

Nama : Az – Zahra Chikal E

NIM : 1103213039

Kelas : TK-45-05

RESUME UAS CHAPTER 9

Optimasi model adalah proses memodifikasi model yang telah kita latih untuk membuatnya lebih efisien. Modifikasi ini sangat penting karena perangkat keras yang kita gunakan selama pelatihan dan inferensi biasanya sangat berbeda. Spesifikasi perangkat keras pada saat inferensi umumnya lebih kecil, itulah sebabnya optimasi model perlu dilakukan. Sebagai contoh, kita melatih model di GPU dengan performa tinggi, sementara proses inferensi model akan dijalankan di perangkat edge (misalnya, mikrokomputer, perangkat mobile, IoT, dll.). Tentu saja, perangkat-perangkat ini memiliki spesifikasi yang berbeda dan cenderung lebih kecil. Melakukan optimasi model sangat penting agar model kita dapat berjalan dengan lancar pada perangkat dengan spesifikasi yang lebih rendah.

EDGE TPU

```
[ ] def set_input_tensor(interpreter, input):
    input_details = interpreter.get_input_details()[0]
    tensor_index = input_details['index']
    input_tensor = interpreter.tensor(tensor_index)()
    # Inputs for the TFLite model must be uint8, so we quantize our input data.
    # NOTE: This step is necessary only because we're receiving input data from
    # ImageDataGenerator, which rescaled all image data to float [0,1]. When using
    # bitmap inputs, they're already uint8 [0,255] so this can be replaced with:
    #   input_tensor[:, :] = input
    scale, zero_point = input_details['quantization']
    input_tensor[:, :] = np.uint8(input / scale + zero_point)

def classify_image(interpreter, input):
    set_input_tensor(interpreter, input)
    interpreter.invoke()
    output_details = interpreter.get_output_details()[0]
    output = interpreter.get_tensor(output_details['index'])
    # Outputs from the TFLite model are uint8, so we dequantize the results:
    scale, zero_point = output_details['quantization']
    output = scale * (output - zero_point)
    top_1 = np.argmax(output)
    return top_1

interpreter = tf.lite.Interpreter('mobilenet_v2_1.0_224_quant.tflite')
interpreter.allocate_tensors()

# Collect all inference predictions in a list
batch_prediction = []
batch_truth = np.argmax(batch_labels, axis=1)

for i in range(len(batch_images)):
    prediction = classify_image(interpreter, batch_images[i])
    batch_prediction.append(prediction)

# Compare all predictions to the ground truth
tflite_accuracy = tf.keras.metrics.Accuracy()
tflite_accuracy(batch_prediction, batch_truth)
print("Quant TF Lite accuracy: {:.3%}".format(tflite_accuracy.result()))
```

Quant TF Lite accuracy: 96.875%

```

[ ] def set_input_tensor(interpreter, input):
    input_details = interpreter.get_input_details()[0]
    tensor_index = input_details['index']
    input_tensor = interpreter.tensor(tensor_index())[0]
    # Inputs for the TFLite model must be uint8, so we quantize our input data.
    # NOTE: This step is necessary only because we're receiving input data from
    # ImageDataGenerator, which rescaled all image data to float [0,1]. When using
    # bitmap inputs, they're already uint8 [0,255] so this can be replaced with:
    #   input_tensor[:, :] = input
    scale, zero_point = input_details['quantization']
    input_tensor[:, :] = np.uint8(input / scale + zero_point)

def classify_image(interpreter, input):
    set_input_tensor(interpreter, input)
    interpreter.invoke()
    output_details = interpreter.get_output_details()[0]
    output = interpreter.get_tensor(output_details['index'])
    # Outputs from the TFLite model are uint8, so we dequantize the results:
    scale, zero_point = output_details['quantization']
    output = scale * (output - zero_point)
    top_1 = np.argmax(output)
    return top_1

interpreter = tf.lite.Interpreter('mobilenet_v2_1.0_224_quant.tflite')
interpreter.allocate_tensors()

# Collect all inference predictions in a list
batch_prediction = []
batch_truth = np.argmax(batch_labels, axis=1)

for i in range(len(batch_images)):
    prediction = classify_image(interpreter, batch_images[i])
    batch_prediction.append(prediction)

# Compare all predictions to the ground truth
tfLite_accuracy = tf.keras.metrics.Accuracy()
tfLite_accuracy(batch_prediction, batch_truth)
print("Quant TF Lite accuracy: {:.3%}".format(tfLite_accuracy.result()))

```

Quant TF Lite accuracy: 93.750%

OPENVINO

```

for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

MODEL_DIR = pathlib.Path("./models")
MODEL_DIR.mkdir(exist_ok=True)
torch.save(model.state_dict(), MODEL_DIR / "original_model.p")

```

Train Epoch: 1 [0/60000 (0%)] Loss: 2.365148
 Train Epoch: 1 [32/60000 (0%)] Loss: 2.288796
 Train Epoch: 1 [64/60000 (0%)] Loss: 2.128072
 Train Epoch: 1 [96/60000 (0%)] Loss: 1.932519
 Train Epoch: 1 [128/60000 (0%)] Loss: 2.018445
 Train Epoch: 1 [160/60000 (0%)] Loss: 2.027848
 Train Epoch: 1 [192/60000 (0%)] Loss: 1.444902
 Train Epoch: 1 [224/60000 (0%)] Loss: 1.683165
 Train Epoch: 1 [256/60000 (0%)] Loss: 1.679698
 Train Epoch: 1 [288/60000 (0%)] Loss: 1.257255
 Train Epoch: 1 [320/60000 (1%)] Loss: 1.346398
 Train Epoch: 1 [352/60000 (1%)] Loss: 1.146433
 Train Epoch: 1 [384/60000 (1%)] Loss: 1.201220
 Train Epoch: 1 [416/60000 (1%)] Loss: 1.000956

```

[ ]  def test_ov(model, data_loader):
      compiled_model = ov.compile_model(model)
      test_loss = 0
      correct = 0
      for data, target in data_loader:
          output = torch.tensor(compiled_model(data)[0])
          test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
          pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
          correct += pred.eq(target.view_as(pred)).sum().item()

      test_loss /= len(data_loader.dataset)

      return 100. * correct / len(data_loader.dataset)

acc = test_ov(ov_model, test_loader)
print(f"Accuracy of original model: {acc}")

qacc = test_ov(quantized_model, test_loader)
print(f"Accuracy of quantized model: {qacc}")

→ Accuracy of original model: 96.55
Accuracy of quantized model: 96.65

```

ONNX

```

[ ]  model.train()

for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

    MODEL_DIR = pathlib.Path("./onnx_models")
    MODEL_DIR.mkdir(exist_ok=True)
    torch.save(model.state_dict(), MODEL_DIR / "original_model.p")

→ Train Epoch: 1 [58144/60000 (97%)]      Loss: 0.004201
    Train Epoch: 1 [58176/60000 (97%)]      Loss: 0.013117
    Train Epoch: 1 [58208/60000 (97%)]      Loss: 0.002426
    Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.067058
    Train Epoch: 1 [58272/60000 (97%)]      Loss: 0.014984
    Train Epoch: 1 [58304/60000 (97%)]      Loss: 0.046276
    Train Epoch: 1 [58336/60000 (97%)]      Loss: 0.027480
    Train Epoch: 1 [58368/60000 (97%)]      Loss: 0.110185

```

```

[ ] def test_onnx(model_name, data_loader):
    onnx_model = onnx.load(model_name)
    onnx.checker.check_model(onnx_model)
    ort_session = onnxruntime.InferenceSession(model_name)
    test_loss = 0
    correct = 0
    for data, target in data_loader:
        ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(data)}
        output = ort_session.run(None, ort_inputs)[0]
        output = torch.from_numpy(output)
        test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
        pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
        correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(data_loader.dataset)

    return 100. * correct / len(data_loader.dataset)

acc = test_onnx(MODEL_DIR / "mnist_model.onnx", test_loader)
print(f"Accuracy of the original model is {acc}%")

qacc = test_onnx(MODEL_DIR / "mnist_model_quant.onnx", test_loader)
print(f"Accuracy of the quantized model is {qacc}%")

[ ] static_qacc = test_onnx(model_static_quant, test_loader)
print(f"Accuracy of the static quantized model is {static_qacc}%")

[ ] Accuracy of the original model is 96.47%
Accuracy of the quantized model is 96.53%

[ ] Accuracy of the static quantized model is 96.51%

```

TORCH

```

[ ]     def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(12 * 13 * 13, 10)

    def forward(self, x):
        x = x.reshape(-1, 1, 28, 28)
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc(x)
        output = F.log_softmax(x, dim=1)
        return output

train_loader = torch.utils.data.DataLoader(train_dataset, 32)
test_loader = torch.utils.data.DataLoader(test_dataset, 32)

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

epochs = 1

model = Net().to(device)
optimizer = optim.Adam(model.parameters())

model.train()

for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

```

[] Train Epoch: 1 [58144/60000 (97%)] Loss: 0.005648
[] Train Epoch: 1 [58176/60000 (97%)] Loss: 0.014392
[] Train Epoch: 1 [58208/60000 (97%)] Loss: 0.005538

```

[ ] def test(model, device, data_loader, quantized=False):
    model.to(device)
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in data_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(data_loader.dataset)

    return 100. * correct / len(data_loader.dataset)

original_acc = test(model, "cpu", test_loader)
quantized_acc = test(quantized_model, "cpu", test_loader)

print('Original model accuracy: {:.0f}%'.format(original_acc))
print('Quantized model accuracy: {:.0f}%'.format(quantized_acc))

→ Original model accuracy: 97%
Quantized model accuracy: 96%

[ ] quantized_acc = test(model_int8, "cpu", test_loader, quantized=True)
print('Post quantized model accuracy: {:.0f}%'.format(quantized_acc))

→ Post quantized model accuracy: 96%

```

TMO

```

[ ] # Run predictions on every image in the "test" dataset.
prediction_digits = []
for test_image in test_images:
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the digit with highest
    # probability.
    output = interpreter.tensor(output_index)
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

    # Compare prediction results with ground truth labels to calculate accuracy.
    accurate_count = 0
    for index in range(len(prediction_digits)):
        if prediction_digits[index] == test_labels[index]:
            accurate_count += 1
    accuracy = accurate_count * 1.0 / len(prediction_digits)

    return accuracy

interpreter = tf.lite.Interpreter(model_path=str(tflite_model_file))
interpreter.allocate_tensors()
print("Original model accuracy = ", evaluate_model(interpreter))

interpreter_quant = tf.lite.Interpreter(model_path=str(tflite_model_quant_file))
interpreter_quant.allocate_tensors()
print("Quantized model accuracy = ", evaluate_model(interpreter_quant))

→ Original model accuracy = 0.96
Quantized model accuracy = 0.9659

```

```
[ ] _, baseline_model_accuracy = model.evaluate(
    test_images, test_labels, verbose=0)
_, model_for_pruning_accuracy = model_for_pruning.evaluate(
    test_images, test_labels, verbose=0)

print('Baseline test accuracy:', baseline_model_accuracy)
print('Pruned test accuracy:', model_for_pruning_accuracy)

→ Baseline test accuracy: 0.965399980545044
Pruned test accuracy: 0.965399980545044
```

OPTIMUM

```
[ ] def infer_ImageNet(classification_model, processor, image):
    inputs = processor(images=image, return_tensors="pt")
    outputs = classification_model(**inputs)
    logits = outputs.logits
    predicted_class_idx = logits.argmax(-1).item()
    return classification_model.config.id2label[predicted_class_idx]

# Get sample image
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)

res = infer_ImageNet(model, preprocessor, image)
print("Original model prediction:", res)

quantized_model = ORTModelForImageClassification.from_pretrained(model_quantized_path)
dq_res = infer_ImageNet(quantized_model, preprocessor, image)
print("Quantized model prediction:", dq_res)
display(image)
```

→ Original model prediction: Egyptian cat
 Quantized model prediction: Egyptian cat



```
[ ]     def preprocess_fn(ex, processor):
[ ]         return processor(ex["image"])

calibration_dataset = quantizer.get_calibration_dataset(
    "zh-plus/tiny-imagenet",
    preprocess_function=partial(preprocess_fn, processor=preprocessor),
    num_samples=50,
    dataset_split="train",
)

# Create the calibration configuration containing the parameters related to calibration.
calibration_config = AutoCalibrationConfig.minmax(calibration_dataset)

# Perform the calibration step: computes the activations quantization ranges
ranges = quantizer.fit(
    dataset=calibration_dataset,
    calibration_config=calibration_config,
    operators_to_quantize=static_qconfig.operators_to_quantize,
)

# Apply static quantization on the model
model_quantized_path_static = quantizer.quantize(
    save_dir="models/vit-base-patch16-224-quantized-static",
    calibration_tensors_range=ranges,
    quantization_config=static_qconfig,
)

```

→ README.md: 100% [██████████] 3.90k/3.90k [00:00<00:00, 226kB/s]

dataset_infos.json: 100% [██████████] 3.52k/3.52k [00:00<00:00, 194kB/s]

(...)00000-of-00001- [██████████] 146M/146M [00:00<00:00, 199MB/s]

1359597a978bc4fa.parquet: 100%

(...)00000-of-00001- [██████████] 14.6M/14.6M [00:00<00:00, 116MB/s]

70d52db3c749a935.parquet: 100%

Generating train split: 100% [██████████] 100000/100000 [00:05<00:00, 61991.93 examples/s]

Generating valid split: 100% [██████████] 10000/10000 [00:00<00:00, 119011.32 examples/s]

Map: 100% [██████████] 50/50 [00:00<00:00, 153.71 examples/s]

```
Fast image processor class <class 'transformers.models.vit.image_processing_vit.FastViTImageProcessor'>
```

```
[ ]     static_quantized_model = ORTModelForImageClassification.from_pretrained(model_quantized_path_
sq_res = infer_ImageNet(static_quantized_model, preprocessor, image)
print("Quantized model prediction (static):", sq_res)
```

→ Quantized model prediction (static): Egyptian cat

RESUME UAS CHAPTER 10

Pernahkah Anda kesulitan mencari data yang dibutuhkan untuk menyelesaikan masalah, baik itu masalah terkait pembelajaran mesin atau pengembangan lainnya? Entah karena datanya tidak tersedia, bersifat tertutup, atau terlalu mahal dan memakan waktu untuk didapatkan. Lalu, bagaimana cara mengatasinya? Salah satu solusinya adalah dengan menggunakan data sintetis. Data sintetis adalah data yang dihasilkan oleh sebuah model dan bisa digunakan menggantikan data asli atau dikombinasikan dengan data asli. Model yang dimaksud bukan hanya model pembelajaran mesin atau pembelajaran mendalam saja, tetapi juga bisa berupa model matematika atau statistik sederhana, seperti persamaan diferensial yang menggambarkan sistem fisik atau ekonomi.

