Fine-tuning Florence-2 - Microsoft's Cutting-edge Vision Language Models

Published June 24, 2024

andito
Andres Marafioti

merve
Merve Noyan

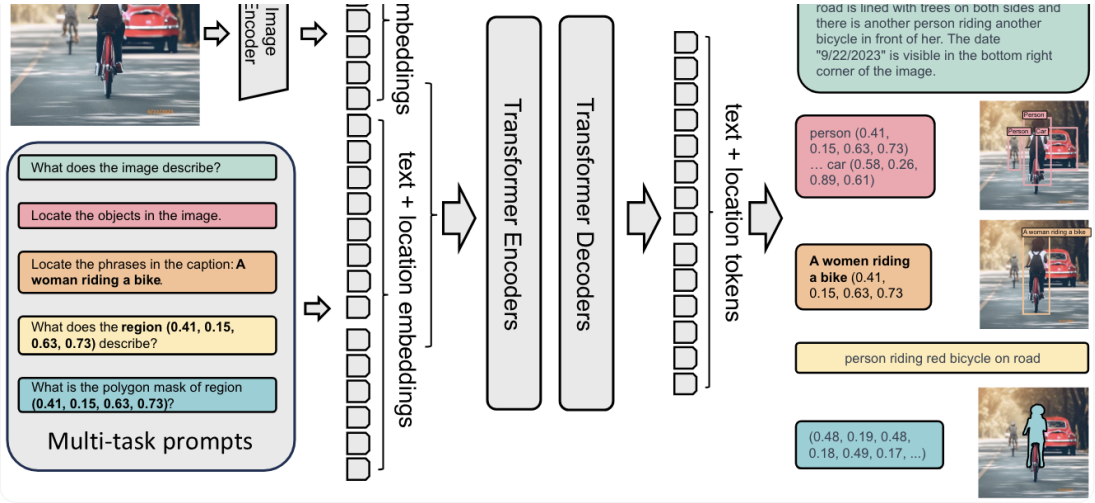
SkalskiP
Piotr Skalski
guest

Florence-2, released by Microsoft in June 2024, is a foundation vision-language model. This model is very attractive because of its small size (0.2B and 0.7B) and strong performance on a variety of computer vision and vision-language tasks.

Florence supports many tasks out of the box: captioning, object detection, OCR, and more. However, your task or domain might not be supported, or you may want to better control the model's output for your task. That's when you will need to fine-tune.

In this post, we show an example on fine-tuning Florence on DocVQA. The authors report that Florence 2 can perform visual question answering (VQA), but the released models don't include VQA capability. Let's see what we can do!

Pre-training Details and Architecture



Florence-2 Architecture

Regardless of the computer vision task being performed, Florence-2 formulates the problem as a sequence-to-sequence task. Florence-2 takes an image and text as inputs, and generates text as output. The model has a simple structure. It uses a DaViT vision encoder to convert images into visual embeddings, and BERT to convert text prompts into text and location embeddings. The resulting embeddings are then processed by a standard encoder-decoder transformer architecture, generating text and location tokens. Florence-2's strength doesn't stem from its architecture, but from the massive dataset it was pre-trained on. The authors noted that leading computer vision datasets typically contain limited information - WIT only includes image/caption pairs, SA-1B only contains images and associated segmentation masks. Therefore, they decided to build a new FLD-5B dataset containing a wide range of information about each image - boxes, masks, captions, and grounding. The dataset creation process was largely automated. The authors used off-the-shelf task-specific models and a set of heuristics and quality checks to clean the obtained results. The result was a new dataset containing over 5 billion annotations for 126 million images, which was used to pre-train the Florence-2 model.

