resnet_mdmm_paca

July 22, 2025

0.1 In this tutorial we create a CNN and dataloaders, and train / prune the model.

[1]:
```
import os
os.environ["KERAS_BACKEND"] = "torch" # Needs to be set, some pruning layers as␣
 ↪well as the quantizers are Keras
import keras
keras.config.set_backend("torch")
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
keras.backend.set_image_data_format("channels_first")
```

[2]:
```
try:
    os.chdir("/home/das214/PQuant/mdmm_dev/src")
except:
    pass

for f in os.listdir(os.getcwd()):
    print(f)
```

```
pquant
data
smartpixels
```

[3]:
```
model = torchvision.models.resnet18()
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)
model = model.to(device)

model
```

```
Using device: cuda
```

[3]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

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

```
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

0.2 Add pruning and quantization

Begin prunning with MDMM pruning with Unstructured Sparsity metric function

[4]:
```
from pquant import get_default_config
from IPython.display import JSON

pruning_method = "mdmm"
config = get_default_config(pruning_method)
JSON(config)
```

[4]:
```
<IPython.core.display.JSON object>
```

[5]:
```
# Replace layers with compressed layers
from pquant import add_compression_layers
input_shape = (256,3,32,32)
model = add_compression_layers(model, config, input_shape)
model
```

[5]:
```
ResNet(
  (conv1): CompressedLayerConv2d(
    (pruning_layer): <MDMM name=mdmm, built=True>
  )
```

```
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_1, built=True>
      )
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_2, built=True>
      )
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_3, built=True>
      )
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_4, built=True>
      )
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_5, built=True>
      )
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_6, built=True>
      )
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
```

```
        (0): CompressedLayerConv2d(
          (pruning_layer): <MDMM name=mdmm_7, built=True>
        )
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_8, built=True>
      )
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_9, built=True>
      )
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_10, built=True>
      )
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_11, built=True>
      )
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): CompressedLayerConv2d(
          (pruning_layer): <MDMM name=mdmm_12, built=True>
        )
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_13, built=True>
      )
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_14, built=True>
      )
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_15, built=True>
      )
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_16, built=True>
      )
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): CompressedLayerConv2d(
          (pruning_layer): <MDMM name=mdmm_17, built=True>
        )
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_18, built=True>
      )
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): CompressedLayerConv2d(
        (pruning_layer): <MDMM name=mdmm_19, built=True>
      )
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): CompressedLayerLinear(
    (pruning_layer): <MDMM name=mdmm_20, built=True>
```

```
  )
)
```

[6]:
```
import torchvision.transforms as transforms
from pquant import get_layer_keep_ratio, get_model_losses
from quantizers.fixed_point.fixed_point_ops import get_fixed_quantizer
from tqdm import tqdm


def get_cifar10_data(batch_size):
    normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    train_transform = transforms.Compose([transforms.RandomHorizontalFlip(),␣
 ↪transforms.RandomCrop(32, padding=4),
                                          transforms.ToTensor(), normalize])
    test_transform = transforms.Compose([transforms.ToTensor(), normalize])
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,␣
 ↪transform=train_transform)
    valset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                       download=True, transform=test_transform)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=4,␣
 ↪pin_memory=True)

    val_loader = torch.utils.data.DataLoader(valset, batch_size=batch_size,
                                         shuffle=False, num_workers=4,␣
 ↪pin_memory=True)

    return train_loader, val_loader

# Set up input quantizer
quantizer = get_fixed_quantizer(overflow_mode="SAT")

def train_resnet(model, trainloader, device, loss_func,
                 epoch, optimizer, scheduler, *args, **kwargs):
    """
    One epoch of training with a live ETA/throughput bar.
    """
    model.train()

    with tqdm(trainloader,
              desc=f"Train || Epoch {epoch}",
              total=len(trainloader),
              unit="batch",
              dynamic_ncols=True) as pbar:

        for inputs, labels in pbar:
```

```
            inputs, labels = inputs.to(device, non_blocking=True), labels.
 ↪to(device, non_blocking=True)
            inputs = quantizer(inputs, k=torch.tensor(1.), i=torch.tensor(0.),␣
 ↪f=torch.tensor(7.))

            optimizer.zero_grad(set_to_none=True)               # cleaner␣
 ↪gradient reset
            outputs = model(inputs)
            loss = loss_func(outputs, labels)
            losses = get_model_losses(model, torch.tensor(0.).to(device))
            loss += losses
            loss.backward()
            optimizer.step()

            if scheduler is not None:
                scheduler.step()

            pbar.set_postfix(loss=f"{loss.item():.4f} ")

