Results

December 19, 2024

[ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

import torch
import torchvision.models
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data.dataloader import DataLoader
from torchvision.datasets import ImageFolder
from torch.utils.data import random_split
import torchvision.transforms as transforms
from tqdm import tqdm
import torch.optim as optim

from torchvision.models import resnet50, ResNet50_Weights
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

[ ]:

```
dir(torchvision.models)
```

[ ]:

```
['AlexNet',
 'AlexNet_Weights',
 'ConvNeXt',
 'ConvNeXt_Base_Weights',
 'ConvNeXt_Large_Weights',
 'ConvNeXt_Small_Weights',
 'ConvNeXt_Tiny_Weights',
 'DenseNet',
 'DenseNet121_Weights',
 'DenseNet161_Weights',
 'DenseNet169_Weights',
 'DenseNet201_Weights',
 'EfficientNet',
 'EfficientNet_B0_Weights',
 'EfficientNet_B1_Weights',
 'EfficientNet_B2_Weights',
 'EfficientNet_B3_Weights',
 'EfficientNet_B4_Weights',
 'EfficientNet_B5_Weights',
 'EfficientNet_B6_Weights',
 'EfficientNet_B7_Weights',
 'EfficientNet_V2_L_Weights',
 'EfficientNet_V2_M_Weights',
 'EfficientNet_V2_S_Weights',
 'GoogLeNet',
 'GoogLeNetOutputs',
 'GoogLeNet_Weights',
 'Inception3',
 'InceptionOutputs',
 'Inception_V3_Weights',
 'MNASNet',
 'MNASNet0_5_Weights',
 'MNASNet0_75_Weights',
 'MNASNet1_0_Weights',
 'MNASNet1_3_Weights',
 'MaxVit',
 'MaxVit_T_Weights',
 'MobileNetV2',
 'MobileNetV3',
 'MobileNet_V2_Weights',
 'MobileNet_V3_Large_Weights',
 'MobileNet_V3_Small_Weights',
 'RegNet',
 'RegNet_X_16GF_Weights',
 'RegNet_X_1_6GF_Weights',
 'RegNet_X_32GF_Weights',
 'RegNet_X_3_2GF_Weights',
 'RegNet_X_400MF_Weights',
 'RegNet_X_800MF_Weights',
 'RegNet_X_8GF_Weights',
 'RegNet_Y_128GF_Weights',
 'RegNet_Y_16GF_Weights',
 'RegNet_Y_1_6GF_Weights',
 'RegNet_Y_32GF_Weights',
 'RegNet_Y_3_2GF_Weights',
 'RegNet_Y_400MF_Weights',
 'RegNet_Y_800MF_Weights',
 'RegNet_Y_8GF_Weights',
 'ResNeXt101_32X8D_Weights',
 'ResNeXt101_64X4D_Weights',
 'ResNeXt50_32X4D_Weights',
 'ResNet',
 'ResNet101_Weights',
 'ResNet152_Weights',
 'ResNet18_Weights',
 'ResNet34_Weights',
 'ResNet50_Weights',
 'ShuffleNetV2',
 'ShuffleNet_V2_X0_5_Weights',
 'ShuffleNet_V2_X1_0_Weights',
 'ShuffleNet_V2_X1_5_Weights',
 'ShuffleNet_V2_X2_0_Weights',
 'SqueezeNet',
 'SqueezeNet1_0_Weights',
 'SqueezeNet1_1_Weights',
 'SwinTransformer',
 'Swin_B_Weights',
 'Swin_S_Weights',
 'Swin_T_Weights',
 'Swin_V2_B_Weights',
 'Swin_V2_S_Weights',
 'Swin_V2_T_Weights',
 'VGG',
 'VGG11_BN_Weights',
 'VGG11_Weights',
 'VGG13_BN_Weights',
 'VGG13_Weights',
 'VGG16_BN_Weights',
 'VGG16_Weights',
 'VGG19_BN_Weights',
 'VGG19_Weights',
 'ViT_B_16_Weights',
 'ViT_B_32_Weights',
 'ViT_H_14_Weights',
 'ViT_L_16_Weights',
 'ViT_L_32_Weights',
 'VisionTransformer',
 'Weights',
 'WeightsEnum',
 'Wide_ResNet101_2_Weights',
 'Wide_ResNet50_2_Weights',
 '_GoogLeNetOutputs',
 '_InceptionOutputs',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_api',
 '_meta',
 '_utils',
 'alexnet',
 'convnext',
 'convnext_base',
 'convnext_large',
 'convnext_small',
 'convnext_tiny',
 'densenet',
 'densenet121',
 'densenet161',
 'densenet169',
 'densenet201',
 'detection',
 'efficientnet',
 'efficientnet_b0',
 'efficientnet_b1',
 'efficientnet_b2',
 'efficientnet_b3',
 'efficientnet_b4',
 'efficientnet_b5',
 'efficientnet_b6',
 'efficientnet_b7',
 'efficientnet_v2_l',
 'efficientnet_v2_m',
 'efficientnet_v2_s',
 'get_model',
 'get_model_builder',
 'get_model_weights',
 'get_weight',
 'googlenet',
 'inception',
 'inception_v3',
 'list_models',
 'maxvit',
 'maxvit_t',
 'mnasnet',
 'mnasnet0_5',
 'mnasnet0_75',
 'mnasnet1_0',
 'mnasnet1_3',
 'mobilenet',
 'mobilenet_v2',
 'mobilenet_v3_large',
 'mobilenet_v3_small',
 'mobilenetv2',
 'mobilenetv3',
 'optical_flow',
 'quantization',
 'regnet',
 'regnet_x_16gf',
 'regnet_x_1_6gf',
 'regnet_x_32gf',
 'regnet_x_3_2gf',
 'regnet_x_400mf',
 'regnet_x_800mf',
 'regnet_x_8gf',
 'regnet_y_128gf',
 'regnet_y_16gf',
 'regnet_y_1_6gf',
 'regnet_y_32gf',
 'regnet_y_3_2gf',
 'regnet_y_400mf',
 'regnet_y_800mf',
 'regnet_y_8gf',
 'resnet',
 'resnet101',
 'resnet152',
 'resnet18',
 'resnet34',
 'resnet50',
 'resnext101_32x8d',
 'resnext101_64x4d',
 'resnext50_32x4d',
 'segmentation',
 'shufflenet_v2_x0_5',
 'shufflenet_v2_x1_0',
 'shufflenet_v2_x1_5',
 'shufflenet_v2_x2_0',
 'shufflenetv2',
 'squeezenet',
 'squeezenet1_0',
 'squeezenet1_1',
 'swin_b',
 'swin_s',
 'swin_t',
 'swin_transformer',
 'swin_v2_b',
 'swin_v2_s',
 'swin_v2_t',
 'vgg',
 'vgg11',
 'vgg11_bn',
 'vgg13',
 'vgg13_bn',
 'vgg16',
 'vgg16_bn',
 'vgg19',
 'vgg19_bn',
 'video',
 'vision_transformer',
 'vit_b_16',
 'vit_b_32',
 'vit_h_14',
 'vit_l_16',
 'vit_l_32',
 'wide_resnet101_2',
 'wide_resnet50_2']
```

```
[ ]: resnet50(weights=ResNet50_Weights.DEFAULT)
```

```
[ ]: ResNet(
       (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
     bias=False)
       (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
       (relu): ReLU(inplace=True)
       (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
     ceil_mode=False)
       (layer1): Sequential(
         (0): Bottleneck(
           (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
           (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
           (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
     bias=False)
           (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
           (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
           (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
           (relu): ReLU(inplace=True)
           (downsample): Sequential(
             (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
             (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
           )
         )
```

```
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

[ ]:

```
class ImageClassificationModel_FC(nn.Module):
    def __init__(self):
        super(ImageClassificationModel_FC, self).__init__()
        self.fc1 = nn.Linear(3 * 300 * 300, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128 ,128)
        self.fc4 = nn.Linear(128 ,128)
        self.fc5 = nn.Linear(128 ,128)
        self.fc6 = nn.Linear(128 ,128)

        self.dropout = nn.Dropout(0.5)

        self.fcm = nn.Linear(128, 3)
    def forward(self, x):
        x = x.reshape(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = torch.relu(self.fc6(x))

        x = self.dropout(x)

        x = self.fcm(x)

        return x
```

[ ]:

