Mempersiapkan

```
# For this notebook to run with updated APIs, we need torch 1.12+ and
torchvision 0.13+
try:
    import torch
    import torchvision
    assert int(torch.__version__.split(".")[1]) >= 12, "torch version
should be 1.12+"
    assert int(torchvision.__version__.split(".")[1]) >= 13,
"torchvision version should be 0.13+"
    print(f"torch version: {torch.__version__}")
    print(f"torchvision version: {torchvision.__version__}")
except:
    print(f"[INFO] torch/torchvision versions not as required,
installing nightly versions.")
    !pip3 install -U torch torchvision torchaudio --extra-index-url
https://download.pytorch.org/whl/cu113
    import torch
    import torchvision
    print(f"torch version: {torch.__version__}")
    print(f"torchvision version: {torchvision.__version__}")
```

```
torch version: 1.13.0.dev20220824+cu113
torchvision version: 0.14.0.dev20220824+cu113
```

```
# Continue with regular imports
import matplotlib.pyplot as plt
import torch
import torchvision

from torch import nn
from torchvision import transforms

# Try to get torchinfo, install it if it doesn't work
try:
    from torchinfo import summary
except:
    print("[INFO] Couldn't find torchinfo... installing it.")
    !pip install -q torchinfo
    from torchinfo import summary

# Try to import the going_modular directory, download it from GitHub
if it doesn't work
try:
    from going_modular.going_modular import data_setup, engine
    from helper_functions import download_data, set_seeds,
plot_loss_curves
except:
    # Get the going_modular scripts
```

```
    print("[INFO] Couldn't find going_modular or helper_functions
scripts... downloading them from GitHub.")
    !git clone https://github.com/mrdbourke/pytorch-deep-learning
    !mv pytorch-deep-learning/going_modular .
    !mv pytorch-deep-learning/helper_functions.py . # get the
helper_functions.py script
    !rm -rf pytorch-deep-learning
    from going_modular.going_modular import data_setup, engine
    from helper_functions import download_data, set_seeds,
plot_loss_curves

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

```
'cuda'
```

##Mendapatkan data

Kita dapat mendownload data menggunakan fungsi download_data() yang kita buat di 07. Pelacakan Eksperimen PyTorch bagian 1 dari helper_functions.py.

```
# Download pizza, steak, sushi images from GitHub
data_20_percent_path =
download_data(source="https://github.com/mrdbourke/pytorch-deep-
learning/raw/main/data/pizza_steak_sushi_20_percent.zip",

destination="pizza_steak_sushi_20_percent")

data_20_percent_path
```

```
[INFO] data/pizza_steak_sushi_20_percent directory exists, skipping
download.
```

```
PosixPath('data/pizza_steak_sushi_20_percent')
```

```
# Setup directory paths to train and test images
train_dir = data_20_percent_path / "train"
test_dir = data_20_percent_path / "test"
```

Membuat ekstraktor fitur EffNetB2

```
# 1. Setup pretrained EffNetB2 weights
effnetb2_weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT

# 2. Get EffNetB2 transforms
effnetb2_transforms = effnetb2_weights.transforms()

# 3. Setup pretrained model
effnetb2 =
torchvision.models.efficientnet_b2(weights=effnetb2_weights) # could
```

```
also use weights="DEFAULT"

# 4. Freeze the base layers in the model (this will freeze all layers
to begin with)
for param in effnetb2.parameters():
    param.requires_grad = False

# Check out EffNetB2 classifier head
effnetb2.classifier
```

```
Sequential(
  (0): Dropout(p=0.3, inplace=True)
  (1): Linear(in_features=1408, out_features=1000, bias=True)
)
```

```
# 5. Update the classifier head
effnetb2.classifier = nn.Sequential(
    nn.Dropout(p=0.3, inplace=True), # keep dropout layer same
    nn.Linear(in_features=1408, # keep in_features same
              out_features=3)) # change out_features to suit our
number of classes
```

Membuat fungsi untuk membuat ekstraktor fitur EffNetB2

Kami akan menyebutnya create_effnetb2_model() dan itu akan memerlukan sejumlah kelas yang dapat disesuaikan dan parameter benih acak untuk reproduktifitas.

```
def create_effnetb2_model(num_classes:int=3,
                          seed:int=42):
    """Creates an EfficientNetB2 feature extractor model and
transforms.

    Args:
        num_classes (int, optional): number of classes in the
classifier head.
            Defaults to 3.
        seed (int, optional): random seed value. Defaults to 42.

    Returns:
        model (torch.nn.Module): EffNetB2 feature extractor model.
        transforms (torchvision.transforms): EffNetB2 image
transforms.
    """
    # 1, 2, 3. Create EffNetB2 pretrained weights, transforms and
model
    weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT
    transforms = weights.transforms()
    model = torchvision.models.efficientnet_b2(weights=weights)

    # 4. Freeze all layers in base model
```

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

    # 5. Change classifier head with random seed for reproducibility
    torch.manual_seed(seed)
    model.classifier = nn.Sequential(
        nn.Dropout(p=0.3, inplace=True),
        nn.Linear(in_features=1408, out_features=num_classes),
    )

    return model, transforms

effnetb2, effnetb2_transforms = create_effnetb2_model(num_classes=3,
                                                      seed=42)

from torchinfo import summary

# # Print EffNetB2 model summary (uncomment for full output)
# summary(effnetb2,
#         input_size=(1, 3, 224, 224),
#         col_names=["input_size", "output_size", "num_params",
"trainable"],
#         col_width=20,
#         row_settings=["var_names"])
```

Membuat DataLoader untuk EffNetB2

Kami akan menggunakan batch_size 32 dan mengubah gambar kami menggunakan effnetb2_transforms jadi formatnya sama dengan model effnetb2 yang dilatih.

```
# Setup DataLoaders
from going_modular.going_modular import data_setup
train_dataloader_effnetb2, test_dataloader_effnetb2, class_names =
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=effnetb2_transforms,

batch_size=32)
```

Pelatihan ekstraktor fitur EffNetB2

Kita dapat melakukannya dengan membuat pengoptimal (kita akan menggunakan torch.optim.Adam() dengan kecepatan pembelajaran 1e-3), fungsi kerugian (kita akan menggunakan torch.nn.CrossEntropyLoss() untuk klasifikasi kelas jamak) dan kemudian meneruskannya serta DataLoaders kami ke engine.train() fungsi yang kami buat di 05. Bagian PyTorch Going Modular 4.

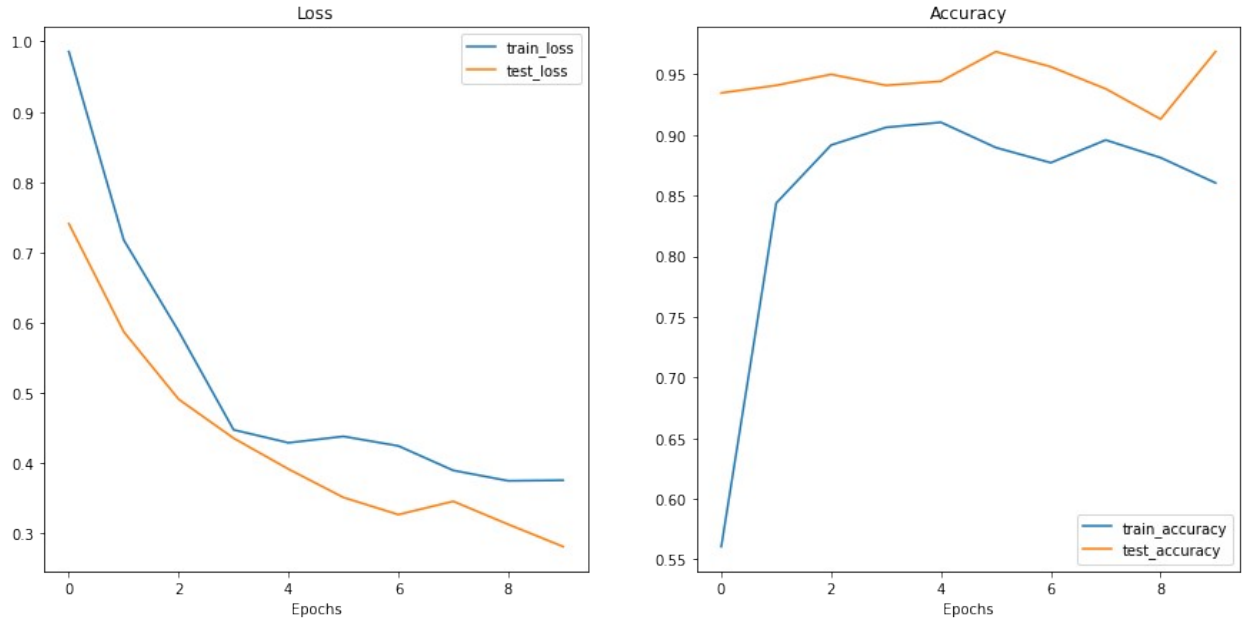
```
from going_modular.going_modular import engine

# Setup optimizer
optimizer = torch.optim.Adam(params=effnetb2.parameters(),
                             lr=1e-3)
# Setup loss function
loss_fn = torch.nn.CrossEntropyLoss()

# Set seeds for reproducibility and train the model
set_seeds()
effnetb2_results = engine.train(model=effnetb2,

train_dataloader=train_dataloader_effnetb2,

test_dataloader=test_dataloader_effnetb2,
                                epochs=10,
                                optimizer=optimizer,
                                loss_fn=loss_fn,
                                device=device)
```