    # ----- Diagnostics on Last mini-batch -----
    print(f"Loss={loss_func(outputs, labels).item():.4f} | Reg={loss.item() -␣
 ↪loss_func(outputs, labels).item():.4f}")

def validate_resnet(model, testloader, device, loss_func, epoch, *args,␣
 ↪**kwargs):
    """
    Validation with progress bar and accuracy summary.
    """
    model.eval()
    correct = total = 0


    with torch.no_grad():
        with tqdm(testloader,
                  desc=f"Val   || Epoch {epoch}",
                  total=len(testloader),
                  unit="batch",
                  dynamic_ncols=True) as pbar:

            for inputs, labels in pbar:
                inputs, labels = inputs.to(device, non_blocking=True), labels.
 ↪to(device, non_blocking=True)
                inputs = quantizer(inputs, k=torch.tensor(1.), i=torch.tensor(0.
 ↪), f=torch.tensor(7.))
                outputs = model(inputs)
                _, predicted = outputs.max(1)
```

```
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()

                running_acc = 100. * correct / total
                pbar.set_postfix(acc=f"{running_acc:.2f}%")

    ratio = get_layer_keep_ratio(model)
    print(f"Accuracy: {correct/total*100:.2f}% | Remaining weights: {ratio*100:.
 ↪2f}% \n")



BATCH_SIZE = 256
train_loader, val_loader = get_cifar10_data(BATCH_SIZE)
```

[7]:
```
from torch.optim.lr_scheduler import CosineAnnealingLR

optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=0.0001,␣
 ↪momentum=0.9)
scheduler = CosineAnnealingLR(optimizer, 200)
loss_function = nn.CrossEntropyLoss()
```

[8]:
```
from pquant import iterative_train
"""
Inputs to train_resnet we defined previously are:
          model, trainloader, device, loss_func, epoch, optimizer, scheduler,␣
 ↪**kwargs
"""

trained_model = iterative_train(model = model,
                                config = config,
                                train_func = train_resnet,
                                valid_func = validate_resnet,
                                trainloader = train_loader,
                                testloader = val_loader,
                                device = device,
                                loss_func = loss_function,
                                optimizer = optimizer,
                                scheduler = scheduler
                                )
```

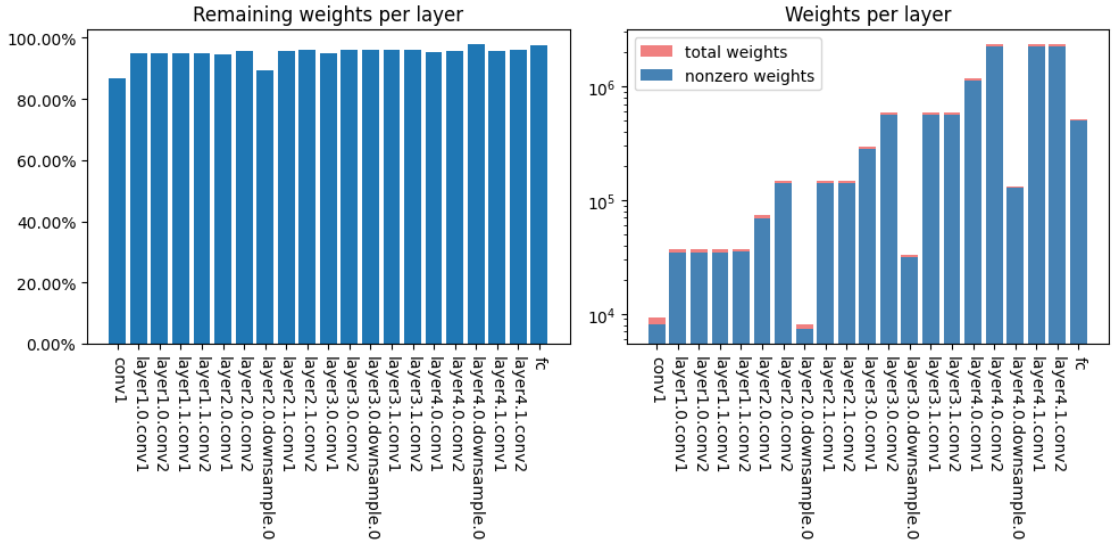
```
Train || Epoch 0: 100%|      | 196/196 [00:09<00:00, 21.43batch/s,
loss=12.3054]

Loss=1.3920 | Reg=10.9134

Val   || Epoch 0: 100%|      | 196/196 [00:05<00:00, 33.03batch/s,
acc=47.99%]
```

