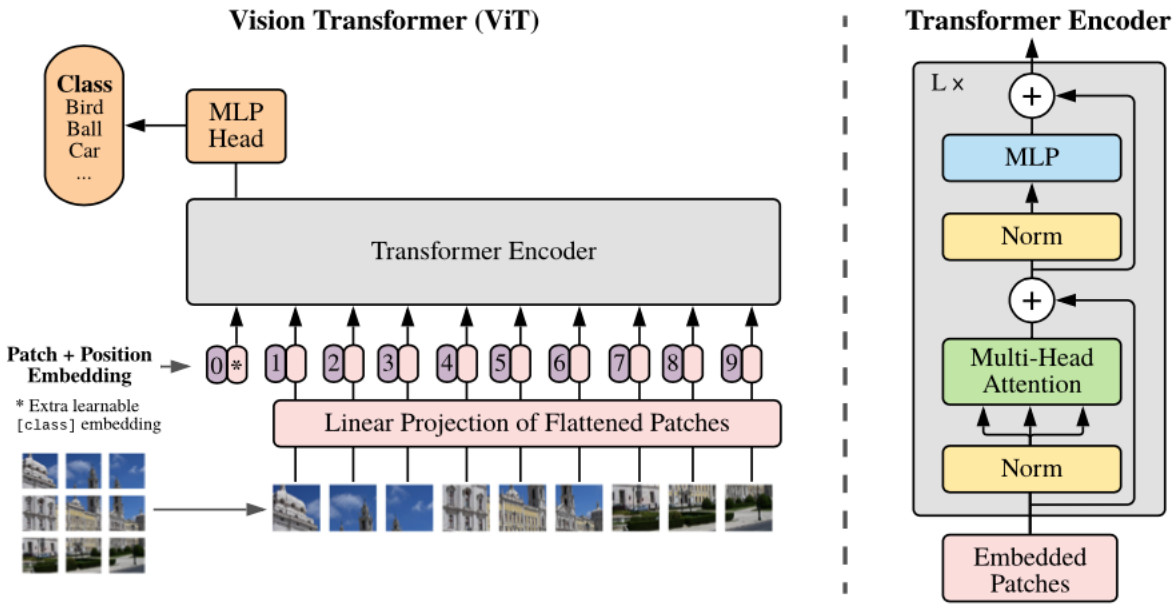
P2: ViT for Image Classification (30%)

Quick intro: Vision Transformer (ViT) by Google Brain

The Vision Transformer (ViT) is basically BERT, but applied to images. It attains excellent results compared to state-of-the-art convolutional networks. In order to provide images to the model, each image is split into a sequence of fixed-size patches (typically of resolution 16x16 or 32x32), which are linearly embedded. One also adds a [CLS] token at the beginning of the sequence in order to classify images. Next, one adds absolute position embeddings and provides this sequence to the Transformer encoder.



- Paper: https://arxiv.org/abs/2010.11929
- Official repo (in JAX): https://github.com/google-research/vision_transformer

Performing Inference with the Vision Transformer for Image Classification

```
1 !pip install 'transformers[torch]'
```

```
Requirement already satisfied: transformers[torch] in /usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (0.19.
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (4.66.1)
Requirement already satisfied: torch!=1.12.0,>=1.10 in /usr/local/lib/python3.10/dist-packages (from transformers[torch]) (2.1.0+cu118)
Collecting accelerate>=0.20.3 (from transformers[torch])
  Downloading accelerate-0.25.0-py3-none-any.whl (265 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 265.7/265.7 kB 6.8 MB/s eta 0:00:00
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from accelerate>=0.20.3->transformers[torch]) (5.9.5)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4-
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (3.2
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (3.1.2
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch])
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers[torch])
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers[torch]) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers[torch]) (2.0.7
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers[torch]) (2023.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch!=1.12.0,>=1.10->transforme
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch!=1.12.0,>=1.10->transformers[t
```

```
Installing collected packages: accelerate
Successfully installed accelerate-0.25.0
```

Let's start by installing the relevant libraries, and load the model from the hub, then move it to GPU.

