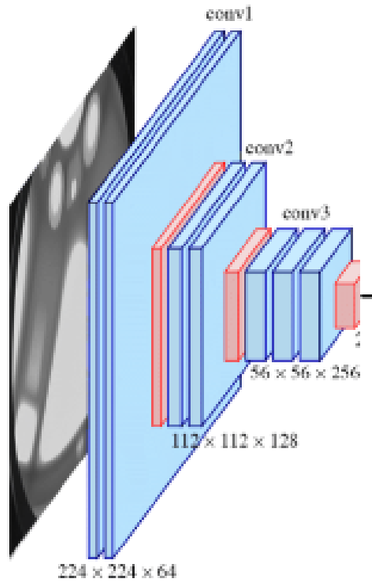
Computer vision

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

datacamp

Vision models



Feature extraction

$c_1, \ldots, c_N$

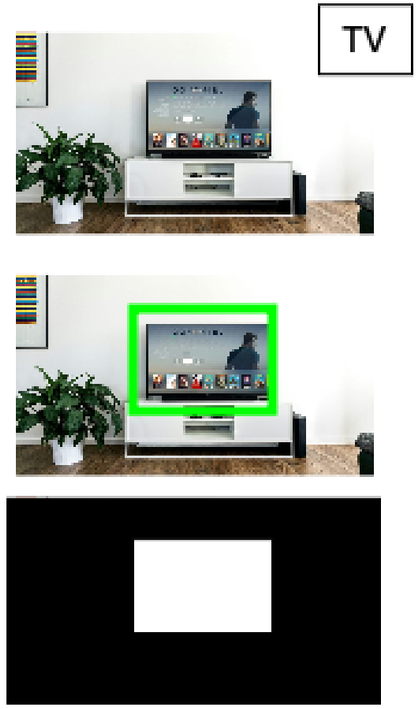
Classification

$[[x_{min}, x_{max}, y_{min}, y_{max}] \ldots]$

Object detection

Z_{dxd}

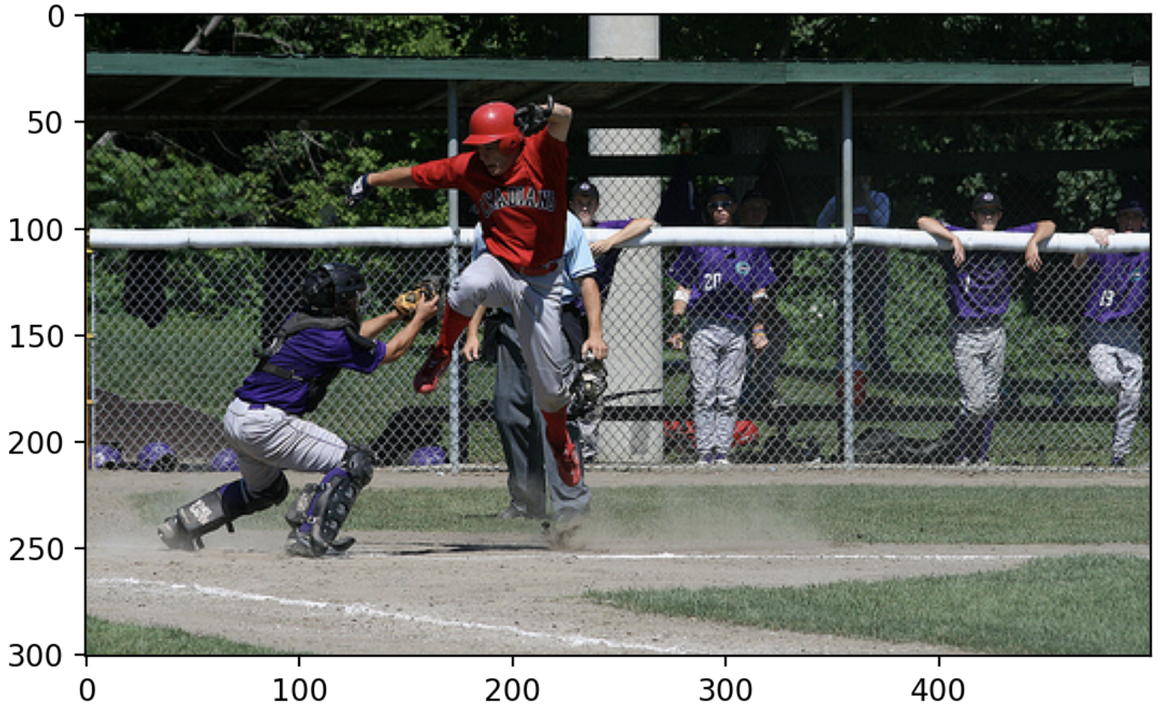
Segmentation



[1] https://arxiv.org/abs/1409.1556

Classification

```
from datasets import load_dataset
dataset = load_dataset("nlphuji/flickr30k")
image = dataset['test'][134]["image"]
```




```
from transformert import pipeline
pipe = pipeline("image-classification",
"google/mobilenet_v2_1.0_224") # 224x224 input

pred = pipe(image)
print("Predicted class:", pred[0]['label'])
```

```
Predicted class: ballplayer, baseball player
```