Menurut definisi Royal Society, data sintetis adalah data yang dihasilkan menggunakan model matematika atau algoritma yang dirancang khusus untuk menyelesaikan tugas tertentu dalam ilmu data. Perlu dicatat bahwa data sintetis hanya meniru data asli dan tidak berasal dari kejadian nyata. Idealnya, data sintetis harus memiliki sifat statistik yang mirip dengan data asli yang ingin digantikan. Data sintetis bisa sangat berguna, misalnya untuk meningkatkan model AI, melindungi data sensitif, dan mengurangi bias dalam data.

Image_labeling_BLIP_2

```
[ ] # URL for the dataset images
url = "https://huggingface.co/datasets/hf-vision/course-assets/resolve/main/label_dataset_owlv2"

# Number of images to process
num_images = 10

# Prompt for image labeling
prompt = "This is a photo of"

# Function to process and label an image
def add_images(idx):
    # Load processor and model
    processor = AutoProcessor.from_pretrained("Salesforce/blip2-opt-2.7b")
    model = Blip2ForConditionalGeneration.from_pretrained(
        "Salesforce/blip2-opt-2.7b", device_map="auto", load_in_8bit=True
    )

    # Get the image URL
    image_url = f"{url}/{idx}.jpeg"

    # Download the image
    response = requests.get(image_url)
    image_content = response.content
    image = Image.open(BytesIO(image_content))

    # Prepare inputs for the model
    inputs = processor(image, text=prompt, return_tensors="pt").to(
        device, torch.float16
    )

    # Generate labels
    generated_ids = model.generate(**inputs, max_new_tokens=20)
    label = processor.batch_decode(generated_ids, skip_special_tokens=True)[0].strip()

    return {
        "image": image,
        "label": label,
    }
```

```
[ ] from datasets import Dataset

username = "Hazelnut27"
repo_id = "labeled_images_demo_BLIP2"

# Create a dataset from the generator
ds = Dataset.from_generator(generate_entries)
ds.push_to_hub(f"{username}/{repo_id}")

→ Uploading the dataset shards: 100% [██████████] 1/1 [00:00<00:00,
Map: 100% [██████████] 10/10 [00:00<00:00, 352.87 examples/s]

Creating parquet from Arrow format: 100% [██████████] 1/1 [00:00<00:00
CommitInfo(commit_url='https://huggingface.co/datasets/Hazelnut27/labeled_images_demo',
commit_message='Upload dataset', commit_description='',
oid='deb350555ec4c3b7ea59ce3a1719f2bef6145656', pr_url=None,
repo_url=RepoUrl('https://huggingface.co/datasets/Hazelnut27/labeled_images_demo_BLIP2'),
endpoint='https://huggingface.co', repo_type='dataset',
repo_id='Hazelnut27/labeled_images_demo_BLIP2'), pr_revision=None, pr_num=None)

[ ] from datasets import load_dataset

# Load the dataset
dataset = load_dataset("kfahn/labeled_images_demo_BLIP2")

→ README.md: 100% [██████████] 307/307 [00:00<00:00, 15.3kB/s]
train-00000-of-00001.parquet: 100%
Generating train split: 100% [██████████] 10/10 [00:00<00:00, 177.92 examples/s]
```

Let's take a look at one of the images.

```
[ ] dataset["train"]["image"][0]
```



Now let's take a look at the label: it is 'a conference room with a table and chairs.' The model provided an accurate label!

```
[ ] dataset["train"]["label"][0]

→ 'a conference room with chairs and a table'
```

Blenderproc_examples

```
[ ] import os

os.environ["LD_PRELOAD"] = ""

!apt remove libtcmalloc-minimal4
!apt install libtcmalloc-minimal4

os.environ["LD_PRELOAD"] = "/usr/lib/x86_64-linux-gnu/libtcmalloc_minimal.so.4.5.9"

[ ] /sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libumf.so.0 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link
/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a symbolic link

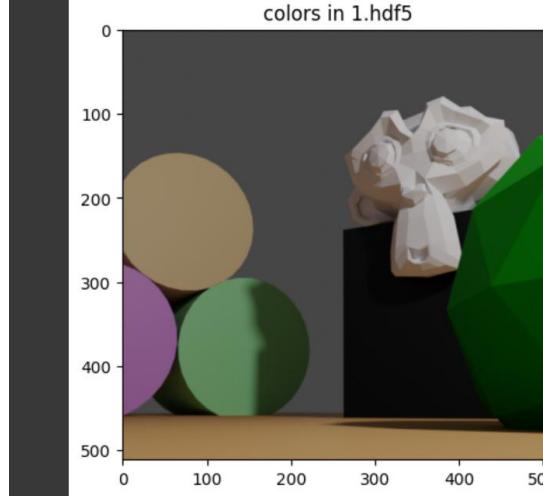
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  libtcmalloc-minimal4
0 upgraded, 1 newly installed, 0 to remove and 49 not upgraded.
Need to get 98.2 kB of archives.
After this operation, 382 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 libtcmalloc-minimal4 amd64 2.9.1-0ubu

[ ] !blenderproc run examples/basics/basic/main.py examples/resources/camera_positions examples/resources/lighting

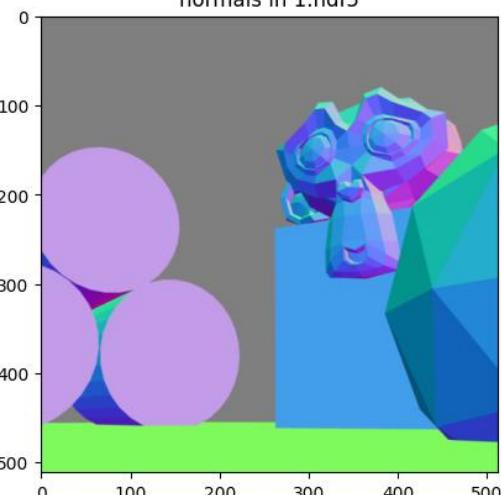
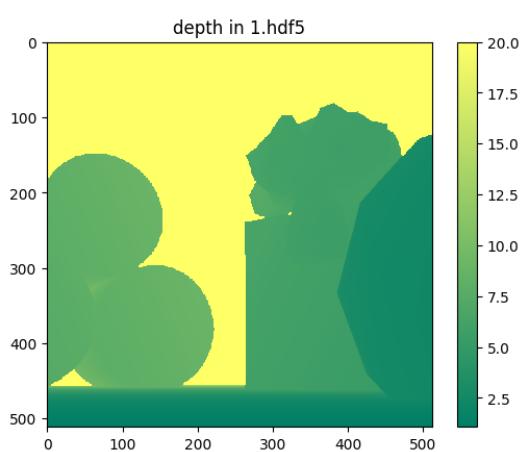
[ ] Using temporary directory: /dev/shm/blender_proc_567b52b91abc4992a5921e323b2e000a
Using blender in ./blender-4.2.1-linux-x64
Blender 4.2.1 LTS (hash 396f546c9d82 built 2024-08-19 23:32:23)
Installing pip package wheel None
Installing pip package wheel None
Collecting wheel
  Downloading wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
  Downloading wheel-0.45.1-py3-none-any.whl (72 kB)
  Installing collected packages: wheel
    WARNING: The script wheel is installed in '/content/BlenderProc/blender-4.2.1-linux-x64/custom_scripts'
    Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location
Successfully installed wheel-0.45.1
Installing pip package pyyaml 6.0.1
Collecting pyyaml==6.0.1
  Downloading PyYAML-6.0.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.9 kB)
  Downloading PyYAML-6.0.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (757 kB)
    757.7/757.7 kB 26.1 MB/s eta 0:00:00
  Installing collected packages: pyyaml
  Successfully installed pyyaml-6.0.1
  Installing pip package imageio 2.34.1
Collecting imageio==2.34.1
  Downloading imageio-2.34.1-py3-none-any.whl.metadata (4.9 kB)
  Requirement already satisfied: numpy in ./blender-4.2.1-linux-x64/4.2/python/lib/python3.11/
  Collecting pillow>=8.3.2 (from imageio==2.34.1)
    Downloading pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (9.1 kB)
  Downloading imageio-2.34.1-py3-none-any.whl (313 kB)
  Downloading pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl (4.5 MB)
    4.5/4.5 MB 96.1 MB/s eta 0:00:00
  Installing collected packages: pillow, imageio
  WARNING: The scripts imageio_download_bin and imageio_remove_bin are installed in '/content/BlenderProc/blender-4.2.1-linux-x64/custom_scripts'
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location
  Successfully installed imageio-2.34.1 pillow-11.1.0
  Installing pip package gitpython 3.1.43
```

```
[ ] # Visualize the generated data (1.hdf5)
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/basic/output/1.hdf5"
```

```
→ examples/basics/basic/output/1.hdf5:
Keys: 'colors': (512, 512, 3), 'depth': (512, 512), 'normals': (512, 512, 3)
```



normals in 1.hdf5



```
[ ] !blenderproc run examples/basics/camera_object_pose/main.py examples/basics/camera_object_pose
```

```
→ Using temporary directory: /dev/shm/blender_proc_f83feb2be7e34a1a97732da53b333ee6
Using blender in ./blender-4.2.1-linux-x64
Blender 4.2.1 LTS (hash 396f546c9d82 built 2024-08-19 23:32:23)
```

```
Selecting render devices...
```

```
Device Tesla T4 of type OPTIX found and used.
```

```
Device Intel Xeon CPU @ 2.20GHz of type CPU found and used.
```

```
PLY import of 'obj_000004.ply' took 305.13 ms
```

```
Rendering 1 frames of colors, depth...
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

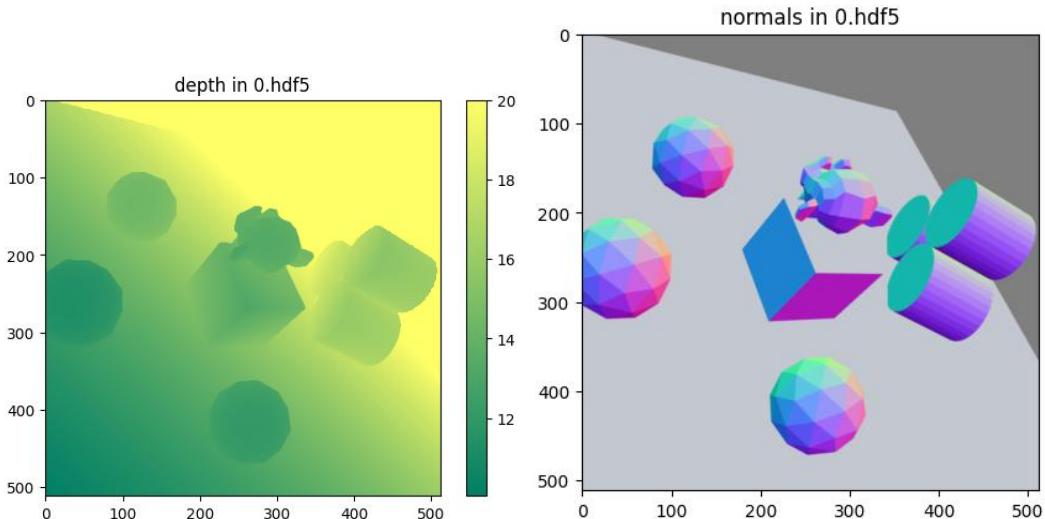
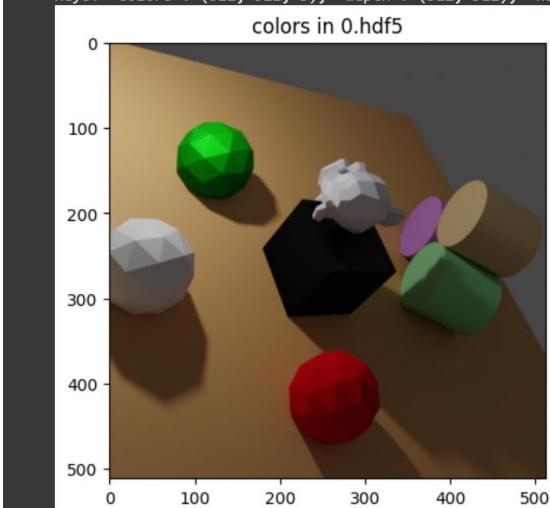
```
Total -----
```