{"model_id":"128923b0faf94e0591a606c01403fbb5","version_major":2,"version_minor":0}

```
Epoch: 1 | train_loss: 0.9856 | train_acc: 0.5604 | test_loss: 0.7408 | test_acc: 0.9347
Epoch: 2 | train_loss: 0.7175 | train_acc: 0.8438 | test_loss: 0.5869 | test_acc: 0.9409
Epoch: 3 | train_loss: 0.5876 | train_acc: 0.8917 | test_loss: 0.4909 | test_acc: 0.9500
Epoch: 4 | train_loss: 0.4474 | train_acc: 0.9062 | test_loss: 0.4355 | test_acc: 0.9409
Epoch: 5 | train_loss: 0.4290 | train_acc: 0.9104 | test_loss: 0.3915 | test_acc: 0.9443
Epoch: 6 | train_loss: 0.4381 | train_acc: 0.8896 | test_loss: 0.3512 | test_acc: 0.9688
Epoch: 7 | train_loss: 0.4245 | train_acc: 0.8771 | test_loss: 0.3268 | test_acc: 0.9563
Epoch: 8 | train_loss: 0.3897 | train_acc: 0.8958 | test_loss: 0.3457 | test_acc: 0.9381
Epoch: 9 | train_loss: 0.3749 | train_acc: 0.8812 | test_loss: 0.3129 | test_acc: 0.9131
Epoch: 10 | train_loss: 0.3757 | train_acc: 0.8604 | test_loss: 0.2813 | test_acc: 0.9688
```

Memeriksa kurva kerugian EffNetB2

```
from helper_functions import plot_loss_curves

plot_loss_curves(effnetb2_results)
```



###Menyimpan ekstraktor fitur EffNetB2

Kami akan menetapkan target_dir ke "models" dan model_name ke "09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_percent.pth" (sedikit komprehensif tapi setidaknya kita tahu apa yang terjadi).

```
from going_modular.going_modular import utils

# Save the model
utils.save_model(model=effnetb2,
                 target_dir="models",

model_name="09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi
_20_percent.pth")
```

```
[INFO] Saving model to:
models/09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_p
ercent.pth
```

Memeriksa ukuran ekstraktor fitur EffNetB2

Untuk memeriksa ukuran model kita dalam byte, kita dapat menggunakan pathlib.Path.stat("path_to_model").st_size Python dan kemudian kita dapat mengonversinya (kira-kira) menjadi megabyte dengan membaginya dengan (1024*1024).

```
from pathlib import Path

# Get the model size in bytes then convert to megabytes
pretrained_effnetb2_model_size =
Path("models/09_pretrained_effnetb2_feature_extractor_pizza_steak_sush
i_20_percent.pth").stat().st_size // (1024*1024) # division converts
```

```
bytes to megabytes (roughly)
print(f"Pretrained EffNetB2 feature extractor model size:
{pretrained_effnetb2_model_size} MB")
```

```
Pretrained EffNetB2 feature extractor model size: 29 MB
```

Mengumpulkan statistik ekstraktor fitur EffNetB2

Kita dapat melakukannya dengan menghitung jumlah elemen (atau pola/bobot) di effnetb2.parameters(). Kita akan mengakses jumlah elemen di setiap parameter menggunakan metode torch.numel() (kependekan dari "number of elements").

```
# Count number of parameters in EffNetB2
effnetb2_total_params = sum(torch.numel(param) for param in
effnetb2.parameters())
effnetb2_total_params
```

```
7705221
```

```
# Create a dictionary with EffNetB2 statistics
effnetb2_stats = {"test_loss": effnetb2_results["test_loss"][-1],
                  "test_acc": effnetb2_results["test_acc"][-1],
                  "number_of_parameters": effnetb2_total_params,
                  "model_size (MB)": pretrained_effnetb2_model_size}
effnetb2_stats
```

```
{'test_loss': 0.28128674924373626,
 'test_acc': 0.96875,
 'number_of_parameters': 7705221,
 'model_size (MB)': 29}
```

Membuat ekstraktor fitur ViT

Kita akan mulai dengan membuat fungsi bernama create_vit_model() yang akan sangat mirip dengan create_effnetb2_model() kecuali tentu saja mengembalikan model ekstraktor fitur ViT dan mentransformasikannya, bukan EffNetB2.

```
# Check out ViT heads layer
vit = torchvision.models.vit_b_16()
vit.heads
```

```
Sequential(
  (head): Linear(in_features=768, out_features=1000, bias=True)
)
```

```
def create_vit_model(num_classes:int=3,
                     seed:int=42):
    """Creates a ViT-B/16 feature extractor model and transforms.
```

```
    Args:
        num_classes (int, optional): number of target classes.
Defaults to 3.
        seed (int, optional): random seed value for output layer.
Defaults to 42.

    Returns:
        model (torch.nn.Module): ViT-B/16 feature extractor model.
        transforms (torchvision.transforms): ViT-B/16 image
transforms.
    """
    # Create ViT_B_16 pretrained weights, transforms and model
    weights = torchvision.models.ViT_B_16_Weights.DEFAULT
    transforms = weights.transforms()
    model = torchvision.models.vit_b_16(weights=weights)

    # Freeze all layers in model
    for param in model.parameters():
        param.requires_grad = False

    # Change classifier head to suit our needs (this will be
trainable)
    torch.manual_seed(seed)
    model.heads = nn.Sequential(nn.Linear(in_features=768, # keep this
the same as original model
                                          out_features=num_classes)) #
update to reflect target number of classes

    return model, transforms

# Create ViT model and transforms
vit, vit_transforms = create_vit_model(num_classes=3,
                                       seed=42)

from torchinfo import summary

# # Print ViT feature extractor model summary (uncomment for full
output)
# summary(vit,
#         input_size=(1, 3, 224, 224),
#         col_names=["input_size", "output_size", "num_params",
"trainable"],
#         col_width=20,
#         row_settings=["var_names"])
```

Membuat DataLoader untuk ViT

```
# Setup ViT DataLoaders
from going_modular.going_modular import data_setup
train_dataloader_vit, test_dataloader_vit, class_names =
```

```
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=vit_transforms,

batch_size=32)
```

Pelatihan ekstraktor fitur ViT

Mari kita latih model ekstraktor fitur ViT selama 10 periode menggunakan fungsi engine.train() dengan torch.optim.Adam() dan kecepatan pembelajaran 1e-3 sebagai pengoptimal dan torch.nn.CrossEntropyLoss() sebagai fungsi kerugian.

```
from going_modular.going_modular import engine

# Setup optimizer
optimizer = torch.optim.Adam(params=vit.parameters(),
                             lr=1e-3)
# Setup loss function
loss_fn = torch.nn.CrossEntropyLoss()

# Train ViT model with seeds set for reproducibility
set_seeds()
vit_results = engine.train(model=vit,
                           train_dataloader=train_dataloader_vit,
                           test_dataloader=test_dataloader_vit,
                           epochs=10,
                           optimizer=optimizer,
                           loss_fn=loss_fn,
                           device=device)
```