Original performance on VQA

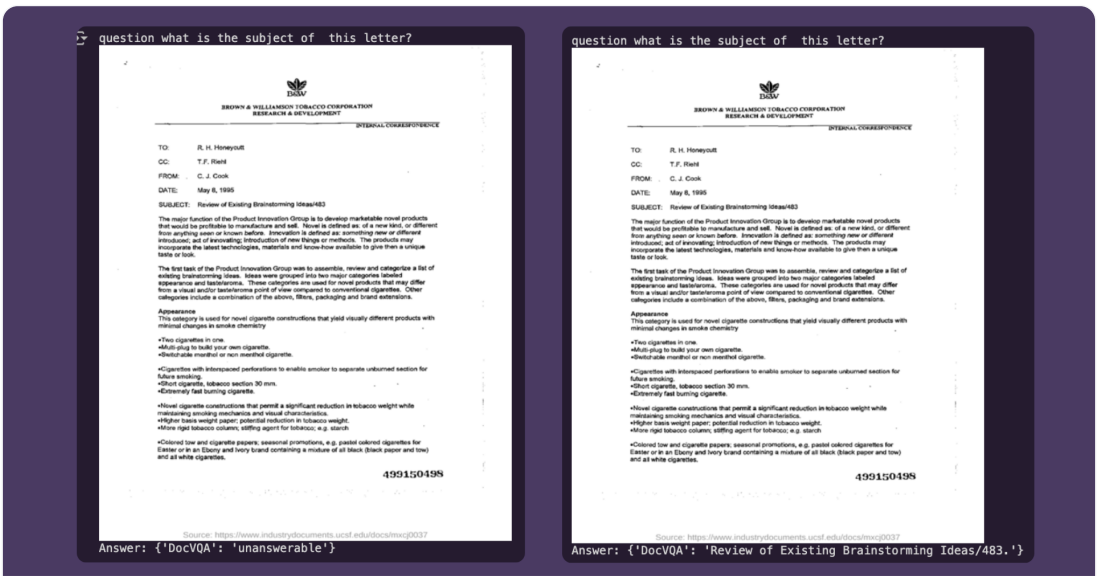
We experimented with various methods to adapt the model for VQA (Visual

Question Answering) responses. The most effective approach we found was region-to-description prompting, though it doesn't fully align with VQA tasks. Captioning provides descriptive information about the image but doesn't allow for direct question input. We also tested several "unsupported" prompts such as " <VQA>", "<vqa>", and "<Visual question answering>". Unfortunately, these attempts yielded unusable results.

Performance on DocVQA after fine-tuning

We measure performance using the Levenshtein's similarity, the standard metric for the DocVQA dataset. Before fine-tuning, the similarity between the model's predictions and the ground truth on the validation dataset was 0, as the outputs were not close to the ground truth. After fine-tuning with the training set for seven epochs, the similarity score on the validation set improved to 57.0. We created a 🤗 space to demo the fine-tuned model. While the model performs well for DocVQA, there is room for improvement in general document understanding. However, it successfully completes the tasks, showcasing Florence-2's potential for fine-tuning on downstream tasks. To develop an exceptional VQA model, we recommend further fine-tuning Florence-2 usingThe Cauldron. We have already provided the necessary code on our GitHub page.

To give a solid example, below we provide two inference results before and after fine-tuning. You can also try the model here.



Before and After Fine-tuning

Fine-tuning Details

For pre-training, the authors used a batch size of 2048 for the base model and 3072 for the large one. They also describe a performance improvement when fine-tuning with an unfrozen image encoder, compared with freezing it.

We conducted our experiments with a much lower resource setup, to explore what the model would be capable of in more constrained fine-tuning environments. We froze the vision encoder and used a batch size of 6 on a single A100 GPU in Colab, or a batch size of 1 with a T4. In parallel, we conducted an experiment with more resources, fine-tuning the entire model with a batch size of 64. This training process took 70 minutes on a cluster equipped with 8 H100 GPUs. This trained model can be found here.

In every case, we found a small learning rate of 1e-6 to be beneficial for training. With larger learning rates the model will quickly overfit the training set.

Code Walkthrough

If you want to follow along, you can find our Colab fine-tuning notebook here. We will be fine-tuning the Florence-2-base-ft checkpoint on the DocVQA dataset. Let's start by installing the dependencies.

```
!pip install -q datasets flash_attn timm einops
```

Load the DocVQA dataset from the Hugging Face Hub.

```
import torch
from datasets import load_dataset

data = load_dataset("HuggingFaceM4/DocumentVQA")
```

We can load the model and processor using the `AutoModelForCausalLM` and `AutoProcessor` classes from the transformers library. We need to pass `trust_remote_code=True` because the model uses custom code – it has not been natively integrated into transformers yet. We will also freeze the vision encoder to make fine-tuning less expensive.

```
from transformers import AutoModelForCausalLM, AutoProcessor
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = AutoModelForCausalLM.from_pretrained(
    "microsoft/Florence-2-base-ft",
    trust_remote_code=True,
    revision='refs/pr/6'
).to(device)
processor = AutoProcessor.from_pretrained("microsoft/Florence-2-base-f
    trust_remote_code=True, revision='refs/pr/6')

for param in model.vision_tower.parameters():
  param.is_trainable = False
```

