install and import necessary libraries

In [12]:
```
!pip install -q sacremoses
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
```

In [39]:
```
!pip install -q sacrebleu
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
```

In [40]:
```
!pip install -q evaluate
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv
```

In [13]:
```
import warnings
warnings.filterwarnings("ignore")
```

Preparing the data

In [1]:
```
from datasets import load_dataset
```

In [2]:
```
dataset = load_dataset("csv", data_files="/kaggle/input/pretraindedsw/combined.csv"
```

```
Downloading and preparing dataset csv/default to /root/.cache/huggingface/dataset
s/csv/default-65bbae727179b21d/0.0.0/433e0ccc46f9880962cc2b12065189766fbb2bee57a22
1866138fb9203c83519...
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/defau
lt-65bbae727179b21d/0.0.0/433e0ccc46f9880962cc2b12065189766fbb2bee57a221866138fb92
03c83519. Subsequent calls will reuse this data.
  0%|          | 0/1 [00:00<?, ?it/s]
```

In [3]:
```
# a view on the dataset object
dataset
```

Out[3]:
```
DatasetDict({
    train: Dataset({
        features: ['English sentence', 'Swahii Translation'],
        num_rows: 8492
    })
})
```

split dataset for training and modelling

In [5]:
```
split_datasets = dataset["train"].train_test_split(train_size=0.9, seed=20)
split_datasets
```

Out[5]:
```
DatasetDict({
    train: Dataset({
        features: ['English sentence', 'Swahii Translation'],
        num_rows: 7642
    })
    test: Dataset({
        features: ['English sentence', 'Swahii Translation'],
        num_rows: 850
    })
})
```

In [6]:
```
# a look on the dataset
split_datasets["train"][10]["English sentence"], split_datasets["train"][10]["Swahi
```

Out[6]:
```
('We are going to build a new wall around the school.',
 'Tutaunda ukuta mpya kuzunguka shule.')
```

load pretrained model

In [18]:
```
from transformers import pipeline
# swahili pretraiined model from hugging face
model_checkpoint = "Helsinki-NLP/opus-mt-en-swc"
translator = pipeline("translation", model=model_checkpoint)
```

In [21]:
```
# test pretrained model
translator("i will not go to school today")
```

Out[21]:
```
[{'translation_text': 'Halitaenda shuleni leo'}]
```

Tokenization

In [22]:
```
from transformers import AutoTokenizer

model_checkpoint = "Helsinki-NLP/opus-mt-en-swc"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint, return_tensors="pt")
```

In [23]:
```
en_sentence = split_datasets["train"][1]["English sentence"]
sw_sentence = split_datasets["train"][1]["Swahii Translation"]

inputs = tokenizer(en_sentence, text_target=sw_sentence)
inputs
```

Out[23]:
```
{'input_ids': [413, 250, 114, 349, 8836, 3, 0], 'attention_mask': [1, 1, 1, 1, 1,
1, 1], 'labels': [1155, 260, 194, 2448, 2138, 3, 0]}
```

In [32]:
```
type(inputs)
```

Out[32]:
```
transformers.tokenization_utils_base.BatchEncoding
```

In [29]:
```
wrong_targets = tokenizer(sw_sentence)
print(tokenizer.convert_ids_to_tokens(wrong_targets["input_ids"]))
print(tokenizer.convert_ids_to_tokens(inputs["labels"]))
```

```
['▁M', 'tu', '▁hu', 'yo', '▁an', 'a', '▁ma', 'ga', 'ri', '▁m', 'aw', 'ili',
'.', '</s>']
['▁Mtu', '▁huyo', '▁ana', '▁magari', '▁mawili', '.', '</s>']
```

In [30]:
```
max_length = 128


def preprocess_function(examples):
    inputs = examples['English sentence']
    targets = examples['Swahii Translation']
    model_inputs = tokenizer(
        inputs, text_target=targets, max_length=max_length, truncation=True
    )
    return model_inputs
```

In [13]:
```
inputs
```

Out[13]:
```
{'input_ids': [413, 250, 114, 349, 8836, 3, 0], 'attention_mask': [1, 1, 1, 1, 1,
1, 1], 'labels': [1155, 260, 194, 2448, 2138, 3, 0]}
```

In [34]:
```
tokenized_datasets = split_datasets.map(
    preprocess_function,
    batched=True,
    remove_columns=split_datasets["train"].column_names,
)
```

```
  0%|          | 0/8 [00:00<?, ?ba/s]
  0%|          | 0/1 [00:00<?, ?ba/s]
```

fine tuning model with keras

In [36]:
```
from transformers import TFAutoModelForSeq2SeqLM

model = TFAutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/opus-mt-en-swc", from
```

```
All PyTorch model weights were used when initializing TFMarianMTModel.

All the weights of TFMarianMTModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, yo
u can already use TFMarianMTModel for predictions without further training.
```

Data collation

In [37]:
```
from transformers import DataCollatorForSeq2Seq

data_collator = DataCollatorForSeq2Seq(tokenizer, model=model, return_tensors="tf")
```