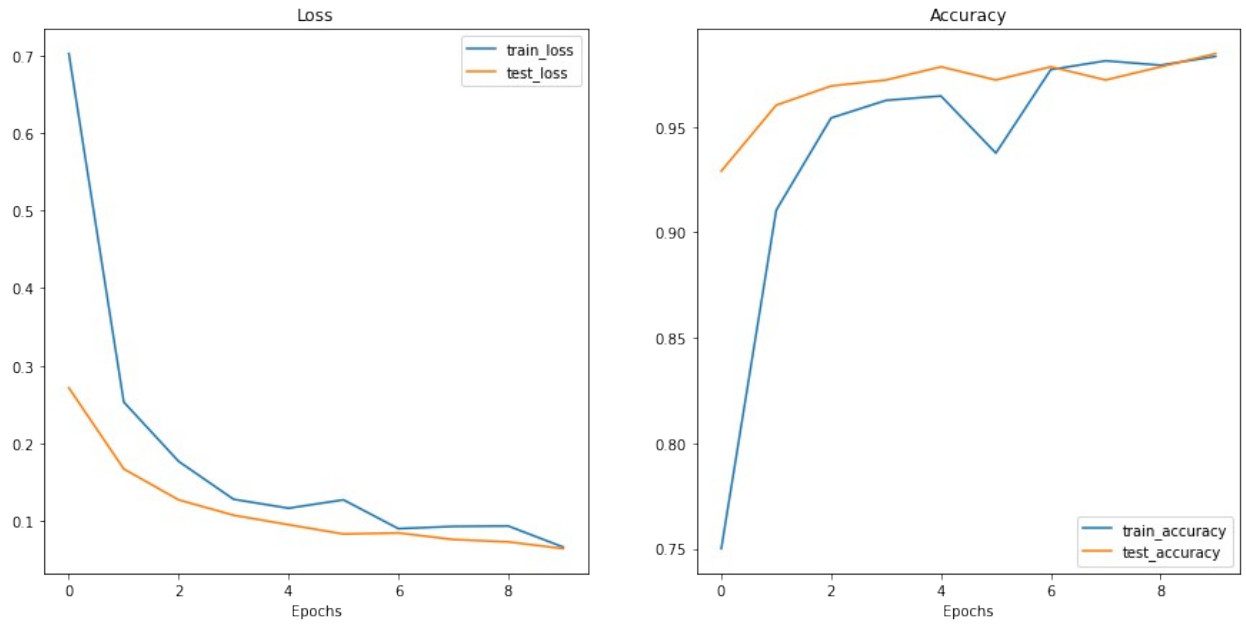
{"model_id":"ded441208c9540c28035eb5ea07b4b39","version_major":2,"version_minor":0}

```
Epoch: 1 | train_loss: 0.7023 | train_acc: 0.7500 | test_loss: 0.2714 | test_acc: 0.9290
Epoch: 2 | train_loss: 0.2531 | train_acc: 0.9104 | test_loss: 0.1669 | test_acc: 0.9602
Epoch: 3 | train_loss: 0.1766 | train_acc: 0.9542 | test_loss: 0.1270 | test_acc: 0.9693
Epoch: 4 | train_loss: 0.1277 | train_acc: 0.9625 | test_loss: 0.1072 | test_acc: 0.9722
Epoch: 5 | train_loss: 0.1163 | train_acc: 0.9646 | test_loss: 0.0950 | test_acc: 0.9784
Epoch: 6 | train_loss: 0.1270 | train_acc: 0.9375 | test_loss: 0.0830 | test_acc: 0.9722
Epoch: 7 | train_loss: 0.0899 | train_acc: 0.9771 | test_loss: 0.0844 | test_acc: 0.9784
```

```
Epoch: 8 | train_loss: 0.0928 | train_acc: 0.9812 | test_loss: 0.0759
| test_acc: 0.9722
Epoch: 9 | train_loss: 0.0933 | train_acc: 0.9792 | test_loss: 0.0729
| test_acc: 0.9784
Epoch: 10 | train_loss: 0.0662 | train_acc: 0.9833 | test_loss: 0.0642
| test_acc: 0.9847
```

Memeriksa kurva kehilangan ViT

```
from helper_functions import plot_loss_curves

plot_loss_curves(vit_results)
```



Menyimpan ekstraktor fitur ViT

Kita dapat melakukannya menggunakan fungsi utils.save_model() yang kita buat di 05. Bagian PyTorch Menjadi Modular 5.

```
# Save the model
from going_modular.going_modular import utils

utils.save_model(model=vit,
                 target_dir="models",

model_name="09_pretrained_vit_feature_extractor_pizza_steak_sushi_20_p
ercent.pth")
```

```
[INFO] Saving model to:
models/09_pretrained_vit_feature_extractor_pizza_steak_sushi_20_percen
t.pth
```

Memeriksa ukuran ekstraktor fitur ViT

Untuk memeriksa ukuran model kita dalam byte, kita dapat menggunakan pathlib.Path.stat("path_to_model").st_size Python dan kemudian kita dapat mengonversinya (kira-kira) menjadi megabyte dengan membaginya dengan (1024*1024).

```
from pathlib import Path

# Get the model size in bytes then convert to megabytes
pretrained_vit_model_size =
Path("models/09_pretrained_vit_feature_extractor_pizza_steak_sushi_20_
percent.pth").stat().st_size // (1024*1024) # division converts bytes
to megabytes (roughly)
print(f"Pretrained ViT feature extractor model size:
{pretrained_vit_model_size} MB")
```

```
Pretrained ViT feature extractor model size: 327 MB
```

Mengumpulkan statistik ekstraktor fitur ViT

```
# Count number of parameters in ViT
vit_total_params = sum(torch.numel(param) for param in
vit.parameters())
vit_total_params
```

```
85800963
```

```
# Create ViT statistics dictionary
vit_stats = {"test_loss": vit_results["test_loss"][-1],
             "test_acc": vit_results["test_acc"][-1],
             "number_of_parameters": vit_total_params,
             "model_size (MB)": pretrained_vit_model_size}

vit_stats
```

```
{'test_loss': 0.06418210905976593,
 'test_acc': 0.984659090909091,
 'number_of_parameters': 85800963,
 'model_size (MB)': 327}
```

Membuat prediksi dengan model terlatih kami dan mengatur

waktunya

Untuk melakukannya, kita akan menggunakan pathlib.Path("target_dir").glob("/.jpg")) Python untuk menemukan semua jalur file di direktori target dengan ekstensi .jpg (semua gambar pengujian kami).

```
from pathlib import Path

# Get all test data paths
print(f"[INFO] Finding all filepaths ending with '.jpg' in directory:
{test_dir}")
test_data_paths = list(Path(test_dir).glob("*/*.jpg"))
test_data_paths[:5]
```

```
[INFO] Finding all filepaths ending with '.jpg' in directory:
data/pizza_steak_sushi_20_percent/test

[PosixPath('data/pizza_steak_sushi_20_percent/test/steak/831681.jpg'),

PosixPath('data/pizza_steak_sushi_20_percent/test/steak/3100563.jpg'),

PosixPath('data/pizza_steak_sushi_20_percent/test/steak/2752603.jpg'),
 PosixPath('data/pizza_steak_sushi_20_percent/test/steak/39461.jpg'),
 PosixPath('data/pizza_steak_sushi_20_percent/test/steak/730464.jpg')]
```

Membuat fungsi untuk membuat prediksi di seluruh kumpulan data pengujian

```
import pathlib
import torch

from PIL import Image
from timeit import default_timer as timer
from tqdm.auto import tqdm
from typing import List, Dict

# 1. Create a function to return a list of dictionaries with sample,
truth label, prediction, prediction probability and prediction time
def pred_and_store(paths: List[pathlib.Path],
                   model: torch.nn.Module,
                   transform: torchvision.transforms,
                   class_names: List[str],
                   device: str = "cuda" if torch.cuda.is_available()
else "cpu") -> List[Dict]:

    # 2. Create an empty list to store prediction dictionaires
    pred_list = []

    # 3. Loop through target paths
    for path in tqdm(paths):

        # 4. Create empty dictionary to store prediction information
for each sample
        pred_dict = {}

        # 5. Get the sample path and ground truth class name
```

```
        pred_dict["image_path"] = path
        class_name = path.parent.stem
        pred_dict["class_name"] = class_name

        # 6. Start the prediction timer
        start_time = timer()

        # 7. Open image path
        img = Image.open(path)

        # 8. Transform the image, add batch dimension and put image on
target device
        transformed_image = transform(img).unsqueeze(0).to(device)

        # 9. Prepare model for inference by sending it to target
device and turning on eval() mode
        model.to(device)
        model.eval()

        # 10. Get prediction probability, predicition label and
prediction class
        with torch.inference_mode():
            pred_logit = model(transformed_image) # perform inference
on target sample
            pred_prob = torch.softmax(pred_logit, dim=1) # turn logits
into prediction probabilities
            pred_label = torch.argmax(pred_prob, dim=1) # turn
prediction probabilities into prediction label
            pred_class = class_names[pred_label.cpu()] # hardcode
prediction class to be on CPU

            # 11. Make sure things in the dictionary are on CPU
(required for inspecting predictions later on)
            pred_dict["pred_prob"] =
round(pred_prob.unsqueeze(0).max().cpu().item(), 4)
            pred_dict["pred_class"] = pred_class

            # 12. End the timer and calculate time per pred
            end_time = timer()
            pred_dict["time_for_pred"] = round(end_time-start_time, 4)

        # 13. Does the pred match the true label?
        pred_dict["correct"] = class_name == pred_class

        # 14. Add the dictionary to the list of preds
        pred_list.append(pred_dict)

    # 15. Return list of prediction dictionaries
    return pred_list
```