```
          ----- Rendering frame 1 of 1
```

```
[ ] !blenderproc run examples/basics/camera_sampling/main.py examples/resources/scene.obj examples  
↳ Total 0:00:09 Rendering frame 5 of 5  
↳ Total 0:00:09 Rendering frame 5 of 5
```

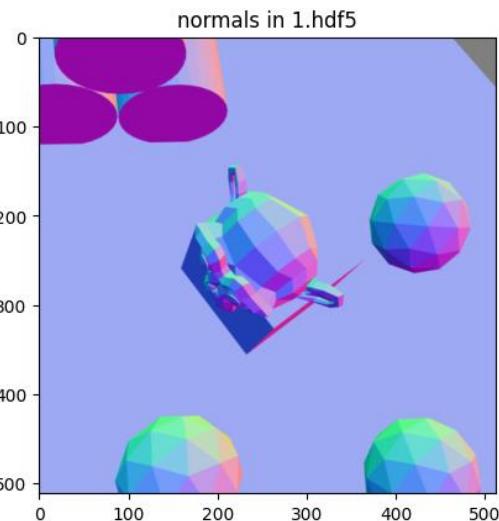
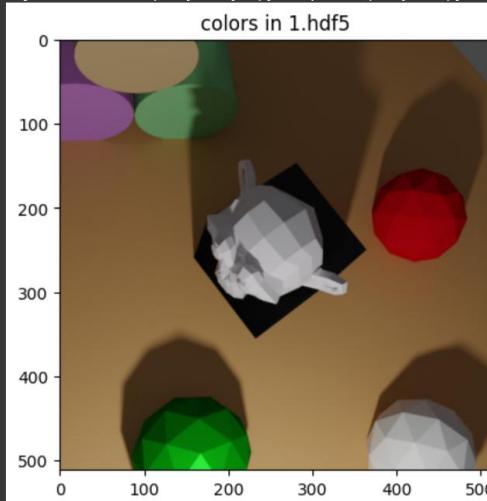
```
[ ] # visualize the generated data (0.hdf5)  
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/camera_sampling/output/0.hdf5"
```

```
→ examples/basics/camera_sampling/output/0.hdf5:  
Keys: 'colors': (512, 512, 3), 'depth': (512, 512), 'normals': (512, 512, 3)
```



```
[ ] # visualize the generated data (1.hdf5)
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/camera_sampling/output/1.hdf5"
```

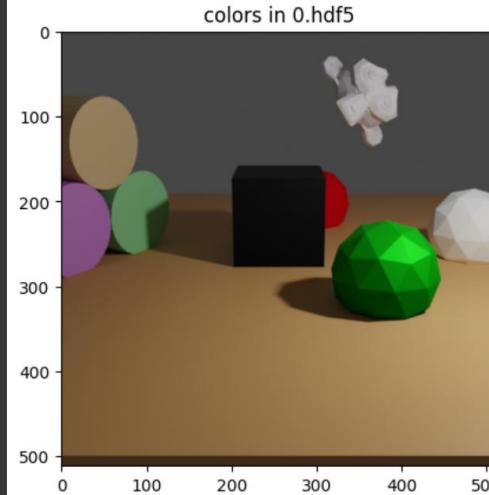
```
examples/basics/camera_sampling/output/1.hdf5:
Keys: 'colors': (512, 512, 3), 'depth': (512, 512), 'normals': (512, 512, 3)
```



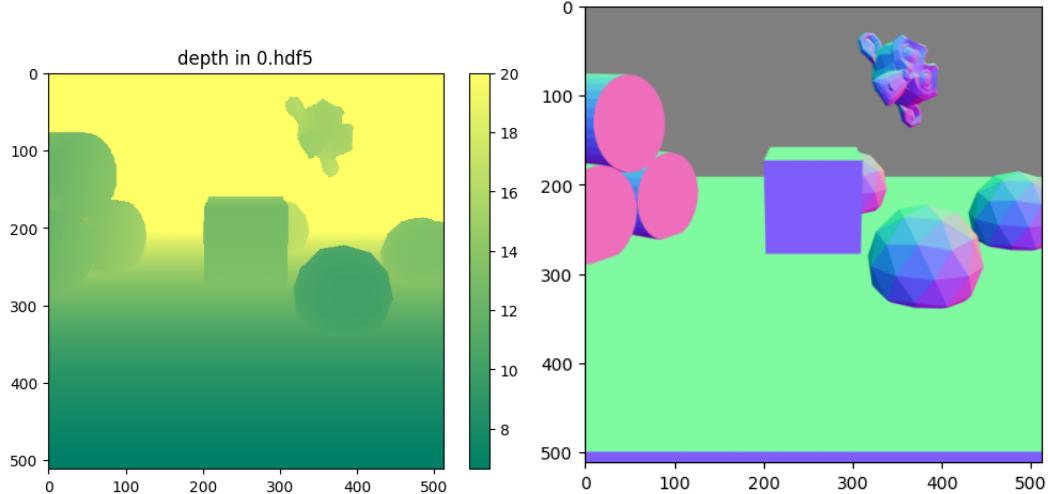
```
[ ] !blenderproc run examples/basics/entity_manipulation/main.py examples/resources/scene.obj example
```

```
[ ] # visualize the generated data (0.hdf5)
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/entity_manipulation/output/0.hdf5"
```

```
examples/basics/entity_manipulation/output/0.hdf5:
Keys: 'colors': (512, 512, 3), 'depth': (512, 512), 'normals': (512, 512, 3)
```

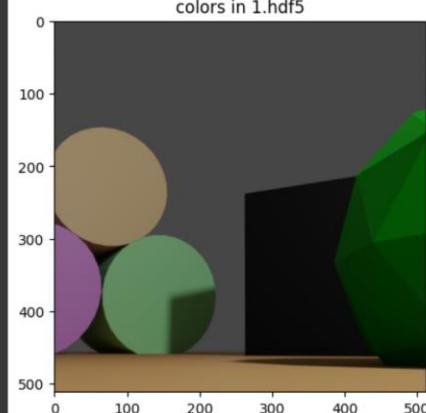


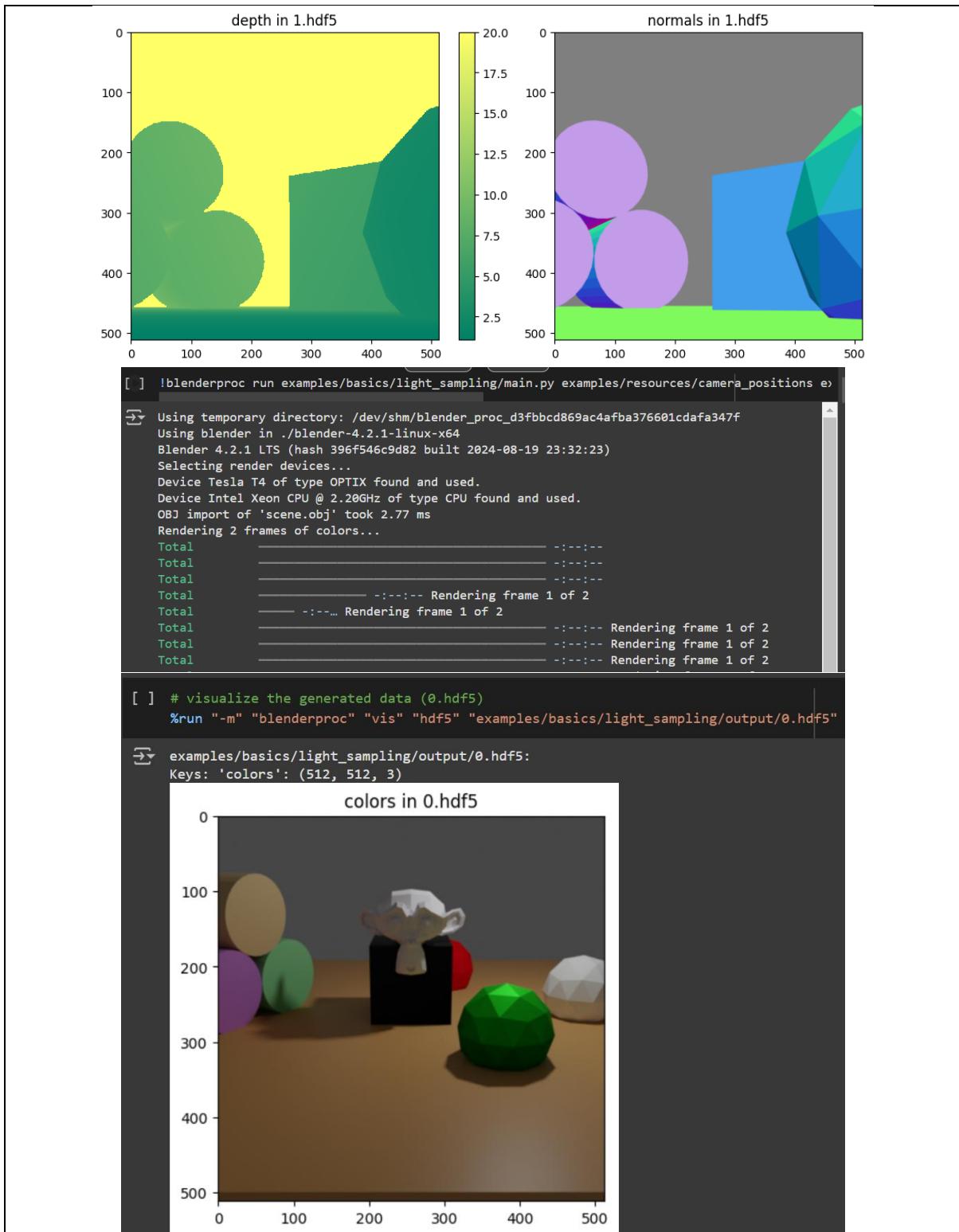
normals in 0.hdf5



```
[ ] # visualize the generated data (1.hdf5)
%run "-m" "blenderproc" "vis" "hdfs" "examples/basics/entity_manipulation/output/1.hdf5"
```

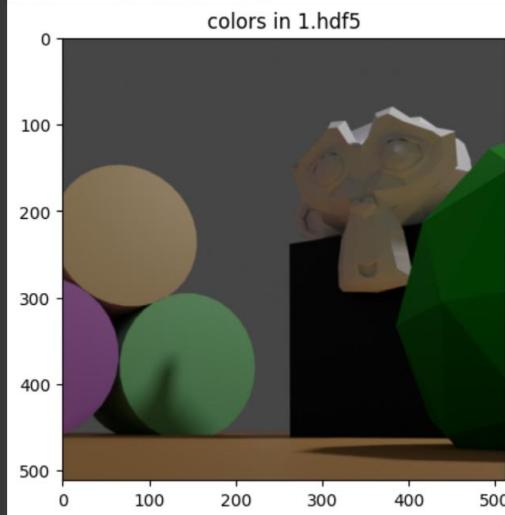
```
examples/basics/entity_manipulation/output/1.hdf5:
Keys: 'colors': (512, 512, 3), 'depth': (512, 512), 'normals': (512, 512, 3)
```





```
[ ] # visualize the generated data (1.hdf5)
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/light_sampling/output/1.hdf5"
```