```
class ImageClassificationModel_CNN(nn.Module):
    def __init__(self):
        super(ImageClassificationModel_CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding = 0)
        self.dropout = nn.Dropout(0.5)

        self.fc1 = nn.Linear(64 * 37 * 37, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 3)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))

        x = x.view(-1, 64 * 37 * 37)

        x = self.dropout(x)

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

[ ]:

```
class building_block_method(nn.Module):
    def __init__(self, num_classes, fc_requires_grad=True):
        super(building_block_method, self).__init__()
        resnet_model = resnet50(weights=ResNet50_Weights.DEFAULT)
        in_features = resnet_model.fc.in_features
        resnet_model.fc = nn.Identity()

        for param in resnet_model.parameters():
            param.requires_grad = False

        self.resnet = resnet_model
        self.fc1 = nn.Linear(in_features, 32)
        self.fc1.requires_grad = False
        self.fc2 = nn.Linear(32, 64)
        self.fc2.requires_grad = fc_requires_grad
        self.fc3 = nn.Linear(64, num_classes)
        self.fc3.requires_grad = fc_requires_grad

    def forward(self,x):
        x = self.resnet(x)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

[ ]:

```
class ImageDataLoader:
    def __init__(self):
        transform = transforms.Compose([
        transforms.Resize((300, 300)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])

        root_dir = 'ProjectImages'
        full_train_dataset = ImageFolder(root=root_dir, transform=transform)
        self.classes = full_train_dataset.classes

        total_size = len(full_train_dataset)
        train_size = int(total_size * 0.7)
        val_size = int(total_size * 0.1)
        test_size = total_size - train_size - val_size

        self.train_data, self.val_data, self.test_data = random_split(full_train_dataset, [train_size, val_size, test_size])

        self.train_loader = DataLoader(self.train_data, batch_size=32, shuffle=True, num_workers=2, pin_memory=True)
        self.val_loader = DataLoader(self.val_data, batch_size=32,␣
↪shuffle=True, num_workers=2, pin_memory=True)
        self.test_loader = DataLoader(self.test_data, batch_size=32,␣
↪shuffle=True, num_workers=2, pin_memory=True)
```

[ ]:

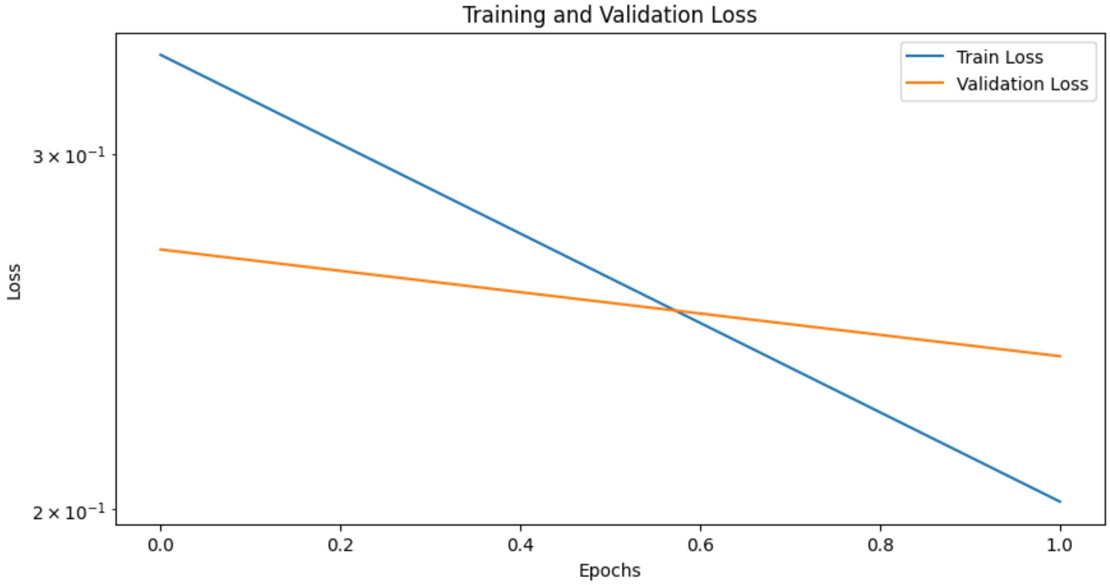
```
class ModelTrainer:
    def __init__(self, model, train_loader, val_loader, test_loader, classes,␣
↪epochs=10, learning_rate=0.001):
        self.model = model
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.test_loader = test_loader
        self.classes = classes
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(model.parameters(), lr=learning_rate)

        self.epochs = epochs

        self.train_losses = []
        self.val_losses = []


    def train(self, device):
        self.model.to(device)
        for epoch in range(self.epochs):
            self.model.train()
            train_loss = 0
            with tqdm(total=len(self.train_loader), desc=f"Epoch {epoch + 1}/
↪{self.epochs}", unit="batch", leave=False) as pbar:
                for features, labels in self.train_loader:
                    features, labels = features.to(device), labels.to(device)
                    outputs = self.model(features)
                    loss = self.criterion(outputs, labels)
                    train_loss += loss.item()

                    self.optimizer.zero_grad()
                    loss.backward()
                    self.optimizer.step()

                    pbar.set_postfix({'train_loss': f'{loss.item():.4f}'})
                    pbar.update(1)

            train_loss /= len(self.train_loader)
            val_loss = self.validate(device)

            self.train_losses.append(train_loss)
            self.val_losses.append(val_loss)
            print(f"Epoch {epoch + 1}/{self.epochs} - Train Loss: {train_loss:.
↪4f}, Validation Loss: {val_loss:.4f}")


    def validate(self, device):
        self.model.eval()
        val_loss = 0
        with torch.no_grad():
            for features, labels in self.val_loader:
                features, labels = features.to(device), labels.to(device)
                outputs = self.model(features)
                loss = self.criterion(outputs, labels)
                val_loss += loss.item()

        val_loss /= len(self.val_loader)
        return val_loss

    def plot_loss(self):
        plt.figure(figsize=(10, 5))
        plt.plot(self.train_losses, label='Train Loss')
        plt.plot(self.val_losses, label='Validation Loss')
        plt.title('Training and Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.yscale('log')
        plt.legend()
        plt.show()

    def evaluate(self, device):
        self.model.eval()
        test_loss = 0
        all_labels = []
        all_preds = []

        total = 0
        correct = 0

        correct_predictions = []
        incorrect_predictions = []


        with torch.no_grad():
            for features, labels in self.test_loader:
                features, labels = features.to(device), labels.to(device)
                outputs = self.model(features.float())
                loss = self.criterion(outputs, labels)
                test_loss += loss.item()

                output_prob = F.softmax(outputs, dim=1).cpu().numpy()

                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

                all_labels.extend(labels.cpu().numpy())
                all_preds.extend(predicted.cpu().numpy())

                for f, l, p, o, prob in zip(features, labels, predicted,␣
↪outputs, output_prob):
                    if l == p and len(correct_predictions) < 10:
                        correct_predictions.append((f, l, p, o, prob))
                    elif l != p and len(incorrect_predictions) < 10:
                        incorrect_predictions.append((f, l, p, o, prob))
                    if len(correct_predictions) >= 10 and␣
↪len(incorrect_predictions) >= 10:
                        break

        print("\n----- Correct Predictions -----")
        for f, l, p, o, prob in correct_predictions:
            f_permute = f.permute(1, 2, 0).cpu().numpy()

            restored_image = (f_permute * [0.229, 0.224, 0.225]) + [0.485, 0.
↪456, 0.406]
            restored_image = np.clip(restored_image, 0, 1)

            plt.imshow(restored_image, cmap='gray')
            plt.title(f'Predicted: {self.classes[p]}, Actual: {self.
↪classes[l]}')
            plt.show()
            print(f"{o}->{prob}: {self.classes[l]} -> {self.classes[p]}")

        print("\n----- Incorrect Predictions -----")
        for f, l, p, o, prob in incorrect_predictions:
            f_permute = f.permute(1, 2, 0).cpu().numpy()

            restored_image = (f_permute * [0.229, 0.224, 0.225]) + [0.485, 0.
↪456, 0.406]
            restored_image = np.clip(restored_image, 0, 1)

            plt.imshow(restored_image, cmap='gray')
            plt.title(f'Predicted: {self.classes[p]}, Actual: {self.
↪classes[l]}')
```

