Transformers vs. Fastai in image classification task - Part2

Which one is better and more suitable for learning?

From Recreate Model with Transformers #2, one thing that I noticed while creating the model was how tedious the whole process was. Unlike using Fast.ai, it requires a lot of coding. Thus, I decided to make some straightforward functions that do some of the tedious work for us.

Install necessary packages

In [15]:
```
from datasets import load_dataset, DatasetDict, load_from_disk
import random
from PIL import ImageDraw, ImageFont, Image
from transformers import ViTFeatureExtractor
import torch
import numpy as np
from datasets import load_metric
from transformers import ViTForImageClassification
from transformers import TrainingArguments
from transformers import Trainer
```

Some functions and parameters from #1

Similar to the #1, I created some parameters such as `feature_extractor` , and `metric` . I also reused some of the same functions.

In [16]:
```
model_name_or_path = 'google/vit-base-patch16-224-in21k'
feature_extractor = ViTFeatureExtractor.from_pretrained(model_name_or_p
```

```
loading feature extractor configuration file https://huggingface.co/g
oogle/vit-base-patch16-224-in21k/resolve/main/preprocessor_config.jso
n (https://huggingface.co/google/vit-base-patch16-224-in21k/resolve/m
ain/preprocessor_config.json) from cache at /scratch/cs344/huggingfac
e/transformers/7c7f3e780b30eeeacd3962294e5154788caa6d9aa555ed6d5c2f0d
2c485eba18.c322cbf30b69973d5aae6c0866f5cba198b5fe51a2fe259d2a506827ec
6274bc
Feature extractor ViTFeatureExtractor {
  "do_normalize": true,
  "do_resize": true,
  "feature_extractor_type": "ViTFeatureExtractor",
  "image_mean": [
    0.5,
    0.5,
    0.5
  ],
  "image_std": [
    0.5,
    0.5,
    0.5
  ],
  "resample": 2,
  "size": 224
}
```

In [17]:
```
def transform(example_batch):
    # Take a list of PIL images and turn them to pixel values
    inputs = feature_extractor([x for x in example_batch['image']], ret

    # Don't forget to include the labels!
    inputs['label'] = example_batch['label']
    return inputs
prepared_ds = ds.with_transform(transform)
```

In [18]:
```
metric = load_metric("accuracy")
def compute_metrics(p):
    return metric.compute(predictions=np.argmax(p.predictions, axis=1),
```

In [19]:
```
def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in batch])
        'labels': torch.tensor([x['label'] for x in batch])
    }
```

Load Dataset

Here, I made a function that creates a dataset for you. It takes three parameters, which are `train_path` , `test_path` , and `valid_path` . Each path has to include subfolders for each category. (e.g. train/dogs & train/cats, test/dogs & test/cats)

In [20]:
```
def create_ds(train_path, test_path, valid_path):
         ds = DatasetDict({
            "train": load_dataset("imagefolder", data_dir= train_path,
            "test": load_dataset("imagefolder", data_dir= test_path, sp
            "validation": load_dataset("imagefolder", data_dir= valid_p
         return ds
```

In [21]:
```
tr_p = '/home/tt35/Desktop/pics_transformers/train'
te_p = '/home/tt35/Desktop/pics_transformers/test'
va_p = '/home/tt35/Desktop/pics_transformers/validation'
ds = create_ds(tr_p, te_p, va_p)
```

```
Resolving data files:   0%|          | 0/107 [00:00<?, ?it/s]

Using custom data configuration default-ac41a5ade1cccd36
Reusing dataset image_folder (/scratch/cs344/huggingface/datasets/ima
ge_folder/default-ac41a5ade1cccd36/0.0.0/ee92df8e96c6907f3c851a987be3
fd03d4b93b247e727b69a8e23ac94392a091)

Resolving data files:   0%|          | 0/34 [00:00<?, ?it/s]

Using custom data configuration default-73eb95b38a320fdb
Reusing dataset image_folder (/scratch/cs344/huggingface/datasets/ima
ge_folder/default-73eb95b38a320fdb/0.0.0/ee92df8e96c6907f3c851a987be3
fd03d4b93b247e727b69a8e23ac94392a091)

Resolving data files:   0%|          | 0/27 [00:00<?, ?it/s]

Using custom data configuration default-a28225d917e12e11
Reusing dataset image_folder (/scratch/cs344/huggingface/datasets/ima
ge_folder/default-a28225d917e12e11/0.0.0/ee92df8e96c6907f3c851a987be3
fd03d4b93b247e727b69a8e23ac94392a091)
```

Create Model

I wrote the function that creates a model. It takes a dataset and model name. Some examples of the possible model name can be found [here (https://huggingface.co/models?other=vit)](https://huggingface.co/models?other=vit). However, it should be noted that some models in the website are not compatible to the function. It is recommended to use [vit-base-patch16-224-in21k (https://huggingface.co/google/vit-base-patch16-224-in21k)](https://huggingface.co/google/vit-base-patch16-224-in21k) as a default.