Let's now fine-tune the model! We'll build a training PyTorch Dataset in which we'll prepend a <DocVQA> prefix to each question from the dataset.

```
import torch from torch.utils.data import Dataset

class DocVQADataset(Dataset):

    def __init__(self, data):
        self.data = data
```

```
    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        example = self.data[idx]
        question = "<DocVQA>" + example['question']
        first_answer = example['answers'][0]
        image = example['image'].convert("RGB")
        return question, first_answer, image
```

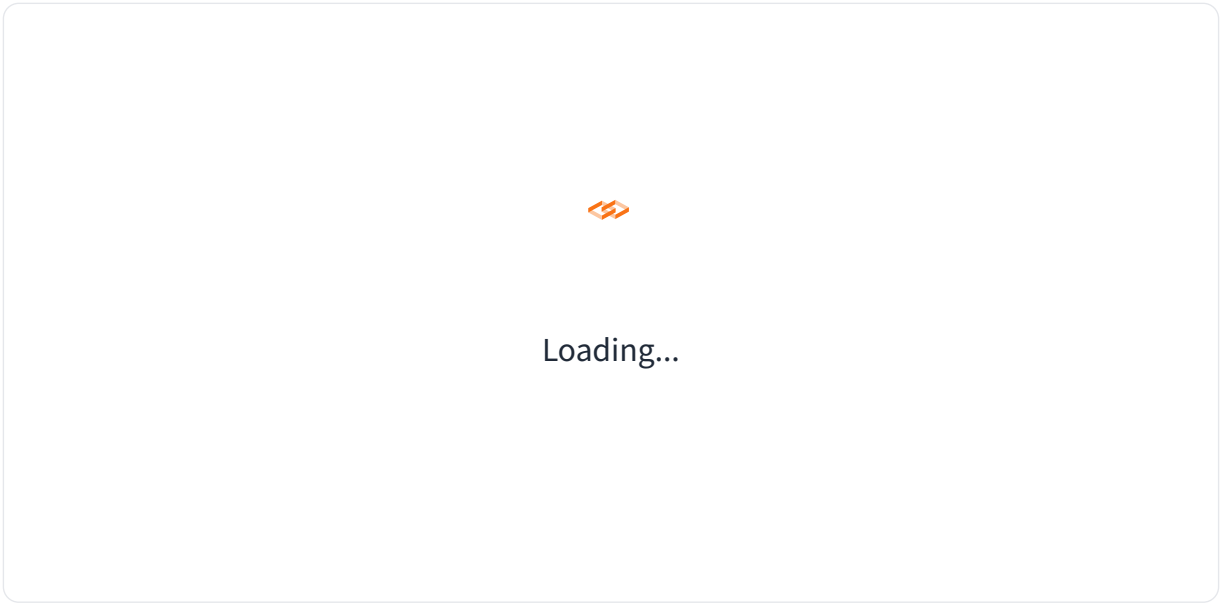
We'll now build the data collator that builds training batches from the dataset samples, and start training. In A100 with 40GB memory, we can fit in 6 examples. If you're training on T4, you can use a batch size of 1.

```
import os
from torch.utils.data import DataLoader
from tqdm import tqdm
from transformers import AdamW, get_scheduler

def collate_fn(batch):
    questions, answers, images = zip(*batch)
    inputs = processor(text=list(questions), images=list(images), retu
    return inputs, answers

train_dataset = DocVQADataset(data['train'])
val_dataset = DocVQADataset(data['validation'])
batch_size = 6
num_workers = 0

train_loader = DataLoader(train_dataset, batch_size=batch_size,
                          collate_fn=collate_fn, num_workers=num_worke
val_loader = DataLoader(val_dataset, batch_size=batch_size,
                          collate_fn=collate_fn, num_workers=num_worke
```

