# Introduction

The semantic segmentation adds class labels to each pixel to give a more detailed knowledge about the picture. The project is a fine-tuning project which trains a previously trained DeepLabV3\_ResNet50 on 300 training and valuation images of the diverse COCO 2017 dataset, exhibiting how transfer learning can enhance the accuracy of the segmentation process without using much time or computing resources.1

# Literature Review

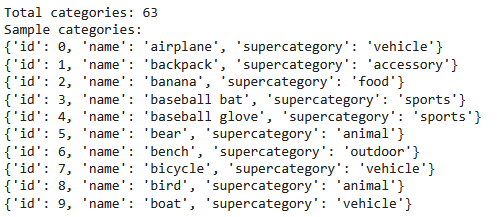
The proposed by (Zhao, Qiao and Dou, 2019) method is an interactive segmentation approach based on CNN that operates on remote sensing images, the intersection of four extreme points (top, bottom, left, right) is provided as guidance in input. The approach has 84.4% mIoU, which is 23.1% better than other methods, and it increases by 88.9% after the insertion of a fifth corrective point to the most inaccurate area. Pros are: a much higher precision (10.3% more on the MS-COCO benchmark dataset), high robustness to point perturbation (falling by only 0.9% on the metric mIoU at a perturbation factor of 20), and a great saving of annotation time to 6.7 seconds per picture, versus 80 percent as before using bounding boxes. It is about as well as 40.6 percent better with respect to mIoU compared to the available ways, such as GrabCut and RandomWalker. Moreover, the approach has good scaling with a smaller amount of training data and the cost of annotation will decrease. But on the negative side there is a limitation to select point only manually, which can lead to inconsistency and possibly there may be a limitation to deal with complex and overlapping objects. In spite of the above, the technique offers a sound balance between accuracy, effectiveness, and ease of annotations in remote sensing.

In a paper by (Wang et al., 2018) they present BIFSeg, an interactive frame based on deep learning segmentation, a framework based on CNNs and bounding boxes and optionally scribbles, to be applied to medical images. It has strong segmentation of objects that are not seen before, which resolves zero-shot learning issues. Incorporating such fine-tuning with a weighted loss increases the accuracy of the image at a significantly better level than CRF post-processing. When applied in 2D fetal MRI and 3D brain tumor segmentation, BIFSeg outperforms or even matches the performance of state-of-the-art CNNs with less user interaction and less time required. The strengths are the good generalizability, effective inference that fits the use of laptops, and better segmentation that requires little input. Nevertheless, unsupervised fine-tuning is potentially weak on complicated scenarios absent user corrections, and performance is determined by bounding box or scribbles supplied by a user. Notwithstanding the above limitations, BIFSeg manages to strike the right balance between accuracy, adaptability, and efficiency, and as such, it is suitable in a clinical interactive segmentation space.

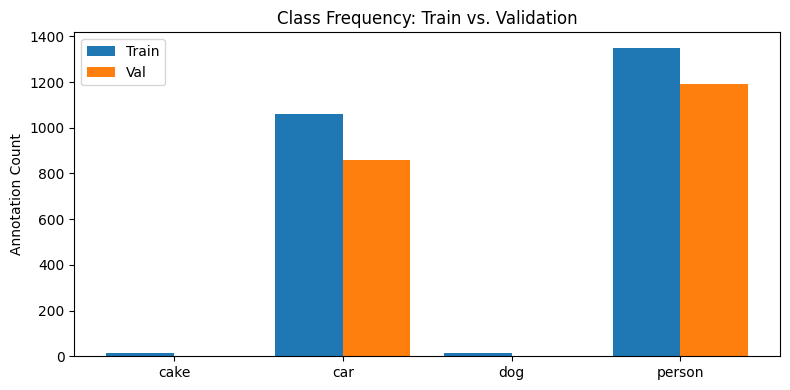
The challenge tackled by this study by (Behboodi et al., 2020) is to segmentation of ultrasound (US) images, which is a challenging problem in the presence of noise and scarce annotated data. The authors suggest the application of US natural images and simulated images as auxiliary data sets in order to pre-train a U-Net segmentation network, and then fine-tune to a relatively small set of in vivo breast US. In their experiments, using fine tuning a pre-trained network on a set of 19 real images gave about 21 percent more accurate segmentation than training a network from scratch. In addition, pre-training with simulated US images will perform better in regards to pre-training with natural images when the number of auxiliary images is equal, but will take significantly shorter times to be trained. The method uses transfer learning to surmount the data deficiency with medical US segmentation. Further study attempts to confirm this methodology on other segmentations and data sets.