Membuat dan menentukan waktu prediksi dengan EffNetB2

```
# Make predictions across test dataset with EffNetB2
effnetb2_test_pred_dicts = pred_and_store(paths=test_data_paths,
                                          model=effnetb2,

transform=effnetb2_transforms,
                                          class_names=class_names,
                                          device="cpu") # make
predictions on CPU

{"model_id":"9b516a8ba5ce4603a25ae0b6d5f8573a","version_major":2,"vers
ion_minor":0}

# Inspect the first 2 prediction dictionaries
effnetb2_test_pred_dicts[:2]

[{'image_path':
PosixPath('data/pizza_steak_sushi_20_percent/test/steak/831681.jpg'),
  'class_name': 'steak',
  'pred_prob': 0.9293,
  'pred_class': 'steak',
  'time_for_pred': 0.0494,
  'correct': True},
 {'image_path':
PosixPath('data/pizza_steak_sushi_20_percent/test/steak/3100563.jpg'),
  'class_name': 'steak',
  'pred_prob': 0.9534,
  'pred_class': 'steak',
  'time_for_pred': 0.0264,
  'correct': True}]

# Turn the test_pred_dicts into a DataFrame
import pandas as pd
effnetb2_test_pred_df = pd.DataFrame(effnetb2_test_pred_dicts)
effnetb2_test_pred_df.head()

                                          image_path class_name
pred_prob  \
0  data/pizza_steak_sushi_20_percent/test/steak/8...      steak
0.9293
1  data/pizza_steak_sushi_20_percent/test/steak/3...      steak
0.9534
2  data/pizza_steak_sushi_20_percent/test/steak/2...      steak
0.7532
3  data/pizza_steak_sushi_20_percent/test/steak/3...      steak
0.5935
4  data/pizza_steak_sushi_20_percent/test/steak/7...      steak
0.8959

  pred_class  time_for_pred  correct
```

```
0      steak          0.0494     True
1      steak          0.0264     True
2      steak          0.0256     True
3      steak          0.0263     True
4      steak          0.0269     True
```

```
# Check number of correct predictions
effnetb2_test_pred_df.correct.value_counts()
```

```
True     145
False      5
Name: correct, dtype: int64
```

```
# Find the average time per prediction
effnetb2_average_time_per_pred =
round(effnetb2_test_pred_df.time_for_pred.mean(), 4)
print(f"EffNetB2 average time per prediction:
{effnetb2_average_time_per_pred} seconds")
```

```
EffNetB2 average time per prediction: 0.0269 seconds
```

```
# Add EffNetB2 average prediction time to stats dictionary
effnetb2_stats["time_per_pred_cpu"] = effnetb2_average_time_per_pred
effnetb2_stats
```

```
{'test_loss': 0.28128674924373626,
 'test_acc': 0.96875,
 'number_of_parameters': 7705221,
 'model_size (MB)': 29,
 'time_per_pred_cpu': 0.0269}
```

Membuat dan menentukan waktu prediksi dengan ViT

Dan kami akan menyimpan prediksi pada CPU melalui device="cpu" (perpanjangan alami di sini adalah menguji waktu prediksi pada CPU dan GPU).

```
# Make list of prediction dictionaries with ViT feature extractor
model on test images
vit_test_pred_dicts = pred_and_store(paths=test_data_paths,
                                     model=vit,
                                     transform=vit_transforms,
                                     class_names=class_names,
                                     device="cpu")
```

```
{"model_id":"2c6e2d8224d84f1c9158c60fe3f7fd9f","version_major":2,"vers
ion_minor":0}
```

```
# Check the first couple of ViT predictions on the test dataset
vit_test_pred_dicts[:2]
```

```
[{'image_path':
PosixPath('data/pizza_steak_sushi_20_percent/test/steak/831681.jpg'),
  'class_name': 'steak',
  'pred_prob': 0.9933,
  'pred_class': 'steak',
  'time_for_pred': 0.1313,
  'correct': True},
 {'image_path':
PosixPath('data/pizza_steak_sushi_20_percent/test/steak/3100563.jpg'),
  'class_name': 'steak',
  'pred_prob': 0.9893,
  'pred_class': 'steak',
  'time_for_pred': 0.0638,
  'correct': True}]

# Turn vit_test_pred_dicts into a DataFrame
import pandas as pd
vit_test_pred_df = pd.DataFrame(vit_test_pred_dicts)
vit_test_pred_df.head()
```

	image_path	class_name	pred_prob
0	data/pizza_steak_sushi_20_percent/test/steak/8...	steak	0.9933
1	data/pizza_steak_sushi_20_percent/test/steak/3...	steak	0.9893
2	data/pizza_steak_sushi_20_percent/test/steak/2...	steak	0.9971
3	data/pizza_steak_sushi_20_percent/test/steak/3...	steak	0.7685
4	data/pizza_steak_sushi_20_percent/test/steak/7...	steak	0.9499

	pred_class	time_for_pred	correct
0	steak	0.1313	True
1	steak	0.0638	True
2	steak	0.0627	True
3	steak	0.0632	True
4	steak	0.0641	True

```
# Count the number of correct predictions
vit_test_pred_df.correct.value_counts()
```

```
True     148
False      2
Name: correct, dtype: int64

# Calculate average time per prediction for ViT model
vit_average_time_per_pred =
round(vit_test_pred_df.time_for_pred.mean(), 4)
```

```
print(f"ViT average time per prediction: {vit_average_time_per_pred}
seconds")
```

```
ViT average time per prediction: 0.0641 seconds
```

```
# Add average prediction time for ViT model on CPU
vit_stats["time_per_pred_cpu"] = vit_average_time_per_pred
vit_stats
```

```
{'test_loss': 0.06418210905976593,
 'test_acc': 0.984659090909091,
 'number_of_parameters': 85800963,
 'model_size (MB)': 327,
 'time_per_pred_cpu': 0.0641}
```

Membandingkan hasil model, waktu dan ukuran prediksi

Untuk melakukannya, mari ubah kamus effnetb2_stats dan vit_stats menjadi pandas DataFrame.

```
# Turn stat dictionaries into DataFrame
df = pd.DataFrame([effnetb2_stats, vit_stats])

# Add column for model names
df["model"] = ["EffNetB2", "ViT"]

# Convert accuracy to percentages
df["test_acc"] = round(df["test_acc"] * 100, 2)

df
```

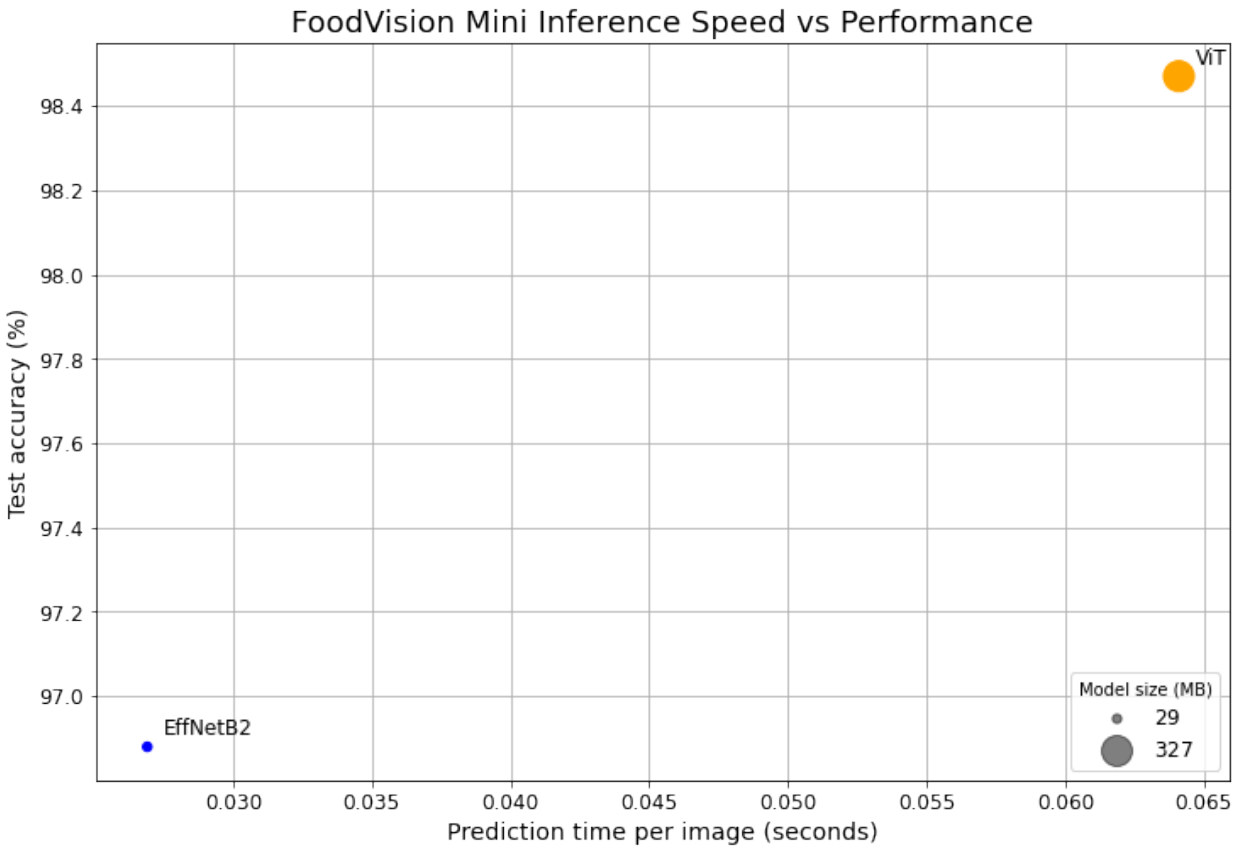
```
   test_loss  test_acc  number_of_parameters  model_size (MB)  \
0   0.281287     96.88               7705221               29
1   0.064182     98.47              85800963              327

   time_per_pred_cpu     model
0             0.0269  EffNetB2
1             0.0641       ViT
```

```
# Compare ViT to EffNetB2 across different characteristics
pd.DataFrame(data=(df.set_index("model").loc["ViT"] /
df.set_index("model").loc["EffNetB2"]), # divide ViT statistics by
EffNetB2 statistics
             columns=["ViT to EffNetB2 ratios"]).T
```

```
                        test_loss  test_acc  number_of_parameters  \
ViT to EffNetB2 ratios   0.228173  1.016412             11.135432

                        model_size (MB)  time_per_pred_cpu
ViT to EffNetB2 ratios         11.275862             2.3829
```

