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.dev20220620+cu113
torchvision version: 0.14.0.dev20220620+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
except:
    # Get the going_modular scripts
    print("[INFO] Couldn't find going_modular scripts... downloading
them from GitHub.")
```

```
    !git clone https://github.com/mrdbourke/pytorch-deep-learning
    !mv pytorch-deep-learning/going_modular .
    !rm -rf pytorch-deep-learning
    from going_modular.going_modular import data_setup, engine

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
'cuda'
```

Dapatkan datanya

Mari kita tulis beberapa kode untuk mendownload dataset pizza_steak_sushi.zip dari kursus GitHub dan kemudian unzip.

```
import os
import zipfile

from pathlib import Path

import requests

# Setup path to data folder
data_path = Path("data/")
image_path = data_path / "pizza_steak_sushi"

# If the image folder doesn't exist, download it and prepare it...
if image_path.is_dir():
    print(f"{image_path} directory exists.")
else:
    print(f"Did not find {image_path} directory, creating one...")
    image_path.mkdir(parents=True, exist_ok=True)

    # Download pizza, steak, sushi data
    with open(data_path / "pizza_steak_sushi.zip", "wb") as f:
        request = requests.get("https://github.com/mrdbourke/pytorch-
deep-learning/raw/main/data/pizza_steak_sushi.zip")
        print("Downloading pizza, steak, sushi data...")
        f.write(request.content)

    # Unzip pizza, steak, sushi data
    with zipfile.ZipFile(data_path / "pizza_steak_sushi.zip", "r") as
zip_ref:
        print("Unzipping pizza, steak, sushi data...")
        zip_ref.extractall(image_path)

    # Remove .zip file
    os.remove(data_path / "pizza_steak_sushi.zip")
```

```
data/pizza_steak_sushi directory exists.
```

Sekarang mari kita buat jalur ke direktori pelatihan dan pengujian kita.

```
# Setup Dirs
train_dir = image_path / "train"
test_dir = image_path / "test"
```

Buat Kumpulan Data dan Pemuat Data

Membuat transformasi untuk torchvision.models (pembuatan manual)

buat serangkaian torchvision.transforms

```
# Create a transforms pipeline manually (required for torchvision < 0.13)
manual_transforms = transforms.Compose([
    transforms.Resize((224, 224)), # 1. Reshape all images to 224x224 (though some models may require different sizes)
    transforms.ToTensor(), # 2. Turn image values to between 0 & 1
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # 3. A mean of [0.485, 0.456, 0.406] (across each colour channel)
                         std=[0.229, 0.224, 0.225]) # 4. A standard deviation of [0.229, 0.224, 0.225] (across each colour channel),
])
```

Kami akan menetapkan batch_size=32 sehingga model kami melihat kumpulan kecil yang terdiri dari 32 sampel sekaligus.

```
# Create training and testing DataLoaders as well as get a list of class names
train_dataloader, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=manual_transforms, # resize, convert images to between 0 & 1 and normalize them

batch_size=32) # set mini-batch size to 32

train_dataloader, test_dataloader, class_names
```

```
(<torch.utils.data.dataloader.DataLoader at 0x7fa9429a3a60>,
 <torch.utils.data.dataloader.DataLoader at 0x7fa9429a37c0>,
 ['pizza', 'steak', 'sushi'])
```

Membuat transformasi untuk torchvision.models (pembuatan otomatis)

```
# Get a set of pretrained model weights
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
# .DEFAULT = best available weights from pretraining on ImageNet
weights
```

```
EfficientNet_B0_Weights.IMAGENET1K_V1
```

Ini pada dasarnya mengatakan "dapatkan transformasi data yang digunakan untuk melatih EfficientNet_B0_Weights di ImageNet".

```
# Get the transforms used to create our pretrained weights
auto_transforms = weights.transforms()
auto_transforms
```

```
ImageClassification(
    crop_size=[224]
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BICUBIC
)
```

Kita bisa menggunakan auto_transforms untuk membuat DataLoaders dengan create_dataloaders() seperti sebelumnya.