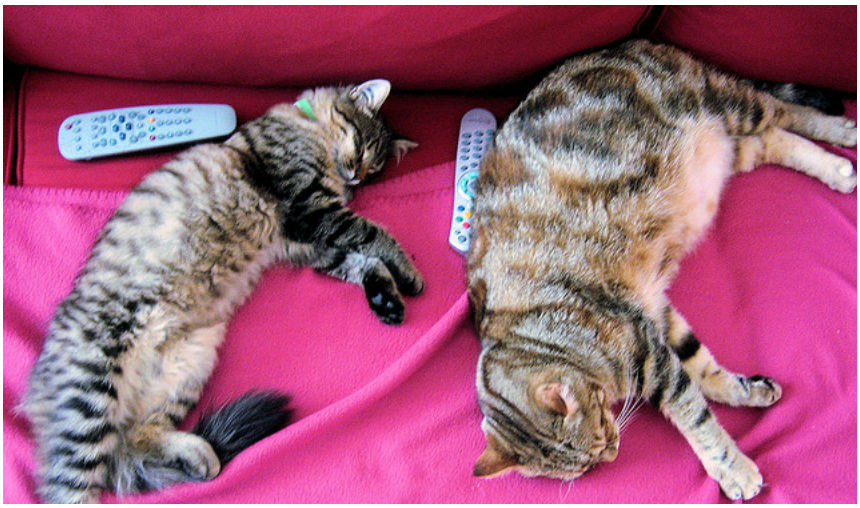
```
from transformers import ViTForImageClassification
import torch

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

# NOTE: Here we are using a pretrained ViT-B model
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')
model.eval()
model.to(device)
```

```
config.json: 100%                                                 69.7k/69.7k [00:00<00:00, 2.24MB/s]

model.safetensors: 100%                                           346M/346M [00:01<00:00, 208MB/s]
```

```
ViTForImageClassification(
  (vit): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): ViTSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (output): ViTSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (layernorm_after): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
    (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (classifier): Linear(in_features=768, out_features=1000, bias=True)
)
```

Let's simply load in an image to let the ViT model perform inference on it.

```
from PIL import Image
import requests

url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
im = Image.open(requests.get(url, stream=True).raw)
im
```



```
1 im.size
```

```
(640, 480)
```

Pre-Processing: As mentioned in the paper, the model accepts an input resolution of `224x224`. The sample we have here is (640, 480). Here we use `ViTFeatureExtractor` to take care of `resizing` + `normalization`.

```
1 from transformers import ViTFeatureExtractor
2
3 feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224')
4 # You can check out its implementation here: https://huggingface.co/transformers/_modules/transformers/models/vit/feature_extraction_vit.
5 encoding = feature_extractor(images=im, return_tensors="pt")
6 encoding.keys()
```

```
preprocessor_config.json: 100%  160/160 [00:00<00:00, 10.7kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/vit/feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtract
  warnings.warn(
dict_keys(['pixel_values'])
```

```
1 # Check the input after applying
2 encoding['pixel_values'].shape
```

```
torch.Size([1, 3, 224, 224])
```

Forward Pass: Let's send the image through the ViT model, which consists of a BERT-like encoder and a linear classification head on top of the last hidden state of the [CLS] token.

```
1 im_input = encoding['pixel_values'].to(device)
2
3 # Feedforward
4 outputs = model(im_input)
5 logits = outputs.logits
6 logits.shape
```

```
torch.Size([1, 1000])
```

```
1 prediction = logits.argmax(-1)
2 print("Predicted class:", model.config.id2label[prediction.item()])
```

```
Predicted class: Egyptian cat
```

Questions:

1. Please find another image from the web (or you can upload them from local as well, any image that is good for classification is fine), and use the ViT-B model to get the predictions.

```
# TODO: Please write your codes for Q1 here

url = 'https://i0.wp.com/blog.pensoft.net/wp-content/uploads/2021/09/covergushter.jpg?fit=900%2C506&ssl=1'
im = Image.open(requests.get(url, stream=True).raw)
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224')
# You can check out its implementation here: https://huggingface.co/transformers/_modules/transformers/models/vit/feature_extraction_vit.
encoding = feature_extractor(images=im, return_tensors="pt")
im_input = encoding['pixel_values'].to(device)

# Feedforward
outputs = model(im_input)
logits = outputs.logits
prediction = logits.argmax(-1)
print("Predicted class:", model.config.id2label[prediction.item()])
```

```
Predicted class: agama
/usr/local/lib/python3.10/dist-packages/transformers/models/vit/feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtract
  warnings.warn(
```

2. As we know, pre-training a ViT model often requires lots of data and computation resources and that's why we use the pre-trained model weights provided by the official. Please visit the [HuggingFace model hub](#) for ViT model, and choose one to replace the `google/vit-base-patch16-224` in the following codes. Run the inference using the new pre-trained model checkpoints again and get the predictions.