Memvisualisasikan trade-off kecepatan vs. kinerja

```
# 1. Create a plot from model comparison DataFrame
fig, ax = plt.subplots(figsize=(12, 8))
scatter = ax.scatter(data=df,
                     x="time_per_pred_cpu",
                     y="test_acc",
                     c=["blue", "orange"], # what colours to use?
                     s="model_size (MB)") # size the dots by the model
sizes

# 2. Add titles, labels and customize fontsize for aesthetics
ax.set_title("FoodVision Mini Inference Speed vs Performance",
fontsize=18)
ax.set_xlabel("Prediction time per image (seconds)", fontsize=14)
ax.set_ylabel("Test accuracy (%)", fontsize=14)
ax.tick_params(axis='both', labelsize=12)
ax.grid(True)

# 3. Annotate with model names
for index, row in df.iterrows():
    ax.annotate(text=row["model"], # note: depending on your version
of Matplotlib, you may need to use "s=..." or "text=...", see:
https://github.com/faustomorales/keras-ocr/issues/183#issuecomment-
977733270
                xy=(row["time_per_pred_cpu"]+0.0006, row["test_acc"]
+0.03),
                size=12)

# 4. Create a legend based on model sizes
handles, labels = scatter.legend_elements(prop="sizes", alpha=0.5)
model_size_legend = ax.legend(handles,
                              labels,
                              loc="lower right",
                              title="Model size (MB)",
                              fontsize=12)

# Save the figure
plt.savefig("images/09-foodvision-mini-inference-speed-vs-
performance.jpg")

# Show the figure
plt.show()
```



Menghidupkan FoodVision Mini dengan membuat demo Gradio

```
# Import/install Gradio
try:
    import gradio as gr
except:
    !pip -q install gradio
    import gradio as gr

print(f"Gradio version: {gr.__version__}")
```

```
Gradio version: 3.1.4
```

Membuat fungsi untuk memetakan input dan output kita

```
# Put EffNetB2 on CPU
effnetb2.to("cpu")

# Check the device
next(iter(effnetb2.parameters())).device
```

```
device(type='cpu')
```

```python
from typing import Tuple, Dict

def predict(img) -> Tuple[Dict, float]:
    """Transforms and performs a prediction on img and returns
prediction and time taken.
    """
    # Start the timer
    start_time = timer()

    # Transform the target image and add a batch dimension
    img = effnetb2_transforms(img).unsqueeze(0)

    # Put model into evaluation mode and turn on inference mode
    effnetb2.eval()
    with torch.inference_mode():
        # Pass the transformed image through the model and turn the
prediction logits into prediction probabilities
        pred_probs = torch.softmax(effnetb2(img), dim=1)

    # Create a prediction label and prediction probability dictionary
for each prediction class (this is the required format for Gradio's
output parameter)
    pred_labels_and_probs = {class_names[i]: float(pred_probs[0][i])
for i in range(len(class_names))}

    # Calculate the prediction time
    pred_time = round(timer() - start_time, 5)

    # Return the prediction dictionary and prediction time
    return pred_labels_and_probs, pred_time

import random
from PIL import Image

# Get a list of all test image filepaths
test_data_paths = list(Path(test_dir).glob("*/*.jpg"))

# Randomly select a test image path
random_image_path = random.sample(test_data_paths, k=1)[0]

# Open the target image
image = Image.open(random_image_path)
print(f"[INFO] Predicting on image at path: {random_image_path}\n")

# Predict on the target image and print out the outputs
pred_dict, pred_time = predict(img=image)
print(f"Prediction label and probability dictionary: \n{pred_dict}")
print(f"Prediction time: {pred_time} seconds")
```

```
[INFO] Predicting on image at path:
data/pizza_steak_sushi_20_percent/test/pizza/3770514.jpg
```

```
Prediction label and probability dictionary:
{'pizza': 0.9785208702087402, 'steak': 0.01169557310640812, 'sushi':
0.009783552028238773}
Prediction time: 0.027 seconds
```

Membuat daftar contoh gambar

Kelas Antarmuka Gradio mengambil daftar contoh sebagai parameter opsional (gradio.Interface(examples=List[Any])).

```
# Create a list of example inputs to our Gradio demo
example_list = [[str(filepath)] for filepath in
random.sample(test_data_paths, k=3)]
example_list
```

```
[['data/pizza_steak_sushi_20_percent/test/sushi/804460.jpg'],
 ['data/pizza_steak_sushi_20_percent/test/steak/746921.jpg'],
 ['data/pizza_steak_sushi_20_percent/test/steak/2117351.jpg']]
```

Membangun antarmuka Gradio

```
import gradio as gr

# Create title, description and article strings
title = "FoodVision Mini 🍕🥩🍣 "
description = "An EfficientNetB2 feature extractor computer vision
model to classify images of food as pizza, steak or sushi."
article = "Created at [09. PyTorch Model
Deployment](https://www.learnpytorch.io/09_pytorch_model_deployment/).
"

# Create the Gradio demo
demo = gr.Interface(fn=predict, # mapping function from input to
output
                    inputs=gr.Image(type="pil"), # what are the
inputs?
                    outputs=[gr.Label(num_top_classes=3,
label="Predictions"), # what are the outputs?
                             gr.Number(label="Prediction time (s)")],
# our fn has two outputs, therefore we have two outputs
                    examples=example_list,
                    title=title,
                    description=description,
                    article=article)

# Launch the demo!
demo.launch(debug=False, # print errors locally?
            share=True) # generate a publically shareable URL?
```

```
Running on local URL:  http://127.0.0.1:7860/
Running on public URL: https://27541.gradio.app

This share link expires in 72 hours. For free permanent hosting, check
out Spaces: https://huggingface.co/spaces

<IPython.core.display.HTML object>

(<gradio.routes.App at 0x7f122dd0f0d0>,
 'http://127.0.0.1:7860/',
 'https://27541.gradio.app')
```

Membuat folder demo untuk menyimpan file aplikasi FoodVision Mini kami

Kita dapat menggunakan pathlib.Path("path_to_dir") Python untuk membuat jalur direktori dan pathlib.Path("path_to_dir").mkdir() untuk membuatnya.

```
import shutil
from pathlib import Path

# Create FoodVision mini demo path
foodvision_mini_demo_path = Path("demos/foodvision_mini/")

# Remove files that might already exist there and create new directory
if foodvision_mini_demo_path.exists():
    shutil.rmtree(foodvision_mini_demo_path)
    foodvision_mini_demo_path.mkdir(parents=True, # make the parent
folders?
                                    exist_ok=True) # create it even if
it already exists?
else:
    # If the file doesn't exist, create it anyway
    foodvision_mini_demo_path.mkdir(parents=True,
                                    exist_ok=True)

# Check what's in the folder
!ls demos/foodvision_mini/
```

Membuat folder berisi contoh gambar untuk digunakan dengan demo FoodVision Mini kami

```
import shutil
from pathlib import Path

# 1. Create an examples directory
foodvision_mini_examples_path = foodvision_mini_demo_path / "examples"
foodvision_mini_examples_path.mkdir(parents=True, exist_ok=True)
```

```
# 2. Collect three random test dataset image paths
foodvision_mini_examples =
[Path('data/pizza_steak_sushi_20_percent/test/sushi/592799.jpg'),

Path('data/pizza_steak_sushi_20_percent/test/steak/3622237.jpg'),

Path('data/pizza_steak_sushi_20_percent/test/pizza/2582289.jpg')]

# 3. Copy the three random images to the examples directory
for example in foodvision_mini_examples:
    destination = foodvision_mini_examples_path / example.name
    print(f"[INFO] Copying {example} to {destination}")
    shutil.copy2(src=example, dst=destination)
```

```
[INFO] Copying data/pizza_steak_sushi_20_percent/test/sushi/592799.jpg
to demos/foodvision_mini/examples/592799.jpg
[INFO] Copying
data/pizza_steak_sushi_20_percent/test/steak/3622237.jpg to
demos/foodvision_mini/examples/3622237.jpg
[INFO] Copying
data/pizza_steak_sushi_20_percent/test/pizza/2582289.jpg to
demos/foodvision_mini/examples/2582289.jpg
```

```
import os

# Get example filepaths in a list of lists
example_list = [["examples/" + example] for example in
os.listdir(foodvision_mini_examples_path)]
example_list
```

```
[['examples/3622237.jpg'], ['examples/592799.jpg'],
['examples/2582289.jpg']]
```