```
# Create training and testing DataLoaders as well as get a list of
class names
train_dataloader, test_dataloader, class_names =
data_setup.create_dataloaders(train_dir=train_dir,

test_dir=test_dir,

transform=auto_transforms, # perform same data transforms on our own
data as the pretrained model

batch_size=32) # set mini-batch size to 32

train_dataloader, test_dataloader, class_names
```

```
(<torch.utils.data.dataloader.DataLoader at 0x7fa942951460>,
 <torch.utils.data.dataloader.DataLoader at 0x7fa942951550>,
 ['pizza', 'steak', 'sushi'])
```

Mendapatkan model terlatih

Model terlatih manakah yang sebaiknya Anda gunakan?

Menyiapkan model terlatih

Kita dapat menyiapkan bobot ImageNet yang telah dilatih EfficientNet_B0 menggunakan kode yang sama seperti yang kita gunakan untuk membuat transformasi.

```
# OLD: Setup the model with pretrained weights and send it to the
target device (this was prior to torchvision v0.13)
# model =
torchvision.models.efficientnet_b0(pretrained=True).to(device) # OLD
method (with pretrained=True)

# NEW: Setup the model with pretrained weights and send it to the
target device (torchvision v0.13+)
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
# .DEFAULT = best available weights
model = torchvision.models.efficientnet_b0(weights=weights).to(device)

#model # uncomment to output (it's very long)
```

Mendapatkan ringkasan model kita dengan torchinfo.summary()

Untuk mempelajari lebih lanjut tentang model kita, mari gunakan metode ringkasan() torchinfo.

```
# Print a summary using torchinfo (uncomment for actual output)
summary(model=model,
        input_size=(32, 3, 224, 224), # make sure this is
"input_size", not "input_shape"
        # col_names=["input_size"], # uncomment for smaller output
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"]
)
```

```
==========================================================================
==========================================================================
Layer (type (var_name))                                   Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                               [32, 3,
224, 224]    [32, 1000]           --                   True
├─Sequential (features)                                   [32, 3,
224, 224]    [32, 1280, 7, 7]     --                   True
│    └─Conv2dNormActivation (0)                           [32, 3,
```

```
224, 224]    [32, 32, 112, 112]   --                   True
|    |    └─Conv2d (0)                                      [32, 3,
224, 224]    [32, 32, 112, 112]   864                  True
|    |    └─BatchNorm2d (1)                                 [32, 32,
112, 112]   [32, 32, 112, 112]   64                   True
|    |    └─SiLU (2)                                        [32, 32,
112, 112]   [32, 32, 112, 112]   --                   --
|    └─Sequential (1)                                       [32, 32,
112, 112]   [32, 16, 112, 112]   --                   True
|    |    └─MBConv (0)                                      [32, 32,
112, 112]   [32, 16, 112, 112]   1,448                True
|    └─Sequential (2)                                       [32, 16,
112, 112]   [32, 24, 56, 56]     --                   True
|    |    └─MBConv (0)                                      [32, 16,
112, 112]   [32, 24, 56, 56]     6,004                True
|    |    └─MBConv (1)                                      [32, 24,
56, 56]     [32, 24, 56, 56]     10,710               True
|    └─Sequential (3)                                       [32, 24,
56, 56]     [32, 40, 28, 28]     --                   True
|    |    └─MBConv (0)                                      [32, 24,
56, 56]     [32, 40, 28, 28]     15,350               True
|    |    └─MBConv (1)                                      [32, 40,
28, 28]     [32, 40, 28, 28]     31,290               True
|    └─Sequential (4)                                       [32, 40,
28, 28]     [32, 80, 14, 14]     --                   True
|    |    └─MBConv (0)                                      [32, 40,
28, 28]     [32, 80, 14, 14]     37,130               True
|    |    └─MBConv (1)                                      [32, 80,
14, 14]     [32, 80, 14, 14]     102,900              True
|    |    └─MBConv (2)                                      [32, 80,
14, 14]     [32, 80, 14, 14]     102,900              True
|    └─Sequential (5)                                       [32, 80,
14, 14]     [32, 112, 14, 14]    --                   True
|    |    └─MBConv (0)                                      [32, 80,
14, 14]     [32, 112, 14, 14]    126,004              True
|    |    └─MBConv (1)                                      [32, 112,
14, 14]    [32, 112, 14, 14]    208,572              True
|    |    └─MBConv (2)                                      [32, 112,
14, 14]    [32, 112, 14, 14]    208,572              True
|    └─Sequential (6)                                       [32, 112,
14, 14]    [32, 192, 7, 7]      --                   True
|    |    └─MBConv (0)                                      [32, 112,
14, 14]    [32, 192, 7, 7]      262,492              True
|    |    └─MBConv (1)                                      [32, 192,
7, 7]      [32, 192, 7, 7]      587,952              True
|    |    └─MBConv (2)                                      [32, 192,
7, 7]      [32, 192, 7, 7]      587,952              True
|    |    └─MBConv (3)                                      [32, 192,
7, 7]      [32, 192, 7, 7]      587,952              True
```