```
→ examples/basics/light_sampling/output/1.hdf5:
Keys: 'colors': (512, 512, 3)
```

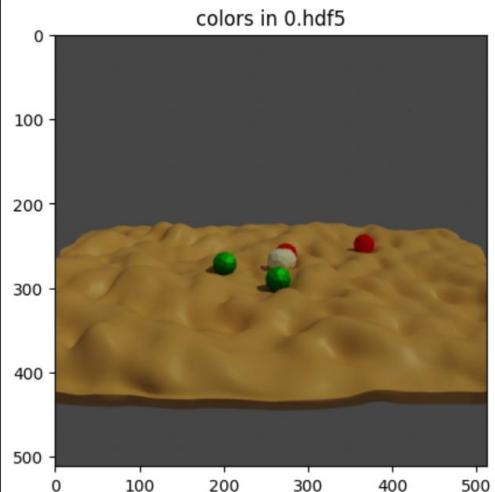


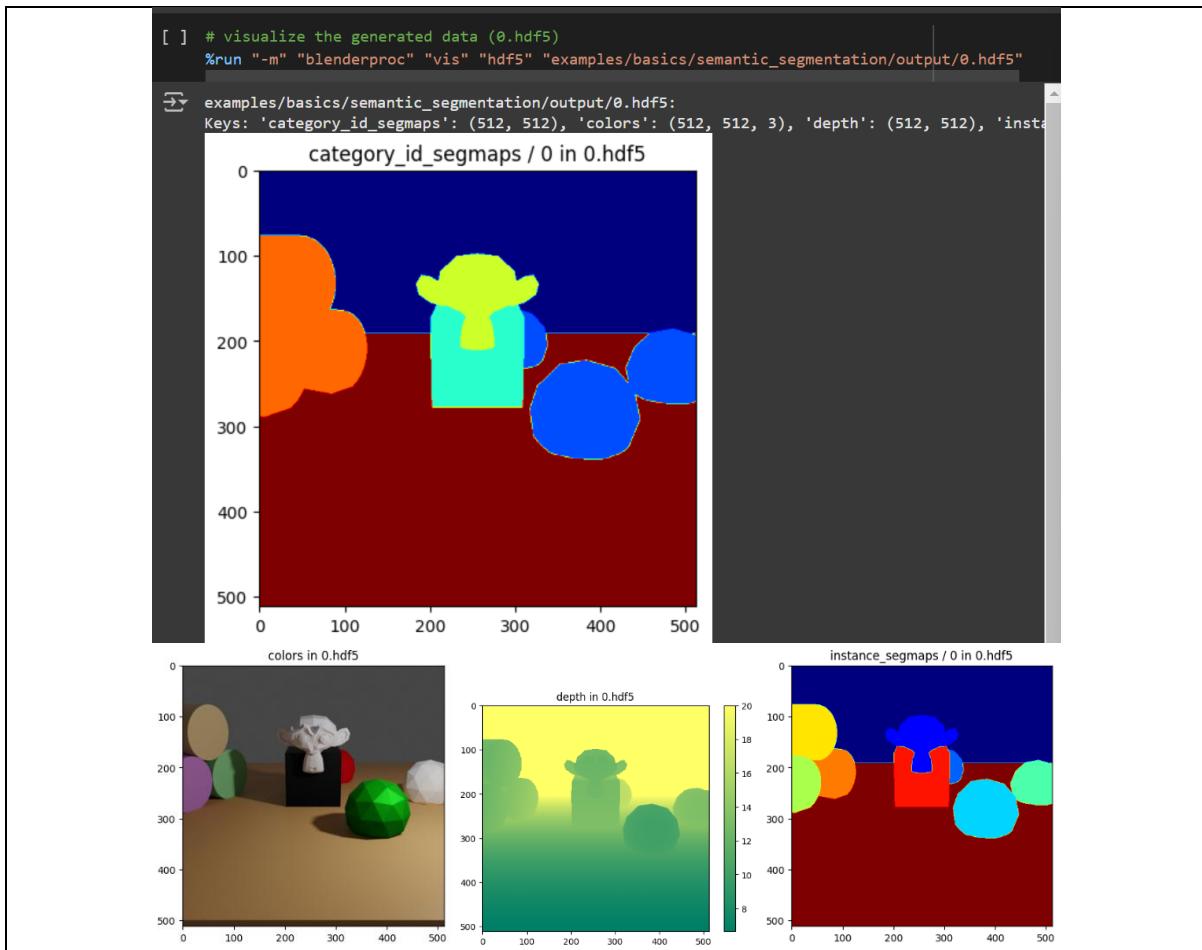
```
[ ] !blenderproc run examples/basics/physics_positioning/main.py examples/basics/physics_positioning/output/0.hdf5
```

```
→ Total ----- Rendering frame 1 of 1
```

```
[ ] # visualize the generated data (0.hdf5)
%run "-m" "blenderproc" "vis" "hdf5" "examples/basics/physics_positioning/output/0.hdf5"
```

```
→ examples/basics/physics_positioning/output/0.hdf5:
Keys: 'colors': (512, 512, 3)
```





```

[ ] import math
import matplotlib.pyplot as plt

num_images_to_display = 6

def plot_images(images, title=None):
    plt.figure(figsize=(20, 20))
    for i in range(num_images_to_display):
        ax = plt.subplot(1, num_images_to_display, i+1)
        if title is not None:
            plt.title(title)
        plt.imshow(images[i])
        plt.axis("off")

plot_images(images)

[ ] from datasets import load_dataset
dataset = load_dataset("imagefolder", data_dir="fractal_flame")

[ ] Resolving data files: 100% 100/100 [00:00<00:00, 5469.38it/s]
[ ] Downloading data: 100% 100/100 [00:00<00:00, 6142.53files/s]
[ ] Generating train split: 100/0 [00:00<00:00, 1280.48 examples/s]

▼ Now we can push the dataset to the Hub.

[ ] user_name = "Hazelnut27"
repo_name = "fractal_flame"

[ ] dataset.push_to_hub(f"{user_name}/{repo_name}")

[ ] Uploading the dataset shards: 100% 1/1 [00:08<00:00, 8.38s/it]
Map: 100% 100/100 [00:00<00:00, 627.10 examples/s]
Creating parquet from Arrow format: 100% 1/1 [00:00<00:00, 3.70ba/s]
CommitInfo(commit_url='https://huggingface.co/datasets/Hazelnut27/fractal_flame/commit/bfe403e...', commit_message='Upload dataset', commit_description='', oid='bfe403ec0bc1f29ccab97fdd40bbdbfe69ab3bf', pr_url=None, repo_url=RepoUrl('https://huggingface.co/datasets/Hazelnut27/fractal_flame'), endpoint='https://huggingface.co', repo_type='dataset', repo_id='Hazelnut27/fractal_flame', pr_revision=None, pr_num=None)

```

OWLV2_labeled_image_dataset_with_annotations

```

[ ] import os
import requests
from PIL import Image
from io import BytesIO
from transformers import OwlV2Processor, OwlV2ForObjectDetection

# We will grab some images from the hf-vision course-assets
url = "https://huggingface.co/datasets/hf-vision/course-assets/resolve/main/label_dataset_owl"

# Alternately, you can load images from a local folder
# path = ""

# Set the text queries
text_queries = ["dog", "table", "chair", "book", "magazine", "bookcase", "fireplace", "plant"]

# Set the number of images to generate
num_images = 10

# Set the score threshold (a lower value makes it more likely that an object is detected)
score_threshold = 0.20

# Create a directory for the annotated images
annotated_images_dir = "/content/annotated_images"
os.makedirs(annotated_images_dir, exist_ok=True)

# Write a function to add images to the dataset
def add_images(idx, text_queries):
    processor = OwlV2Processor.from_pretrained("google/owlv2-base-patch16-ensemble")
    model = OwlV2ForObjectDetection.from_pretrained("google/owlv2-base-patch16-ensemble")

    # Get the URL for an image
    image_url = f'{url}/{idx}.jpeg'

    # Download the image
    response = requests.get(image_url)
    image_content = response.content

    # Open the image using PIL
    image = Image.open(BytesIO(image_content))

[ ] from datasets import Dataset

username = "Hazelnut27"
repo_id = "labeled_images_demo"

# Create a dataset from the generator
ds = Dataset.from_generator(generate_entries)
ds.push_to_hub(f'{username}/{repo_id}')

[ ] Uploading the dataset shards: 100% [1/1] [00:00<00:00, 3.17it/s]
Map: 100% [10/10] [00:00<00:00, 272.39 examples/s]
Creating parquet from Arrow format: 100% [1/1] [00:00<00:00, 45.92ba/s]
README.md: 100% [353/353] [00:00<00:00, 15.0kB/s]

No files have been modified since last commit. Skipping to prevent empty commit.
WARNING:huggingface_hub.hf_api:No files have been modified since last commit. Skipping to prevent empty commit.
CommitInfo(commit_url='https://huggingface.co/datasets/Hazelnut27/labeled_images_demo/commit/c:commit_message='Upload dataset', commit_description='', oid='c2d96975d625626427aa65ac71aee6f12e80c904', pr_url=None, repo_url=RepoUrl('https://huggingface.co/datasets/Hazelnut27/labeled_images_demo', endpoint='https://huggingface.co', repo_type='dataset', repo_id='Hazelnut27/labeled_images_demo'), pr_revision=None, pr_num=None)

[ ] from datasets import load_dataset

# Load the dataset
dataset = load_dataset('Hazelnut27/labeled_images_demo')

[ ] train-00000-of-00001.parquet: 100%
Generating train split: 100% [10/10] [00:00<00:00, 259.87 examples/s]

```

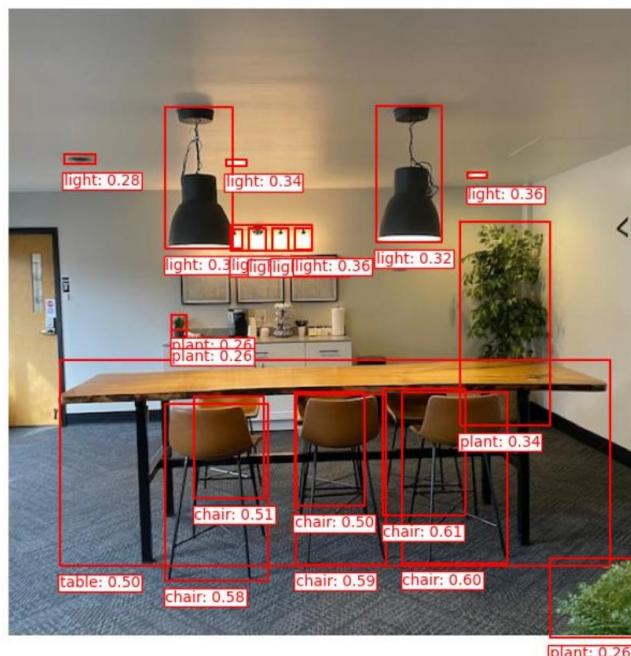
Let's take a look at one of the image, label pairs in our new dataset. The image shows a table and chairs with some lights and a plant.

```
[ ] dataset["train"][0]["image"]
```

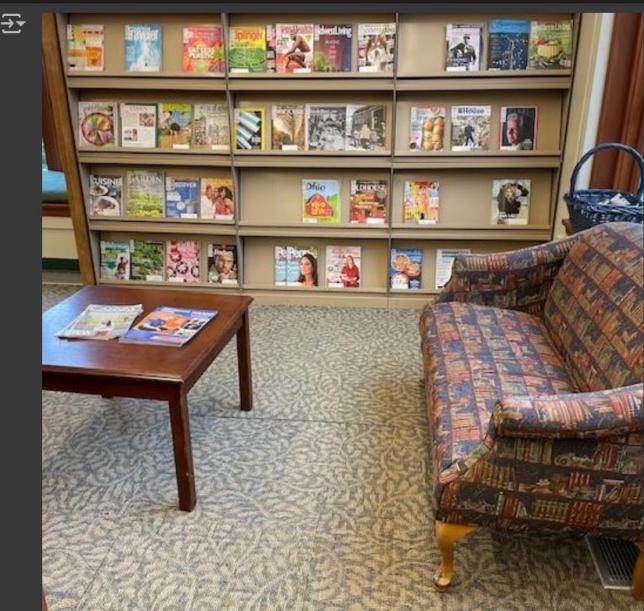


```
[ ] dataset["train"][0]["label"]
```

```
→ '10 lights, 4 plants, 1 table, 6 chairs'
```



```
[ ] dataset["train"][7]["image"]
```



```
[ ] dataset["train"][7]["label"]
```

```
→ '5 books, 51 magazines, 1 dog, 4 bookcases, 1 table'
```

```
[ ] dataset["train"][7]["annotated_image"]
```



Synthetic_lung_images_hf_course

```
[ ] # load the dataset of lung images
dataset = load_dataset("hf-vision/chest-xray-pneumonia")

↳ /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
README.md: 100% | 2.06k/2.06k [00:00<00:00, 42.2kB/s]

train-00000-of-00007.parquet: 100% | 446M/446M [00:21<00:00, 21.0MB/s]

train-00001-of-00007.parquet: 100% | 385M/385M [00:17<00:00, 23.6MB/s]

train-00002-of-00007.parquet: 100% | 68.9M/68.9M [00:03<00:00, 21.2MB/s]

train-00003-of-00007.parquet: 100% | 74.1M/74.1M [00:05<00:00, 17.6MB/s]

train-00004-of-00007.parquet: 100% | 59.9M/59.9M [00:02<00:00, 22.0MB/s]

train-00005-of-00007.parquet: 100% | 57.6M/57.6M [00:02<00:00, 21.3MB/s]

train-00006-of-00007.parquet: 100% | 57.8M/57.8M [00:02<00:00, 23.1MB/s]