Memindahkan model EffNetB2 terlatih kami ke direktori demo FoodVision Mini

```
import shutil

# Create a source path for our target model
effnetb2_foodvision_mini_model_path =
"models/09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_
percent.pth"

# Create a destination path for our target model
effnetb2_foodvision_mini_model_destination = foodvision_mini_demo_path
/ effnetb2_foodvision_mini_model_path.split("/")[1]

# Try to move the file
try:
    print(f"[INFO] Attempting to move
```

```
{effnetb2_foodvision_mini_model_path} to
{effnetb2_foodvision_mini_model_destination}")

    # Move the model
    shutil.move(src=effnetb2_foodvision_mini_model_path,
                dst=effnetb2_foodvision_mini_model_destination)

    print(f"[INFO] Model move complete.")

# If the model has already been moved, check if it exists
except:
    print(f"[INFO] No model found at
{effnetb2_foodvision_mini_model_path}, perhaps its already been
moved?")
    print(f"[INFO] Model exists at
{effnetb2_foodvision_mini_model_destination}:
{effnetb2_foodvision_mini_model_destination.exists()}")
```

```
[INFO] Attempting to move
models/09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_p
ercent.pth to
demos/foodvision_mini/09_pretrained_effnetb2_feature_extractor_pizza_s
teak_sushi_20_percent.pth
[INFO] Model move complete.
```

Mengubah model EffNetB2 menjadi skrip Python (model.py)

```
%%writefile demos/foodvision_mini/model.py
import torch
import torchvision

from torch import nn


def create_effnetb2_model(num_classes:int=3,
                          seed:int=42):
    """Creates an EfficientNetB2 feature extractor model and
transforms.

    Args:
        num_classes (int, optional): number of classes in the
classifier head.
            Defaults to 3.
        seed (int, optional): random seed value. Defaults to 42.

    Returns:
        model (torch.nn.Module): EffNetB2 feature extractor model.
        transforms (torchvision.transforms): EffNetB2 image
transforms.
    """
```

```
    # Create EffNetB2 pretrained weights, transforms and model
    weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT
    transforms = weights.transforms()
    model = torchvision.models.efficientnet_b2(weights=weights)

    # Freeze all layers in base model
    for param in model.parameters():
        param.requires_grad = False

    # Change classifier head with random seed for reproducibility
    torch.manual_seed(seed)
    model.classifier = nn.Sequential(
        nn.Dropout(p=0.3, inplace=True),
        nn.Linear(in_features=1408, out_features=num_classes),
    )

    return model, transforms
```

```
Writing demos/foodvision_mini/model.py
```

Mengubah aplikasi FoodVision Mini Gradio menjadi skrip Python (app.py)

```
%%writefile demos/foodvision_mini/app.py
### 1. Imports and class names setup ###
import gradio as gr
import os
import torch

from model import create_effnetb2_model
from timeit import default_timer as timer
from typing import Tuple, Dict

# Setup class names
class_names = ["pizza", "steak", "sushi"]

### 2. Model and transforms preparation ###

# Create EffNetB2 model
effnetb2, effnetb2_transforms = create_effnetb2_model(
    num_classes=3, # len(class_names) would also work
)

# Load saved weights
effnetb2.load_state_dict(
    torch.load(

f="09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_percent.pth",
        map_location=torch.device("cpu"),  # load to CPU
```

```
    )
)

### 3. Predict function ###

# Create predict function
def predict(img) -> Tuple[Dict, float]:
    """Transforms and performs a prediction on img and returns
prediction and time taken.
    """
    # Start the timer
    start_time = timer()

    # Transform the target image and add a batch dimension
    img = effnetb2_transforms(img).unsqueeze(0)

    # Put model into evaluation mode and turn on inference mode
    effnetb2.eval()
    with torch.inference_mode():
        # Pass the transformed image through the model and turn the
prediction logits into prediction probabilities
        pred_probs = torch.softmax(effnetb2(img), dim=1)

    # Create a prediction label and prediction probability dictionary
for each prediction class (this is the required format for Gradio's
output parameter)
    pred_labels_and_probs = {class_names[i]: float(pred_probs[0][i])
for i in range(len(class_names))}

    # Calculate the prediction time
    pred_time = round(timer() - start_time, 5)

    # Return the prediction dictionary and prediction time
    return pred_labels_and_probs, pred_time

### 4. Gradio app ###

# Create title, description and article strings
title = "FoodVision Mini 🍕🥩🍣"
description = "An EfficientNetB2 feature extractor computer vision
model to classify images of food as pizza, steak or sushi."
article = "Created at [09. PyTorch Model
Deployment](https://www.learnpytorch.io/09_pytorch_model_deployment/).
"

# Create examples list from "examples/" directory
example_list = [["examples/" + example] for example in
os.listdir("examples")]

# Create the Gradio demo
```

```
demo = gr.Interface(fn=predict, # mapping function from input to
output
                    inputs=gr.Image(type="pil"), # what are the
inputs?
                    outputs=[gr.Label(num_top_classes=3,
label="Predictions"), # what are the outputs?
                             gr.Number(label="Prediction time (s)")],
# our fn has two outputs, therefore we have two outputs
                    # Create examples list from "examples/" directory
                    examples=example_list,
                    title=title,
                    description=description,
                    article=article)

# Launch the demo!
demo.launch()
```

```
Writing demos/foodvision_mini/app.py
```

Membuat file persyaratan untuk FoodVision Mini (requirements.txt)

```
%%writefile demos/foodvision_mini/requirements.txt
torch==1.12.0
torchvision==0.13.0
gradio==3.1.4
```

```
Writing demos/foodvision_mini/requirements.txt
```

Mengunduh file aplikasi FoodVision Mini kami

```
!ls demos/foodvision_mini
```

```
09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_percent.
pth
app.py
examples
model.py
requirements.txt
```

```
# Change into and then zip the foodvision_mini folder but exclude
certain files
!cd demos/foodvision_mini && zip -r ../foodvision_mini.zip * -x
"*.pyc" "*.ipynb" "*__pycache__*" "*ipynb_checkpoints*"

# Download the zipped FoodVision Mini app (if running in Google Colab)
try:
    from google.colab import files
    files.download("demos/foodvision_mini.zip")
except:
```

```
    print("Not running in Google Colab, can't use
google.colab.files.download(), please manually download.")
```

```
updating:
09_pretrained_effnetb2_feature_extractor_pizza_steak_sushi_20_percent.
pth (deflated 8%)
updating: app.py (deflated 57%)
updating: examples/ (stored 0%)
updating: examples/3622237.jpg (deflated 0%)
updating: examples/592799.jpg (deflated 1%)
updating: examples/2582289.jpg (deflated 17%)
updating: model.py (deflated 56%)
updating: requirements.txt (deflated 4%)
Not running in Google Colab, can't use google.colab.files.download(),
please manually download.
```

Mengunggah ke Memeluk Wajah

```
# IPython is a library to help make Python interactive
from IPython.display import IFrame

# Embed FoodVision Mini Gradio demo
IFrame(src="https://hf.space/embed/mrdbourke/foodvision_mini/+",
width=900, height=750)
```

```
<IPython.lib.display.IFrame at 0x7f122dd77700>
```

Membuat model dan transformasi untuk FoodVision Big

```
# Create EffNetB2 model capable of fitting to 101 classes for Food101
effnetb2_food101, effnetb2_transforms =
create_effnetb2_model(num_classes=101)

from torchinfo import summary

# # Get a summary of EffNetB2 feature extractor for Food101 with 101
output classes (uncomment for full output)
# summary(effnetb2_food101,
#         input_size=(1, 3, 224, 224),
#         col_names=["input_size", "output_size", "num_params",
"trainable"],
#         col_width=20,
#         row_settings=["var_names"])

# Create Food101 training data transforms (only perform data
augmentation on the training images)
food101_train_transforms = torchvision.transforms.Compose([
    torchvision.transforms.TrivialAugmentWide(),
    effnetb2_transforms,
])
```

```
print(f"Training transforms:\n{food101_train_transforms}\n")
print(f"Testing transforms:\n{effnetb2_transforms}")
```

```
Training transforms:
Compose(
    TrivialAugmentWide(num_magnitude_bins=31,
interpolation=InterpolationMode.NEAREST, fill=None)
    ImageClassification(
    crop_size=[288]
    resize_size=[288]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BICUBIC
)
)

Testing transforms:
ImageClassification(
    crop_size=[288]
    resize_size=[288]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BICUBIC
)
```