Object detection

```
dataset['test'][52]["image"]
```

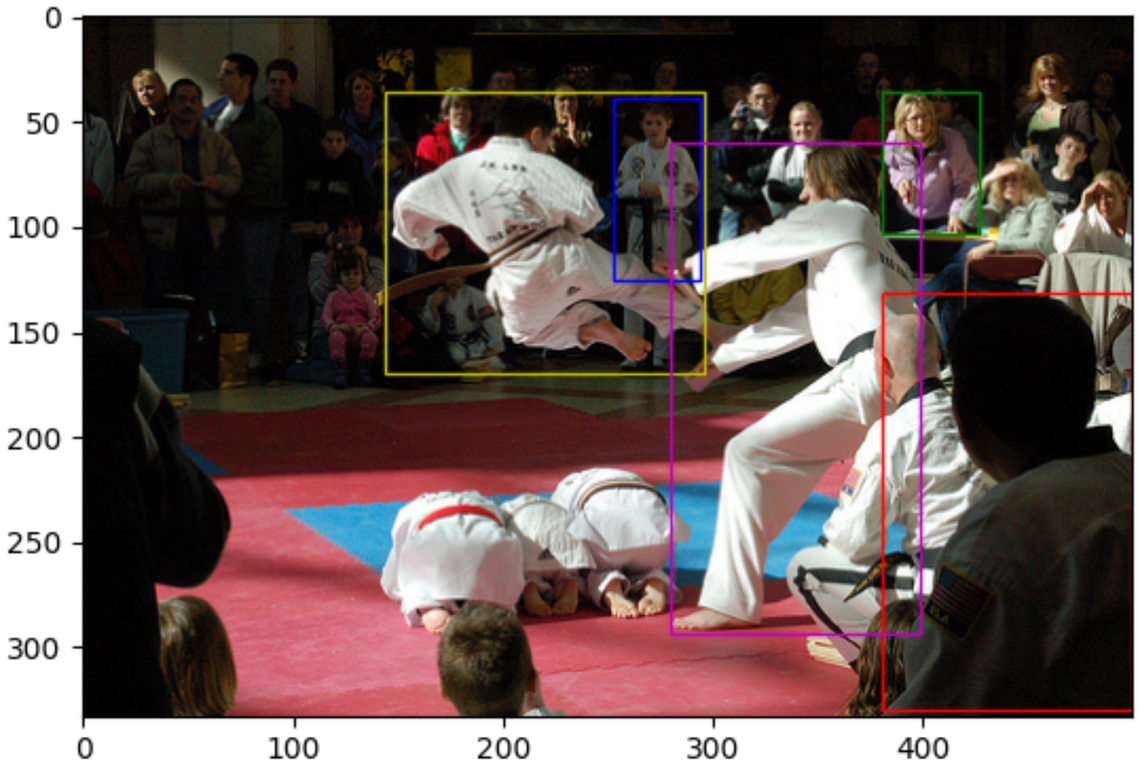
Object detection

```
pipe = pipeline("object-detection", "facebook/detr-resnet-50", revision="no_timm")
outputs = pipe(image, threshold=0.95)

for obj in outputs:
    box = obj['box']
    print(f"Detected {obj['label']} with confidence {obj['score']:.2f}
    at ({box['xmin']}, {box['ymin']}) to ({box['xmax']}, {box['ymax']})")
```