```
|    └─Sequential (7)                                        [32, 192,
7, 7]      [32, 320, 7, 7]      --                   True
|    |    └─MBConv (0)                                       [32, 192,
7, 7]      [32, 320, 7, 7]      717,232              True
|    └─Conv2dNormActivation (8)                              [32, 320,
7, 7]      [32, 1280, 7, 7]     --                   True
|    |    └─Conv2d (0)                                       [32, 320,
7, 7]      [32, 1280, 7, 7]     409,600              True
|    |    └─BatchNorm2d (1)                                  [32,
1280, 7, 7]     [32, 1280, 7, 7]     2,560                True
|    |    └─SiLU (2)                                         [32,
1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─AdaptiveAvgPool2d (avgpool)                               [32,
1280, 7, 7]     [32, 1280, 1, 1]     --                   --
├─Sequential (classifier)                                   [32,
1280]           [32, 1000]           --                   True
|    └─Dropout (0)                                          [32,
1280]           [32, 1280]           --                   --
|    └─Linear (1)                                           [32,
1280]           [32, 1000]           1,281,000            True
==========================================================================
==========================================================================
Total params: 5,288,548
Trainable params: 5,288,548
Non-trainable params: 0
Total mult-adds (G): 12.35
==========================================================================
==========================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3452.35
Params size (MB): 21.15
Estimated Total Size (MB): 3492.77
==========================================================================
==========================================================================
```

Membekukan model dasar dan mengubah lapisan keluaran agar sesuai dengankebutuhan kita

parameter dengan require_grad=False adalah "tidak dapat dilatih" atau "dibekukan" pada tempatnya

```python
# Freeze all base layers in the "features" section of the model (the
feature extractor) by setting requires_grad=False
for param in model.features.parameters():
    param.requires_grad = False
```

Lapisan pengklasifikasi baru kita harus berada di perangkat yang sama dengan model kita.

```
# Set the manual seeds
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# Get the length of class_names (one output unit for each class)
output_shape = len(class_names)

# Recreate the classifier layer and seed it to the target device
model.classifier = torch.nn.Sequential(
    torch.nn.Dropout(p=0.2, inplace=True),
    torch.nn.Linear(in_features=1280,
                    out_features=output_shape, # same number of output
units as our number of classes
                    bias=True)).to(device)
```

Lapisan keluaran diperbarui, mari kita lihat ringkasan lain dari model kita dan lihat apa yang berubah.

```
# # Do a summary *after* freezing the features and changing the output
classifier layer (uncomment for actual output)
summary(model,
        input_size=(32, 3, 224, 224), # make sure this is
"input_size", not "input_shape" (batch_size, color_channels, height,
width)
        verbose=0,
        col_names=["input_size", "output_size", "num_params",
"trainable"],
        col_width=20,
        row_settings=["var_names"]
)
```

```
==========================================================================
==========================================================================
Layer (type (var_name))                                      Input
Shape          Output Shape         Param #              Trainable
==========================================================================
==========================================================================
EfficientNet (EfficientNet)                                  [32, 3,
224, 224]    [32, 3]              --                   Partial
├─Sequential (features)                                      [32, 3,
224, 224]    [32, 1280, 7, 7]     --                   False
│    └─Conv2dNormActivation (0)                              [32, 3,
224, 224]    [32, 32, 112, 112]   --                   False
│    │    └─Conv2d (0)                                       [32, 3,
224, 224]    [32, 32, 112, 112]   (864)                False
│    │    └─BatchNorm2d (1)                                  [32, 32,
112, 112]    [32, 32, 112, 112]   (64)                 False
│    │    └─SiLU (2)                                         [32, 32,
112, 112]    [32, 32, 112, 112]   --                   --
```

