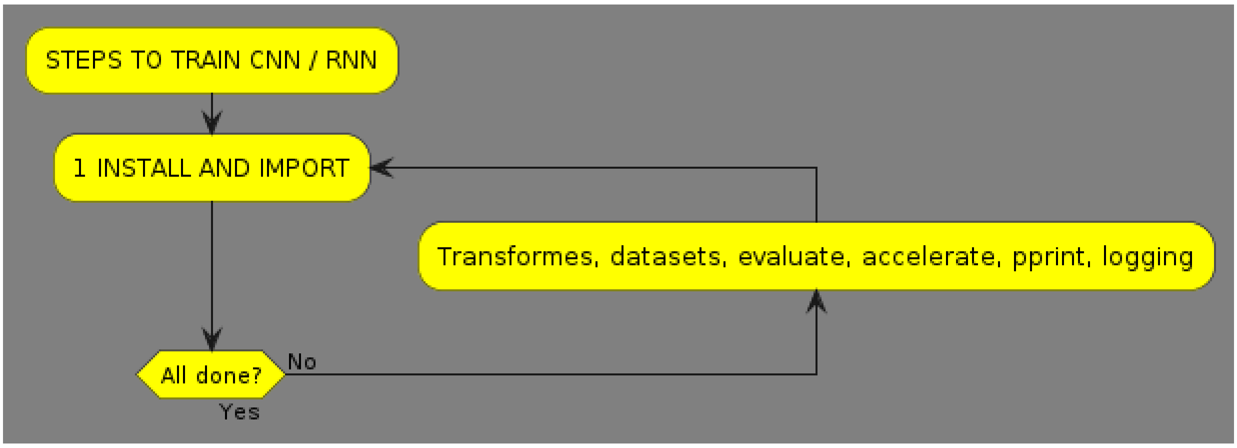
Deep learning week2 exercise 1

Your task is to carefully study the notebooks, and write a step-by-step summary of key steps to train and evaluate such a model. Keep in mind that many of these steps will be applicable throughout the course, even if the specific model differs. Therefore, it is essential to grasp the key concepts. As most of the code is shared in these two notebooks, writing just one summary is enough, but in the model building part, you should refer to both CNN/RNN implementations.

1 Install and import



Setup

in this phase install via pip:

- `transformers` is a popular deep learning package
- `datasets` provides support for loading, creating, and manipulating datasets
- `evaluate` is a library for easily evaluating machine learning models and datasets
- `accelerate` is a wrapper we need to install in order to train torch models using a transformers trainer

install all of the above:

```
!pip3 install -q transformers datasets evaluate accelerate
```

- `pprint` to formulate prints for some data structures

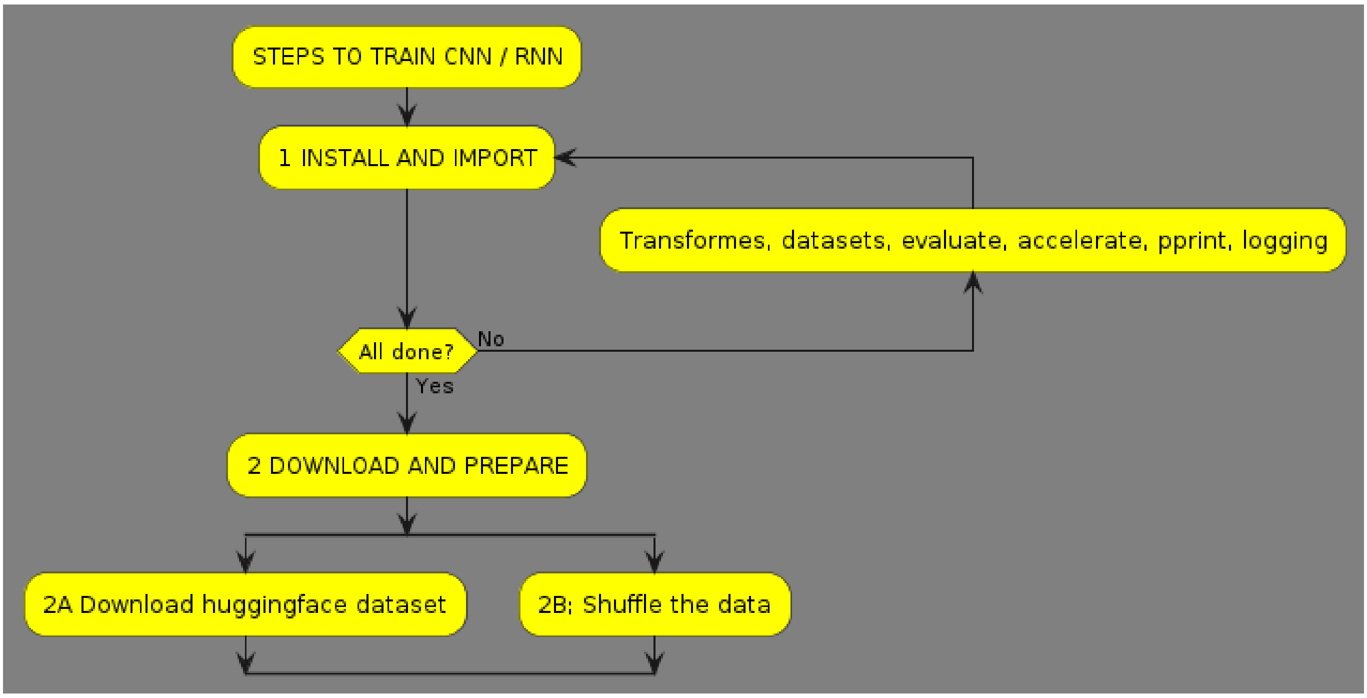
```
from pprint import PrettyPrinter
pprint = PrettyPrinter(compact=True).pprint
```