Mendapatkan data untuk FoodVision Big

```
from torchvision import datasets

# Setup data directory
from pathlib import Path
data_dir = Path("data")

# Get training data (~750 images x 101 food classes)
train_data = datasets.Food101(root=data_dir, # path to download data
to
                              split="train", # dataset split to get
                              transform=food101_train_transforms, #
perform data augmentation on training data
                              download=True) # want to download?

# Get testing data (~250 images x 101 food classes)
test_data = datasets.Food101(root=data_dir,
                             split="test",
                             transform=effnetb2_transforms, # perform
normal EffNetB2 transforms on test data
                             download=True)

# Get Food101 class names
food101_class_names = train_data.classes
```

```
# View the first 10
food101_class_names[:10]
```

```
['apple_pie',
 'baby_back_ribs',
 'baklava',
 'beef_carpaccio',
 'beef_tartare',
 'beet_salad',
 'beignets',
 'bibimbap',
 'bread_pudding',
 'breakfast_burrito']
```

Membuat subkumpulan kumpulan data Food101 untuk eksperimen yang lebih cepat

```
def split_dataset(dataset:torchvision.datasets, split_size:float=0.2,
seed:int=42):
    """Randomly splits a given dataset into two proportions based on
split_size and seed.

    Args:
        dataset (torchvision.datasets): A PyTorch Dataset, typically
one from torchvision.datasets.
        split_size (float, optional): How much of the dataset should
be split?
            E.g. split_size=0.2 means there will be a 20% split and an
80% split. Defaults to 0.2.
        seed (int, optional): Seed for random generator. Defaults to
42.

    Returns:
        tuple: (random_split_1, random_split_2) where random_split_1
is of size split_size*len(dataset) and
            random_split_2 is of size (1-split_size)*len(dataset).
    """
    # Create split lengths based on original dataset length
    length_1 = int(len(dataset) * split_size) # desired length
    length_2 = len(dataset) - length_1 # remaining length

    # Print out info
    print(f"[INFO] Splitting dataset of length {len(dataset)} into
splits of size: {length_1} ({int(split_size*100)}%), {length_2}
({int((1-split_size)*100)}%)")

    # Create splits with given random seed
    random_split_1, random_split_2 =
torch.utils.data.random_split(dataset,
```

```
lengths=[length_1, length_2],

generator=torch.manual_seed(seed)) # set the random seed for
reproducible splits
    return random_split_1, random_split_2

# Create training 20% split of Food101
train_data_food101_20_percent, _ = split_dataset(dataset=train_data,
                                                 split_size=0.2)

# Create testing 20% split of Food101
test_data_food101_20_percent, _ = split_dataset(dataset=test_data,
                                                split_size=0.2)

len(train_data_food101_20_percent), len(test_data_food101_20_percent)
```

```
[INFO] Splitting dataset of length 75750 into splits of size: 15150
(20%), 60600 (80%)
[INFO] Splitting dataset of length 25250 into splits of size: 5050
(20%), 20200 (80%)

(15150, 5050)
```

Mengubah kumpulan data Food101 menjadi DataLoaders

```
import os
import torch

BATCH_SIZE = 32
NUM_WORKERS = 2 if os.cpu_count() <= 4 else 4 # this value is very
experimental and will depend on the hardware you have available,
Google Colab generally provides 2x CPUs

# Create Food101 20 percent training DataLoader
train_dataloader_food101_20_percent =
torch.utils.data.DataLoader(train_data_food101_20_percent,

batch_size=BATCH_SIZE,

shuffle=True,

num_workers=NUM_WORKERS)
# Create Food101 20 percent testing DataLoader
test_dataloader_food101_20_percent =
torch.utils.data.DataLoader(test_data_food101_20_percent,

batch_size=BATCH_SIZE,

shuffle=False,
```

```
num_workers=NUM_WORKERS)
```

Pelatihan FoodVision Model Besar

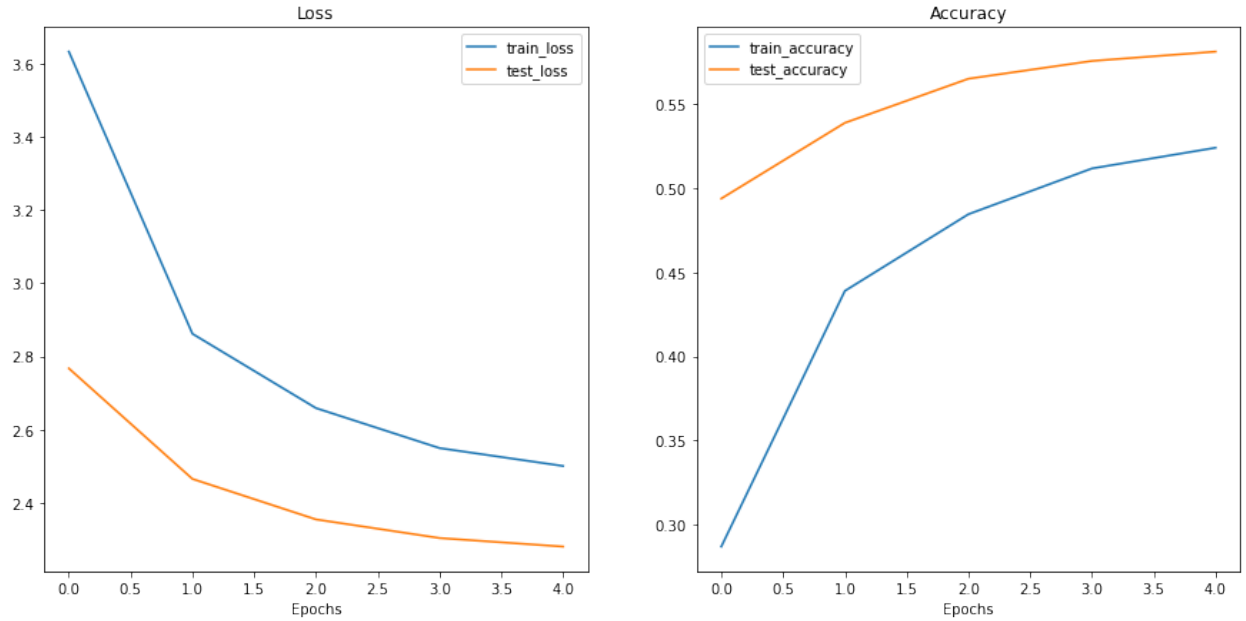
```
from going_modular.going_modular import engine

# Setup optimizer
optimizer = torch.optim.Adam(params=effnetb2_food101.parameters(),
                             lr=1e-3)

# Setup loss function
loss_fn = torch.nn.CrossEntropyLoss(label_smoothing=0.1) # throw in a
little label smoothing because so many classes

# Want to beat original Food101 paper with 20% of data, need 56.4%+
acc on test dataset
set_seeds()
effnetb2_food101_results = engine.train(model=effnetb2_food101,

train_dataloader=train_dataloader_food101_20_percent,

test_dataloader=test_dataloader_food101_20_percent,
                                        optimizer=optimizer,
                                        loss_fn=loss_fn,
                                        epochs=5,
                                        device=device)
```


```
{"model_id":"41ba5e6cee154970aa960f83784b421f","version_major":2,"vers
ion_minor":0}
```


```
Epoch: 1 | train_loss: 3.6317 | train_acc: 0.2869 | test_loss: 2.7670
| test_acc: 0.4937
Epoch: 2 | train_loss: 2.8615 | train_acc: 0.4388 | test_loss: 2.4653
| test_acc: 0.5387
Epoch: 3 | train_loss: 2.6585 | train_acc: 0.4844 | test_loss: 2.3547
| test_acc: 0.5649
Epoch: 4 | train_loss: 2.5494 | train_acc: 0.5116 | test_loss: 2.3038
| test_acc: 0.5755
Epoch: 5 | train_loss: 2.5006 | train_acc: 0.5239 | test_loss: 2.2805
| test_acc: 0.5810
```

Memeriksa kurva kerugian model Besar FoodVision

```
from helper_functions import plot_loss_curves

# Check out the loss curves for FoodVision Big
plot_loss_curves(effnetb2_food101_results)
```



Menyimpan dan memuat FoodVision Big

```
from going_modular.going_modular import utils

# Create a model path
effnetb2_food101_model_path =
"09_pretrained_effnetb2_feature_extractor_food101_20_percent.pth"

# Save FoodVision Big model
utils.save_model(model=effnetb2_food101,
                 target_dir="models",
                 model_name=effnetb2_food101_model_path)
```

```
[INFO] Saving model to:
models/09_pretrained_effnetb2_feature_extractor_food101_20_percent.pth
```

```
# Create Food101 compatible EffNetB2 instance
loaded_effnetb2_food101, effnetb2_transforms =
create_effnetb2_model(num_classes=101)

# Load the saved model's state_dict()
loaded_effnetb2_food101.load_state_dict(torch.load("models/09_pretrain
ed_effnetb2_feature_extractor_food101_20_percent.pth"))
```

```
<All keys matched successfully>
```