```
|    └─Sequential (1)                                                  [32, 32,
112, 112]   [32, 16, 112, 112]   --                    False
|    |    └─MBConv (0)                                                [32, 32,
112, 112]   [32, 16, 112, 112]   (1,448)               False
|    └─Sequential (2)                                                  [32, 16,
112, 112]   [32, 24, 56, 56]     --                    False
|    |    └─MBConv (0)                                                [32, 16,
112, 112]   [32, 24, 56, 56]     (6,004)               False
|    |    └─MBConv (1)                                                [32, 24,
56, 56]     [32, 24, 56, 56]     (10,710)              False
|    └─Sequential (3)                                                  [32, 24,
56, 56]     [32, 40, 28, 28]     --                    False
|    |    └─MBConv (0)                                                [32, 24,
56, 56]     [32, 40, 28, 28]     (15,350)              False
|    |    └─MBConv (1)                                                [32, 40,
28, 28]     [32, 40, 28, 28]     (31,290)              False
|    └─Sequential (4)                                                  [32, 40,
28, 28]     [32, 80, 14, 14]     --                    False
|    |    └─MBConv (0)                                                [32, 40,
28, 28]     [32, 80, 14, 14]     (37,130)              False
|    |    └─MBConv (1)                                                [32, 80,
14, 14]     [32, 80, 14, 14]     (102,900)             False
|    |    └─MBConv (2)                                                [32, 80,
14, 14]     [32, 80, 14, 14]     (102,900)             False
|    └─Sequential (5)                                                  [32, 80,
14, 14]     [32, 112, 14, 14]    --                    False
|    |    └─MBConv (0)                                                [32, 80,
14, 14]     [32, 112, 14, 14]    (126,004)             False
|    |    └─MBConv (1)                                                [32, 112,
14, 14]    [32, 112, 14, 14]    (208,572)             False
|    |    └─MBConv (2)                                                [32, 112,
14, 14]    [32, 112, 14, 14]    (208,572)             False
|    └─Sequential (6)                                                  [32, 112,
14, 14]    [32, 192, 7, 7]      --                    False
|    |    └─MBConv (0)                                                [32, 112,
14, 14]    [32, 192, 7, 7]      (262,492)             False
|    |    └─MBConv (1)                                                [32, 192,
7, 7]      [32, 192, 7, 7]      (587,952)             False
|    |    └─MBConv (2)                                                [32, 192,
7, 7]      [32, 192, 7, 7]      (587,952)             False
|    |    └─MBConv (3)                                                [32, 192,
7, 7]      [32, 192, 7, 7]      (587,952)             False
|    └─Sequential (7)                                                  [32, 192,
7, 7]      [32, 320, 7, 7]      --                    False
|    |    └─MBConv (0)                                                [32, 192,
7, 7]      [32, 320, 7, 7]      (717,232)             False
|    └─Conv2dNormActivation (8)                                        [32, 320,
7, 7]      [32, 1280, 7, 7]     --                    False
|    |    └─Conv2d (0)                                                [32, 320,
```

```
7, 7]      [32, 1280, 7, 7]     (409,600)            False
│    │    └─BatchNorm2d (1)                                  [32,
1280, 7, 7]     [32, 1280, 7, 7]     (2,560)              False
│    │    └─SiLU (2)                                         [32,
1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─AdaptiveAvgPool2d (avgpool)                                [32,
1280, 7, 7]     [32, 1280, 1, 1]     --                   --
├─Sequential (classifier)                                    [32,
1280]           [32, 3]              --                   True
│    └─Dropout (0)                                           [32,
1280]           [32, 1280]           --                   --
│    └─Linear (1)                                           [32,
1280]           [32, 3]              3,843                True
==========================================================================
==========================================================================
Total params: 4,011,391
Trainable params: 3,843
Non-trainable params: 4,007,548
Total mult-adds (G): 12.31
==========================================================================
==========================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3452.09
Params size (MB): 16.05
Estimated Total Size (MB): 3487.41
==========================================================================
==========================================================================
```

Train model

Dan kami akan tetap menggunakan torch.optim.Adam() sebagai pengoptimal kami dengan lr=0.001.