```
Accuracy: 47.99% | Remaining weights: 96.49%


Train || Epoch 1: 100%|      | 196/196 [00:04<00:00, 48.80batch/s,
loss=1.5023]

Loss=1.5023 | Reg=0.0000

Val   || Epoch 1: 100%|      | 196/196 [00:02<00:00, 82.07batch/s,
acc=45.43%]

Accuracy: 45.43% | Remaining weights: 95.88%
```

[9]:
```python
import numpy as np
from pquant import remove_pruning_from_model
import matplotlib.pyplot as plt
# Remove compression layers, leaves Quantized activations in place
model = remove_pruning_from_model(trained_model, config)

# Plot remaining weights
names = []
remaining = []
total_w = []
nonzeros = []
for n, m in trained_model.named_modules():
    if isinstance(m, (torch.nn.Conv1d, torch.nn.Conv2d, torch.nn.Linear)):
        names.append(n)
        nonzero = np.count_nonzero(m.weight.detach().cpu())
        remaining_pct = nonzero / m.weight.numel()
        remaining.append(remaining_pct)
        total_w.append(m.weight.numel())
        nonzeros.append(nonzero)
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].bar(range(len(names)), remaining)
ax[0].set_xticks(range(len(names)))
ax[0].set_xticklabels(names)
ax[0].tick_params(axis='x', labelrotation=270)
new_ytick = []
for i in ax[0].get_yticklabels():
    ytick = f"{float(i.get_text()) * 100:.2f}%"
    new_ytick.append(ytick)
ax[0].set_yticklabels(new_ytick)
ax[0].title.set_text("Remaining weights per layer")

ax[1].bar(range(len(nonzeros)), total_w, color="lightcoral", label="total␣
 ↪weights")
ax[1].bar(range(len(nonzeros)), nonzeros, color="steelblue", label="nonzero␣
 ↪weights")
```

```
ax[1].set_xticks(range(len(names)))
ax[1].set_xticklabels(names)
ax[1].tick_params(axis='x', labelrotation=270)
ax[1].title.set_text("Weights per layer")
ax[1].legend()
ax[1].set_yscale("log")

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_1859277/1596906618.py:29: UserWarning: set_ticklabels() should
only be used with a fixed number of ticks, i.e. after set_ticks() or using a
FixedLocator.
  ax[0].set_yticklabels(new_ytick)
```



[10]:
```
import math

def visualize_conv_patterns(model, layer_name, max_patterns_to_show=16):
    """
    Visualizes the dominant binary patterns for a specific convolutional layer.
    """
    target_module = None
    for name, module in model.named_modules():
        if name == layer_name and hasattr(module, "pruning_layer"):
            pruning_layer = module.pruning_layer
            if hasattr(pruning_layer, "metric_fn") and "PACAPattern" in␣
 ↪pruning_layer.metric_fn.__class__.__name__:
                target_module = module
                break
```

```
    if target_module is None:
        print(f"Error: Could not find a PACA-pruned layer named '{layer_name}'.
↪")
        return

    metric_fn = target_module.pruning_layer.metric_fn
    # Ensure dominant patterns are selected based on the final weights
    metric_fn._select_dominant_patterns(target_module.weight)

    patterns = metric_fn.dominant_patterns
    if patterns is None or patterns.shape[0] == 0:
        print(f"No dominant patterns found for layer '{layer_name}'.")
        return

    # Convert to NumPy for plotting
    patterns_np = keras.ops.convert_to_numpy(patterns)

    # Get original kernel shape from the weight tensor
    kernel_h, kernel_w = target_module.weight.shape[2], target_module.weight.
↪shape[3]

    num_patterns = min(patterns_np.shape[0], max_patterns_to_show)

    # Create a subplot grid for the patterns
    cols = math.ceil(math.sqrt(num_patterns))
    rows = math.ceil(num_patterns / cols)

    fig, axes = plt.subplots(rows, cols, figsize=(cols * 2, rows * 2))
    if isinstance(axes, plt.Axes):
        axes = [axes]
    else:
        axes = axes.flatten()


    fig.suptitle(f"Dominant Patterns for Layer: {layer_name} (found␣
↪{patterns_np.shape[0]})", fontsize=16)

    for i in range(num_patterns):
        pattern_2d = patterns_np[i].reshape(kernel_h, kernel_w)
        print(pattern_2d.shape)
        print(pattern_2d)
        axes[i].imshow(pattern_2d, cmap='binary', vmin=0, vmax=1)
        axes[i].set_title(f"Pattern {i+1}")
        axes[i].set_xticks([])
        axes[i].set_yticks([])
```