# EDA

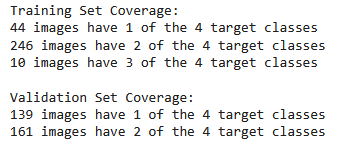
Below image shows few sample categories in the given dataset. But the task of pre training is limited to only 4 categories – dog, person, cake and car.



Below image shows that the frequency of dog and cake are very low in train and even lower in validation data. But there are a lot of car and person classes in both datasets. The model will learn better in the cases of car and person since the frequency is higher.



The spread of classes is as below



# Methodology

The steps involved in the methodology are starting with mounting Google Drive and loading the dataset JSON files. The annotations and images are sifted, taking account merely of the target classes. To gain insight into the dataset composition, data exploration is undertaken using the distributions of these categories, the number of annotations, and size of images.

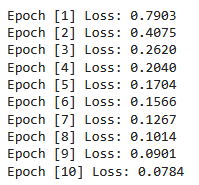
A custom dataset class is made to use images and create segmentation masks using polygon annotations. Albumentations apply such data augmentations as resizing, flipping, and normalization. Different transformations are specified on training and validation datasets.

The filtered data are packaged as PyTorch DataLoaders to be batched. An existing DeepLabV3-ResNet50 model is altered by changing its classifier head in order to predict target classes. Training of the model will be on the Adam optimizer and cross-entropy loss.

To test, one of the classes of the unlabeled dataset loads images with transformations. The end of the inference process is predictions which are created and displayed through overlapping segmentation masks on original images.

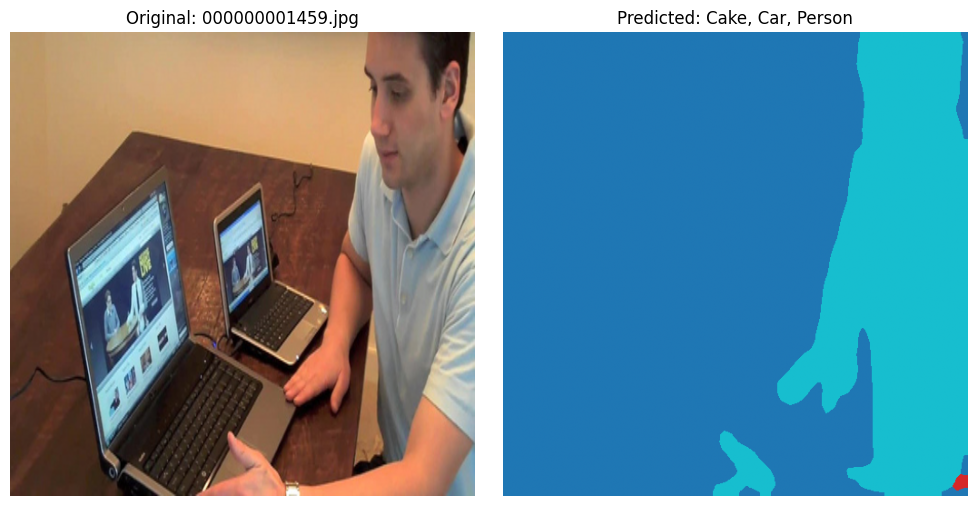
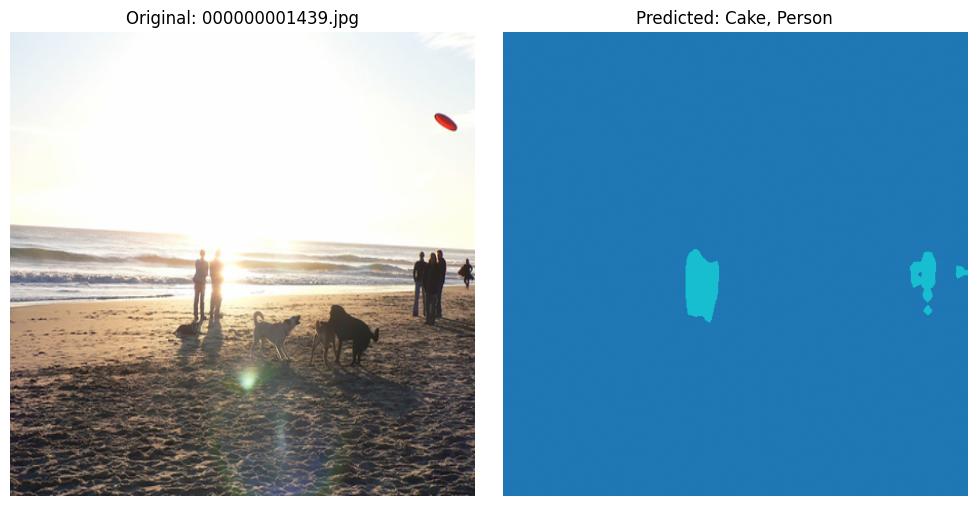
# Results