```
# Define loss and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Set the random seeds
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# Start the timer
from timeit import default_timer as timer
start_time = timer()

# Setup training and save the results
results = engine.train(model=model,
                       train_dataloader=train_dataloader,
                       test_dataloader=test_dataloader,
```

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

# End the timer and print out how long it took
end_time = timer()
print(f"[INFO] Total training time: {end_time-start_time:.3f}
seconds")
```


```
{"model_id":"b61588abc1df499286a8e260d139026b","version_major":2,"vers
ion_minor":0}
```

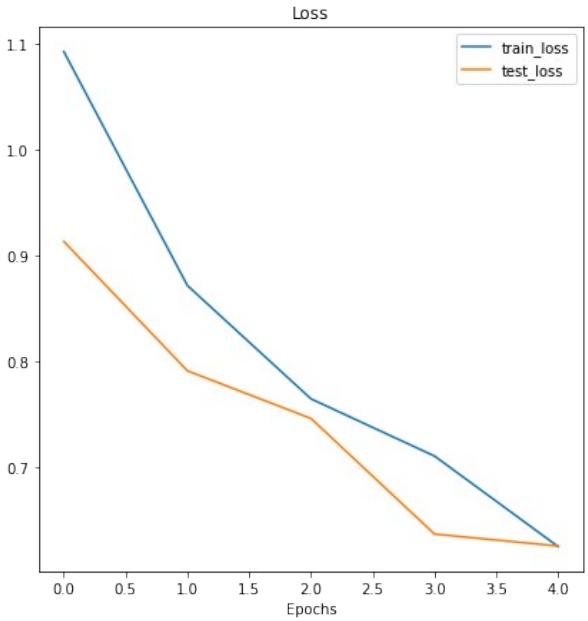

```
Epoch: 1 | train_loss: 1.0924 | train_acc: 0.3984 | test_loss: 0.9133
| test_acc: 0.5398
Epoch: 2 | train_loss: 0.8717 | train_acc: 0.7773 | test_loss: 0.7912
| test_acc: 0.8153
Epoch: 3 | train_loss: 0.7648 | train_acc: 0.7930 | test_loss: 0.7463
| test_acc: 0.8561
Epoch: 4 | train_loss: 0.7108 | train_acc: 0.7539 | test_loss: 0.6372
| test_acc: 0.8655
Epoch: 5 | train_loss: 0.6254 | train_acc: 0.7852 | test_loss: 0.6260
| test_acc: 0.8561
[INFO] Total training time: 8.977 seconds
```

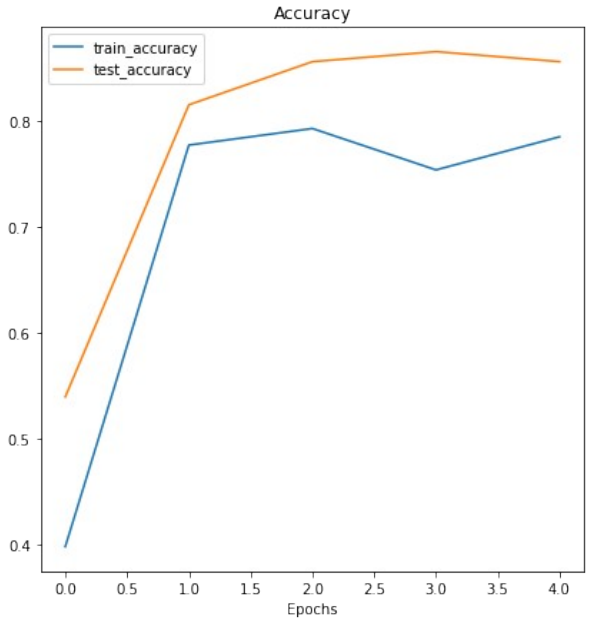
Evaluasi model dengan memplot kurva kerugian