- `logging` to reduce `transformers` verbose logging ops. Remove this to get all low level info also

```
```

```
import logging
logging.disable(logging.INFO)
```

2 Download and prepare data



2A Download huggingface dataset

- Import `datasets` and download for example the imdb dataset from huggingface

```
import datasets
#https://huggingface.co/docs/datasets/main/en/package_reference/loading_met
hods#datasets.load_dataset
dataset = datasets.load_dataset("imdb")
```

- Check the quality / splits

```
print(dataset)

DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
```

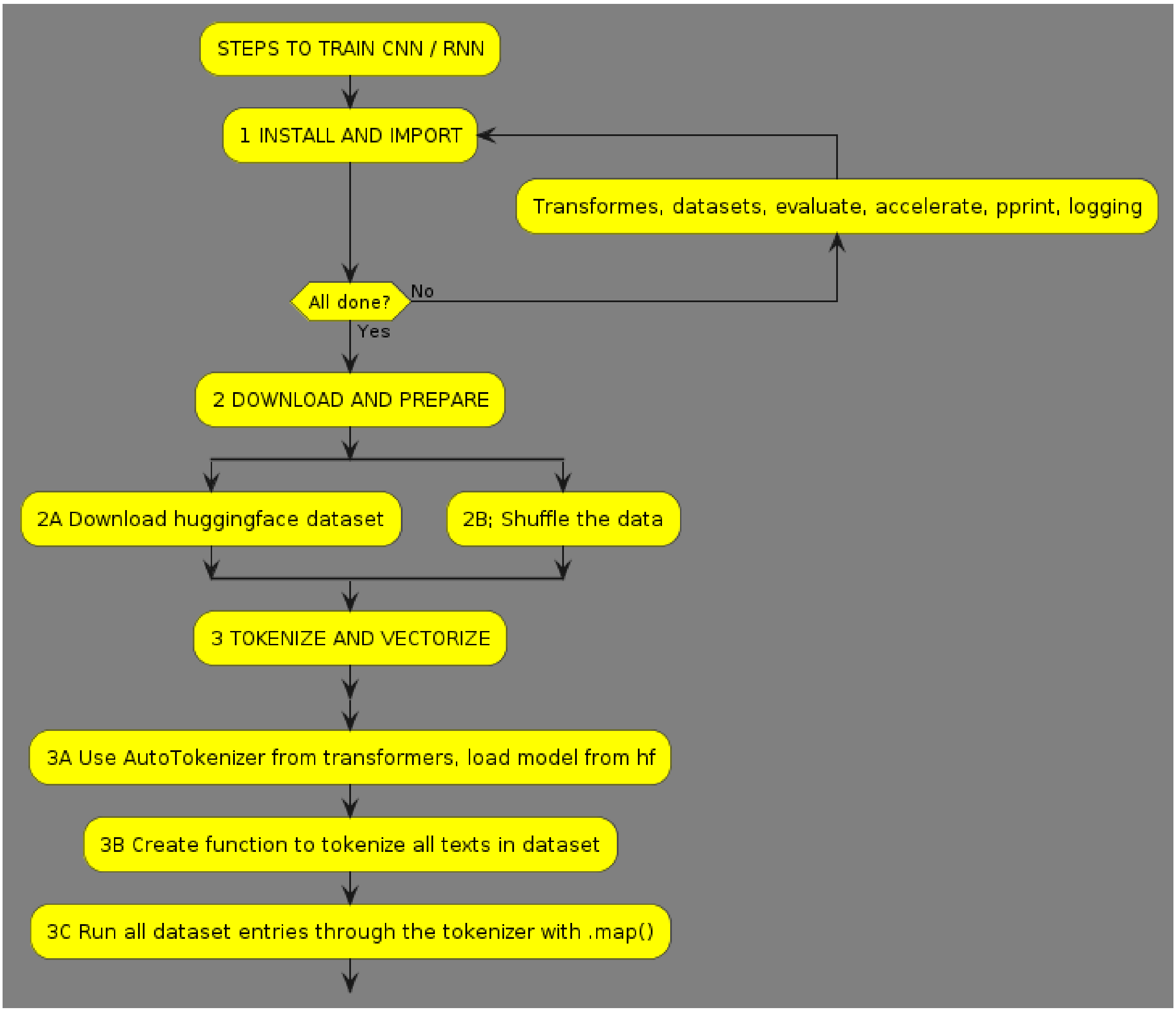
```
            num_rows: 50000
        })
    })
```