In [22]:
```
def create_model(ds, model_name_or_path = 'google/vit-base-patch16-224-
        mn = model_name_or_path
        labels = ds['train'].features['label'].names
        model = ViTForImageClassification.from_pretrained(
        mn,
        num_labels=len(labels),
        id2label={str(i): c for i, c in enumerate(labels)},
        label2id={c: str(i) for i, c in enumerate(labels)}
        )

        return model
```

In [23]:
```
model = create_model(ds, model_name_or_path)
```

```
loading configuration file https://huggingface.co/google/vit-base-patch16-224-in21k/resolve/main/config.json (https://huggingface.co/google/vit-base-patch16-224-in21k/resolve/main/config.json) from cache at /scratch/cs344/huggingface/transformers/7bba26dd36a6ff9f6a9b19436dec361727bea03ec70fbfa82b70628109163eaa.92995a56e2eabab0c686015c4ad8275b4f9cbd858ed228f6a08936f2c31667e7
Model config ViTConfig {
  "_name_or_path": "google/vit-base-patch16-224-in21k",
  "architectures": [
    "ViTModel"
  ],
  "attention_probs_dropout_prob": 0.0,
  "encoder_stride": 16,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.0,
  "hidden_size": 768,
  "id2label": {
    "0": "CF",
    "1": "SB"
  },
  "image_size": 224,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "CF": "0",
    "SB": "1"
  },
  "layer_norm_eps": 1e-12,
  "model_type": "vit",
  "num_attention_heads": 12,
  "num_channels": 3,
  "num_hidden_layers": 12,
  "patch_size": 16,
  "qkv_bias": true,
  "transformers_version": "4.17.0"
}

loading weights file https://huggingface.co/google/vit-base-patch16-224-in21k/resolve/main/pytorch_model.bin (https://huggingface.co/google/vit-base-patch16-224-in21k/resolve/main/pytorch_model.bin) from cache at /scratch/cs344/huggingface/transformers/d01bfc4a52063e6f2cc1bc7063192e012043a7c6d8e75981bb6afbb9dc911001.e4710baf72bd00d091aab2ae692d487c057734cf044ba421696823447b95521e
Some weights of the model checkpoint at google/vit-base-patch16-224-in21k were not used when initializing ViTForImageClassification: ['pooler.dense.bias', 'pooler.dense.weight']
- This IS expected if you are initializing ViTForImageClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing ViTForImageClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
Some weights of ViTForImageClassification were not initialized from
 the model checkpoint at google/vit-base-patch16-224-in21k and are n
ewly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be abl
e to use it for predictions and inference.
```

Make a Trainer

Unlike #1, this function let you choose various parameters such as batch_size, and epohcs.

In [24]:
```
def make_trainer(model, batch_size=8, epochs=3, learning_rate=2e-4, sav
    prepared_ds = ds.with_transform(transform)
    trainer1 = Trainer(
        model=model,
        args=TrainingArguments(
          output_dir="./vit-base",
          per_device_train_batch_size=batch_size,
          evaluation_strategy="steps",
          num_train_epochs=epochs,
          fp16=True,
          save_steps=save_steps,
          eval_steps=eval_steps,
          logging_steps=logging_steps,
          learning_rate=learning_rate,
          save_total_limit=save_total_limit,
          remove_unused_columns=False,
          push_to_hub=False,
          report_to='tensorboard',
          load_best_model_at_end=True,
        ),
        data_collator=collate_fn,
        compute_metrics=compute_metrics,
        train_dataset=prepared_ds["train"],
        eval_dataset=prepared_ds["validation"],
        tokenizer=feature_extractor,
    )

    return trainer1
```

In [25]:
```
trainer = make_trainer
```

In [26]:
```
trainer = make_trainer(model, batch_size=8, epochs=3, learning_rate=2e-
```

```
PyTorch: setting up devices
Using amp half precision backend
```

Model Testing and Evaluation

Model testing and evaluation are done in the same way as #1 as I believe it does not require intensive coding.

In [27]:
```
random.seed(100)
train_results = trainer.train()
trainer.save_model()
trainer.log_metrics("train", train_results.metrics)
trainer.save_metrics("train", train_results.metrics)
trainer.save_state()
```

```
***** Running training *****
  Num examples = 107
  Num Epochs = 3
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 
8
  Gradient Accumulation steps = 1
  Total optimization steps = 42
```

[42/42 00:28, Epoch 3/3]

Step	Training Loss	Validation Loss

```
Training completed. Do not forget to share your model on huggingface.
co/models =)


Saving model checkpoint to ./vit-base
Configuration saved in ./vit-base/config.json
Model weights saved in ./vit-base/pytorch_model.bin
Feature extractor saved in ./vit-base/preprocessor_config.json

***** train metrics *****
  epoch                    =        3.0
  total_flos               = 23166582GF
  train_loss               =     0.1605
  train_runtime            = 0:00:29.81
  train_samples_per_second =     10.766
  train_steps_per_second   =      1.409
```

In [28]:
```
metrics = trainer.evaluate(prepared_ds['test'])
trainer.log_metrics("eval", metrics)
trainer.save_metrics("eval", metrics)
```

```
***** Running Evaluation *****
  Num examples = 34
  Batch size = 8
```

[5/5 00:02]

```
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =        1.0
  eval_loss               =     0.0214
  eval_runtime            = 0:00:03.04
  eval_samples_per_second =     11.168
  eval_steps_per_second   =      1.642
```