```
    # Hide any unused subplots
    for j in range(num_patterns, len(axes)):
        axes[j].axis('off')

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

# --- How to use it in your notebook after training is finished ---
visualize_conv_patterns(model, 'layer1.0.conv1')
```

```
Error: Could not find a PACA-pruned layer named 'layer1.0.conv1'.
```

0.3 Add PACA prunning

After pruning we will have multiple patterns, so we force all of them to have a lower num,ber of dominant patterns

[11]:
```
import yaml

with open("pquant/configs/config_mdmm_paca.yaml", 'r') as f:
    config = yaml.safe_load(f)
JSON(config)
```

[11]:
```
<IPython.core.display.JSON object>
```

[13]:
```
from pquant import add_compression_layers
input_shape = (256,3,32,32)
model = add_compression_layers(model, config, input_shape)
model
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 3
      1 from pquant import add_compression_layers
      2 input_shape = (256,3,32,32)
----> 3 model = add_compression_layers(model, config, input_shape)
      4 model

File ~/PQuant/mdmm_dev/src/pquant/core/compressed_layers.py:25, in␣
 ↪add_compression_layers(model, config, input_shape)
     20 if keras.backend.backend() == "torch":
     21     from pquant.core.torch_impl.compressed_layers_torch import (
     22         add_compression_layers_torch,
     23     )
---> 25     return add_compression_layers_torch(model, config, input_shape)
     26 else:
     27     from pquant.core.tf_impl.compressed_layers_tf import␣
 ↪add_compression_layers_tf
```

```
File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪215, in add_compression_layers_torch(model, config, input_shape)
    213 model = disable_pruning_from_layers(model, config)
    214 model = add_layer_specific_quantization_to_model(model, config)
--> 215 model(torch.rand(input_shape, device=next(model.parameters()).device))
    216 return model

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torchvision/models/resnet.py:285, in ResNet.forward(self, x)
    284 def forward(self, x: Tensor) -> Tensor:
--> 285     return self._forward_impl(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torchvision/models/resnet.py:268, in ResNet._forward_impl(self, x)
    266 def _forward_impl(self, x: Tensor) -> Tensor:
    267     # See note [TorchScript super()]
--> 268     x = self.conv1(x)
    269     x = self.bn1(x)
    270     x = self.relu(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)
```

```
File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪161, in CompressedLayerConv2d.forward(self, x)
    160 def forward(self, x):
--> 161     weight, bias = self.prune_and_quantize(self.weight, self.bias)
    162     if self.pruning_method == "wanda":
    163         self.pruning_layer.collect_input(x, weight, self.training)

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪119, in CompressedLayerBase.prune_and_quantize(self, weight, bias)
    117 else:
    118     weight, bias = self.quantize(weight, bias)
--> 119     weight = self.prune(weight)
    120 return weight, bias

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪110, in CompressedLayerBase.prune(self, weight)
    108 def prune(self, weight):
    109     if self.enable_pruning:
--> 110         weight = self.pruning_layer(weight)
    111     return weight

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:122, in filter_traceback.<locals>.
 ↪error_handler(*args, **kwargs)
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
```

```
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:515, in MDMM.
 ↪call(self, weight)
    513         weight = weight * self.get_hard_mask(weight)
    514 else:
--> 515     self.penalty_loss = self.constraint_layer(weight)
    517 return weight

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:61, in Constraint.
 ↪call(self, weight)
     59 def call(self, weight):
     60     """Calculates the penalty from a given infeasibility measure."""
---> 61     raw_infeasibility = self.get_infeasibility(weight)
     62     infeasibility = self.pipe_infeasibility(raw_infeasibility)
```

```
     64     if self.use_grad_:

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:108, in␣
 ↪EqualityConstraint.get_infeasibility(self, weight)
    107 def get_infeasibility(self, weight):
--> 108     metric_value = self.metric_fn(weight)
    109     infeasibility = metric_value - self.target_value
    110     # return ops.abs(infeasibility)

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:410, in␣
 ↪PACAPatternMetric.__call__(self, weight)
    406 if len(weight.shape) != 4:
    407     # For non-conv layers, return zero loss.
    408     return ops.convert_to_tensor(0.0, dtype=weight.dtype)
--> 410 self._select_dominant_patterns(weight)
    411 distances = self._pattern_distances(weight)
    412 min_distances = ops.min(distances, axis=1)