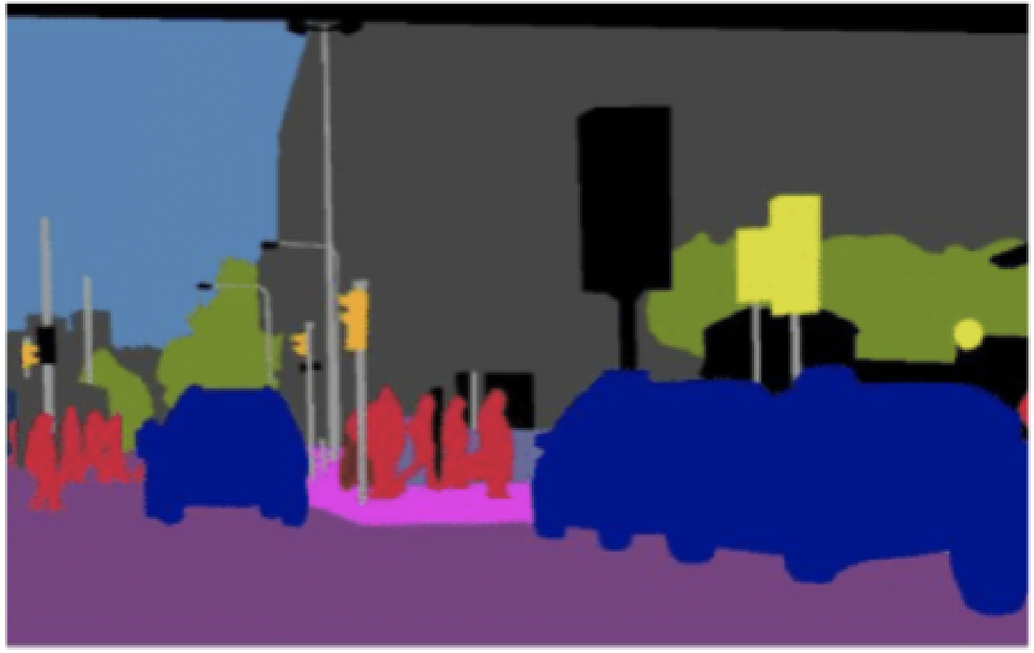
```
Detected person with confidence 0.97 at (381, 131) to (499, 330)
Detected person with confidence 0.96 at (381, 36) to (427, 103)
Detected person with confidence 0.98 at (253, 39) to (294, 125)
Detected person with confidence 1.00 at (144, 36) to (296, 170)
Detected person with confidence 0.95 at (280, 60) to (399, 294)
```

Object detection

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
ax = plt.gca()
colors = ['r', 'g', 'b', 'y', 'm', 'c', 'k']
plt.imshow(image)
for n, obj in enumerate(outputs):
    box = obj['box']
    rect = patches.Rectangle(
      (box['xmin'], box['ymin']),
      box['xmax']-box['xmin'],
      box['ymax']-box['ymin'],
      linewidth=1,
      edgecolor=colors[n], facecolor='none')
    ax.add_patch(rect)
plt.show()
```



Segmentation



- Output: 2D array with same dimensions as input
- Background removal: each pixel is `1` (foreground) or `0` (background)
- Image $\times$ Output → Image with background removed

Segmentation

```
pipe = pipeline("image-segmentation",
                model="briaai/RMBG-1.4",
                trust_remote_code=True)

outputs = pipe(image)