The model is trained for 10 epochs and the main metrics for measuring the performance is the loss metric. The model started with high loss and trained well with a very low loss at the end of the last epoch as shown in the below picture.



Since the model was taking a long time to train, using a greater number of epochs was not computationally feasible.

The below pictures show that the model is able to correctly identify humans and a little bit of car, but it is not able to identify dogs at all, since the distribution of dogs and cake are very low in the training data.



# References

Behboodi, B. *et al.* (2020) “Breast lesion segmentation in ultrasound images with limited annotated data,” in *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*. IEEE, pp. 1834–1837.

Wang, G. *et al.* (2018) “Interactive medical image segmentation using deep learning with image-specific fine tuning,” *IEEE transactions on medical imaging*, 37(7), pp. 1562–1573. Available at: https://doi.org/10.1109/TMI.2018.2791721.

Zhao, L., Qiao, P. and Dou, Y. (2019) “Aircraft Segmentation Based On Deep Learning framework : from extreme points to remote sensing image segmentation,” in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, pp. 1362–1366.

# Code

from google.colab import drive

drive.mount('/content/drive')

import os

folder\_path = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset'

if os.path.exists(folder\_path):

print(f"Contents of '{folder\_path}':")

for item in os.listdir(folder\_path):

print(item)

else:

print(f"Folder not found: '{folder\_path}'")

import json

train\_json\_path = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/train-300/labels.json'

val\_json\_path = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/validation-300/labels.json'

with open(train\_json\_path) as f:

train\_labels = json.load(f)

with open(val\_json\_path) as f:

val\_labels = json.load(f)

train\_categories = train\_labels['categories']

print(f"Total categories: {len(train\_categories)}")

print("Sample categories:")

for cat in train\_categories[:10]:

print(cat)

target\_class\_names = ['cake', 'car', 'dog', 'person']

id\_to\_name = {cat['id']: cat['name'].lower() for cat in train\_labels['categories']}

name\_to\_id = {v: k for k, v in id\_to\_name.items()}

target\_category\_ids = {name\_to\_id[name] for name in target\_class\_names}

print("Target category IDs:", target\_category\_ids)

target\_class\_names = ['cake', 'car', 'dog', 'person']

val\_id\_to\_name = {cat['id']: cat['name'].lower() for cat in val\_labels['categories']}

val\_name\_to\_id = {v: k for k, v in val\_id\_to\_name.items()}

val\_target\_category\_ids = {val\_name\_to\_id[name] for name in target\_class\_names}

print("Target category IDs:", val\_target\_category\_ids)

filtered\_annotations = [

ann for ann in train\_labels['annotations']

if ann['category\_id'] in target\_category\_ids

]

valid\_image\_ids = {ann['image\_id'] for ann in filtered\_annotations}

filtered\_images = [

img for img in train\_labels['images']

if img['id'] in valid\_image\_ids

]

filtered\_train\_labels = {

'info': train\_labels['info'],

'licenses': train\_labels['licenses'],

'categories': [cat for cat in train\_labels['categories'] if cat['id'] in target\_category\_ids],