(NOTE: Some of the pre-trained weights may be too large to fit for your GPU, choose the one that works.)

```
# TODO: Please write your codes for Q2 here
# NOTE: Here we are using a ViT-B model on ImageNet 2012 (1 million images, 1,000 classes) at resolution 224x224, replace it with the one
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-384')
model.eval()
model.to(device)
```

```
config.json: 100%                                   69.7k/69.7k [00:00<00:00, 1.25MB/s]

model.safetensors: 100%                             347M/347M [00:01<00:00, 169MB/s]

ViTForImageClassification(
  (vit): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): ViTSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (output): ViTSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (layernorm_after): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
    (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (classifier): Linear(in_features=768, out_features=1000, bias=True)
)
```

3. **Clearly** explain why the [CLS] logits have a shape of [1, 1000]. What if we train the model on CIFAR-10 from scratch, will this shape of logits change? (Yes/No), How? (The shape).

THE [CLS] Logits have an nx1000 shape because it has 1000 labels. If we train it from scratch, it will change to the number of labels we give, or "nxm" where m is the number of labels

4. What if I already had a trained model on ImageNet1k, in this case, a `vit-base-patch16-224` checkpoint , and now I have a new dataset, e.g., CIFAR-10, coming with multiple new classes. What do you think is the best strategy to adapt my existing models to the new datasets?

Here is the link fine-tuning for transfer learning for your reference. You can read it first and then come back to answer this question.

The answer to the above question depends deeply on how many images each has. In addition, we are assuming that the images on both pretrained ccheckpoints and the current dataset are similar.

Since the Imagenet has around a million images and this vitbase has 14 million similar images, we can just pick up where we left off, and fine tune the model weights further. We can be relatively confident in the lack of over tuning.

Fine-tune the Vision Transformer on CIFAR-10

In this section, we are going to fine-tune a pre-trained Vision Transformer on the CIFAR-10 dataset. This dataset is a collection of 60,000 32x32 colour images in 10 classes, with 6000 images per class, which we have been familiar with in the past homeworks.

We will prepare the data using HuggingFace datasets, and train the model using Pytorch.

Preprocessing the data: Here we import a small portion of CIFAR-10, and prepare it for the model using `ViTFeatureExtractor`. This feature extractor will resize every image to the resolution that the model expects (let's use the default 224), and normalize the channels.

Note that in the ViT paper, the best results were obtained when fine-tuning at a higher resolution. For this, one interpolates the pre-trained absolute position embeddings.

```
1 !pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.15.0-py3-none-any.whl (521 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.2/521.2 kB 7.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (9.0.0)
Collecting pyarrow-hotfix (from datasets)
  Downloading pyarrow_hotfix-0.6-py3-none-any.whl (7.9 kB)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 17.4 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 21.0 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.19.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.3)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.18.0->datasets) (3.13.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.18.0->data
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2023.11.
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->datasets) (1.16
Installing collected packages: pyarrow-hotfix, dill, multiprocess, datasets
Successfully installed datasets-2.15.0 dill-0.3.7 multiprocess-0.70.15 pyarrow-hotfix-0.6
```

```
from transformers import ViTFeatureExtractor

# NOTE: Here we are using a ViT-B model pre-trained on ImageNet-21k (14 million images, 21,843 classes) at resolution 224x224
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')
```

```
preprocessor_config.json: 100%  160/160 [00:00<00:00, 7.65kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/vit/feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtract
  warnings.warn(
```

```
from datasets import load_dataset

# load cifar10 (only small portion for demonstration purposes)
train_ds, test_ds = load_dataset('cifar10', split=['train[:5000]', 'test[:2000]'])
# split up training into training + validation
splits = train_ds.train_test_split(test_size=0.1)
train_ds = splits['train']
val_ds = splits['test']
```

```
Downloading builder script: 100%  3.61k/3.61k [00:00<00:00, 202kB/s]
Downloading metadata: 100%  1.66k/1.66k [00:00<00:00, 50.9kB/s]
Downloading readme: 100%  5.00k/5.00k [00:00<00:00, 259kB/s]
Downloading data: 100%  170M/170M [00:13<00:00, 14.9MB/s]
Generating train split: 100%  50000/50000 [00:34<00:00, 2645.36 examples/s]
Generating test split: 100%  10000/10000 [00:06<00:00, 2388.84 examples/s]
```