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:346, in␣
 ↪PACAPatternMetric._select_dominant_patterns(self, w)
    343 all_patterns = self._get_patterns_from_weights(w)
    345 # Get unique patterns and their frequencies from the new helper functio
--> 346 unique_patterns, counts =␣
 ↪self._get_unique_patterns_with_counts(all_patterns)
    348 if ops.shape(unique_patterns)[0] == 0:
    349     self.dominant_patterns = unique_patterns

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:314, in␣
 ↪PACAPatternMetric._get_unique_patterns_with_counts(self, all_patterns)
    312 num_bits = ops.shape(all_patterns)[1]
    313 device = all_patterns.device
--> 314 powers_of_2 = ops.power(2.0,␣
 ↪ops.arange(num_bits, dtype=all_patterns.dtype, device=device))
    316 hashes = ops.sum(all_patterns * powers_of_2, axis=1)
    318 # Sort hashes to group identical patterns, then find unique hashes and␣
 ↪their counts.

TypeError: Exception encountered when calling EqualityConstraint.call().

arange() got an unexpected keyword argument 'device'

Arguments received by EqualityConstraint.call():
  • weight=torch.Tensor(shape=torch.Size([64, 3, 7, 7]), dtype=float32)
```

[ ]:
```
def train_resnet(model, trainloader, device, loss_func, epoch, optimizer,␣
 ↪scheduler, *args, **kwargs):
```

```
    first_batch = True
    for data in trainloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        inputs = quantizer(inputs, k=torch.tensor(1.), i=torch.tensor(0.),␣
↪f=torch.tensor(7.)) # 8 bits input quantization
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = loss_func(outputs, labels)
        losses = get_model_losses(model, torch.tensor(0.).to(device))
        loss += losses
        loss.backward()

        if first_batch:
            print("\n ---- Checking Loss values ----")
            print("Loss:", loss_func(outputs, labels).item())
            print("Model Losses:", losses.item())
            print("-------------------------------------------------")

            for name, module in model.named_modules():
                if "conv1" in name and hasattr(module, "pruning_layer"):
                    pruning_layer = module.pruning_layer
                    if hasattr(pruning_layer, "metric_fn") and "PACAPattern" in␣
↪pruning_layer.metric_fn.__class__.__name__:
                        metric_fn = pruning_layer.metric_fn
                        num_patterns = 0
                        if metric_fn.dominant_patterns is not None:
                            num_patterns = metric_fn.dominant_patterns.shape[0]

                        total_dist = metric_fn(module.weight).item()
                        num_kernels = module.weight.shape[0]
                        avg_dist = total_dist / num_kernels if num_kernels > 0␣
↪else 0

                        print(f"--- PACA Stats for {name} at Epoch {epoch} ---")
                        print(f"Num Patterns: {num_patterns}, Avg Pattern Dist:␣
↪{avg_dist:.4f}")
                    ␣
↪print("-------------------------------------------------\n")
                        break
            first_batch = False

        optimizer.step()
        epoch += 1
        if scheduler is not None:
            scheduler.step()
```

```
def validate_resnet(model, testloader, device, loss_func, epoch, *args,␣
 ↪**kwargs):
    correct = 0
    total = 0
    num_paca_patterns = 0
    avg_paca_dist = 0.0
    model.eval()
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            inputs = quantizer(inputs, k=torch.tensor(1.), i=torch.tensor(0.),␣
 ↪f=torch.tensor(7.))
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        for name, module in model.named_modules():
            if "conv1" in name and hasattr(module, "pruning_layer"):
                pruning_layer = module.pruning_layer
                if hasattr(pruning_layer, "metric_fn") and "PACAPattern" in␣
 ↪pruning_layer.metric_fn.__class__.__name__:
                    metric_fn = pruning_layer.metric_fn
                    if metric_fn.dominant_patterns is not None:
                        num_paca_patterns = metric_fn.dominant_patterns.shape[0]

                    total_dist = metric_fn(module.weight).item()
                    num_kernels = module.weight.shape[0]
                    avg_paca_dist = total_dist / num_kernels if num_kernels > 0␣
 ↪else 0
                    break

        ratio = get_layer_keep_ratio(model)
        print(f'Accuracy: {100 * correct / total:.2f}%, '
              f'Remaining Weights: {ratio * 100:.2f}%, '
              f'Num Patterns: {num_paca_patterns}, '
              f'Avg Pattern Dist: {avg_paca_dist:.4f}')