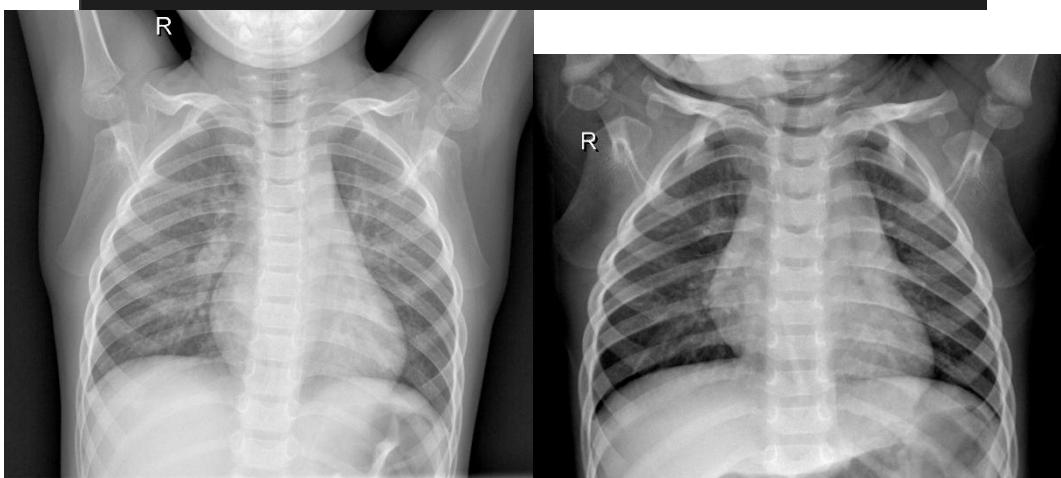
validation-00000-of-00001.parquet: 100% | 3.02M/3.02M [00:00<00:00, 42.9MB/s]

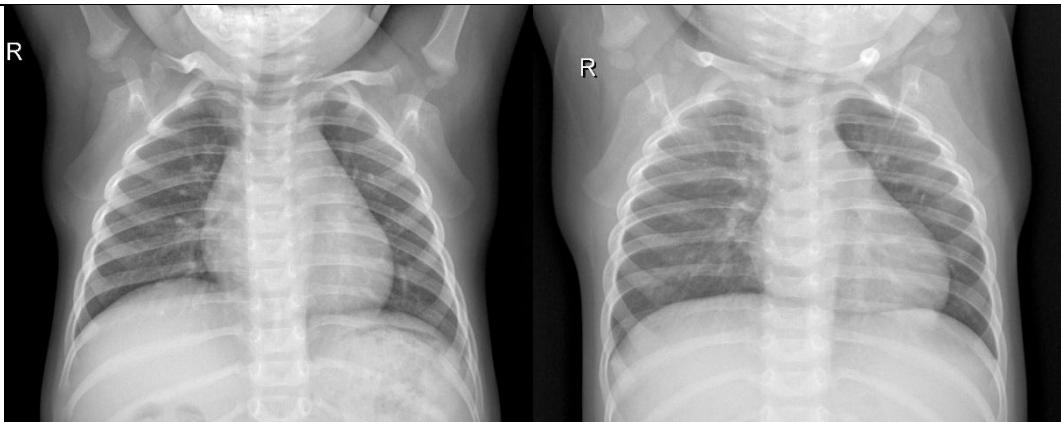
[ ] len(dataset['train'])

↳ 5216

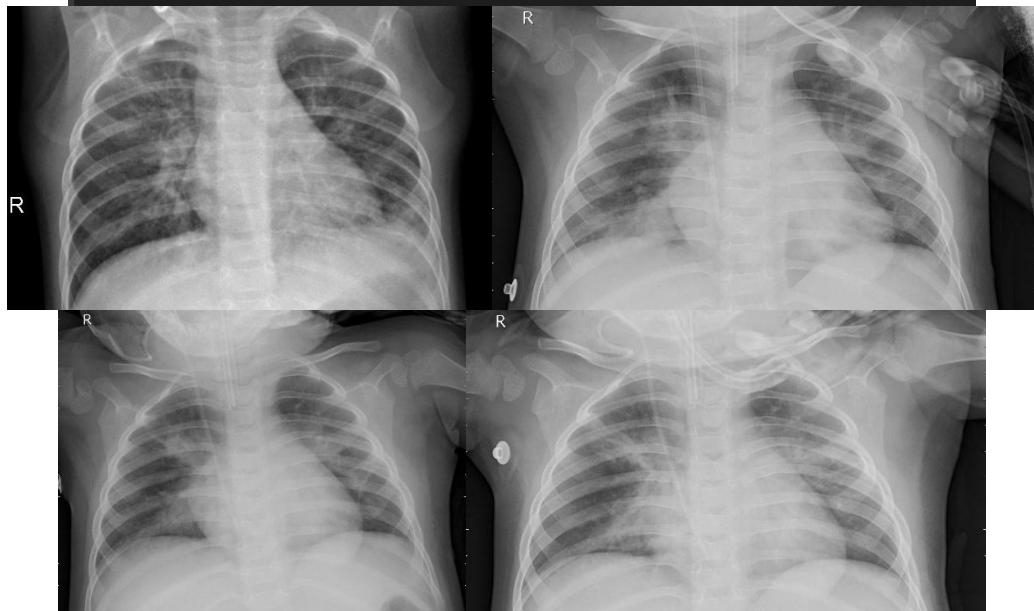
[ ] from PIL import Image
from IPython.display import display

# Display the image
for i in range(4):
    display(dataset["train"][i]['image'])
```





```
[ ] from PIL import Image  
from IPython.display import display  
  
# Display the image  
for i in range(5211,5215):  
    display(dataset["train"][i]['image'])
```



```

[ ] # Create the generator
gen_net = Generator(nb_gpu).to(device)

# Handle multi-GPU if desired
if (device.type == 'cuda') and (nb_gpu > 1):
    gen_net = nn.DataParallel(gen_net, list(range(nb_gpu)))

# Apply the ``weights_init`` function to randomly initialize all weights
gen_net.apply(weights_init)

# Print the model
print(gen_net)

→ Generator(
    (main): Sequential(
        (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (13): Tanh()
    )
)

[ ] # Create the Discriminator
disc_net = Discriminator(nb_gpu).to(device)

# Handle multi-GPU if desired
if (device.type == 'cuda') and (nb_gpu > 1):
    disc_net = nn.DataParallel(disc_net, list(range(nb_gpu)))

# Apply the ``weights_init`` function to randomly initialize all weights
# like this: ```to mean=0, stddev=0.2```.
disc_net.apply(weights_init)

# Print the model
print(disc_net)

→ Discriminator(
    (main): Sequential(
        (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (1): LeakyReLU(negative_slope=0.2, inplace=True)
        (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (4): LeakyReLU(negative_slope=0.2, inplace=True)
        (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (7): LeakyReLU(negative_slope=0.2, inplace=True)
        (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (10): LeakyReLU(negative_slope=0.2, inplace=True)
        (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
        (12): Sigmoid()
    )
)

[ ] # Real image and discriminator thinks it is real - Right decision
target = 1
output= 0.99

print(-(target * math.log(output) + (1-target)*math.log(1-output)))

# Real image and discriminator thinks it is fake - False decision
target = 1
output= 0.01

print(-(target * math.log(output) + (1-target)*math.log(1-output)))

# False image but discriminator thinks it is real - False decision
target = 0
output= 0.99

print(-(target * math.log(output) + (1-target)*math.log(1-output)))

→ 0.01005033585350145
4.605170185988091
4.605170185988091

```

```
[ ] # Load 9 images
nb_images = 9
nb_row = math.ceil(math.sqrt(nb_images))

# Training Loop
data_len = len(dataloader)

# Lists to keep track of progress
img_list = []
G_losses = []
D_losses = []

# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader (depends on batch_size and your number of images)
    for i, data in enumerate(dataloader, 0):

        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))

        ## Train with all-real batch
        disc_net.zero_grad()
        # Format batch
        real_cpu = data['image'].to(device)
        #real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
        # Forward pass real batch through D
        output = disc_net(real_cpu).view(-1)
        # Calculate loss on all-real batch
        errD_real = criterion(output, label)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        D_x = output.mean().item()

        ## Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # Generate fake image batch with G
        fake = gen_net(noise)
        label.fill_(fake_label)
```

```
[ ]
```

```

# Print output training stats every 50 batches (if your dataset is large, printing at
if i % 50 == 0:
    print(f"Epoch: {epoch}/{num_epochs} Batches: {i}/{data_len}\tLoss_D: {errD.item():.4f}\tLoss_G: {errG.item():.4f}")

# Save Losses for plotting later
G_losses.append(errG.item())
D_losses.append(errD.item())

# Generate fake images to see how the generator is doing by saving G's output on fixed_noi
if show_images == True and epoch % 10 == 0:
    with torch.no_grad():
        # Uncomment the line below to generate a new variety of images every time
        #fixed_noise = torch.randn(64, nz, 1, 1, device=device)

        fake = gen_net(fixed_noise).detach().cpu()
        img_list.append(vutils.make_grid(fake[:nb_images], padding=2, normalize=True, nrow=nb_im
        plt.figure(figsize=(3, 3))
        plt.axis("off")
        plt.imshow(np.transpose(img_list[-1], (1, 2, 0)))

    if save_images == True:
        plt.savefig(f'images/epoch_{epoch}_gen_images.png')

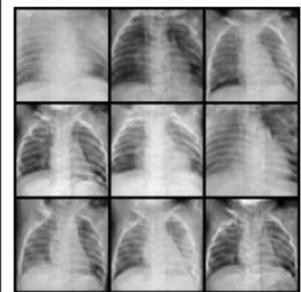
    # Display image
    plt.show()

# Save models each 5 epochs
if epoch % 5 == 0:
    if save_model:
        save_dcgan(gen_net, disc_net, path_checkpoint=f"models/chest_epoch_{epoch}_checkpo

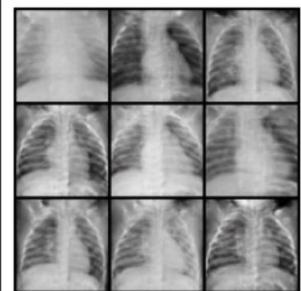
```

```
# Save the final models
save_dcgan(gen_net, disc_net, path_checkpoint="models/chest_final_epoch_checkpoint.pkl")
```

```
→ Epoch: 73/100 Batches: 0/41      Loss_D: 0.3047      Loss_G: 5.4518      D(x): 0.9560      D(G(z)): 0.8498
Epoch: 74/100 Batches: 0/41      Loss_D: 0.2691      Loss_G: 5.0210      D(x): 0.9005      D(G(z)): 0.8498
Epoch: 75/100 Batches: 0/41      Loss_D: 0.3297      Loss_G: 3.8732      D(x): 0.8657      D(G(z)): 0.8498
Epoch: 76/100 Batches: 0/41      Loss_D: 0.2293      Loss_G: 3.7722      D(x): 0.8443      D(G(z)): 0.8498
Epoch: 77/100 Batches: 0/41      Loss_D: 0.2314      Loss_G: 4.0465      D(x): 0.8898      D(G(z)): 0.8498
Epoch: 78/100 Batches: 0/41      Loss_D: 0.3107      Loss_G: 4.3041      D(x): 0.8359      D(G(z)): 0.8498
... Epoch: 79/100 Batches: 0/41      Loss_D: 0.4792      Loss_G: 7.4737      D(x): 0.9498      D(G(z)): 0.8498
```

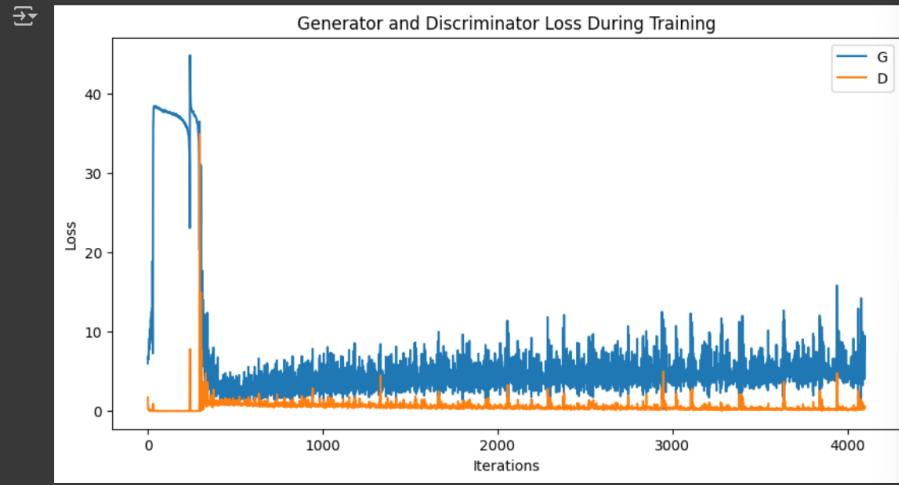