Thanks to HuggingFace Datasets' `.map(function, batched=True)` functionality, we can prepare images very fast (note that it still takes a few minutes on Google Colab).

```
import numpy as np

def preprocess_images(examples):
    # get batch of images
    images = examples['img']
    # convert to list of NumPy arrays of shape (C, H, W)
    images = [np.array(image, dtype=np.uint8) for image in images]
    images = [np.moveaxis(image, source=-1, destination=0) for image in images]
    # preprocess and add pixel_values
    inputs = feature_extractor(images=images)
    examples['pixel_values'] = inputs['pixel_values']

    return examples
```

```
from datasets import Features, ClassLabel, Array3D, Image

# we need to define the features ourselves as both the img and pixel_values have a 3D shape
features = Features({
    'label': ClassLabel(names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']),
    'img': Image(decode=True, id=None),
    'pixel_values': Array3D(dtype="float32", shape=(3, 224, 224)),
})

preprocessed_train_ds = train_ds.map(preprocess_images, batched=True, features=features)
preprocessed_val_ds = val_ds.map(preprocess_images, batched=True, features=features)
preprocessed_test_ds = test_ds.map(preprocess_images, batched=True, features=features)
```

```
Map: 100%  4500/4500 [00:50<00:00, 86.92 examples/s]
Map: 100%  500/500 [00:04<00:00, 107.15 examples/s]
Map: 100%  2000/2000 [00:19<00:00, 103.44 examples/s]
```

Define the model The model itself uses a linear layer on top of a pre-trained `ViTModel`. We place a linear layer on top of the last hidden state of the [CLS] token, which serves as a good representation of an entire image. We also add dropout for regularization.

```
from transformers import ViTModel
```

```
from transformers.modeling_outputs import SequenceClassifierOutput
import torch.nn as nn

class ViTForImageClassification(nn.Module):
    def __init__(self, num_labels=10):
        super(ViTForImageClassification, self).__init__()
        self.vit = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
        self.num_labels = num_labels
        self.dropout = nn.Dropout(0.1)
        self.classifier = nn.Linear(self.vit.config.hidden_size, num_labels)

    def forward(self, pixel_values, labels):
        outputs = self.vit(pixel_values=pixel_values)
        output = self.dropout(outputs.last_hidden_state[:,0])
        logits = self.classifier(output)

        loss = None
        if labels is not None:
          loss_fct = nn.CrossEntropyLoss()
          loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))

        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

To instantiate a `Trainer`, we will need to define three more things. The most important is the `TrainingArguments`, which is a class that contains all the attributes to customize the training. It requires one folder name, which will be used to save the checkpoints of the model, and all other arguments are optional:

```
from transformers import TrainingArguments, Trainer

metric_name = "accuracy"

args = TrainingArguments(
    f"test-cifar-10",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=10,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    metric_for_best_model=metric_name,
    logging_dir='logs',
)
```

Here we set the evaluation to be done at the end of each epoch, tweak the learning rate, set the training and evaluation batch_sizes and customize the number of epochs for training, as well as the weight decay.

Then we will need a data collator that will batch our processed examples together, here the default one will work:

```
from transformers import default_data_collator

data_collator = default_data_collator
```

```
model = ViTForImageClassification()
```

```
config.json: 100%  502/502 [00:00<00:00, 26.7kB/s]
pytorch_model.bin: 100%  346M/346M [00:01<00:00, 338MB/s]
```

We also define a `compute_metrics` function that will be used to compute metrics at evaluation. We use "accuracy", which is available in HuggingFace Datasets.