```
            plt.show()
            print(f"{o}->{prob}: {self.classes[l]} -> {self.classes[p]}")

        test_loss /= len(self.test_loader)
        print(f"Accuracy = {correct/total: .4f}")

        cm = confusion_matrix(all_labels, all_preds)
        print(f"Test Loss: {test_loss:.4f}")
        print("Confusion Matrix:")
        print(cm)
```

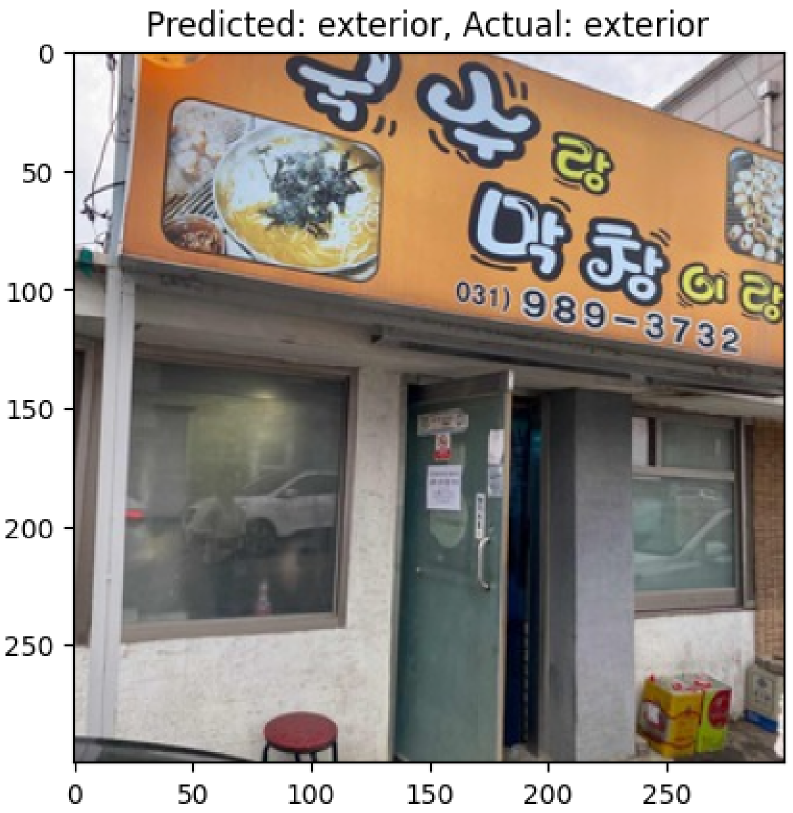
[ ]:

```
if __name__ == '__main__':
    device = torch.device('mps') if torch.backends.mps.is_available() else "cpu"
    print(device)

    model1 = ImageClassificationModel_FC()
    model1.to(device)

    model2 = ImageClassificationModel_CNN()
    model2.to(device)

    resnet50(weights=ResNet50_Weights.DEFAULT)
    model3 = building_block_method(num_classes=3, fc_requires_grad=True)
    model3.to(device)

    dataloader = ImageDataLoader()

    trainer1 = ModelTrainer(model1, dataloader.train_loader, dataloader.
 ↪val_loader, dataloader.test_loader, dataloader.classes, epochs=2)
    trainer1.train(device)
    trainer1.validate(device)
    trainer1.plot_loss()
    trainer1.evaluate(device)

    trainer2 = ModelTrainer(model2, dataloader.train_loader, dataloader.
 ↪val_loader, dataloader.test_loader, dataloader.classes, epochs=2)
    trainer2.train(device)
    trainer2.validate(device)
    trainer2.plot_loss()
    trainer2.evaluate(device)

    trainer3 = ModelTrainer(model3, dataloader.train_loader, dataloader.
 ↪val_loader, dataloader.test_loader, dataloader.classes, epochs=2)
    trainer3.train(device)
    trainer3.validate(device)
    trainer3.plot_loss()
    trainer3.evaluate(device)
```

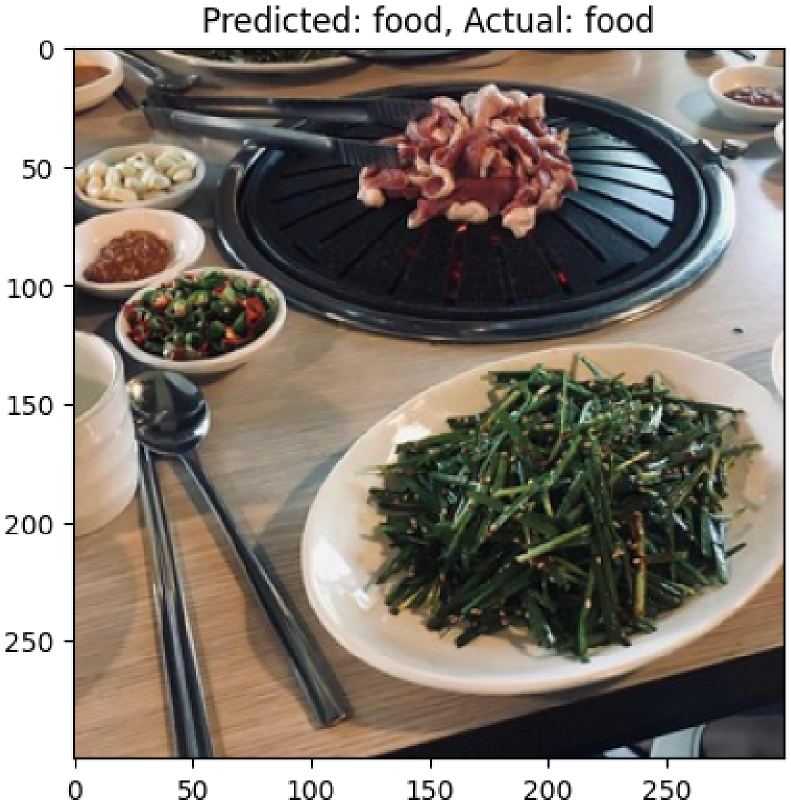
```
mps