```
Epoch: 81/100 Batches: 0/41      Loss_D: 0.1300      Loss_G: 5.6036      D(x): 0.9592      D(G(z)): 0.8498
Epoch: 82/100 Batches: 0/41      Loss_D: 0.3295      Loss_G: 5.1986      D(x): 0.9087      D(G(z)): 0.8498
Epoch: 83/100 Batches: 0/41      Loss_D: 0.5559      Loss_G: 8.3827      D(x): 0.9812      D(G(z)): 0.8498
Epoch: 84/100 Batches: 0/41      Loss_D: 0.3686      Loss_G: 3.3655      D(x): 0.7606      D(G(z)): 0.8498
Epoch: 85/100 Batches: 0/41      Loss_D: 0.2084      Loss_G: 4.9672      D(x): 0.8497      D(G(z)): 0.8498
Epoch: 86/100 Batches: 0/41      Loss_D: 0.3038      Loss_G: 5.4950      D(x): 0.9057      D(G(z)): 0.8498
Epoch: 87/100 Batches: 0/41      Loss_D: 0.1304      Loss_G: 4.8840      D(x): 0.9359      D(G(z)): 0.8498
Epoch: 88/100 Batches: 0/41      Loss_D: 0.1681      Loss_G: 4.8975      D(x): 0.9676      D(G(z)): 0.8498
Epoch: 89/100 Batches: 0/41      Loss_D: 0.3123      Loss_G: 5.2607      D(x): 0.9050      D(G(z)): 0.8498
Epoch: 90/100 Batches: 0/41      Loss_D: 0.2011      Loss_G: 3.7200      D(x): 0.8777      D(G(z)): 0.8498
```



```
[ ] from IPython.display import HTML
import matplotlib.animation as animation
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



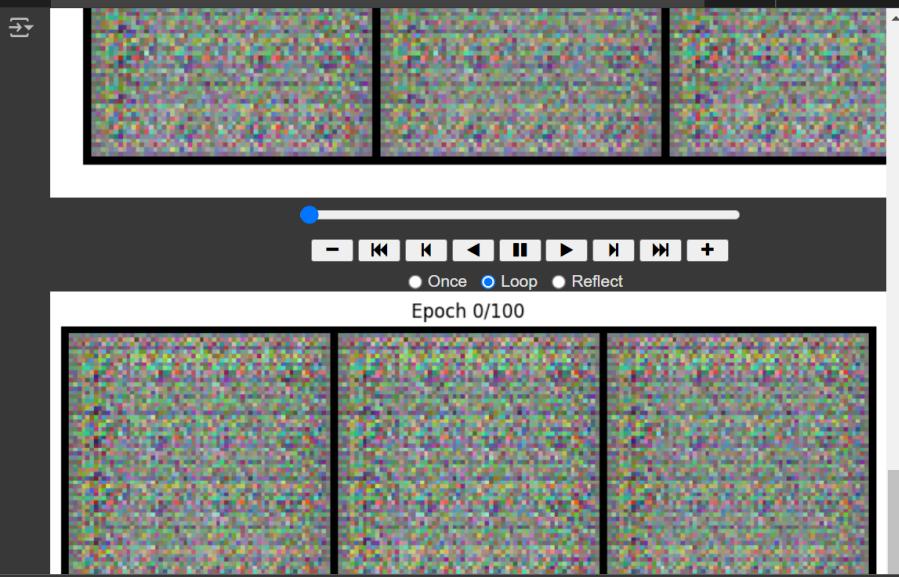
```
[ ] fig = plt.figure(figsize=(8, 8))
plt.axis("off")
plt.tight_layout(pad=2.0, h_pad=2.0, w_pad=2.0)

# Initial empty title and image
title = plt.title("")
img_plot = plt.imshow(np.transpose(img_list[0], (1, 2, 0)), animated=True)

def update_title_and_image(frame):
    img_plot.set_array(np.transpose(img_list[frame], (1, 2, 0)))
    title.set_text(f"Epoch {frame}/{num_epochs}")

ani = animation.FuncAnimation(fig, update_title_and_image, frames=len(img_list), interval=100,
                             cache_frame_data=False)

# Display animation
HTML(ani.to_jshtml())
```



```

[ ] # Number of desired images
num_img = 16
nb_row = math.ceil(math.sqrt(num_img))

# Create a random noise
random_noise = torch.randn(num_img, nz, 1, 1, device=device)

# Instantiate a generator
new_gen= Generator(nb_gpu).to(device)

# Handle multi-GPU if desired
if (device.type == 'cuda') and (nb_gpu > 1):
    new_gen = nn.DataParallel(new_gen, list(range(nb_gpu)))

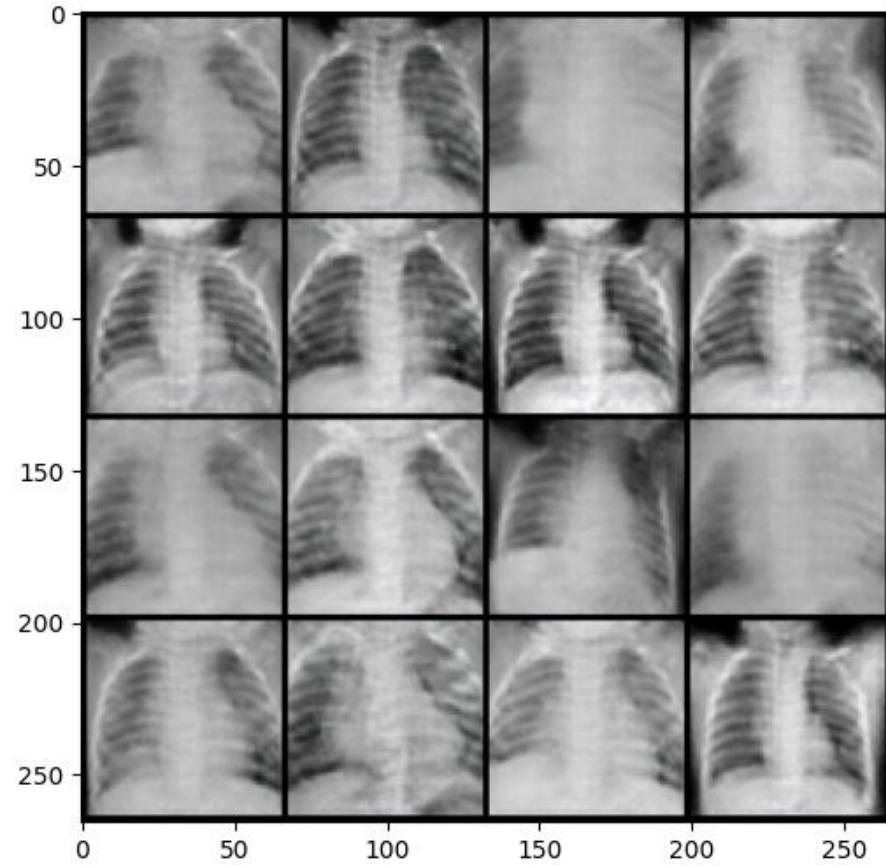
# Load weights from path
checkpoint = torch.load(path_checkpoint, map_location="cpu")
state_dict_g = checkpoint["g_model_state_dict"]
new_gen.load_state_dict(state_dict_g)

# Generate images
with torch.no_grad():
    fake_data = new_gen(random_noise).detach().cpu()

# Plot images
img_grid = vutils.make_grid(fake_data, padding=2, normalize=True, nrow=nb_row).cpu()
img_grid = np.transpose(img_grid, (1, 2, 0))
plt.figure(figsize=(6, 6))
plt.imshow(img_grid)
plt.show()

```

→ <ipython-input-25-31883b1e2340>:16: FutureWarning: You are using `torch.load` with `weights_on



```

[ ] # Set threshold
threshold = 0.70

# Create list to store the images classified as >= threshold
pass_threshold = []

while len(pass_threshold)<64:
    # Create a random noise
    random_noise = torch.randn(num_img, nz, 1, 1, device=device)
    # Generate images for this noise
    with torch.no_grad():
        fake_data = new_gen(random_noise)

    # Set the discriminator to eval mode
    new_dis.eval()

    # Pass the generated fake_data through the discriminator & obtain real/fake probability
    output_probabilities = new_dis(fake_data)

    # Create mask (If prob < threshold, set to False, else True)
    good_image_mask = output_probabilities >= threshold

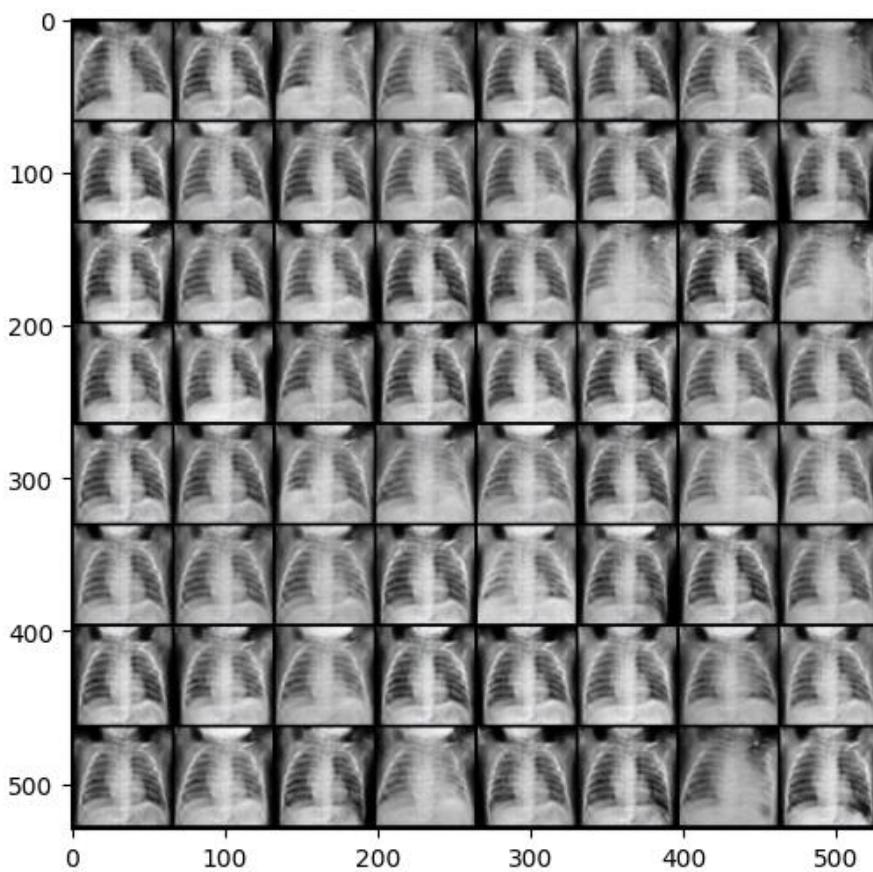
    # Remove extra tensor dims (torch.Size([16, 1, 1, 1]) -> torch.Size([16]))
    good_image_mask = good_image_mask.squeeze()

    # Keep the selected images
    good_images = fake_data[good_image_mask > threshold]

    # If good_images is not None
    if good_images.numel() > 0:
        # Loop through the tensor along the first dimension (index 0), since we have x images
        for i in range(good_images.size(0)):
            # Add selected images
            pass_threshold.append(good_images[i])

# Plot selected images
img_grid = vutils.make_grid(pass_threshold, padding=2, normalize=True, nrow=8).cpu()
img_grid = np.transpose(img_grid, (1, 2, 0))
plt.figure(figsize=(6, 6))
plt.imshow(img_grid)
plt.show()

```



RESUME UAS CHAPTER 11

Unit 11 dari Kursus Komunitas Visi Komputer oleh Hugging Face membahas Zero-Shot Learning (ZSL) dalam visi komputer, yang memungkinkan model mengenali kelas objek yang tidak pernah dilihat selama pelatihan.

A. On Generalization

Model pembelajaran mesin yang baik harus bisa menggeneralisasi, yaitu mengenali objek baru yang mirip dengan data pelatihan. Namun, ada tantangan ketika domain data berbeda. Misalnya, model yang hanya dilatih dengan foto kucing nyata mungkin kesulitan mengenali ilustrasi kartun kucing karena perbedaan gaya visual. Di sinilah adaptasi domain diperlukan untuk menjembatani perbedaan ini.