'images': filtered\_images,

'annotations': filtered\_annotations

}

import os

output\_path = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/train-300-filtered.json'

with open(output\_path, 'w') as f:

json.dump(filtered\_train\_labels, f)

filtered\_val\_annotations = [

ann for ann in val\_labels['annotations']

if ann['category\_id'] in val\_target\_category\_ids

]

valid\_val\_image\_ids = {ann['image\_id'] for ann in filtered\_val\_annotations}

filtered\_val\_images = [

img for img in val\_labels['images']

if img['id'] in valid\_val\_image\_ids

]

filtered\_val\_labels = {

'info': val\_labels['info'],

'licenses': val\_labels['licenses'],

'categories': [cat for cat in val\_labels['categories'] if cat['id'] in target\_category\_ids],

'images': filtered\_val\_images,

'annotations': filtered\_val\_annotations

}

output\_val\_path = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/validation-300-filtered.json'

with open(output\_val\_path, 'w') as f:

json.dump(filtered\_val\_labels, f)

from collections import defaultdict

def analyze\_class\_coverage(annotations, categories):

id\_to\_name = {cat['id']: cat['name'].lower() for cat in categories}

image\_class\_map = defaultdict(set)

for ann in annotations:

cat\_id = ann['category\_id']

class\_name = id\_to\_name.get(cat\_id)

if class\_name:

image\_class\_map[ann['image\_id']].add(class\_name)

return image\_class\_map

def summarize\_class\_coverage(image\_class\_map):

coverage\_count = defaultdict(int)

for img\_id, class\_set in image\_class\_map.items():

coverage\_count[len(class\_set)] += 1

for i in sorted(coverage\_count):

print(f"{coverage\_count[i]} images have {i} of the 4 target classes")

train\_class\_map = analyze\_class\_coverage(filtered\_train\_labels['annotations'], filtered\_train\_labels['categories'])

print("Training Set Coverage:")

summarize\_class\_coverage(train\_class\_map)

val\_class\_map = analyze\_class\_coverage(filtered\_val\_labels['annotations'], filtered\_val\_labels['categories'])

print("\nValidation Set Coverage:")

summarize\_class\_coverage(val\_class\_map)

from collections import Counter

def get\_class\_counts(annotations, categories):

id\_to\_name = {cat['id']: cat['name'].lower() for cat in categories}

class\_counter = Counter()

for ann in annotations:

class\_name = id\_to\_name.get(ann['category\_id'])

if class\_name:

class\_counter[class\_name] += 1

return class\_counter

train\_counts = get\_class\_counts(filtered\_train\_labels['annotations'], filtered\_train\_labels['categories'])

val\_counts = get\_class\_counts(filtered\_val\_labels['annotations'], filtered\_val\_labels['categories'])

print("Train Class Frequencies:")

for cls, count in train\_counts.items():

print(f"{cls:10s}: {count}")

print("\nValidation Class Frequencies:")

for cls, count in val\_counts.items():

print(f"{cls:10s}: {count}")

import matplotlib.pyplot as plt

def plot\_class\_counts(train\_counts, val\_counts):

classes = sorted(set(train\_counts.keys()).union(val\_counts.keys()))

train\_vals = [train\_counts.get(cls, 0) for cls in classes]

val\_vals = [val\_counts.get(cls, 0) for cls in classes]

x = range(len(classes))

plt.figure(figsize=(8, 4))

plt.bar(x, train\_vals, width=0.4, label='Train', align='center')

plt.bar([i + 0.4 for i in x], val\_vals, width=0.4, label='Val', align='center')

plt.xticks([i + 0.2 for i in x], classes)

plt.ylabel('Annotation Count')

plt.title('Class Frequency: Train vs. Validation')

plt.legend()

plt.tight\_layout()

plt.show()

plot\_class\_counts(train\_counts, val\_counts)