BATCH_SIZE = 256
train_loader, val_loader = get_cifar10_data(BATCH_SIZE)
```

0.4 Create loss function, scheduler and optimizer

[ ]:
```
from torch.optim.lr_scheduler import CosineAnnealingLR

optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=0.0001,␣
 ↪momentum=0.9)
scheduler = CosineAnnealingLR(optimizer, 200)
loss_function = nn.CrossEntropyLoss()
```

0.5 Train model

Training time. We use the train_compressed_model function from pquant to train. We need to provide some parameters such as training and validation functions, their input parameters, the model and the config file. The function automatically adds pruning layers and replaces activations with a quantized variant, trains the model, and removes the pruning layers after training is done

[ ]:
```
from pquant import iterative_train
"""
Inputs to train_resnet we defined previously are:
          model, trainloader, device, loss_func, epoch, optimizer, scheduler,␣
 ↪**kwargs
"""

trained_model = iterative_train(model = model,
                                config = config,
                                train_func = train_resnet,
                                valid_func = validate_resnet,
                                trainloader = train_loader,
                                testloader = val_loader,
                                device = device,
                                loss_func = loss_function,
                                optimizer = optimizer,
                                scheduler = scheduler
                                )
```

```
 ---- Checking Loss values ----
Loss: 1.023564100265503
Model Losses: 0.0
-------------------------------------------------
--- PACA Stats for conv1 at Epoch 0 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 70.08%, Remaining Weights: 96.59%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
```

```
Loss: 0.9040290713310242
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 197 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 63.14%, Remaining Weights: 96.59%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.9618150591850281
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 394 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 73.10%, Remaining Weights: 96.58%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.9226782321929932
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 591 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 68.42%, Remaining Weights: 96.58%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.8731341361999512
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 788 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 74.76%, Remaining Weights: 96.57%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.8373536467552185
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 985 ---
```

```
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 70.21%, Remaining Weights: 96.57%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.934232771396637
Model Losses: 0.0
-------------------------------------------------
--- PACA Stats for conv1 at Epoch 1182 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 76.75%, Remaining Weights: 96.57%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.7584898471832275
Model Losses: 0.0
-------------------------------------------------
--- PACA Stats for conv1 at Epoch 1379 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 72.12%, Remaining Weights: 96.57%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.8126096129417419
Model Losses: 0.0
-------------------------------------------------
--- PACA Stats for conv1 at Epoch 1576 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 77.97%, Remaining Weights: 96.56%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.6571393609046936
Model Losses: 0.0
-------------------------------------------------
--- PACA Stats for conv1 at Epoch 1773 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
-------------------------------------------------

Accuracy: 75.92%, Remaining Weights: 96.56%, Num Patterns: 1, Avg Pattern Dist:
```

```
0.0000

 ---- Checking Loss values ----
Loss: 0.7207622528076172
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 1970 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 79.20%, Remaining Weights: 96.56%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.7187369465827942
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 2167 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 75.83%, Remaining Weights: 96.56%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.8391613960266113
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 2364 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 80.06%, Remaining Weights: 96.56%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.7094944715499878
Model Losses: 0.0
--------------------------------------------------
--- PACA Stats for conv1 at Epoch 2561 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
--------------------------------------------------

Accuracy: 76.99%, Remaining Weights: 96.55%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.7293562293052673
```

```
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 2758 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 80.78%, Remaining Weights: 96.55%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.6537322998046875
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 2955 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 79.24%, Remaining Weights: 96.55%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.5786072015762329
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 3152 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 81.20%, Remaining Weights: 96.54%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.6281647086143494
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 3349 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 80.78%, Remaining Weights: 96.54%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.6206632852554321
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 3546 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
```

```
------------------------------------------------

Accuracy: 82.38%, Remaining Weights: 96.54%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.5568910241127014
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 3743 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 82.85%, Remaining Weights: 96.54%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.5789287090301514
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 3940 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 82.67%, Remaining Weights: 96.53%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.46712976694107056
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 4137 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 82.77%, Remaining Weights: 96.53%, Num Patterns: 1, Avg Pattern Dist:
0.0000

 ---- Checking Loss values ----
Loss: 0.47319871187210083
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 4334 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------

Accuracy: 83.30%, Remaining Weights: 96.53%, Num Patterns: 1, Avg Pattern Dist:
0.0000
```