```
from datasets import load_metric
import numpy as np

metric = load_metric("accuracy")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return metric.compute(predictions=predictions, references=labels)
```

```
<ipython-input-22-73d77e7e10a9>:4: FutureWarning: load_metric is deprecated and will be removed in the next major version of datasets. U
  metric = load_metric("accuracy")
```

Downloading builder script: 4.21k/? [00:00<00:00, 200kB/s]

Then we just need to pass all of this along with our datasets to the Trainer:

```
trainer = Trainer(
    model,
    args,
    train_dataset=preprocessed_train_ds,
    eval_dataset=preprocessed_val_ds,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

Train the model: Let's first start up Tensorboard:

We can now finetune our model by just calling the `train` method:

```
trainer.train()
```

[1350/1350 17:27, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.522791	0.952000
2	1.123500	0.279570	0.962000
3	0.270700	0.240133	0.962000

```
TrainOutput(global_step=1350, training_loss=0.5582089855052806, metrics={'train_runtime': 1049.2219, 'train_samples_per_second':
12.867, 'train_steps_per_second': 1.287, 'total_flos': 0.0, 'train_loss': 0.5582089855052806, 'epoch': 3.0})
```

```
# Start tensorboard.
%load_ext tensorboard
%tensorboard --logdir logs/
```

Evaluation: Finally, let's evaluate the model on the test set:

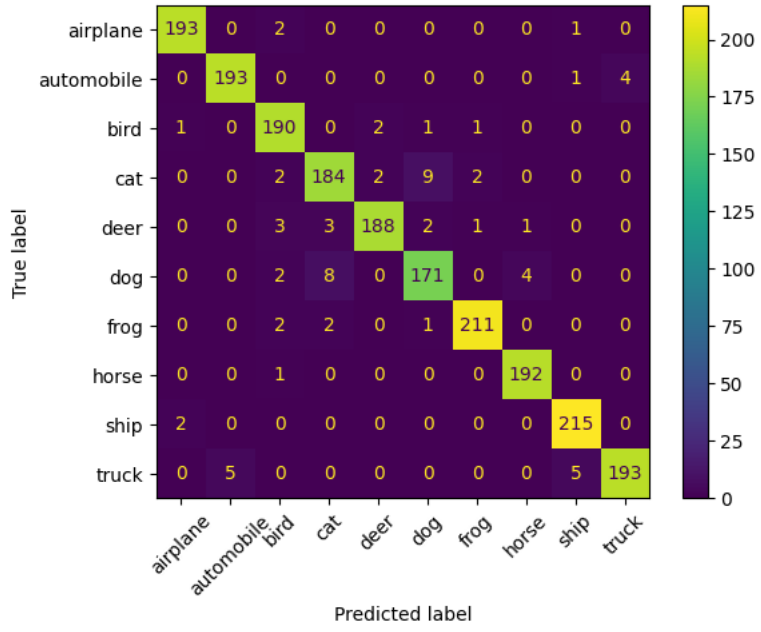
```
1 outputs = trainer.predict(preprocessed_test_ds)
2
3 print(outputs.metrics)
```

```
{'test_loss': 0.23571895062923431, 'test_accuracy': 0.965, 'test_runtime': 106.7509, 'test_samples_per_second': 18.735, 'test_steps_per
```

We can also easily create a confusion matrix using `sklearn`

```
1 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
2
3 y_true = outputs.label_ids
4 y_pred = outputs.predictions.argmax(1)
5
6 labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
7 cm = confusion_matrix(y_true, y_pred)
8 disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
9 disp.plot(xticks_rotation=45)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e5368f456f0>
```



Questions:

5. Fine-tune the model based on CIFAR-10 dataset using given `vit-base-patch16-224-in21k`, which is a ViT-B model pre-trained on ImageNet-21k (14 million images, 21,843 classes) at resolution 224x22, for 3 epochs with end-to-end setting (it might take ~20 mins on Colab). Report the accuracy on the test set, show the whole tensorboard training logs and the confusion matrix from sklearn.

6. Fine-tune the model based on CIFAR-10 dataset using `vit-base-patch16-224-in21k`, which is a ViT-B model pre-trained on ImageNet-21k, for 3 epochs with backbone (encoder) fixed setting (see torch.no_grad() for details). Please also report the accuracy on the test set, show the whole tensorboard training logs and the confusion matrix from sklearn.