B. What Is Zero-Shot Learning

Zero-Shot Learning (ZSL) adalah pendekatan dalam pembelajaran mesin yang memungkinkan model mengenali kelas-kelas yang tidak pernah dilihat selama pelatihan. Ini dicapai dengan memanfaatkan informasi tambahan, seperti deskripsi tekstual atau atribut semantik, yang menggambarkan karakteristik kelas-kelas tersebut. Dengan demikian, model dapat mengaitkan pengetahuan dari kelas yang sudah dikenal ke kelas yang belum pernah dilihat. Dalam ZSL, model diuji dengan kelas yang tidak ada dalam data pelatihan, sehingga set pelatihan dan pengujian tidak saling tumpang tindih. Generalized Zero-Shot Learning (GZSL) memperluas konsep ini dengan mengizinkan model diuji pada kombinasi kelas yang pernah dan belum dilihat, mencerminkan situasi dunia nyata di mana data sangat bervariasi.

Dalam makalah "An embarrassingly simple approach to zero-shot learning" menjelaskan bahwa ZSL terdiri dari pembelajaran bagaimana mengenali konsep baru hanya dengan memiliki deskripsi tentangnya[1]. Dalam konteks segmentasi semantik visual, ZSL memungkinkan model untuk mempelajari kategori yang tidak terlihat hanya berdasarkan gambar berlabel dari kelas yang terlihat[2]

C. History Of Zero-Shot Learning

Zero-Shot Learning (ZSL) adalah metode dalam pembelajaran mesin yang memungkinkan model mengenali kelas yang belum pernah ditemui selama pelatihan. Awalnya, konsep ini diperkenalkan pada tahun 2008 di domain pemrosesan bahasa alami (NLP) oleh Chang et al., yang menyebutnya sebagai *dataless classification*. Pada tahun yang sama, Larochelle et al. mengadaptasi konsep ini untuk visi komputer dengan nama *zero-data learning*. Istilah "zero-shot learning" sendiri mulai digunakan secara resmi oleh Palatucci et al. pada konferensi NeurIPS tahun 2009.

Dalam visi komputer, Zero-Shot Learning memungkinkan model mengenali objek dari kelas yang tidak ada dalam data pelatihan dengan memanfaatkan informasi tambahan seperti deskripsi tekstual atau atribut visual. Ini menjadi relevan dengan kebutuhan mengklasifikasikan banyak kelas meskipun data yang tersedia terbatas. Xian et al.

menyatakan bahwa ZSL adalah bentuk ekstrem dari adaptasi domain, di mana model dituntut untuk mampu menggeneralisasi ke kelas yang benar-benar baru. Perkembangan signifikan terjadi pada tahun 2021 dengan diperkenalkannya model CLIP oleh OpenAI, yang meningkatkan kemampuan ZSL dalam visi komputer.

Konsep lanjutan dari ZSL adalah Generalized Zero-Shot Learning (GZSL), yang menangani skenario di mana data uji mencakup kelas yang sudah pernah dilihat dan yang belum pernah dilihat selama pelatihan. ZSL kini menjadi bidang penelitian penting dalam pembelajaran mesin, dengan aplikasi luas di visi komputer, NLP, dan lainnya. Metode ini memberikan solusi untuk pengenalan kelas baru tanpa memerlukan data pelatihan yang besar, menjadikannya sangat berguna dalam situasi di mana pengumpulan data sulit atau mahal.

D. How Does Zero-shot Learning Work in Computer Vision?

Cara Kerja Zero-Shot Learning dalam Visi Komputer berbeda dengan pemrosesan bahasa alami (NLP) di mana model dapat mempelajari pola dari teks, visi komputer memerlukan informasi tambahan untuk mengenali objek yang belum pernah dilihat. Informasi ini disebut informasi semantik atau informasi tambahan, seperti deskripsi tekstual, vektor atribut, atau embedding label kelas. Model belajar memetakan fitur visual ke ruang semantik bersama, memungkinkan pengenalan kelas baru berdasarkan kesamaan dalam ruang tersebut. Berikut adalah cara kerja ZSL:

1. Representasi Fitur Visual dan Semantik:

Ekstraksi Fitur Visual merupakan Model belajar mengenali pola dalam gambar menggunakan teknik seperti jaringan saraf konvolusi (CNN). Hasilnya adalah data yang mewakili karakteristik penting dari objek dalam gambar.

Representasi Semantik yaitu Setiap kelas objek, baik yang pernah dilihat (seen) maupun yang belum dilihat (unseen), digambarkan melalui deskripsi semantik. Ini bisa berupa atribut seperti "berbulu", "berwarna biru", atau embedding teks yang diperoleh dari deskripsi tulisan.

2. Pemetaan ke Ruang Bersama

Ruang Embedding Bersama merupakan Model memetakan fitur visual dan deskripsi semantik ke ruang yang sama. Di ruang ini, fitur visual dari suatu kelas akan mendekati representasi semantiknya. Hal ini memungkinkan model mengenali kelas yang belum dilihat dengan membandingkan kemiripannya dengan deskripsi semantik.

3. Proses Pelatihan

Pelatihan dengan Kelas yang Terlihat merupakan Model dilatih menggunakan data dari kelas yang terlihat, dengan tujuan agar fitur visual dan deskripsi semantik sesuai di ruang bersama. Jaringan saraf atau metode regresi sering digunakan untuk melatih pemetaan ini.

4. Inferensi pada Kelas yang Tidak Terlihat

Prediksi Kelas Baru: Ketika model menghadapi gambar dari kelas yang belum dilihat, model mengekstraksi fitur visualnya, memetakan ke ruang bersama, dan membandingkannya dengan deskripsi semantik dari semua kelas baru. Model akan memilih kelas dengan kemiripan tertinggi..

E. Metode Zero-Shot Learning

Metode berbasis embedding bekerja dengan memetakan fitur visual dari gambar dan representasi semantik dari kelas ke dalam ruang embedding bersama. Untuk mendapatkan representasi visual, model seperti Convolutional Neural Networks (CNN) digunakan, sementara representasi semantik bisa berupa deskripsi atribut atau embedding teks yang dihasilkan oleh model bahasa seperti Word2Vec atau Transformer. Setelah dipetakan ke ruang yang sama, kesamaan antara representasi visual dan semantik diukur untuk menentukan kelas gambar baru. Keunggulan metode ini adalah kemampuannya untuk bekerja dengan kelas yang tidak terlihat tanpa memerlukan data tambahan, meskipun masih rentan terhadap masalah bias domain jika representasi visual atau semantik tidak cukup menggambarkan karakteristik kelas dengan baik.

Di sisi lain, metode berbasis generatif menggunakan model generatif seperti Generative Adversarial Networks (GANs) atau Variational Autoencoders (VAEs) untuk membuat fitur visual atau data gambar untuk kelas yang tidak terlihat. Dalam pendekatan ini, model generatif dilatih dengan data dari kelas yang terlihat untuk mempelajari hubungan antara representasi semantik dan fitur visual. Dengan cara ini, model bisa menghasilkan data sintetis untuk kelas baru berdasarkan deskripsi semantik yang ada. Data sintetis tersebut kemudian digunakan untuk melatih model klasifikasi tambahan, memungkinkan pengenalan kelas baru meskipun data nyata tidak tersedia. Keunggulan utama dari metode ini adalah kemampuannya untuk mengatasi masalah bias domain dengan menciptakan data tambahan untuk kelas yang tidak terlihat, meskipun membutuhkan lebih banyak sumber daya komputasi dan pelatihan yang lebih kompleks[3].

F. Zero-shot Learning vs. Generalized Zero-shot Learning

Zero-Shot Learning (ZSL) dan Generalized Zero-Shot Learning (GZSL) adalah dua pendekatan dalam pembelajaran mesin yang memungkinkan model untuk mengklasifikasikan kelas yang belum pernah dilihat selama pelatihan. Pada ZSL, model dilatih hanya dengan data dari kelas yang sudah terlihat dan diuji hanya pada kelas yang belum pernah ditemui sebelumnya. Dalam kasus ini, dataset pengujian hanya mencakup sampel dari kelas-kelas yang tidak ada dalam data pelatihan ZSL mengandalkan informasi semantik atau atribut untuk menjembatani kesenjangan antara kelas yang terlihat dan tidak terlihat. Meskipun pendekatan ini berhasil dalam skenario eksperimental, aplikasinya di dunia nyata terbatas karena tidak mempertimbangkan keberadaan kelas terlihat dalam dataset pengujian[4].

Generalized Zero-Shot Learning (GZSL) mengembangkan konsep ZSL dengan memungkinkan pengujian pada data yang mencakup campuran kelas yang terlihat dan tidak terlihat, sehingga lebih sesuai untuk digunakan di dunia nyata. Tantangan utama dalam GZSL adalah kemampuan model untuk membedakan apakah suatu sampel berasal dari kelas yang sudah dikenal atau kelas yang baru. Untuk itu, dibutuhkan pendekatan yang lebih kompleks untuk mengatasi bias yang sering terjadi terhadap kelas yang terlihat selama pelatihan. Beberapa metode populer dalam GZSL termasuk penggunaan model generatif untuk mensintesis data dari kelas yang tidak terlihat atau pemetaan fitur visual ke dalam ruang semantik bersama. Meskipun GZSL dianggap lebih sulit dibandingkan ZSL, metode ini lebih aplikatif karena lebih mencerminkan kondisi operasional dalam pengenalan pola dan klasifikasi objek baru[5].

G. Inductive Zero-shot Learning vs. Transductive Zero-shot Learning

Inductive ZSL melibatkan pelatihan model hanya dengan data dari kelas yang sudah terlihat, tanpa adanya akses ke informasi atau data dari kelas yang belum terlihat. Tujuannya adalah untuk menggeneralisasi pola dari data pelatihan agar bisa mengklasifikasikan contoh dari kelas yang tidak terlihat. Pendekatan ini mengandalkan representasi semantik atau atribut yang menghubungkan kelas yang terlihat dengan kelas yang tidak terlihat, sehingga model bisa mentransfer pengetahuan yang ada. Namun, karena tidak ada informasi tambahan dari kelas yang tidak terlihat, model seringkali menghadapi tantangan domain shift, yaitu perbedaan distribusi antara kelas yang terlihat dan yang tidak terlihat. Hal ini bisa menurunkan akurasi prediksi.

Di sisi lain, Transductive ZSL memanfaatkan informasi tambahan tentang kelas yang tidak terlihat selama pelatihan, meskipun labelnya tidak tersedia. Informasi ini bisa berupa atribut atau contoh tanpa label, yang memungkinkan model untuk menyesuaikan diri dengan distribusi data kelas yang tidak terlihat, mengurangi masalah domain shift. Pendekatan transduktif sering menggunakan teknik seperti self-training atau co-training, di mana model secara iteratif memperbarui diri dengan memprediksi label untuk data tanpa label, lalu menggunakan prediksi tersebut sebagai pseudo-label untuk pelatihan berikutnya. Menurut penelitian, transductive ZSL dapat meningkatkan kinerja model dengan memanfaatkan struktur data dari kelas yang tidak terlihat, sehingga menghasilkan model yang lebih robust dalam mengklasifikasikan data dari kelas yang tidak terlihat[6].

H. Zero-shot Learning Example with Semantic Embeddings

Zero-Shot Learning (ZSL) adalah teknik dalam pembelajaran mesin yang memungkinkan model untuk mengklasifikasikan data dari kelas yang belum pernah ditemukan selama pelatihan. Salah satu komponen penting dalam ZSL adalah embedding semantik, yaitu representasi vektor yang menggambarkan makna atau informasi semantik dari data. Embedding semantik ini mengubah informasi seperti deskripsi teks atau atribut kelas menjadi vektor dalam ruang berdimensi tinggi, di mana jarak dan arah antara vektor ini mencerminkan hubungan semantik mereka. Proses ini biasanya dilakukan menggunakan model pembelajaran tanpa pengawasan, seperti Word2Vec atau GloVe.

Pada ZSL, embedding semantik digunakan untuk menjembatani perbedaan antara kelas yang terlihat (seen) dan kelas yang tidak terlihat (unseen). Model dilatih untuk memetakan fitur visual dari data yang terlihat ke dalam ruang embedding semantik, sehingga model dapat memahami hubungan antara representasi visual dan semantik. Ketika model dihadapkan pada data dari kelas yang tidak terlihat, model dapat memproyeksikan fitur visual ke dalam ruang embedding semantik dan menentukan kelas yang paling sesuai berdasarkan kesamaan dengan representasi semantik yang sudah ada. Pendekatan ini memungkinkan model untuk mentransfer pengetahuan dari kelas yang terlihat ke kelas yang tidak terlihat, sehingga meningkatkan kemampuan model dalam generalisasi. Menurut penelitian, penggunaan embedding semantik yang konsisten dapat meningkatkan kinerja model dalam domain adaptif ZSL dengan memastikan bahwa representasi semantik tetap relevan dan akurat di berbagai domain [7].

I. Comparison to CLIP

Hubungan antara Zero-Shot Learning (ZSL) dan CLIP (Contrastive Language–Image Pre-training) [5] terletak pada tujuan bersama mereka, yaitu memungkinkan model mengenali dan mengklasifikasikan gambar dari kategori yang tidak ada dalam data pelatihan. Namun, CLIP merupakan kemajuan signifikan dan penerapan yang lebih luas dari prinsip-prinsip dasar ZSL, dengan memanfaatkan pendekatan baru dalam pembelajaran dan generalisasi.