import numpy as np

def image\_size\_stats(images):

widths = [img['width'] for img in images]

heights = [img['height'] for img in images]

print(f"Total images: {len(images)}")

print(f"Width - min: {np.min(widths)}, max: {np.max(widths)}, mean: {np.mean(widths):.1f}")

print(f"Height - min: {np.min(heights)}, max: {np.max(heights)}, mean: {np.mean(heights):.1f}")

image\_size\_stats(filtered\_train\_labels['images'])

image\_size\_stats(filtered\_val\_labels['images'])

import matplotlib.pyplot as plt

import cv2

import random

import numpy as np

from collections import defaultdict

import os

def show\_sample\_with\_mask(filtered\_labels, image\_dir):

id\_to\_name = {cat['id']: cat['name'] for cat in filtered\_labels['categories']}

image\_id\_to\_image = {img['id']: img for img in filtered\_labels['images']}

image\_id\_to\_anns = defaultdict(list)

for ann in filtered\_labels['annotations']:

image\_id\_to\_anns[ann['image\_id']].append(ann)

image\_id = random.choice(list(image\_id\_to\_anns.keys()))

image\_info = image\_id\_to\_image[image\_id]

anns = image\_id\_to\_anns[image\_id]

image\_path = os.path.join(image\_dir, image\_info['file\_name'])

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(8, 6))

plt.imshow(image)

for ann in anns:

for seg in ann['segmentation']:

seg\_np = np.array(seg).reshape(-1, 2)

plt.plot(seg\_np[:, 0], seg\_np[:, 1], linewidth=2)

class\_names = [id\_to\_name[ann['category\_id']] for ann in anns]

plt.title(f"Image ID: {image\_id} | Classes: {class\_names}")

plt.axis('off')

plt.show()

train\_image\_dir = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/train-300/data'

show\_sample\_with\_mask(filtered\_train\_labels, train\_image\_dir)

val\_image\_dir = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/validation-300/data'

show\_sample\_with\_mask(filtered\_val\_labels, val\_image\_dir)

import torch

from torch.utils.data import Dataset

import os

import json

import cv2

import numpy as np

from torchvision import transforms

class CocoSegmentationDataset(Dataset):

def \_\_init\_\_(self, data\_dir, label\_json, category\_ids, transform=None):

with open(label\_json, 'r') as f:

labels = json.load(f)

self.image\_dir = os.path.join(data\_dir, "data")

self.images = labels['images']

self.annotations = labels['annotations']

self.categories = labels['categories']

self.transform = transform

self.category\_ids = category\_ids

self.image\_id\_to\_anns = {}

for ann in self.annotations:

if ann['category\_id'] in self.category\_ids:

self.image\_id\_to\_anns.setdefault(ann['image\_id'], []).append(ann)

self.images = [img for img in self.images if img['id'] in self.image\_id\_to\_anns]

self.cat\_to\_idx = {cat\_id: idx for idx, cat\_id in enumerate(category\_ids)}

def \_\_len\_\_(self):

return len(self.images)

def \_\_getitem\_\_(self, idx):

img\_info = self.images[idx]

img\_path = os.path.join(self.image\_dir, img\_info['file\_name'])

image = cv2.imread(img\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

H, W = img\_info['height'], img\_info['width']

mask = np.zeros((H, W), dtype=np.uint8)

for ann in self.image\_id\_to\_anns[img\_info['id']]:

for seg in ann['segmentation']:

if not isinstance(seg, list) or len(seg) < 6:

continue

try:

pts = np.array(seg).reshape(-1, 2).astype(np.int32)

except ValueError:

continue

cv2.fillPoly(mask, [pts], color=self.cat\_to\_idx[ann['category\_id']])

if self.transform:

aug = self.transform(image=image, mask=mask)

image = aug['image']

mask = aug['mask']

return image, mask.long()

import albumentations as A

from albumentations.pytorch import ToTensorV2

train\_transform = A.Compose([

A.Resize(512, 512),

A.HorizontalFlip(p=0.5),

A.Normalize(mean=(0.485, 0.456, 0.406),

std=(0.229, 0.224, 0.225)),

ToTensorV2()