Epoch 1/2 - Train Loss: 0.3357, Validation Loss: 0.2690

Epoch 2/2 - Train Loss: 0.2018, Validation Loss: 0.2382
```



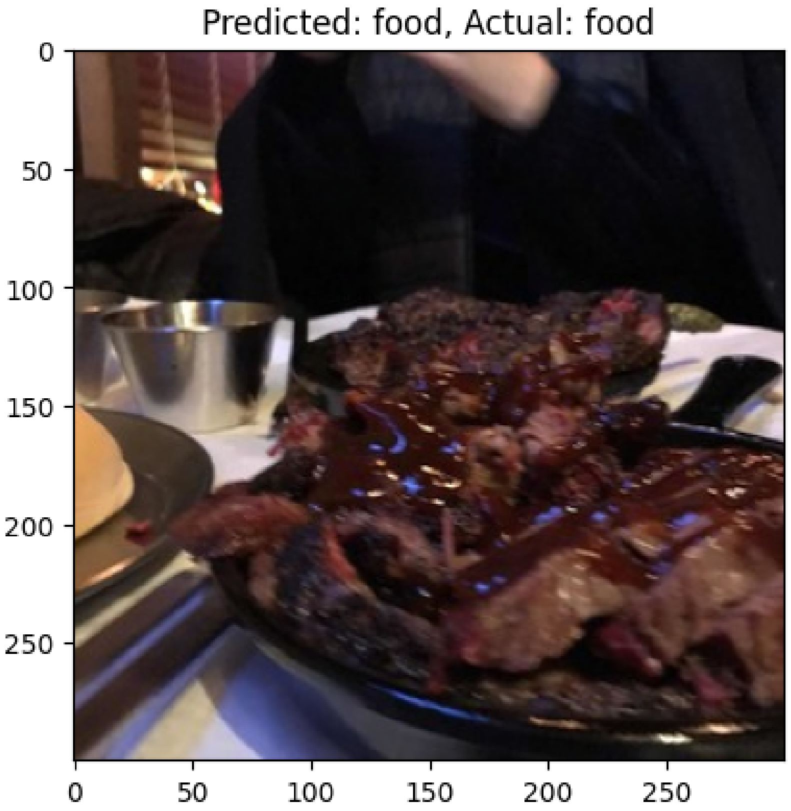
```
----- Correct Predictions -----
```



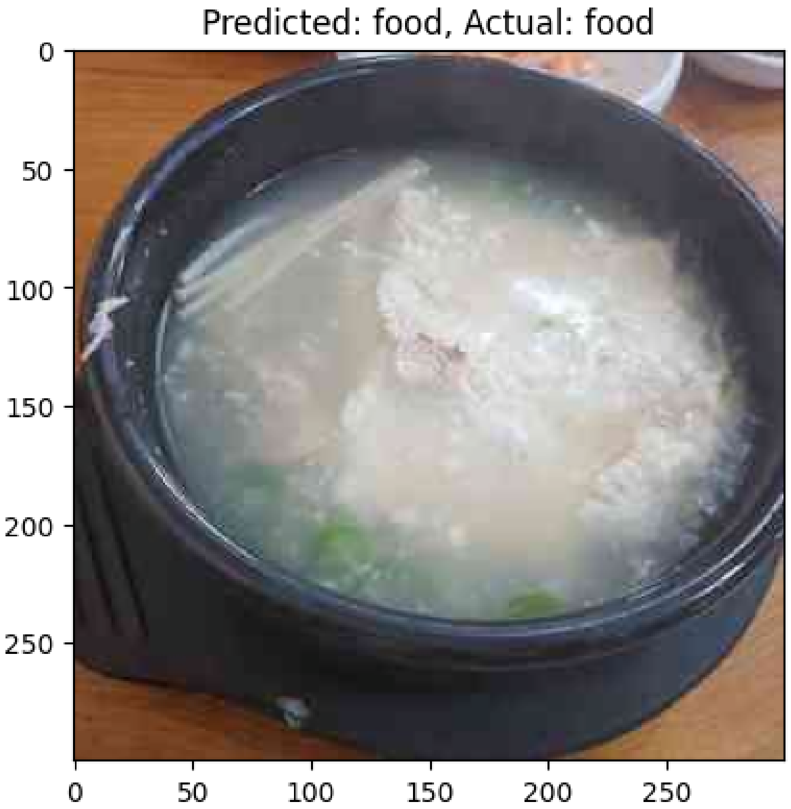
```
tensor([ 3.6561, -4.8687, -1.8997], device='mps:0')->[9.9595296e-01
1.9768013e-04 3.8493443e-03]: exterior -> exterior
```



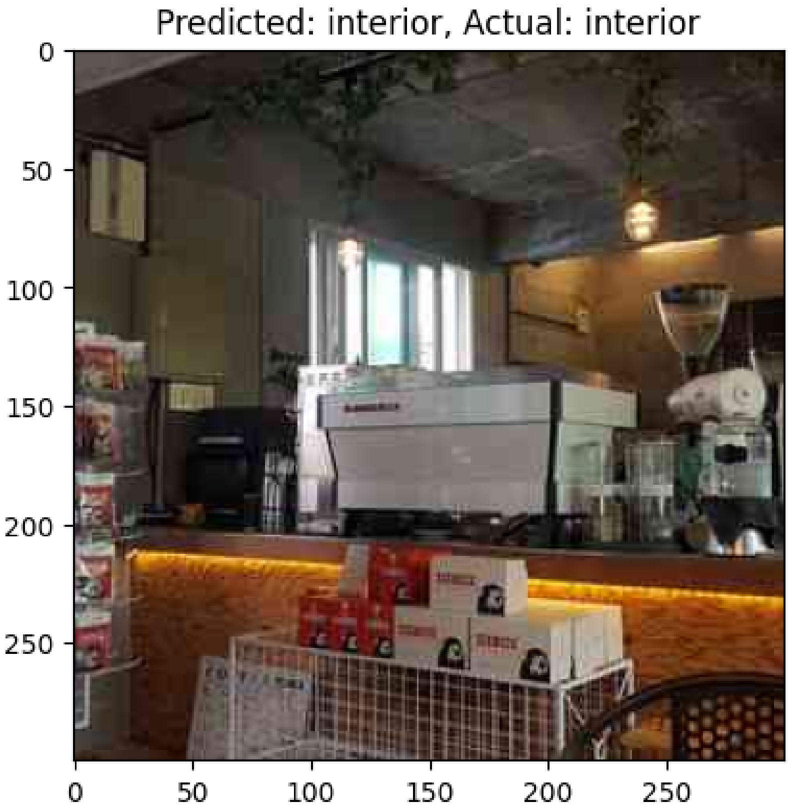
```
tensor([-4.2567,  3.5494, -1.6011], device='mps:0')->[4.0475361e-04
9.9383461e-01 5.7606036e-03]: food -> food
```



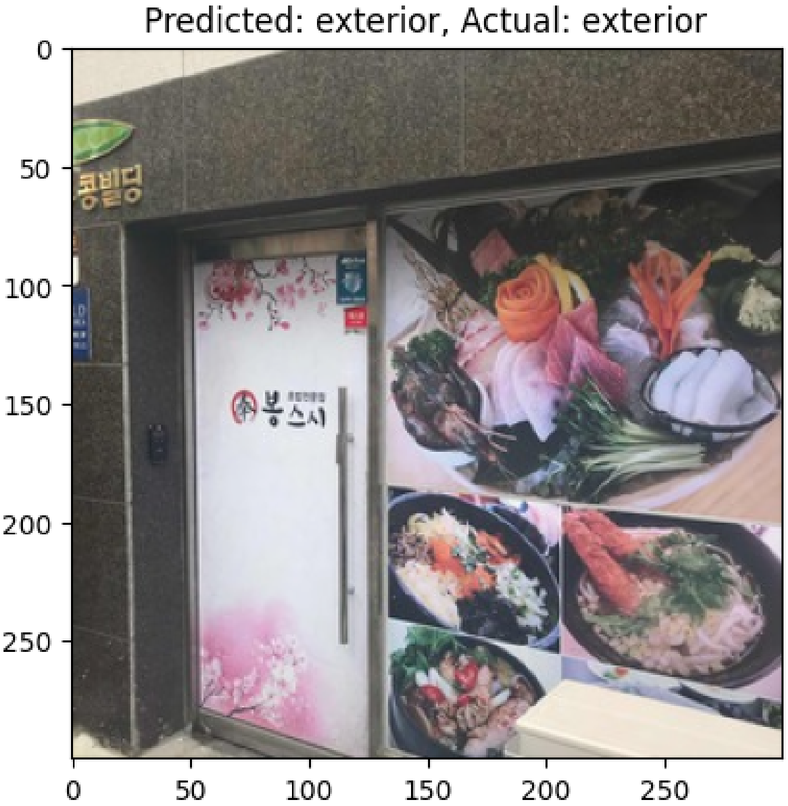
```
tensor([-4.9761,  3.8806, -1.6546], device='mps:0')->[1.4184402e-04
9.9592870e-01 3.9294548e-03]: food -> food
```



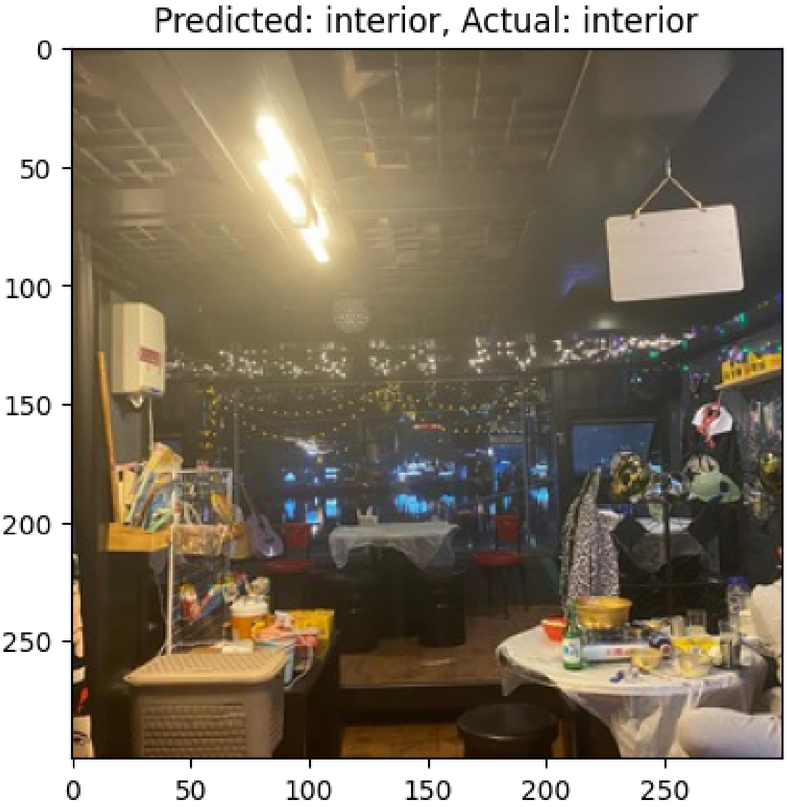
```