```
 ---- Checking Loss values ----
Loss: 0.49258410930633545
Model Losses: 0.0
------------------------------------------------
--- PACA Stats for conv1 at Epoch 4531 ---
Num Patterns: 1, Avg Pattern Dist: 0.0000
------------------------------------------------
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[15], line 7
      1 from pquant import iterative_train
      2 """
      3 Inputs to train_resnet we defined previously are:
      4           model, trainloader, device, loss_func, epoch, optimizer,␣
 ↪scheduler, **kwargs
      5 """
----> 7 trained_model = iterative_train(model = model,
      8                                 config = config,
      9                                 train_func = train_resnet,
     10                                 valid_func = validate_resnet,
     11                                 trainloader = train_loader,
     12                                 testloader = val_loader,
     13                                 device = device,
     14                                 loss_func = loss_function,
     15                                 optimizer = optimizer,
     16                                 scheduler = scheduler
     17                                 )

File ~/PQuant/mdmm_dev/src/pquant/core/train.py:8, in iterative_train(model,␣
 ↪config, train_func, valid_func, **kwargs)
      5 if keras.backend.backend() == "torch":
      6     from pquant.core.torch_impl.train_torch import iterative_train_torc
----> 8     return␣
 ↪iterative_train_torch(model, config, train_func, valid_func, **kwargs)
      9 else:
     10     from pquant.core.tf_impl.train_tf import iterative_train_tf

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/train_torch.py:37, in␣
 ↪iterative_train_torch(model, config, train_func, valid_func, **kwargs)
     35 train_func(model, epoch=epoch, **kwargs)
     36 model.eval()
---> 37 valid_func(model, epoch=epoch, **kwargs)
     38 post_epoch_functions(model, e, training_config["epochs"])
     39 epoch += 1
```

```
Cell In[13], line 82, in validate_resnet(model, testloader, device, loss_func,␣
 ↪epoch, *args, **kwargs)
     80 inputs, labels = inputs.to(device), labels.to(device)
     81 inputs = quantizer(inputs, k=torch.tensor(1.), i=torch.tensor(0.),␣
 ↪f=torch.tensor(7.))
---> 82 outputs = model(inputs)
     83 _, predicted = torch.max(outputs.data, 1)
     84 total += labels.size(0)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torchvision/models/resnet.py:285, in ResNet.forward(self, x)
    284 def forward(self, x: Tensor) -> Tensor:
--> 285     return self._forward_impl(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torchvision/models/resnet.py:268, in ResNet._forward_impl(self, x)
    266 def _forward_impl(self, x: Tensor) -> Tensor:
    267     # See note [TorchScript super()]
--> 268     x = self.conv1(x)
    269     x = self.bn1(x)
    270     x = self.relu(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
```

```
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪161, in CompressedLayerConv2d.forward(self, x)
    160 def forward(self, x):
--> 161     weight, bias = self.prune_and_quantize(self.weight, self.bias)
    162     if self.pruning_method == "wanda":
    163         self.pruning_layer.collect_input(x, weight, self.training)

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪119, in CompressedLayerBase.prune_and_quantize(self, weight, bias)
    117 else:
    118     weight, bias = self.quantize(weight, bias)
--> 119     weight = self.prune(weight)
    120 return weight, bias

File ~/PQuant/mdmm_dev/src/pquant/core/torch_impl/compressed_layers_torch.py:
 ↪110, in CompressedLayerBase.prune(self, weight)
    108 def prune(self, weight):
    109     if self.enable_pruning:
--> 110         weight = self.pruning_layer(weight)
    111     return weight

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:117, in filter_traceback.<locals>.
 ↪error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/layers/layer.py:936, in Layer.__call__(self, *args, **kwargs)
    934         outputs = super().__call__(*args, **kwargs)
```

```
    935 else:
--> 936     outputs = super().__call__(*args, **kwargs)
    937 # Change the layout for the layer output if needed.
    938 # This is useful for relayout intermediate tensor in the model
    939 # to achieve the optimal performance.
    940 distribution = distribution_lib.distribution()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/backend/torch/layer.py:41, in TorchLayer.forward(self, *args,␣
 ↪**kwargs)
     40 def forward(self, *args, **kwargs):
---> 41     return Operation.__call__(self, *args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:117, in filter_traceback.<locals>.
 ↪error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/ops/operation.py:58, in Operation.__call__(self, *args, **kwargs)
     53                 call_fn = self.call
     54     call_fn = traceback_utils.inject_argument_info_in_traceback(
     55         call_fn,
     56         object_name=(f"{self.__class__.__name__}.call()"),
```

```
     57     )
---> 58     return call_fn(*args, **kwargs)
     60 # Plain flow.
     61 if any_symbolic_tensors(args, kwargs):