Hubungan antara CLIP dan ZSL dapat dijelaskan sebagai berikut:

- Baik ZSL maupun CLIP bertujuan untuk mengklasifikasikan gambar ke dalam kelas yang tidak terlihat selama pelatihan. Namun, sementara pendekatan ZSL tradisional sering mengandalkan embedding semantik atau atribut yang telah ditentukan sebelumnya untuk menjembatani kesenjangan antara kelas terlihat dan tidak terlihat, CLIP langsung belajar dari deskripsi bahasa alami. Hal ini memungkinkan CLIP untuk melakukan generalisasi ke berbagai tugas tanpa memerlukan embedding khusus untuk setiap tugas.
- CLIP adalah contoh utama dari pembelajaran multi-modal, di mana model belajar dari data tekstual dan visual secara bersamaan. Pendekatan ini sejalan dengan ZSL, di mana informasi tambahan digunakan untuk meningkatkan kinerja klasifikasi. CLIP melangkah lebih jauh dengan langsung belajar dari teks dan gambar mentah, memungkinkan model memahami dan merepresentasikan hubungan antara konten visual dan deskripsi bahasa.

J. Zero-Shot Learning Evaluation Datasets

Metode ZSL baru diusulkan setiap tahun, membuat penilaian terhadap pendekatan terbaik menjadi tantangan karena variasi metodologi evaluasi. Kerangka evaluasi dan dataset standar lebih disukai untuk menilai berbagai metode ZSL. Sebuah studi perbandingan metode ZSL klasik disajikan dalam [6]. Berikut adalah dataset yang umum digunakan untuk evaluasi ZSL:

1. **Animal with Attributes (AwA)** : Dataset ini digunakan untuk menguji algoritma transfer learning, khususnya klasifikasi berbasis atribut [8]. Dataset ini mencakup

30.475 gambar dari 50 kelas hewan dengan enam representasi fitur untuk setiap gambar.

2. **Caltech-UCSD-Birds-200-2011 (CUB):** Dataset ini dirancang untuk tugas kategorisasi visual tingkat lanjut. Terdiri dari 11.788 gambar dari 200 subkategori burung. Setiap gambar memiliki 1 label subkategori, 15 lokasi bagian tubuh, 312 atribut biner, dan 1 bounding box. Selain itu, sepuluh deskripsi kalimat untuk setiap gambar dikumpulkan melalui Mechanical Turk oleh Amazon, dengan deskripsi yang dirancang secara hati-hati agar tidak mengandung informasi subkategori.
3. **Sun database (SUN):** Database atribut skala besar untuk adegan. Dataset ini terdiri dari 130.519 gambar dengan 899 kategori yang dapat digunakan untuk pemahaman adegan tingkat tinggi dan pengenalan adegan tingkat lanjut.
4. **Attribute Pascal and Yahoo dataset (aPY):** Dataset ini bersifat kasar, terdiri dari 15.339 gambar dari 3 kategori utama (hewan, objek, dan kendaraan), yang selanjutnya dibagi menjadi total 32 subkategori.
5. **ILSVRC2012/ILSVRC2010 (ImNet-2):** The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) mengevaluasi algoritma untuk deteksi objek dan klasifikasi gambar dalam skala besar [9]

RESUME UAS CHAPTER 12

A. ImageNet Roulette: A Case Study on Biases in Classification

Bayangkan kamu bangun pada Minggu pagi dan bermain-main dengan ponsel. kamu menemukan sebuah aplikasi yang mencoba memberikan label sarkastik dan lucu saat kamu mengunggah gambar atau selfie. Tanpa banyak berpikir, kamu mencoba aplikasi tersebut dengan mengunggah selfie, dan mengejutkan, aplikasi itu mengembalikan label yang mengkhawatirkan, seperti menyebut kamu sebagai tersangka kejahatan (yang bisa jadi sangat berbahaya atau keji). Kamu juga melihat di media sosial bahwa aplikasi yang sama memberikan label provokatif pada orang lain, meningkatkan kemungkinan profil rasial dan gender. Beberapa label bahkan menyebut orang sebagai pelaku kejahatan, menghubungkan ciri-ciri wajah dengan etnis tertentu, atau berdasarkan asal-usul seseorang. Label-label ini sering kali sangat ofensif dan berpotensi merugikan kepentingan individu atau menargetkan kelompok tertentu. Aplikasi ini menunjukkan bagaimana teknologi AI dapat memperburuk stereotip atau prasangka. Meski AI membantu membuat hidup lebih nyaman, jika tidak diawasi, AI dapat menimbulkan kerusakan besar. Oleh karena itu, nilai-nilai manusia yang inklusif harus diterapkan dalam pengembangan dan penerapan model AI untuk menghindari dampak negatif.

B. Introduction to ImageNet: A Large-Scale Dataset for Object Recognition

ImageNet adalah dataset skala besar yang dikembangkan untuk benchmark pengenalan objek. Tujuannya adalah memetakan dunia objek untuk meningkatkan kemampuan mesin dalam memahami lingkungan, suatu tugas yang secara alami lebih unggul dilakukan oleh manusia. Dataset ini merupakan salah satu upaya pertama dalam menciptakan dataset skala besar untuk pengenalan objek. Tim ImageNet mengumpulkan data gambar dari berbagai sumber di internet. Dataset asli terdiri dari sekitar 14.197.122 gambar dengan 21.841 kelas, yang disebut sebagai ImageNet-21K. Anotasi data dilakukan secara crowdsourcing melalui Amazon Mechanical Turk. Sub-kumpulan yang lebih kecil, ImageNet-1K, memiliki 1.281.167 gambar pelatihan, 50.000 gambar validasi, dan 100.000 gambar pengujian dengan 1.000 kelas. Dataset ini menjadi dasar dari ImageNet Large Scale Visual Recognition Challenge (ILSVRC), yang merupakan arena kompetisi bagi banyak perusahaan dan laboratorium visi komputer untuk meningkatkan akurasi pelabelan objek. Struktur ImageNet didasarkan pada WordNet, basis data klasifikasi kata yang dikembangkan di Universitas Princeton.

Dalam makalah “ImageNet: A large-scale hierarchical image database” memperkenalkan ImageNet sebagai dataset hierarkis skala besar untuk pengenalan objek [10]. Penulis memaparkan metodologi pengumpulan data gambar dari berbagai sumber online, anotasi data menggunakan crowdsourcing, dan bagaimana dataset ini dibangun berdasarkan struktur hierarki WordNet. ImageNet dirancang untuk memetakan dunia objek dengan presisi tinggi, menyediakan data yang kaya untuk melatih model visi komputer. Dalam penelitian ini, tim juga menjelaskan penggunaan ImageNet untuk meningkatkan akurasi pengenalan objek, terutama dalam membedakan ribuan kategori.

C. Motivation Behind ImageNet Roulette

ImageNet Roulette adalah sebuah eksperimen yang dibuat oleh Trevor Paglen dan Kate Crawford untuk menyoroti adanya bias dalam model AI, terutama dalam hal klasifikasi manusia. Model ini dilatih menggunakan subkategori "person" dari dataset ImageNet dengan framework Caffe. Eksperimen ini mengungkapkan bagaimana model AI dapat menghasilkan label yang provokatif, ofensif, atau stereotipikal, seperti "pecandu" atau "gagal." Temuan ini menimbulkan pertanyaan serius mengenai bias yang ada dalam anotasi dataset, keandalan data AI, serta potensi dampak negatif pada pengguna jika model seperti ini digunakan dalam kehidupan nyata.

Masalah utama yang ditemukan mencakup label yang menyinggung, konsep yang sulit digambarkan secara visual, kurangnya keragaman gambar dalam dataset, dan risiko privasi yang mungkin timbul. Beberapa solusi yang diusulkan termasuk melakukan anotasi manual untuk menghapus label yang tidak pantas, mengevaluasi kembali konsep yang relevan, menyeimbangkan dataset berdasarkan demografi yang lebih beragam, dan menggunakan teknik pengaburan untuk melindungi privasi individu dalam gambar. Eksperimen ini kemudian memicu diskusi etis yang lebih luas di kalangan komunitas AI, menggarisbawahi pentingnya mengembangkan model AI yang lebih inklusif, adil, dan etis agar dapat menghindari bias yang merugikan dan memastikan pengembangan teknologi yang bertanggung jawab.

D. Ethics and Bias in AI

Etika dan bias dalam AI adalah isu penting yang memengaruhi pengembangan dan penggunaan teknologi berbasis kecerdasan buatan. Studi kasus ImageNet Roulette menunjukkan bagaimana bias bawaan dalam data pelatihan dan kurangnya pengawasan dapat menghasilkan model AI yang ofensif dan berbahaya. Eksperimen ini memicu koreksi besar-besaran oleh tim ImageNet, termasuk penghapusan label yang tidak pantas dan pengaburan wajah untuk melindungi privasi. Contoh lainnya adalah teknologi deepfake, yang memiliki potensi kreatif tetapi juga dapat disalahgunakan untuk menyebarkan disinformasi dan kebencian, dengan dampak serius pada kehidupan korban. Etika AI mencakup prinsip-prinsip seperti transparansi, keadilan, tanggung jawab, dan keberlanjutan untuk memastikan pengembangan AI yang aman dan inklusif. Organisasi seperti UNESCO dan komunitas internasional telah menetapkan pedoman, seperti Hukum Robotika Asimov dan Prinsip Asilomar, untuk memitigasi risiko dan dampak negatif AI. Dengan melibatkan berbagai pemangku kepentingan, termasuk peneliti, pengembang, pemerintah, dan masyarakat umum, pengembangan AI dapat diarahkan untuk menciptakan dampak positif yang adil dan bermanfaat bagi semua.

E. Hugging Face's efforts: Ethics and Society

Hugging Face berkomitmen untuk mempromosikan pembelajaran mesin yang etis dan inklusif dengan mengedepankan prinsip Good ML, yang meliputi kolaborasi, transparansi, dan tanggung jawab. Mereka menyediakan berbagai alat seperti Model Cards untuk

membantu transparansi dalam model, ruang diskusi untuk komunitas, serta proyek edukasi yang bertujuan untuk memperluas akses pengetahuan tentang AI. Selain itu, Hugging Face juga mengkategorikan proyek-proyek mereka dalam enam kategori berdasarkan etika:

1. **Rigorous**: yang fokus pada dokumentasi dan transparansi
2. **Consentful**: yang menjaga privasi dan otonomi pengguna
3. **Socially Conscious**: yang mendukung masyarakat dengan aplikasi seperti revitalisasi bahasa asli dan aksesibilitas
4. **Sustainable**: yang mencari solusi untuk mengurangi dampak lingkungan dari teknologi AI
5. **Inclusive**: yang bertujuan untuk membawa manfaat AI ke kelompok yang terpinggirkan
6. **Inquisitive**: yang mendorong tantangan terhadap struktur kekuasaan dan ketidakadilan dalam AI. Semua upaya ini mencerminkan dedikasi Hugging Face untuk menciptakan AI yang aman, adil, dan bermanfaat bagi semua orang.

REFRENSI:

- [1] B. Romera-Paredes and P. Torr, “An embarrassingly simple approach to zero-shot learning,” in *International conference on machine learning*, PMLR, 2015, pp. 2152–2161.
- [2] W. Ren, Y. Tang, Q. Sun, C. Zhao, and Q.-L. Han, “Visual semantic segmentation based on few/zero-shot learning: An overview,” *IEEE/CAA Journal of Automatica Sinica*, 2023.
- [3] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, “Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly,” *IEEE Trans Pattern Anal Mach Intell*, vol. 41, no. 9, pp. 2251–2265, 2018.
- [4] C. Li, X. Ye, H. Yang, Y. Han, X. Li, and Y. Jia, “Generalized zero shot learning via synthesis pseudo features,” *IEEE Access*, vol. 7, pp. 87827–87836, 2019.
- [5] R. Xu, S. Shao, B. Liu, and W. Liu, “Generalized zero-shot learning based on manifold alignment,” in *2022 16th IEEE International Conference on Signal Processing (ICSP)*, IEEE, 2022, pp. 202–207.
- [6] B. Liu, L. Hu, Q. Dong, and Z. Hu, “An iterative co-training transductive framework for zero shot learning,” *IEEE Transactions on Image Processing*, vol. 30, pp. 6943–6956, 2021.
- [7] J. Zhang, G. Yang, P. Hu, G. Lin, and F. Lv, “Semantic consistent embedding for domain adaptive zero-shot learning,” *IEEE Transactions on Image Processing*, 2023.
- [8] C. H. Lampert, H. Nickisch, and S. Harmeling, “Learning to detect unseen object classes by between-class attribute transfer,” in *2009 IEEE conference on computer vision and pattern recognition*, IEEE, 2009, pp. 951–958.

- [9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, ieee, 2009, pp. 248–255.
- [10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, ieee, 2009, pp. 248–255.