tensor([-4.4480,  3.1045, -1.0441], device='mps:0')->[5.1639415e-04
9.8395073e-01 1.5532908e-02]: food -> food
```



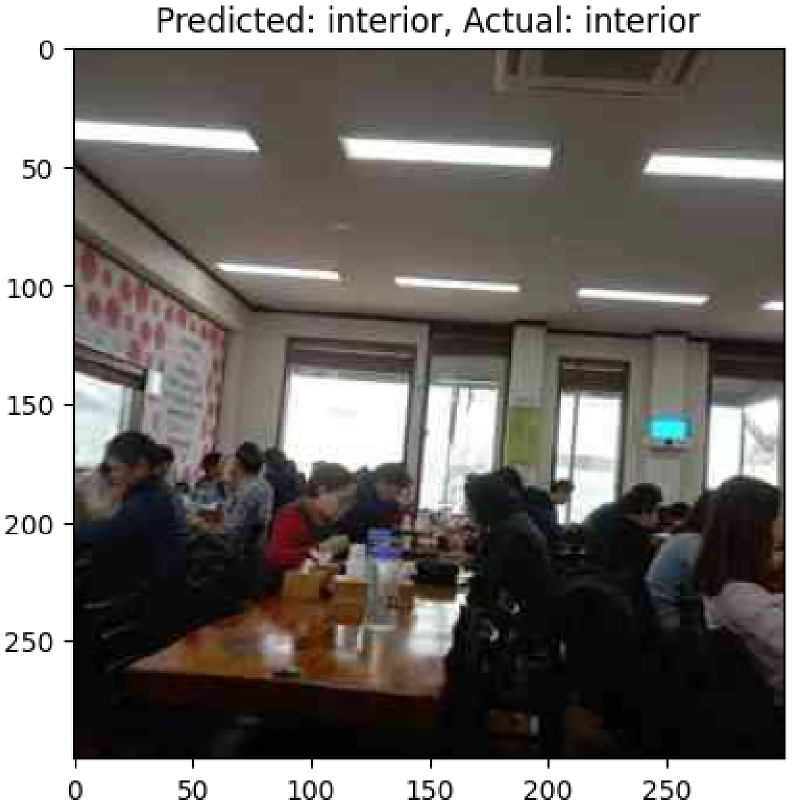
```
tensor([-1.4475, -2.1962,  1.5140], device='mps:0')->[0.04807756 0.02273929
0.9291832 ]: interior -> interior
```



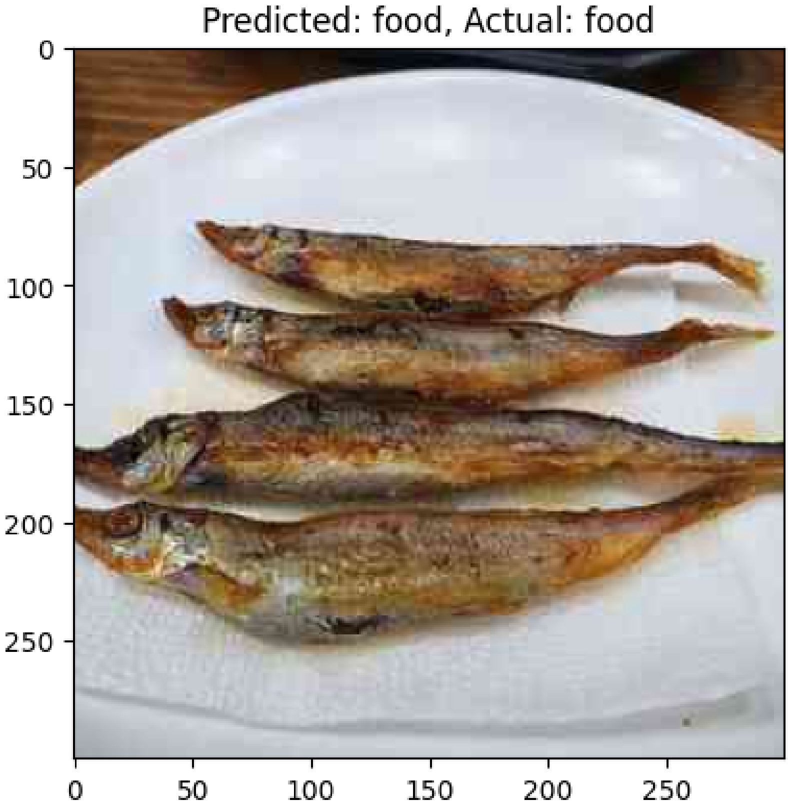
```
tensor([ 0.4371, -2.5603, -0.4303], device='mps:0')->[0.68028396 0.03395645
0.28575963]: exterior -> exterior
```



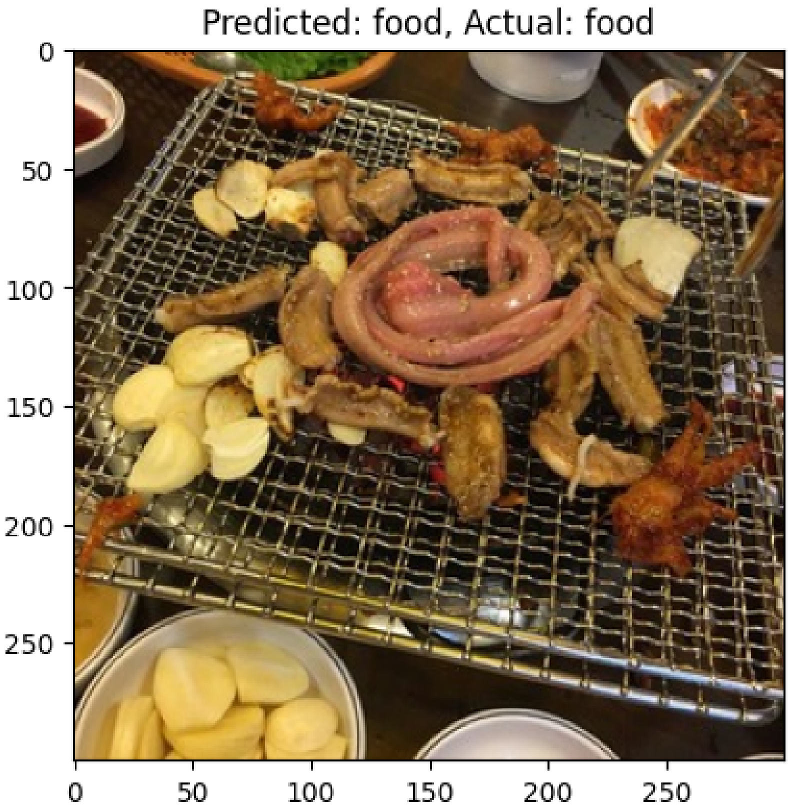
```
tensor([-2.7997, -3.7176,  3.5310], device='mps:0')->[1.7763670e-03
7.0939941e-04 9.9751425e-01]: interior -> interior
```



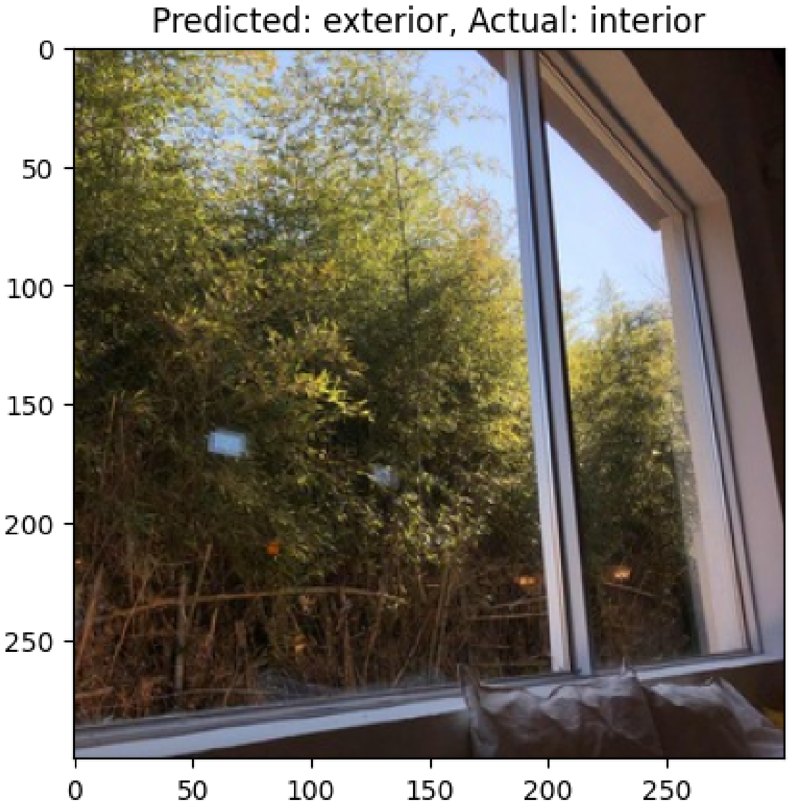
```
tensor([-1.9388, -3.2185,  2.7320], device='mps:0')->[0.00925427 0.00257381
0.9881719 ]: interior -> interior
```