We can train the model now.

```
epochs = 7
optimizer = AdamW(model.parameters(), lr=1e-6)
num_training_steps = epochs * len(train_loader)

lr_scheduler = get_scheduler(name="linear", optimizer=optimizer,
                             num_warmup_steps=0, num_training_steps=nu

for epoch in range(epochs):
    model.train()
    train_loss = 0
    i = -1
    for inputs, answers in tqdm(train_loader, desc=f"Training Epoch {ep
        i += 1
        input_ids = inputs["input_ids"]
        pixel_values = inputs["pixel_values"]
        labels = processor.tokenizer(text=answers, return_tensors="pt"
        outputs = model(input_ids=input_ids, pixel_values=pixel_values
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        train_loss += loss.item()
    avg_train_loss = train_loss / len(train_loader)
    print(f"Average Training Loss: {avg_train_loss}")

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for batch in tqdm(val_loader, desc=f"Validation Epoch {epoch +
            inputs, answers = batch
            input_ids = inputs["input_ids"]
            pixel_values = inputs["pixel_values"]
            labels = processor.tokenizer(text=answers, return_tensors=
            outputs = model(input_ids=input_ids, pixel_values=pixel_va
            loss = outputs.loss
            val_loss += loss.item()

      print(val_loss / len(val_loader))
```

You can save the model and processor by calling `save_pretrained()` on both objects. The fully fine-tuned model is here and the demo is here.



Conclusions

In this post, we showed that Florence-2 can be effectively fine-tuned to a custom dataset, achieving impressive performance on a completely new task in a short amount of time. This capability is particularly valuable for those looking to deploy this small model on devices or use it cost-effectively in production environments. We encourage the open-source community to leverage this fine-tuning tutorial and explore the remarkable potential of Florence-2 for a wide range of new tasks! We can't wait to see your models on the 🤗 Hub!

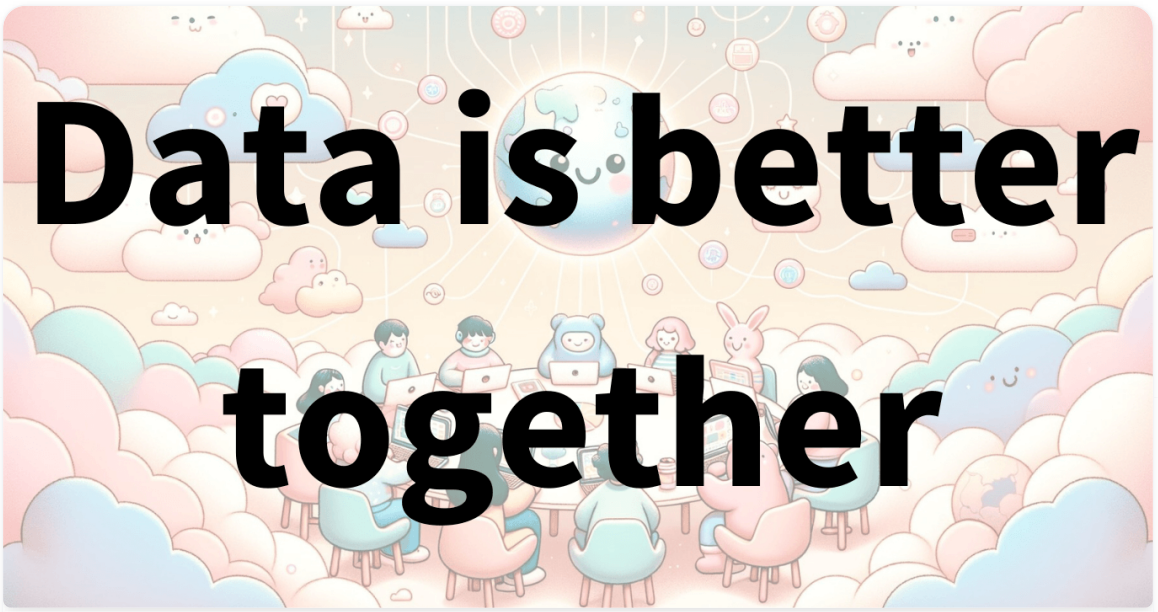
Useful Resources

- Vision Language Models Explained
- Fine tuning Colab
- Fine tuning Github Repo
- Notebook for Florence-2 Inference
- Florence-2 DocVQA Demo

- Florence-2 Demo

We would like to thank Pedro Cuenca for his reviews on this blog post.

More Articles from our Blog



Data Is Better Together: A Look Back and Forward

By davanstrien June 20, 2024 • △ 12

BigCodeBench: Benchmarking Large Language Models on Solving Practical and Challenging Programming Tasks

By terryyz June 18, 2024 guest • △ 28

Company

TOS

Privacy

About

Jobs

Website

Models

Datasets

Spaces

Pricing

Docs