plt.imshow(outputs)
plt.show()
```

Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

Fine-tuning computer vision models

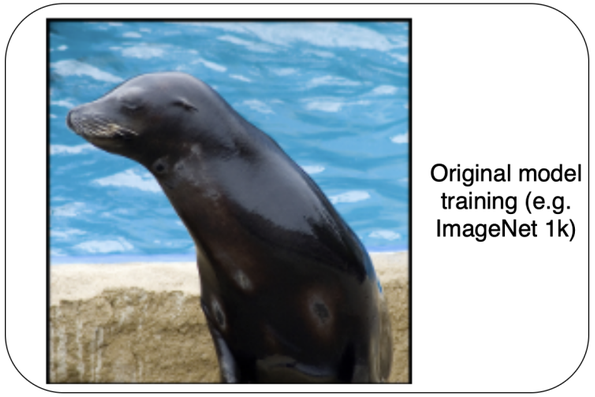
MULTI-MODAL MODELS WITH HUGGING FACE

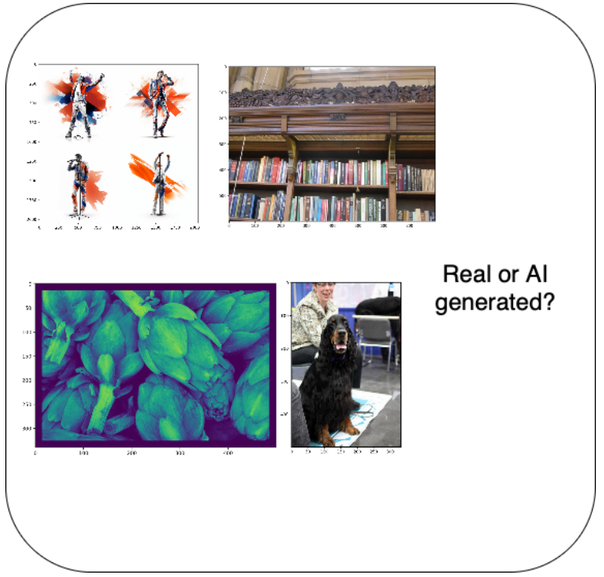
James Chapman
Curriculum Manager, DataCamp

datacamp

Purpose of fine-tuning vision models

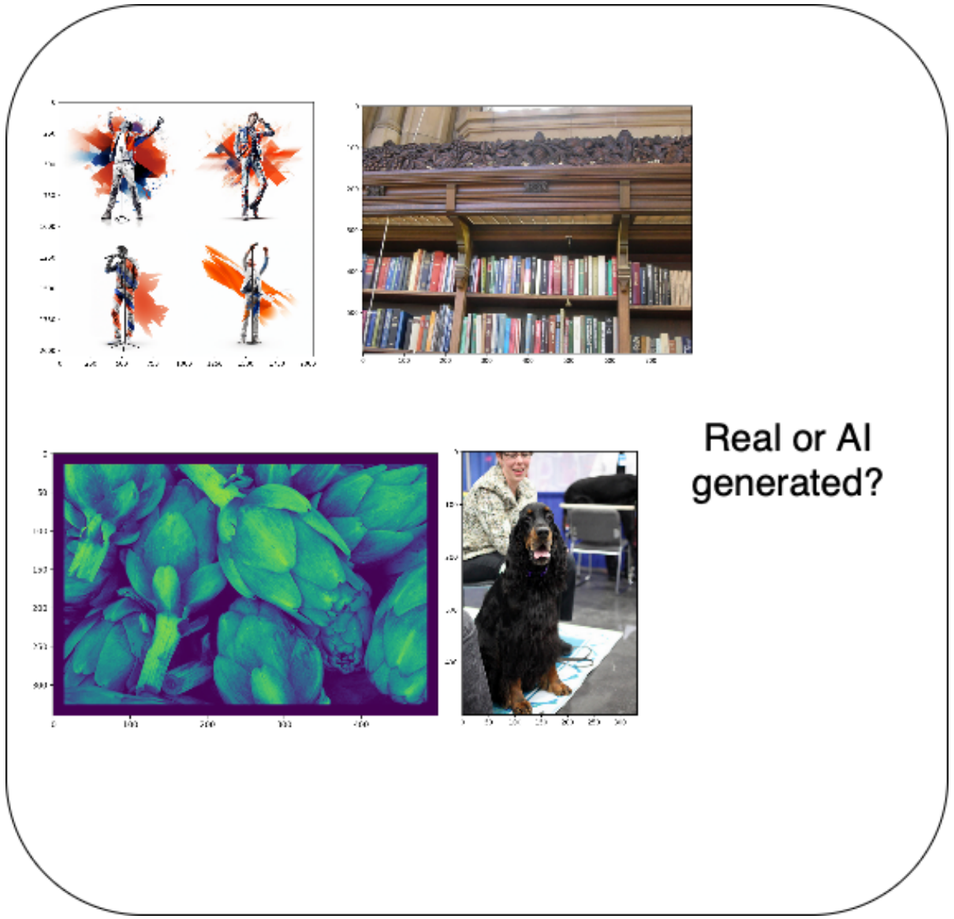
- New classes, e.g., binary classification of real or AI-generated images
- New imaging domains, e.g., X-rays





[1] https://image-net.org/index.php

Fine-tuning vision models



1. Adjust model output to the new predictions
2. Prepare dataset for training
3. Configure training options
4. Train!

Model updates

```
from datasets import load_dataset
dataset = load_dataset("ideepankarsharma2003/Midjourney_v6_Classification_small_shuf
fled")['train']
data_splits = dataset.train_test_split(test_size=0.2, seed=42)

labels = data_splits["train"].features["label"].names
label2id, id2label = dict(), dict()
for i, label in enumerate(labels):
    label2id[label] = str(i)
    id2label[str(i)] = label
```

Model updates

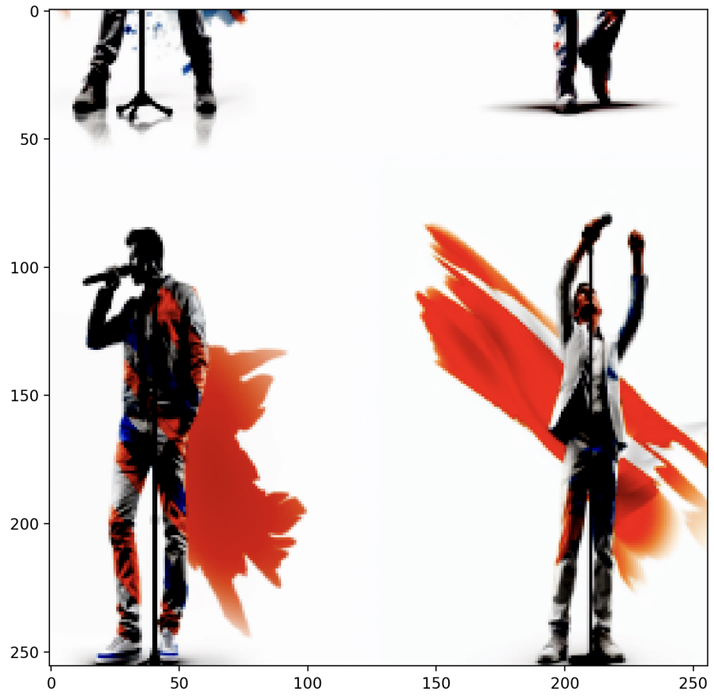
```
from transformers import AutoModelForImageClassification
checkpoint = "google/mobilenet_v2_1.0_224"
model = AutoModelForImageClassification.from_pretrained(
    checkpoint,
    num_labels=len(labels),
    id2label=id2label,
    label2id=label2id,
    ignore_mismatched_sizes=True
)
```

Dataset preparation