```
tensor([-3.2067,  3.1991, -2.0228], device='mps:0')->[0.00164043 0.99300015
0.00535938]: food -> food
```

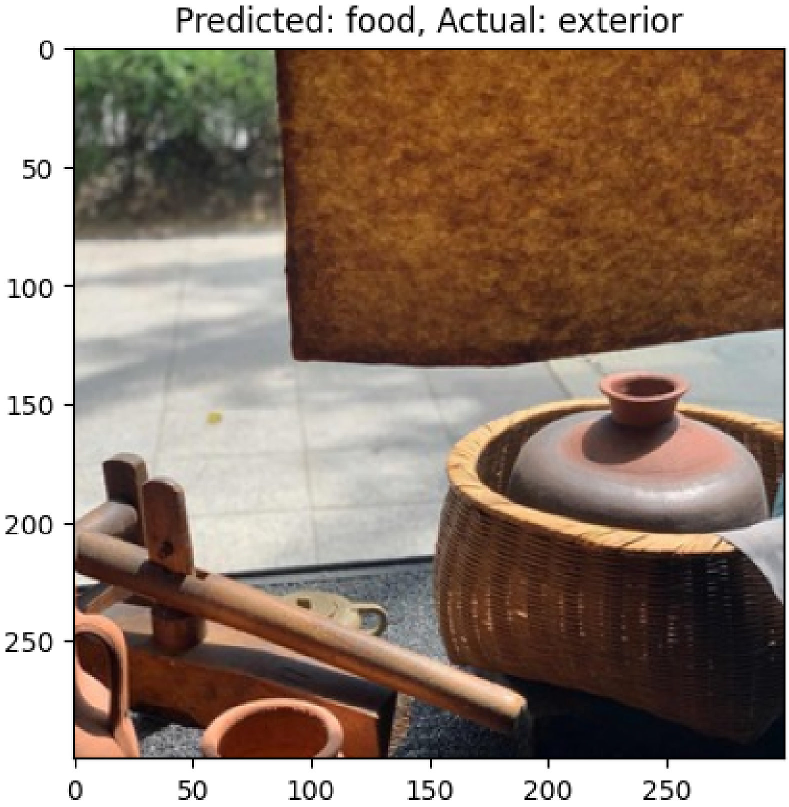


```
tensor([-4.4551,  4.3037, -2.6177], device='mps:0')->[1.5688212e-04
9.9885786e-01 9.8525907e-04]: food -> food