2B Shuffle the data

- Shuffle

```
dataset = dataset.shuffle() #This is never a bad idea, datasets may have
ordering to them, which is not what we want
#del dataset["unsupervised"] Delete the unlabeled part of the dataset to
make things faster
```

3 Tokenize and vectorize



3A

- Import transformers

- Models are listed at https://huggingface.co/models
- Assing pretrained AutoTokenizer to variable

```
import transformers

# Text in IMDB dataset in english, use the bert-cased
MODEL = "bert-base-cased"
tokenizer = transformers.AutoTokenizer.from_pretrained(MODEL)
```

3B

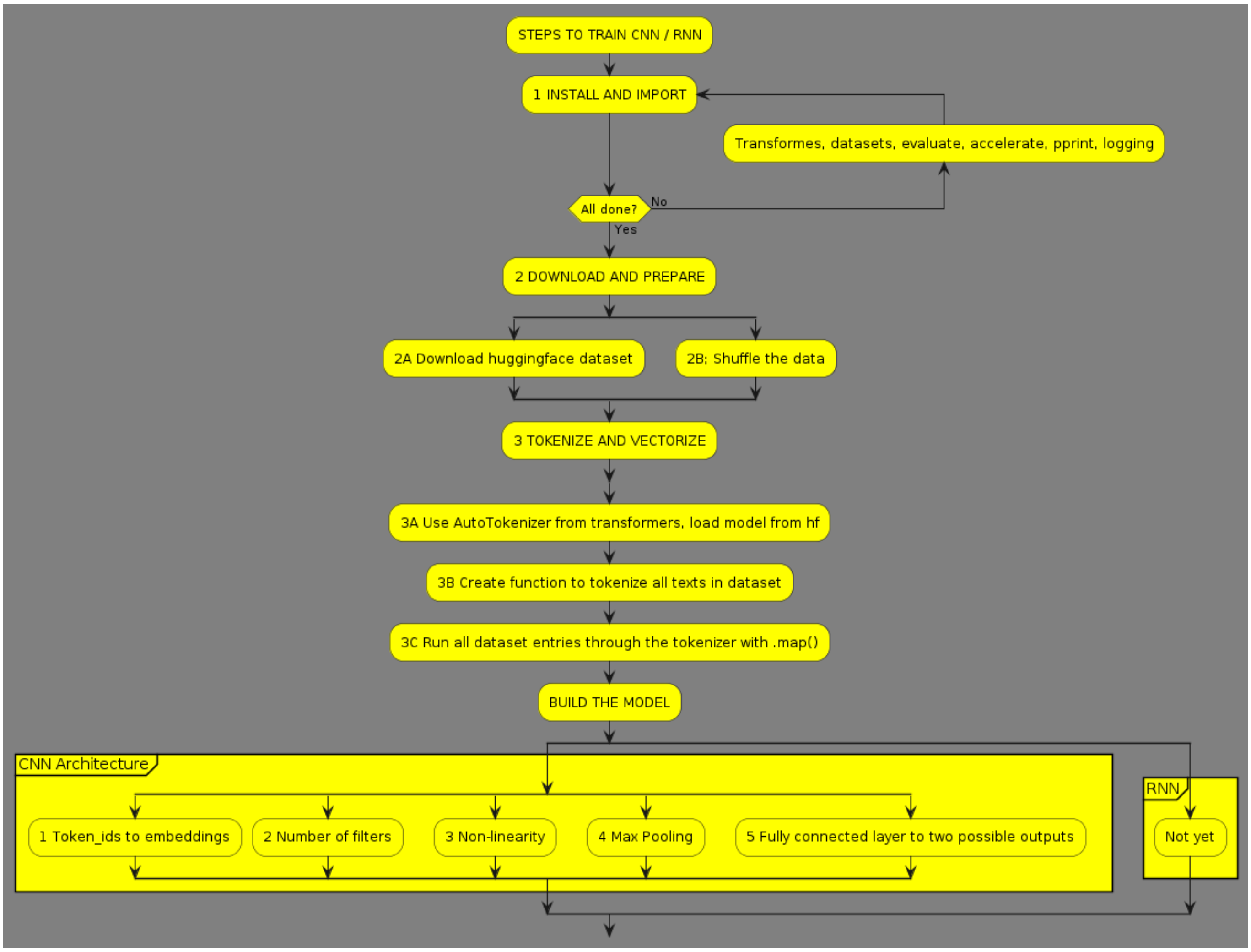
```
def tokenizer(dataset_entry: dict) --> dict:
    return tokenizer(dataset_entry["text"],
                    max_length=128, #limits the maximum length of outputs
to the given length
                    truncation=True) # faster train and potential
performance gains
```