```
from transformers import AutoImageProcessor
image_processor = AutoImageProcessor.from_pretrained(checkpoint)

from torchvision.transforms import Compose, Normalize, ToTensor
normalize = Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
transform = Compose([ToTensor(), normalize])

def transforms(examples):
    examples["pixel_values"] = [transform(img.convert("RGB")) for img in examples["image"]]
    del examples["image"]
    return examples

dataset = dataset.with_transform(transforms)
```

Plotting transformed data

```
import matplotlib.pyplot as plt
plt.imshow(dataset["train"][0]["pixel_values"].permute(1, 2, 0))
plt.show()
```



Training

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="dataset_finetune",
    learning_rate=6e-5,
    gradient_accumulation_steps=4,
    num_train_epochs=3,
    push_to_hub=False
)
```

```
from transformers import Trainer,
    DefaultDataCollator

data_collator = DefaultDataCollator()

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    processing_class=image_processor,
    data_collator=data_collator
)
```

Evaluation

```
predictions = trainer.predict(dataset["test"])
predictions.metrics["test_accuracy"]
```

```
0.455
```

```
trainer.train()
```

```
{..., 'eval_accuracy': 0.93, ...}
```

Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

Speech recognition and audio generation

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

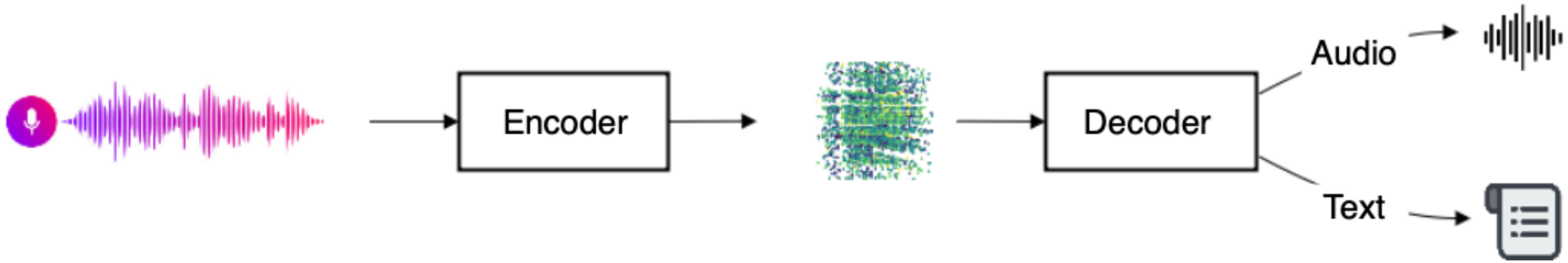
datacamp

Speech

What are the important parts of speech waveforms?

- **Pitch**: avg. frequency for males 85-180Hz, for females 165-255Hz
- **Stress**: language, accent, and emotion specific
- **Rhythm**: influenced by context and emotion and language

Automatic speech recognition



- Text and audio are both *sequential*
- **Examples:** transcription, translation, text-to-speech

Automatic speech recognition

- Tiny model: 39M parameters, 150MB model size
- Training data: 680k hours (labeled)

```
from transformers import WhisperProcessor, WhisperForConditionalGeneration
processor = WhisperProcessor.from_pretrained("openai/whisper-tiny")
model = WhisperForConditionalGeneration.from_pretrained("openai/whisper-tiny")
```

[1] https://github.com/openai/whisper

Automatic speech recognition

```
from datasets import load_dataset, Audio
dataset = load_dataset("CSTR-Edinburgh/vctk")["train"]
dataset = dataset.cast_column("audio", Audio(sampling_rate=16_000))
sample = dataset[0]["audio"]
input_preprocessed = processor(sample["array"],
                               sampling_rate=sample["sampling_rate"],
                               return_tensors="pt")
predicted_ids = model.generate(input_preprocessed.input_features)
transcription = processor.batch_decode(predicted_ids, skip_special_tokens=True)
print(transcription)
```

```
['Please cool Stella.']
```