----- Incorrect Predictions -----
```

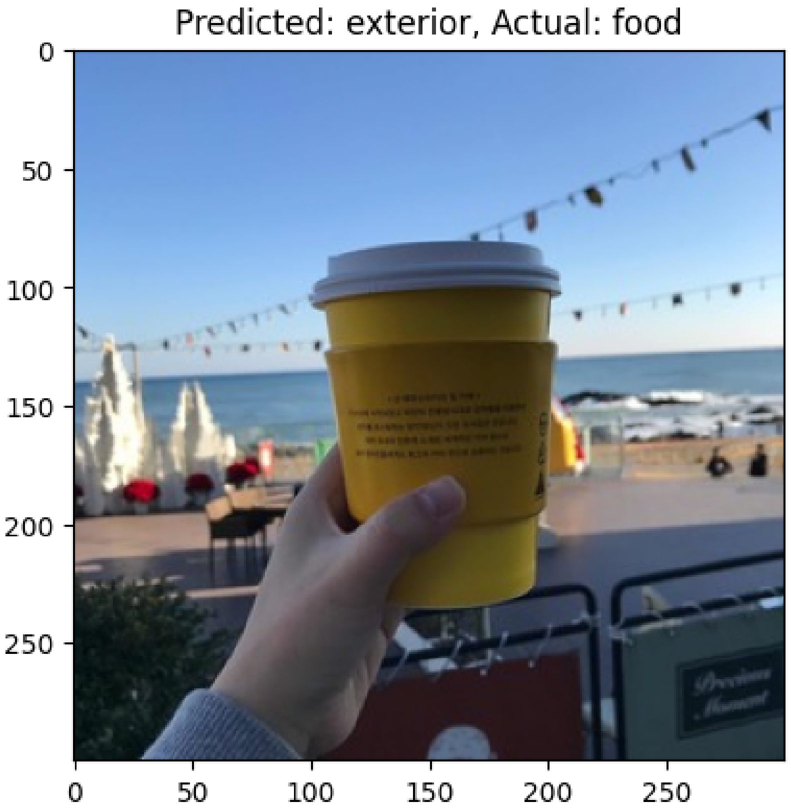


Predicted: exterior, Actual: interior

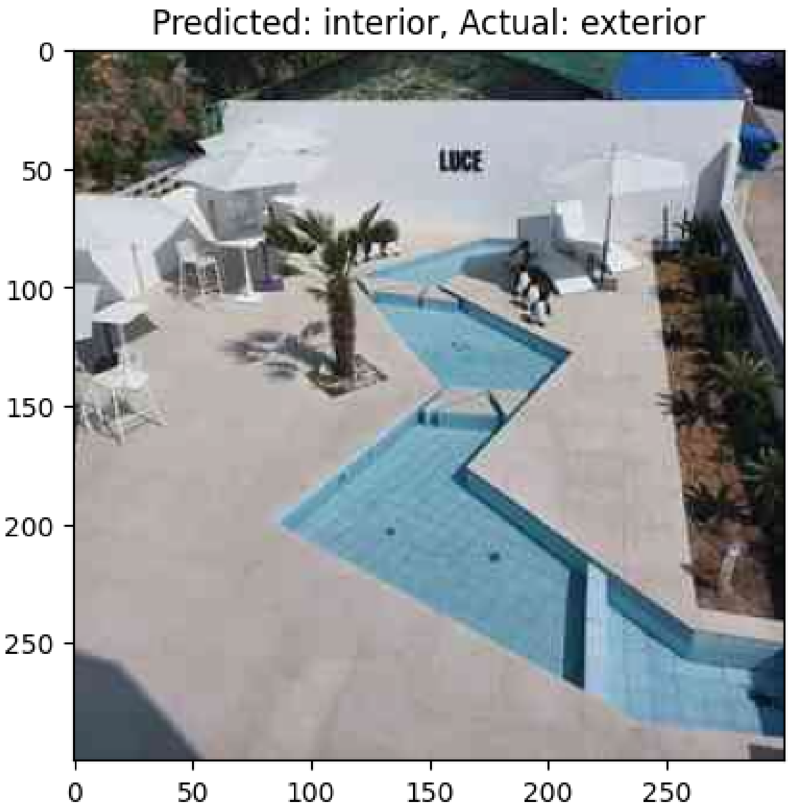
```
tensor([ 0.7594, -2.8345,  0.1924], device='mps:0')->[0.6270555  0.01723866
0.35570583]: interior -> exterior
```



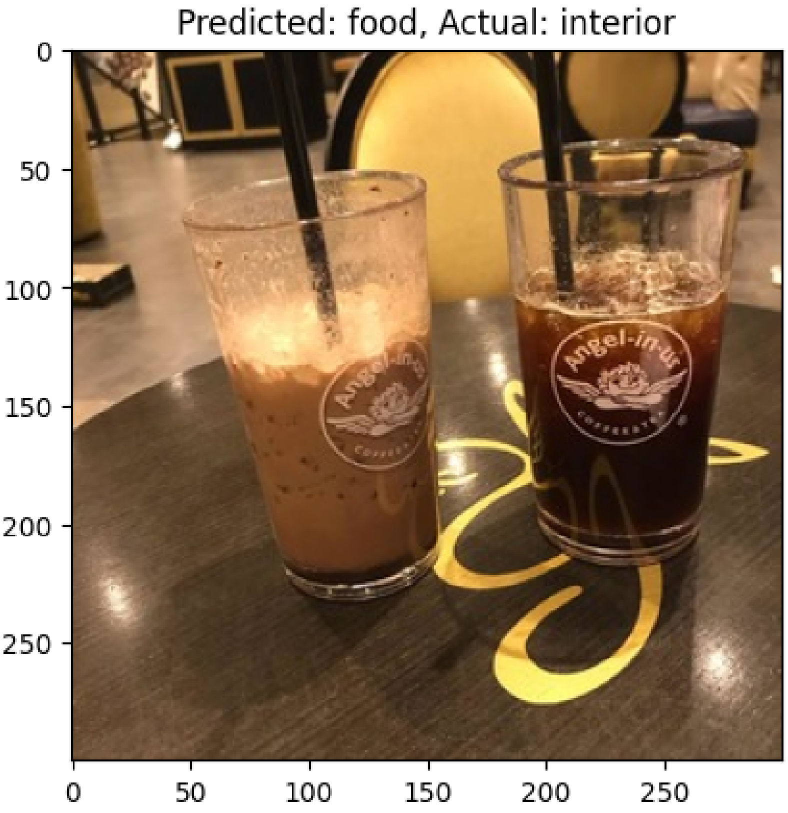
```
tensor([-2.1850,  0.6240, -0.1979], device='mps:0')->[0.04018188 0.66671926
0.29309884]: exterior -> food
```



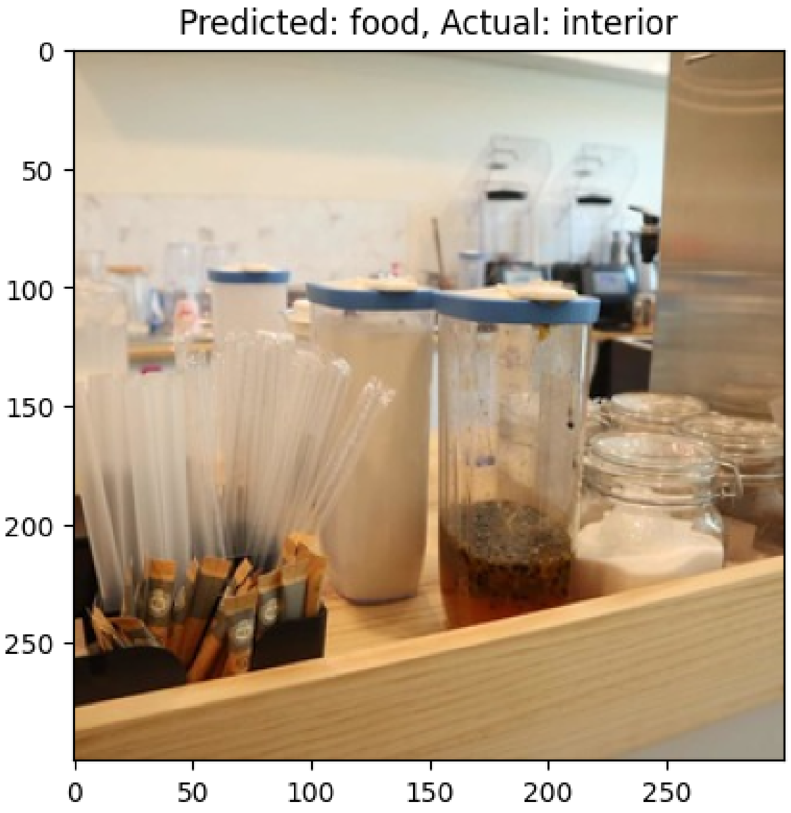
```
tensor([ 0.5220, -1.8772, -0.7506], device='mps:0')->[0.7294539  0.06622441
0.20432161]: food -> exterior
```



```
tensor([-0.1065, -2.8265,  0.3131], device='mps:0')->[0.38653475 0.02546231
0.588003  ]: exterior -> interior
```

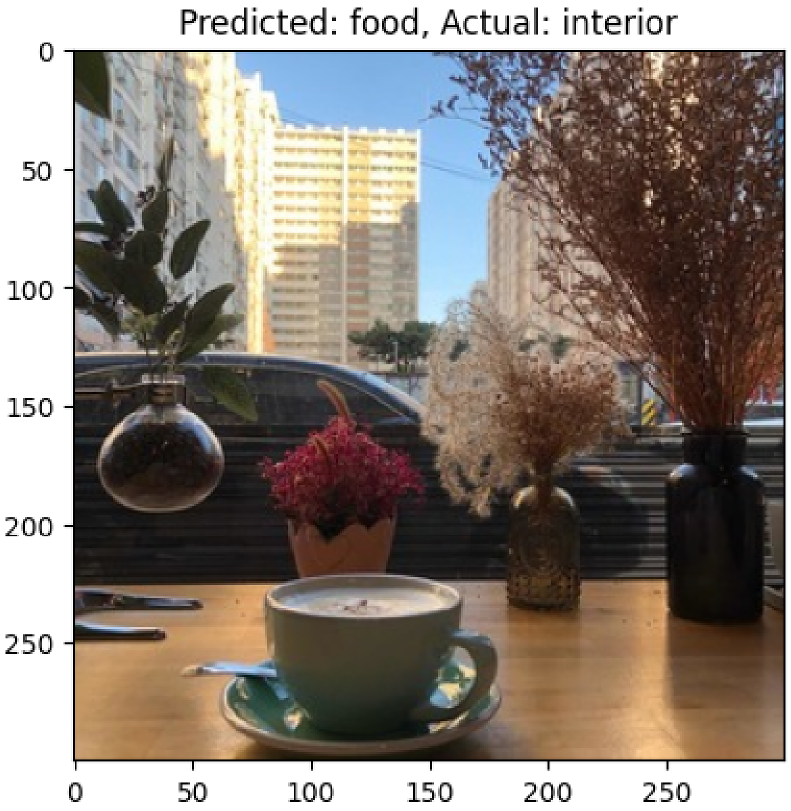