3C

```
# https://huggingface.co/docs/transformers/preprocessing#everything-you-
always-wanted-to-know-about-padding-and-truncation
dataset = dataset.map(tokenizer)
```

4 BUILD THE MODEL

CNN




```
graph TD
   A(STEPS TO TRAIN CNN / RNN) ==> B{1 Install and import};
   B1(transformers, datasets, evaluate, accelerate, pprint, logging) --> B
   B ==> C{2 Download and prepare data};
C1[2A Download huggingface dataset] --> C;
   C2[2B Shuffle the data] --> C;
   C ==> D{3 Tokenize and vectorize};
   D1(3A: Use AutoTokenizer from transformers, model from hugging face) -->
D;
    D2(3B: Create function to tokenize all texts in dataset) --> D;
    D3(3C: Run all dataset entries through the tokenizer with .map) --> D;
    D ==> E(BUILD THE MODEL);

    %% from here is CNN PART
    E ==> E1(Architecture);
    F1(1 Token_ids -> embeddings) --> E1
    F2(2 Number of filters) --> E1
    F3(3 Non-linearity) --> E1
    F4(4 Max Pooling) --> E1
    F5(5 Fully connected layer to two possible outputs) --> E1

    subgraph CNN
        E1
        F1
```

```
        F2
        F3
        F4
        F5
    end
```

```
import torch
BasicConfig = transformers.PretrainedConfig #nice way to start
```

1. Token IDs are mapped to embeddings of a user-specific size (`config.embedding_dim`) in a torch.nn.Embedding layer. Typically initialized with previously leanerd weights, here starts with random

    ```
    # SELF HERE MEANS THE MODEL CLASS, ALL COMES TOGETHER IN THE END UNDER ONE CLASS :-)
    # Embedding layer: vocab size x embedding dim
    self.embeddings = torch.nn.Embedding(
        num_embeddings=config.vocab_size,
        embedding_dim=config.embedding_dim
    )
    ```

2. Number of filters, specified by the user is applied to the matrix formed by the sequence of token embedding in a convolution layer (*think these filters with the image example*)

    ```
    # Convolution layer:
    self.convolution = torch.nn.Conv1d(
        config.embedding_dim,
        config.num_filters,
        config.filter_size,
        padding=1
        )
    ```

3.
4.
5.

DISCARD BELOW JUST TO KEEP

```
graph TD
   A(STEPS TO TRAIN CNN / RNN) ==> B{1 Install and import};
   B1(transformers, datasets, evaluate, accelerate, pprint, logging) --> B
   B ==> C{2 Download and prepare data};
C1[2A Download huggingface dataset] --> C;
```

```
    C2[2B Shuffle the data] --> C;
    C ==> D{3 Tokenize and vectorize};
    D1(3A: Use AutoTokenizer from transformers, model from hugging face) -->
D;
     D2(3B: Create function to tokenize all texts in dataset) --> D;
     D3(3C: Run all dataset entries through the tokenizer with .map) --> D;
     D ==> E(BUILD THE MODEL);

     %% from here is CNN PART
     E ==> E1(Architecture);
     F1(1 Token_ids -> embeddings) --> E1
     F2(2 Number of filters) --> E1
     F3(3 Non-linearity) --> E1
     F4(4 Max Pooling) --> E1
     F5(5 Fully connected layer to two possible outputs) --> E1

     subgraph CNN
         E1
         F1
         F2
         F3
         F4
         F5
     end

     %% from here is RNN PART
     E ==> E2(Architecture);
     G1 --> E2
     G2 --> E2
     G3 --> E2
     G4 --> E2
     G5 --> E2

     subgraph RNN
         E2
         G1
         G2
         G3
         G4
         G5
     end
```