Kita dapat memplot kurva kerugian menggunakan fungsi plot_loss_curves()

```
# Get the plot_loss_curves() function from helper_functions.py,
download the file if we don't have it
try:
    from helper_functions import plot_loss_curves
except:
    print("[INFO] Couldn't find helper_functions.py, downloading...")
    with open("helper_functions.py", "wb") as f:
        import requests
        request =
requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-
deep-learning/main/helper_functions.py")
        f.write(request.content)
    from helper_functions import plot_loss_curves

# Plot the loss curves of our model
plot_loss_curves(results)
```





. Membuat prediksi pada gambar dari set tes

```
from typing import List, Tuple

from PIL import Image

# 1. Take in a trained model, class names, image path, image size, a
transform and target device
def pred_and_plot_image(model: torch.nn.Module,
                        image_path: str,
                        class_names: List[str],
                        image_size: Tuple[int, int] = (224, 224),
                        transform: torchvision.transforms = None,
                        device: torch.device=device):


    # 2. Open image
    img = Image.open(image_path)

    # 3. Create transformation for image (if one doesn't exist)
    if transform is not None:
        image_transform = transform
    else:
        image_transform = transforms.Compose([
            transforms.Resize(image_size),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225]),
        ])

    ### Predict on image ###
```

```
    # 4. Make sure the model is on the target device
    model.to(device)

    # 5. Turn on model evaluation mode and inference mode
    model.eval()
    with torch.inference_mode():
      # 6. Transform and add an extra dimension to image (model
requires samples in [batch_size, color_channels, height, width])
      transformed_image = image_transform(img).unsqueeze(dim=0)

      # 7. Make a prediction on image with an extra dimension and send
it to the target device
      target_image_pred = model(transformed_image.to(device))

    # 8. Convert logits -> prediction probabilities (using
torch.softmax() for multi-class classification)
    target_image_pred_probs = torch.softmax(target_image_pred, dim=1)

    # 9. Convert prediction probabilities -> prediction labels
    target_image_pred_label = torch.argmax(target_image_pred_probs,
dim=1)

    # 10. Plot image with predicted label and probability
    plt.figure()
    plt.imshow(img)
    plt.title(f"Pred: {class_names[target_image_pred_label]} | Prob:
{target_image_pred_probs.max():.3f}")
    plt.axis(False);

# Get a random list of image paths from test set
import random
num_images_to_plot = 3
test_image_path_list = list(Path(test_dir).glob("*/*.jpg")) # get list
all image paths from test data
test_image_path_sample =
random.sample(population=test_image_path_list, # go through all of the
test image paths
                                      k=num_images_to_plot) #
randomly select 'k' image paths to pred and plot

# Make predictions on and plot the images
for image_path in test_image_path_sample:
    pred_and_plot_image(model=model,
                        image_path=image_path,
                        class_names=class_names,
                        # transform=weights.transforms(), # optionally
pass in a specified transform from our pretrained model weights
                        image_size=(224, 224))
```

Pred: sushi | Prob: 0.507

Pred: sushi | Prob: 0.427

Membuat prediksi pada gambar khusus

Kami kemudian akan meneruskannya ke fungsi pred_and_plot_image() yang kami buat di atas dan melihat apa yang terjadi.

```
# Download custom image
import requests

# Setup custom image path
custom_image_path = data_path / "04-pizza-dad.jpeg"

# Download the image if it doesn't already exist
if not custom_image_path.is_file():
    with open(custom_image_path, "wb") as f:
        # When downloading from GitHub, need to use the "raw" file
link
        request =
requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-
deep-learning/main/images/04-pizza-dad.jpeg")
        print(f"Downloading {custom_image_path}...")
        f.write(request.content)
else:
    print(f"{custom_image_path} already exists, skipping download.")

# Predict on custom image
pred_and_plot_image(model=model,
                    image_path=custom_image_path,
                    class_names=class_names)
```

```
data/04-pizza-dad.jpeg already exists, skipping download.
```

Pred: pizza | Prob: 0.499