Memeriksa FoodVision Ukuran model besar

```
from pathlib import Path

# Get the model size in bytes then convert to megabytes
pretrained_effnetb2_food101_model_size = Path("models",
```

```
effnetb2_food101_model_path).stat().st_size // (1024*1024) # division
converts bytes to megabytes (roughly)
print(f"Pretrained EffNetB2 feature extractor Food101 model size:
{pretrained_effnetb2_food101_model_size} MB")
```

```
Pretrained EffNetB2 feature extractor Food101 model size: 30 MB
```

11. Mengubah model FoodVision Big menjadi aplikasi yang dapat

diterapkan

```
from pathlib import Path

# Create FoodVision Big demo path
foodvision_big_demo_path = Path("demos/foodvision_big/")

# Make FoodVision Big demo directory
foodvision_big_demo_path.mkdir(parents=True, exist_ok=True)

# Make FoodVision Big demo examples directory
(foodvision_big_demo_path / "examples").mkdir(parents=True,
exist_ok=True)
```

Mengunduh gambar contoh dan memindahkannya ke contoh direktori

```
# Download and move an example image
!wget https://raw.githubusercontent.com/mrdbourke/pytorch-deep-
learning/main/images/04-pizza-dad.jpeg
!mv 04-pizza-dad.jpeg demos/foodvision_big/examples/04-pizza-dad.jpg

# Move trained model to FoodVision Big demo folder (will error if
model is already moved)
!mv
models/09_pretrained_effnetb2_feature_extractor_food101_20_percent.pth
demos/foodvision_big
```

```
--2022-08-25 14:24:41--
https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/
main/images/04-pizza-dad.jpeg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.111.133, 185.199.110.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2874848 (2.7M) [image/jpeg]
Saving to: '04-pizza-dad.jpeg’

04-pizza-dad.jpeg   100%[===================>]   2.74M  7.85MB/s    in
```

```
0.3s

2022-08-25 14:24:43 (7.85 MB/s) - '04-pizza-dad.jpeg’ saved
[2874848/2874848]
```

Menyimpan nama kelas Food101 ke file (class_names.txt)

```
# Check out the first 10 Food101 class names
food101_class_names[:10]
```

```
['apple_pie',
 'baby_back_ribs',
 'baklava',
 'beef_carpaccio',
 'beef_tartare',
 'beet_salad',
 'beignets',
 'bibimbap',
 'bread_pudding',
 'breakfast_burrito']
```

```
# Create path to Food101 class names
foodvision_big_class_names_path = foodvision_big_demo_path /
"class_names.txt"

# Write Food101 class names list to file
with open(foodvision_big_class_names_path, "w") as f:
    print(f"[INFO] Saving Food101 class names to
{foodvision_big_class_names_path}")
    f.write("\n".join(food101_class_names)) # leave a new line between
each class
```

```
[INFO] Saving Food101 class names to
demos/foodvision_big/class_names.txt
```

```
# Open Food101 class names file and read each line into a list
with open(foodvision_big_class_names_path, "r") as f:
    food101_class_names_loaded = [food.strip() for food in
f.readlines()]

# View the first 5 class names loaded back in
food101_class_names_loaded[:5]
```

```
['apple_pie', 'baby_back_ribs', 'baklava', 'beef_carpaccio',
'beef_tartare']
```

Mengubah model FoodVision Big menjadi skrip Python (model.py)

```
%%writefile demos/foodvision_big/model.py
import torch
```

```
import torchvision

from torch import nn


def create_effnetb2_model(num_classes:int=3,
                          seed:int=42):
    """Creates an EfficientNetB2 feature extractor model and
transforms.

    Args:
        num_classes (int, optional): number of classes in the
classifier head.
            Defaults to 3.
        seed (int, optional): random seed value. Defaults to 42.

    Returns:
        model (torch.nn.Module): EffNetB2 feature extractor model.
        transforms (torchvision.transforms): EffNetB2 image
transforms.
    """
    # Create EffNetB2 pretrained weights, transforms and model
    weights = torchvision.models.EfficientNet_B2_Weights.DEFAULT
    transforms = weights.transforms()
    model = torchvision.models.efficientnet_b2(weights=weights)

    # Freeze all layers in base model
    for param in model.parameters():
        param.requires_grad = False

    # Change classifier head with random seed for reproducibility
    torch.manual_seed(seed)
    model.classifier = nn.Sequential(
        nn.Dropout(p=0.3, inplace=True),
        nn.Linear(in_features=1408, out_features=num_classes),
    )

    return model, transforms
```

```
Overwriting demos/foodvision_big/model.py
```

Mengubah aplikasi FoodVision Big Gradio menjadi skrip Python (app.py)

```
%%writefile demos/foodvision_big/app.py
### 1. Imports and class names setup ###
import gradio as gr
import os
import torch
```

```
from model import create_effnetb2_model
from timeit import default_timer as timer
from typing import Tuple, Dict

# Setup class names
with open("class_names.txt", "r") as f: # reading them in from
class_names.txt
    class_names = [food_name.strip() for food_name in  f.readlines()]

### 2. Model and transforms preparation ###

# Create model
effnetb2, effnetb2_transforms = create_effnetb2_model(
    num_classes=101, # could also use len(class_names)
)

# Load saved weights
effnetb2.load_state_dict(
    torch.load(

f="09_pretrained_effnetb2_feature_extractor_food101_20_percent.pth",
        map_location=torch.device("cpu"),  # load to CPU
    )
)

### 3. Predict function ###

# Create predict function
def predict(img) -> Tuple[Dict, float]:
    """Transforms and performs a prediction on img and returns
prediction and time taken.
    """
    # Start the timer
    start_time = timer()

    # Transform the target image and add a batch dimension
    img = effnetb2_transforms(img).unsqueeze(0)

    # Put model into evaluation mode and turn on inference mode
    effnetb2.eval()
    with torch.inference_mode():
        # Pass the transformed image through the model and turn the
prediction logits into prediction probabilities
        pred_probs = torch.softmax(effnetb2(img), dim=1)

    # Create a prediction label and prediction probability dictionary
for each prediction class (this is the required format for Gradio's
output parameter)
    pred_labels_and_probs = {class_names[i]: float(pred_probs[0][i])
for i in range(len(class_names))}
```

```
    # Calculate the prediction time
    pred_time = round(timer() - start_time, 5)

    # Return the prediction dictionary and prediction time
    return pred_labels_and_probs, pred_time

### 4. Gradio app ###

# Create title, description and article strings
title = "FoodVision Big 🍔👁"
description = "An EfficientNetB2 feature extractor computer vision
model to classify images of food into [101 different classes]
(https://github.com/mrdbourke/pytorch-deep-learning/blob/main/extras/
food101_class_names.txt)."
article = "Created at [09. PyTorch Model
Deployment](https://www.learnpytorch.io/09_pytorch_model_deployment/).
"

# Create examples list from "examples/" directory
example_list = [["examples/" + example] for example in
os.listdir("examples")]

# Create Gradio interface
demo = gr.Interface(
    fn=predict,
    inputs=gr.Image(type="pil"),
    outputs=[
        gr.Label(num_top_classes=5, label="Predictions"),
        gr.Number(label="Prediction time (s)"),
    ],
    examples=example_list,
    title=title,
    description=description,
    article=article,
)

# Launch the app!
demo.launch()
```

```
Overwriting demos/foodvision_big/app.py
```

Membuat file persyaratan untuk FoodVision Big (requirements.txt)

```
%%writefile demos/foodvision_big/requirements.txt
torch==1.12.0
torchvision==0.13.0
gradio==3.1.4
```

```
Overwriting demos/foodvision_big/requirements.txt
```

Mengunduh file aplikasi FoodVision Big kami

```
# Zip foodvision_big folder but exclude certain files
!cd demos/foodvision_big && zip -r ../foodvision_big.zip * -x "*.pyc"
"*.ipynb" "*__pycache__*" "*ipynb_checkpoints*"

# Download the zipped FoodVision Big app (if running in Google Colab)
try:
    from google.colab import files
    files.download("demos/foodvision_big.zip")
except:
    print("Not running in Google Colab, can't use
google.colab.files.download()")
```

```
updating:
09_pretrained_effnetb2_feature_extractor_food101_20_percent.pth
(deflated 8%)
updating: app.py (deflated 54%)
updating: class_names.txt (deflated 48%)
updating: examples/ (stored 0%)
updating: flagged/ (stored 0%)
updating: model.py (deflated 56%)
updating: requirements.txt (deflated 4%)
updating: examples/04-pizza-dad.jpg (deflated 0%)
Not running in Google Colab, can't use google.colab.files.download()
```

Menerapkan aplikasi FoodVision Big kami ke HuggingFace Spaces

```
# IPython is a library to help work with Python iteractively
from IPython.display import IFrame

# Embed FoodVision Big Gradio demo as an iFrame
IFrame(src="https://hf.space/embed/mrdbourke/foodvision_big/+",
width=900, height=750)
```

```
<IPython.lib.display.IFrame at 0x7f145512baf0>
```