In [38]:
```
tf_train_dataset = model.prepare_tf_dataset(
    tokenized_datasets["train"],
```

```
    collate_fn=data_collator,
    shuffle=True,
    batch_size=32,
)
tf_eval_dataset = model.prepare_tf_dataset(
    tokenized_datasets["test"],
    collate_fn=data_collator,
    shuffle=False,
    batch_size=16,
)
```

Setting up Sacrebeu for evaluation

In [41]:
```
import evaluate

metric = evaluate.load("sacrebleu")
```

```
Downloading builder script:   0%|          | 0.00/8.15k [00:00<?, ?B/s]
```

In [45]:
```
# using sacrabeu to
predictions = [
    "Tutaunda ukuta mpya kuzunguka shule yote."
]
references = [
    [
        "Tutatengeneza ukuta mpya kuzunguka shule."
    ]
]
metric.compute(predictions=predictions, references=references)
```

Out[45]:
```
{'score': 43.47208719449914,
 'counts': [5, 3, 2, 1],
 'totals': [7, 6, 5, 4],
 'precisions': [71.42857142857143, 50.0, 40.0, 25.0],
 'bp': 1.0,
 'sys_len': 7,
 'ref_len': 6}
```

function to loop through all translation and give evaluation

In [46]:
```
import numpy as np
import tensorflow as tf
from tqdm import tqdm

generation_data_collator = DataCollatorForSeq2Seq(
    tokenizer, model=model, return_tensors="tf", pad_to_multiple_of=128
)

tf_generate_dataset = model.prepare_tf_dataset(
    tokenized_datasets["test"],
    collate_fn=generation_data_collator,
    shuffle=False,
    batch_size=8,
```

```
)


@tf.function(jit_compile=True)
def generate_with_xla(batch):
    return model.generate(
        input_ids=batch["input_ids"],
        attention_mask=batch["attention_mask"],
        max_new_tokens=128,
    )


def compute_metrics():
    all_preds = []
    all_labels = []

    for batch, labels in tqdm(tf_generate_dataset):
        predictions = generate_with_xla(batch)
        decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=Tru
        labels = labels.numpy()
        labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
        decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
        decoded_preds = [pred.strip() for pred in decoded_preds]
        decoded_labels = [[label.strip()] for label in decoded_labels]
        all_preds.extend(decoded_preds)
        all_labels.extend(decoded_labels)

    result = metric.compute(predictions=all_preds, references=all_labels)
    return {"bleu": result["score"]}
```

Score of pretrained model before fine tuning

In [47]:
```
print(compute_metrics())
```

```
100%|██████████| 107/107 [01:22<00:00,  1.30it/s]
{'bleu': 29.821006218712345}
```

Training and Fine tuning

In [53]:
```
from transformers import create_optimizer
from transformers.keras_callbacks import PushToHubCallback
import tensorflow as tf

num_epochs = 3
num_train_steps = len(tf_train_dataset) * num_epochs

optimizer, schedule = create_optimizer(
    init_lr=5e-5,
    num_warmup_steps=0,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
```

```
)
model.compile(optimizer=optimizer)

# Train in mixed-precision float16
tf.keras.mixed_precision.set_global_policy("mixed_float16")
```

```
No loss specified in compile() - the model's internal loss computation will be use
d as the loss. Don't panic - this is a common way to train TensorFlow models in Tr
ansformers! To disable this behaviour please pass a loss argument, or explicitly p
ass `loss=None` if you do not want your model to compute a loss.
```

save the model to huggingface

In [ ]:
```
# login to hugging face
#from huggingface_hub import notebook_login
#notebook_login()
```

In [56]:
```
from transformers.keras_callbacks import PushToHubCallback

#callback = PushToHubCallback(
    #output_dir="marian-finetuned-kde4-en-to-sw", tokenizer=tokenizer
#)

model.fit(
    tf_train_dataset,
    validation_data=tf_eval_dataset,
    #callbacks=[callback],
    epochs=num_epochs,
)
```

```
Epoch 1/3
238/238 [==============================] - 64s 268ms/step - loss: 0.6788 - val_los
s: 0.9320
Epoch 2/3
238/238 [==============================] - 64s 269ms/step - loss: 0.6547 - val_los
s: 0.9320
Epoch 3/3
238/238 [==============================] - 66s 276ms/step - loss: 0.6559 - val_los
s: 0.9320
```

Out[56]:
```
<keras.callbacks.History at 0x7371dc0e39d0>
```

model accuracy after fine tuning

In [57]:
```
print(compute_metrics())
```

```
100%|██████████| 107/107 [00:28<00:00,  3.77it/s]
{'bleu': 56.81215256729416}
```

In [ ]:
```
#3 bleu score 3 epochhs {'bleu': 56.980206425731886}
# bleu score on 3 epochs {'bleu': 61.74923844322799}
```

Loading the fine tuned pretrained model

In [ ]:
```
#from transformers import pipeline

# Replace this with your own checkpoint
#model_checkpoint = "KigenCHESS/marian-finetuned-kde4-en-to-sw"
#translator = pipeline("translation", model=model_checkpoint)
```

translation on pretrained model

In [ ]:
```
#translator("i will not go to school today")
```