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:156, in inject_argument_info_in_traceback.
 ↪<locals>.error_handler(*args, **kwargs)
    154 bound_signature = None
    155 try:
--> 156     return fn(*args, **kwargs)
    157 except Exception as e:
    158     if hasattr(e, "_keras_call_info_injected"):
    159         # Only inject info for the innermost failing call

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:513, in MDMM.
 ↪call(self, weight)
    511         weight = weight * self.get_hard_mask(weight)
    512 else:
--> 513     self.penalty_loss = self.constraint_layer(weight)
    515 return weight

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:117, in filter_traceback.<locals>.
 ↪error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/layers/layer.py:936, in Layer.__call__(self, *args, **kwargs)
    934         outputs = super().__call__(*args, **kwargs)
    935 else:
--> 936     outputs = super().__call__(*args, **kwargs)
    937 # Change the layout for the layer output if needed.
    938 # This is useful for relayout intermediate tensor in the model
    939 # to achieve the optimal performance.
    940 distribution = distribution_lib.distribution()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1751, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)
```

```
File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪torch/nn/modules/module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/backend/torch/layer.py:41, in TorchLayer.forward(self, *args,␣
 ↪**kwargs)
     40 def forward(self, *args, **kwargs):
---> 41     return Operation.__call__(self, *args, **kwargs)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:117, in filter_traceback.<locals>.
 ↪error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/ops/operation.py:58, in Operation.__call__(self, *args, **kwargs)
     53                 call_fn = self.call
     54     call_fn = traceback_utils.inject_argument_info_in_traceback(
     55         call_fn,
     56         object_name=(f"{self.__class__.__name__}.call()"),
     57     )
---> 58     return call_fn(*args, **kwargs)
     60 # Plain flow.
     61 if any_symbolic_tensors(args, kwargs):

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/utils/traceback_utils.py:156, in inject_argument_info_in_traceback.
 ↪<locals>.error_handler(*args, **kwargs)
    154 bound_signature = None
    155 try:
--> 156     return fn(*args, **kwargs)
    157 except Exception as e:
    158     if hasattr(e, "_keras_call_info_injected"):
    159         # Only inject info for the innermost failing call
```

```
File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:61, in Constraint.
 ↪call(self, weight)
     59 def call(self, weight):
     60     """Calculates the penalty from a given infeasibility measure."""
---> 61     raw_infeasibility = self.get_infeasibility(weight)
     62     infeasibility = self.pipe_infeasibility(raw_infeasibility)
     64     if self.use_grad_:

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:108, in␣
 ↪EqualityConstraint.get_infeasibility(self, weight)
    107 def get_infeasibility(self, weight):
--> 108     metric_value = self.metric_fn(weight)
    109     infeasibility = metric_value - self.target_value
    110     # return ops.abs(infeasibility)

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:409, in␣
 ↪PACAPatternMetric.__call__(self, weight)
    406     return ops.convert_to_tensor(0.0, dtype=weight.dtype)
    408 self._select_dominant_patterns(weight)
--> 409 distances = self._pattern_distances(weight)
    410 min_distances = ops.min(distances, axis=1)
    411 return ops.sum(min_distances)

File ~/PQuant/mdmm_dev/src/pquant/pruning_methods/mdmm.py:386, in␣
 ↪PACAPatternMetric._pattern_distances(self, w)
    384     distances = ops.sum(ops.abs(dom_patterns_exp - w_patterns_exp),␣
 ↪axis=-1)
    385 elif self.distance_metric == 'valued_hamming':
--> 386     abs_diff = ops.abs(dom_patterns_exp - w_patterns_exp)
    387     distances = ops.sum(abs_diff * ops.abs(w_kernels_exp), axis=-1)
    388 elif self.distance_metric == 'cosine':

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/ops/numpy.py:185, in abs(x)
    182 @keras_export(["keras.ops.abs", "keras.ops.numpy.abs"])
    183 def abs(x):
    184     """Shorthand for `keras.ops.absolute`."""
--> 185     return absolute(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/ops/numpy.py:175, in absolute(x)
    173 if any_symbolic_tensors((x,)):
    174     return Absolute().symbolic_call(x)
--> 175 return backend.numpy.absolute(x)

File /depot/cms/conda_envs/das214/pquant-gpu-env/lib/python3.10/site-packages/
 ↪keras/src/backend/torch/numpy.py:249, in absolute(x)
```

```
    247 if standardize_dtype(x.dtype) == "bool":
    248     return x
--> 249 return torch.abs(x)

KeyboardInterrupt:
```

[ ]:
```
visualize_conv_patterns(model, 'layer1.0.conv1')
```

```
(3, 3)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```