7. Think about the difference between these two settings (end-to-end vs. backbone (encoder) fixed). Why we need the second settings, especially when we are facing even larger models (e.g., ViT-L `vit-large-patch16-384-in21k`)?

The larger models can sometimes overshadow the smaller ones, especially when you want to keep some of the data in the smaller models. This way, you can keep the smaller model's key data.

```
features = Features({
    'label': ClassLabel(names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']),
    'img': Image(decode=True, id=None),
    'pixel_values': Array3D(dtype="float32", shape=(3, 224, 224)),
})

preprocessed_train_ds = train_ds.map(preprocess_images, batched=True, features=features)
preprocessed_val_ds = val_ds.map(preprocess_images, batched=True, features=features)
preprocessed_test_ds = test_ds.map(preprocess_images, batched=True, features=features)

class ViTForImageClassification(nn.Module):
    @torch.no_grad()
    def __init__(self, num_labels=10):
        super(ViTForImageClassification, self).__init__()
        self.vit = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
        self.num_labels = num_labels
        self.dropout = nn.Dropout(0.1)
        self.classifier = nn.Linear(self.vit.config.hidden_size, num_labels)

    def forward(self, pixel_values, labels):
        outputs = self.vit(pixel_values=pixel_values)
        output = self.dropout(outputs.last_hidden_state[:,0])
        logits = self.classifier(output)

        loss = None
        if labels is not None:
          loss_fct = nn.CrossEntropyLoss()
          loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))

        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )

metric_name = "accuracy"

args = TrainingArguments(
    f"test-cifar-10",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=10,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    metric_for_best_model=metric_name,
    logging_dir='logs',
)

data_collator = default_data_collator
model = ViTForImageClassification()

metric = load_metric("accuracy")
trainer = Trainer(
    model,
    args,
    train_dataset=preprocessed_train_ds,
    eval_dataset=preprocessed_val_ds,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
trainer.train()
```

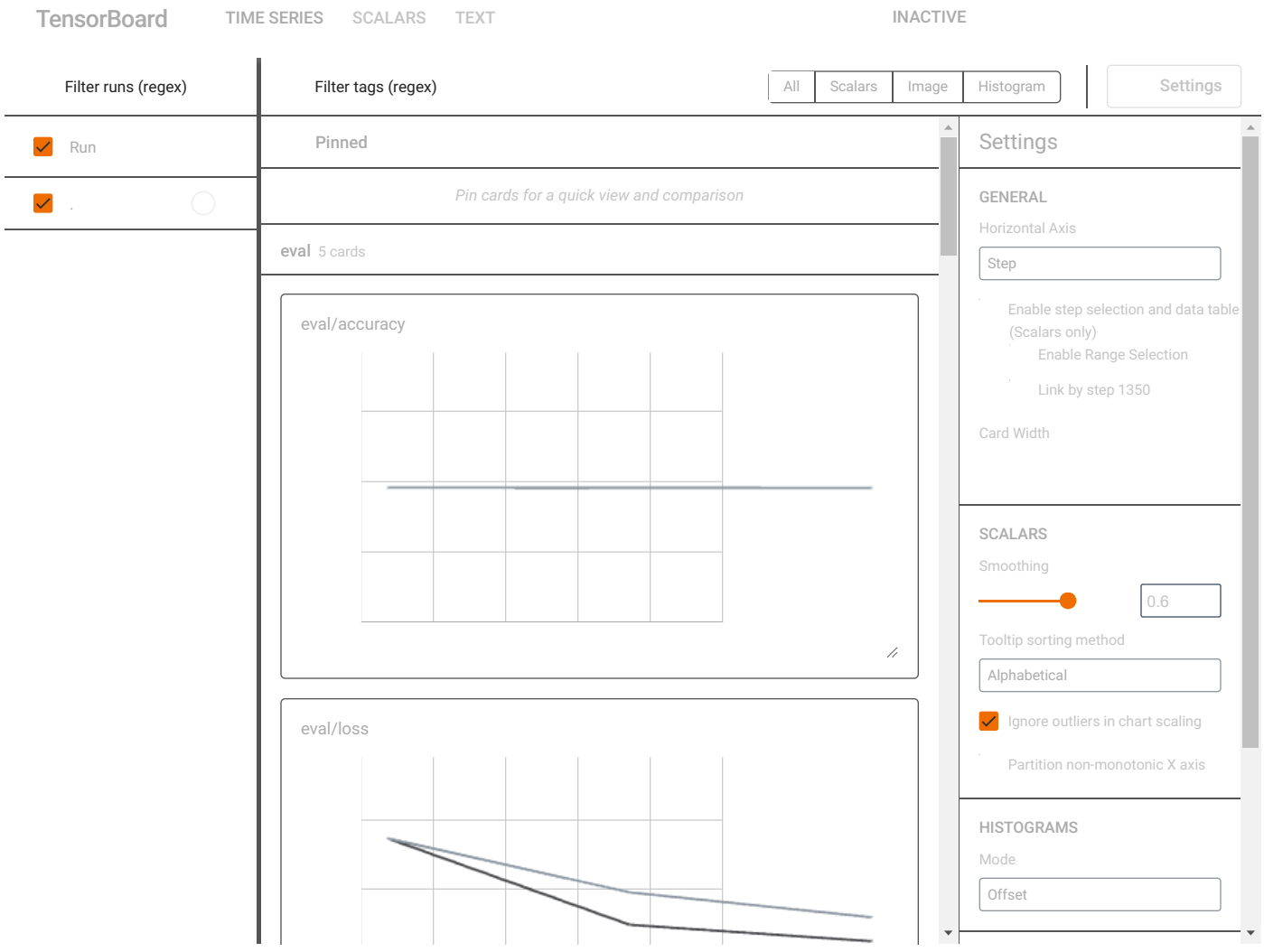
[1350/1350 17:26, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.547445	0.960000
2	1.150500	0.297475	0.958000
3	0.284300	0.250140	0.958000

```
TrainOutput(global_step=1350, training_loss=0.5745002548782914, metrics={'train_runtime': 1047.225, 'train_samples_per_second': 12.891,
'train_steps_per_second': 1.289, 'total_flos': 0.0, 'train_loss': 0.5745002548782914, 'epoch': 3.0})
```