])

val\_transform = A.Compose([

A.Resize(512, 512),

A.Normalize(mean=(0.485, 0.456, 0.406),

std=(0.229, 0.224, 0.225)),

ToTensorV2()

])

from torch.utils.data import DataLoader

category\_ids = [cat['id'] for cat in filtered\_train\_labels['categories']] # use correct ordering

train\_dataset = CocoSegmentationDataset(

data\_dir='/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/train-300',

label\_json='/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/train-300-filtered.json',

category\_ids=category\_ids,

transform=train\_transform

)

val\_dataset = CocoSegmentationDataset(

data\_dir='/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/validation-300',

label\_json='/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/validation-300-filtered.json',

category\_ids=category\_ids,

transform=val\_transform

)

train\_loader = DataLoader(train\_dataset, batch\_size=4, shuffle=True, num\_workers=2)

val\_loader = DataLoader(val\_dataset, batch\_size=4, shuffle=False, num\_workers=2)

import torchvision

from torchvision.models.segmentation import deeplabv3\_resnet50

model = deeplabv3\_resnet50(pretrained=True)

model.classifier[4] = torch.nn.Conv2d(256, 4, kernel\_size=1) # 4 target classes

model = model.cuda()

import torch.nn as nn

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=1e-4)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

for epoch in range(10):

model.train()

running\_loss = 0

for imgs, masks in train\_loader:

imgs, masks = imgs.to(device), masks.to(device)

outputs = model(imgs)['out']

loss = criterion(outputs, masks)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f"Epoch [{epoch+1}] Loss: {running\_loss / len(train\_loader):.4f}")

from torch.utils.data import Dataset

from PIL import Image

import albumentations as A

from albumentations.pytorch import ToTensorV2

import os

class UnlabeledImageDataset(Dataset):

def \_\_init\_\_(self, image\_dir, transform=None):

self.image\_dir = os.path.join(image\_dir)

self.image\_files = sorted(os.listdir(self.image\_dir))

self.transform = transform

def \_\_len\_\_(self):

return len(self.image\_files)

def \_\_getitem\_\_(self, idx):

image\_path = os.path.join(self.image\_dir, self.image\_files[idx])

image = np.array(Image.open(image\_path).convert("RGB"))

if self.transform:

image = self.transform(image=image)['image']

return image, self.image\_files[idx]

test\_transform = A.Compose([

A.Resize(512, 512),

A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),

ToTensorV2()

])

test\_image\_dir = '/content/drive/My Drive/RM\_Segmentation\_Assignment\_dataset/test-30'

test\_dataset = UnlabeledImageDataset(image\_dir=test\_image\_dir, transform=test\_transform)

test\_loader = DataLoader(test\_dataset, batch\_size=1, shuffle=False)

def visualize\_test\_predictions(model, dataloader, device, label\_names):

model.eval()

with torch.no\_grad():

for i, (img, filename) in enumerate(dataloader):

img = img.to(device)

output = model(img)['out']

pred\_mask = torch.argmax(output.squeeze(), dim=0).cpu().numpy()

# Recover original image

img\_np = img.squeeze().cpu().permute(1, 2, 0).numpy()

img\_np = (img\_np \* [0.229, 0.224, 0.225]) + [0.485, 0.456, 0.406] # unnormalize

img\_np = np.clip(img\_np, 0, 1)

# Get unique class indices in prediction

unique\_classes = np.unique(pred\_mask)

predicted\_class\_names = [label\_names[c] for c in unique\_classes if c < len(label\_names)]

# Plot

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(img\_np)

plt.title(f"Original: {filename[0]}")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(pred\_mask, cmap="tab10", vmin=0, vmax=len(label\_names) - 1)

plt.title(f"Predicted: {', '.join(predicted\_class\_names)}")

plt.axis("off")

plt.tight\_layout()

plt.show()

if i >= 30: # Show first 5 examples

break

label\_names = ['Cake', 'Car', 'Dog', 'Person'] # in correct ID order

visualize\_test\_predictions(model, test\_loader, device, label\_names)