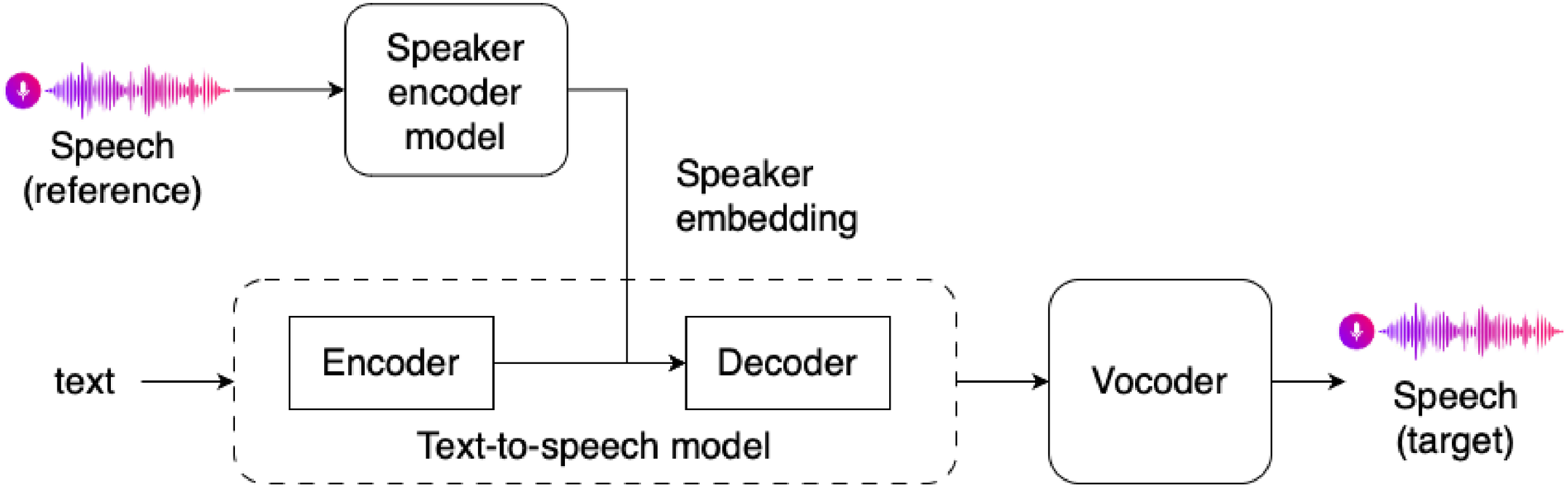
Audio generation

Three components to generate audio:

- **Preprocessor**: resampling and feature extraction
- **Model**: feature transformation
- **Vocoder**: a separate generative model for audio waveforms

```
from transformers import SpeechT5Processor, SpeechT5ForSpeechToSpeech, SpeechT5HifiGan
processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_vc")
model = SpeechT5ForSpeechToSpeech.from_pretrained("microsoft/speecht5_vc")
vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")
```

Speech embeddings



Generating speaker embeddings

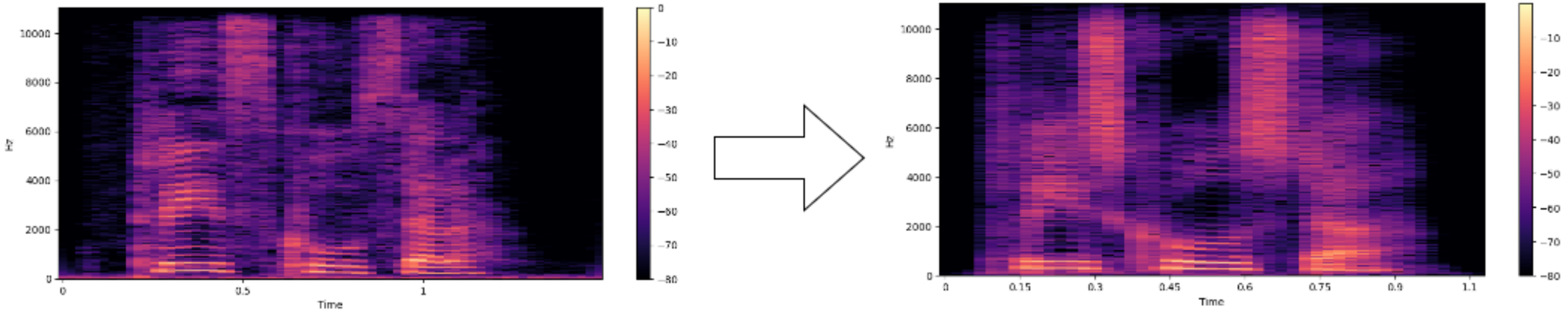
- **Pre-trained encoder**: audio waveform array → encoded array (typically 512 dimensions)

```
from speechbrain.inference.speaker import EncoderClassifier
speaker_model = EncoderClassifier.from_hparams(source="speechbrain/spkrec-xvect-voxceleb")
```

```
speaker_embeddings = speaker_model.encode_batch(torch.tensor(dataset[0]["audio"]["array"]))
speaker_embeddings = torch.nn.functional.normalize(speaker_embeddings, dim=2).unsqueeze(0)
```