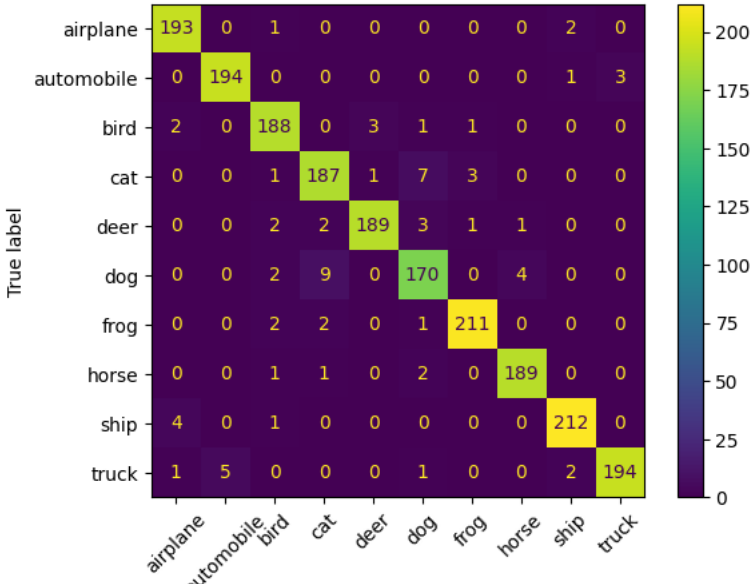
```
# Start tensorboard.
%load_ext tensorboard
%tensorboard --logdir logs/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 12363), started 0:41:09 ago. (Use '!kill 12363' to kill it.)
```




```
outputs = trainer.predict(preprocessed_test_ds)

print(outputs.metrics)
y_true = outputs.label_ids
y_pred = outputs.predictions.argmax(1)

labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(xticks_rotation=45)
```

```
{'test_loss': 0.24328432977199554, 'test_accuracy': 0.9635, 'test_runtime': 108.6428, 'test_samples_per_second': 18.409, 'test_steps_per
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e554b65ce50>
```




```
1 Start coding or generate with AI.
```