CoreNet

Deep neural network framework

1. Define a **Neural Network** (*very* similar to PyTorch)

```python
from corenet.modeling.models import MODEL_REGISTRY
from corenet.modeling.models.base_model import BaseAnyNNModel

class Net(BaseAnyNNModel):
    """A simple 2-layer CNN"""

    def __init__(self, opts: argparse.Namespace) -> None:
        super().__init__(opts)
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.reset_parameters(opts)  # Initialize the weights.

    def forward(self, x: torch.Tensor):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

CoreNet

Deep neural network framework

2. Define a **model config**

```yaml
dataset:
    category: classification
    name: "cifar10"
    train_batch_size0: 4
    val_batch_size0: 4
    eval_batch_size0: 1
model:
    classification:
        name: "two_layer"
        n_classes: 10
    layer:
        # Weight initialization parameters:
        conv_init: "kaiming_normal"
        linear_init: "trunc_normal"
        linear_init_std_dev: 0.02
loss:
    category: classification
    classification:
        name: cross_entropy
optim:
    name: sgd
    sgd:
        momentum: 0.9
```

CoreNet

Deep neural network framework

3. **Evaluate** your model

```python
from corenet.options.opts import get_training_arguments
from corenet.modeling import get_model

model = get_model(opts)
model.eval()

for image_path in ["assets/cat.jpeg", "assets/dog.jpeg"]:
    image = Image.open(image_path).convert("RGB")
    img_transforms = Compose([CenterCrop(600), Resize(size=(32, 32)), PILToTensor()])
    # Transform the image, normalize between 0 and 1
    input_tensor = img_transforms(image)
    input_tensor = input_tensor.to(torch.float).div(255.0)
    # add dummy batch dimension
    input_tensor = input_tensor[None, ...]

    with torch.no_grad():
        logits = model(input_tensor)[0]
        probs = torch.softmax(logits, dim=-1)
        predictions = sorted(zip(probs.tolist(), Cifar10.CLASS_NAMES), reverse=True)
        print(
            "Top 3 Predictions:",
            [f"{cls}: {prob:.1%}" for prob, cls in predictions[:3]],
        )
```