Audio generation

```
inputs = processor(audio=dataset[0]["audio"],
                   sampling_rate=dataset[0]["audio"]["sampling_rate"],
                   return_tensors="pt")

speech = model.generate_speech(inputs["input_values"], speaker_embedding, vocoder=vocoder)
```



Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

Fine-tuning text-to-speech models

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

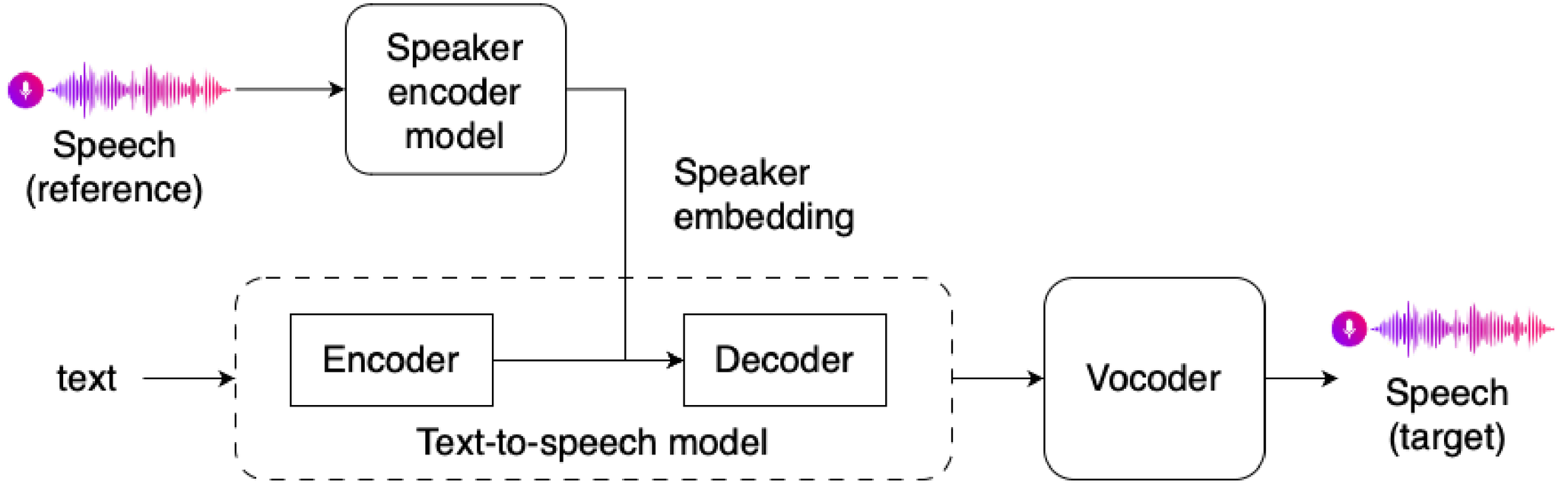
datacamp

Purpose of fine-tuning text-to-speech

- Learn sounds in new languages and dialects
- Apply to new contexts

E.g., large English language for general pretraining model ⇏ realistic Italian speech

Purpose of fine-tuning text-to-speech



- Speaker embedding + text-to-speech model features → generative model
- New speaker embedding are insufficient on their own without fine-tuning

Preparing an audio dataset

VoxPopuli dataset: transcribed speech data for 18 languages from EU parliament

```
from datasets import load_dataset
dataset = load_dataset("facebook/voxpopuli", "it", split="train",
                       trust_remote_code=True)
print(dataset.features)
```

```
['audio', 'raw_text', 'normalized_text', 'gender', 'speaker_id', ... ]
```

- Need to preprocess the audio + add speech embeddings:

```
speaker_model = EncoderClassifier.from_hparams(source="speechbrain/spkrec-xvect-voxceleb",
                                               savedir="pretrained_models/spkrec-xvect-voxceleb")
```

Audio preprocessing

```
from transformers import SpeechT5Processor
processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")

def prepare_dataset(example):
    audio = example["audio"]
    example = processor(text=example["normalized_text"], audio_target=audio["array"],
        sampling_rate=audio["sampling_rate"], return_attention_mask=False)
    example["labels"] = example["labels"][0]
    with torch.no_grad():
        speaker_embeddings = speaker_model.encode_batch(torch.tensor(audio["array"]))
        speaker_embeddings = torch.nn.functional.normalize(speaker_embeddings, dim=2)
        example["speaker_embeddings"] = speaker_embeddings.squeeze().cpu().numpy()
    return example

dataset = dataset.map(prepare_dataset)
```