```
tensor([-2.7150,  2.4054, -1.1648], device='mps:0')->[0.00577655 0.9670046
0.02721885]: interior -> food
```

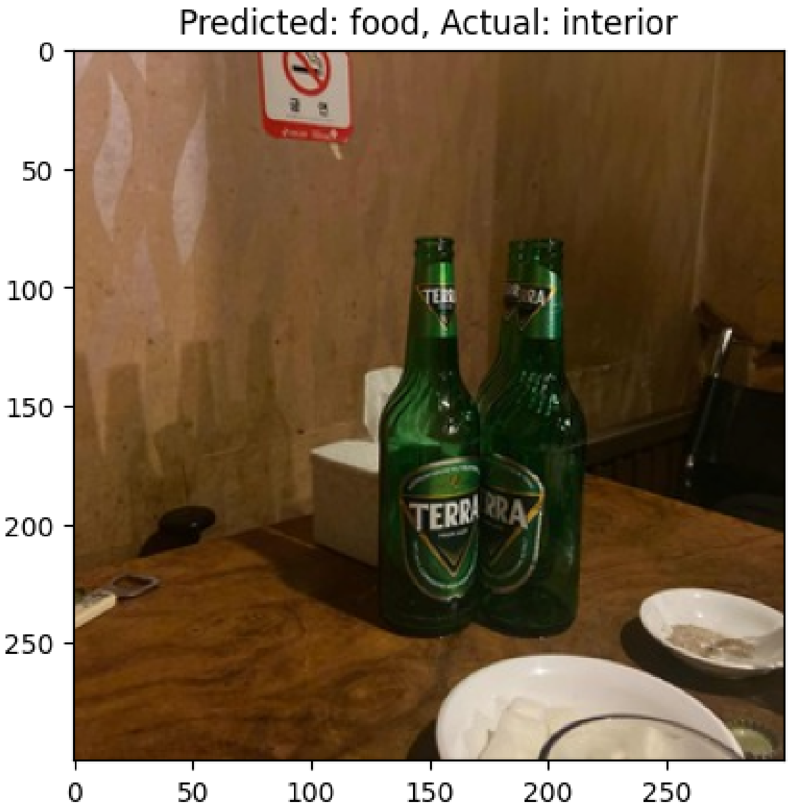


Predicted: food, Actual: interior

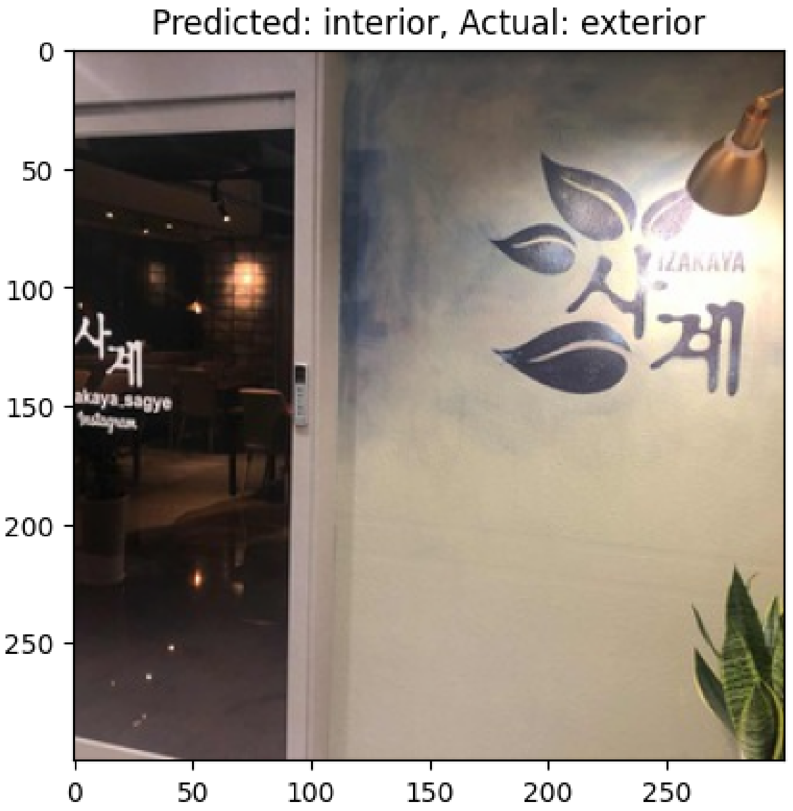
```
tensor([-4.6442,  1.2060,  0.7401], device='mps:0')->[0.00176597 0.6133261
0.38490793]: interior -> food
```



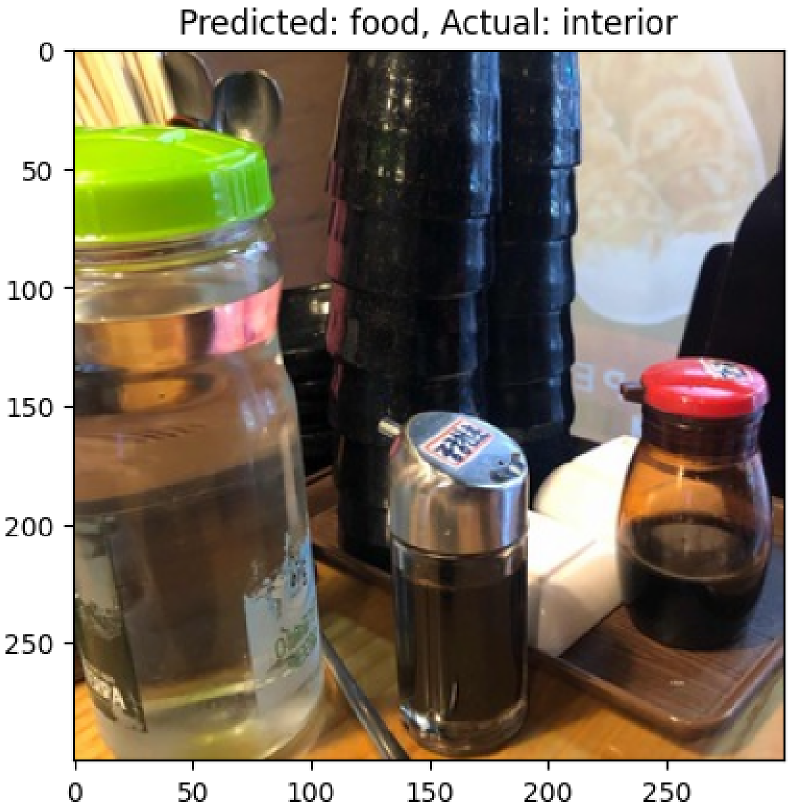
```
tensor([-2.3536,  0.4820, -0.0597], device='mps:0')->[0.03577331 0.60958475
0.35464197]: interior -> food
```



```
tensor([-3.4949,  0.7624,  0.4419], device='mps:0')->[0.0081384  0.57473814
0.41712347]: interior -> food
```



```
tensor([-0.6874, -3.2711,  1.4039], device='mps:0')->[0.10904053 0.00823209
0.8827274 ]: exterior -> interior
```



```
tensor([-3.4620,  2.3771, -0.9500], device='mps:0')->[0.00280255 0.9626429
0.0345545 ]: interior -> food
Accuracy =  0.9374
Test Loss: 0.1725
Confusion Matrix:
[[269   2  33]
 [  1 629   6]
 [ 18  30 449]]
```