Training arguments

```
from transformers import Seq2SeqTrainingArguments


training_args = Seq2SeqTrainingArguments(
        per_device_train_batch_size=4,
        gradient_accumulation_steps=8,
        learning_rate=1e-5,
        warmup_steps=500,
        label_names=["labels"],
        data_collator=data_collator
)
```

Putting it all together

```
model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")
```

Trainer:

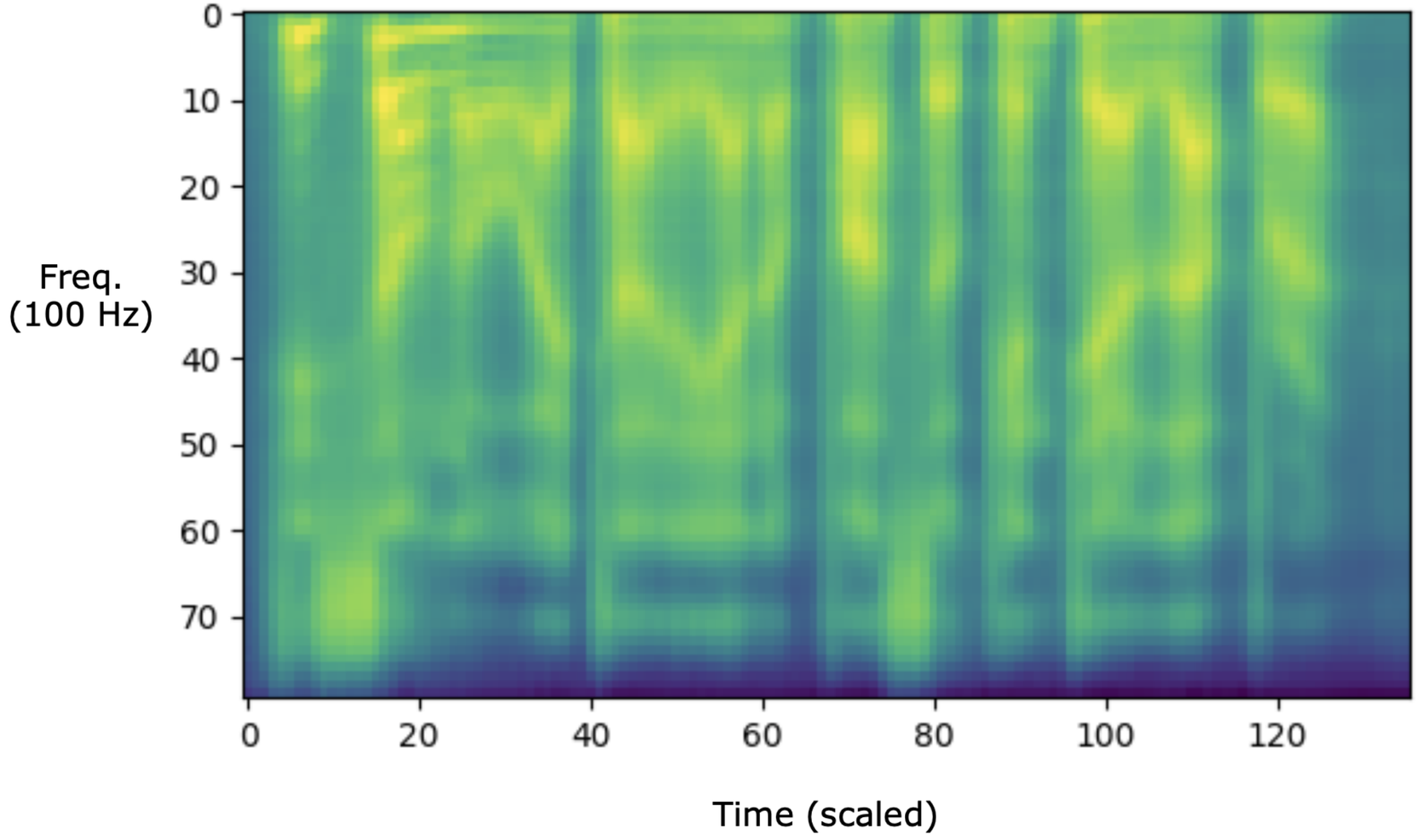
```
trainer = Seq2SeqTrainer(args=training_args, model=model,
    train_dataset=dataset["train"], eval_dataset=dataset["test"],
                          tokenizer=processor)
```

Run the training: `trainer.train()`

Using the new model

```
text = "se sono italiano posso cantare l'opera lirica"


speaker_embedding = torch.tensor(dataset[5]["speaker_embeddings"]).unsqueeze(0)
inputs = processor(text=text, return_tensors="pt")
speech = model.generate_speech(inputs["input_ids"],
                               speaker_embedding,
                               vocoder=vocoder)


make_spectrogram(speech)
```

Using the new model



Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE