[PyTorch Operations] (CheatSheet)

1. Tensor Basics

- **Creating a Tensor**: `torch.tensor([1, 2, 3])`
- **Zeros Tensor**: `torch.zeros(3, 3)`
- **Ones Tensor**: `torch.ones(3, 3)`
- **Random Tensor**: `torch.rand(3, 3)`
- **Arange**: `torch.arange(0, 10)`
- **Linspace**: `torch.linspace(0, 10, steps=100)`
- **Eye (Identity Matrix)**: `torch.eye(3)`
- **Tensor Shape**: `tensor.size()`
- **Reshaping a Tensor**: `tensor.view(3, -1)`
- **Flattening a Tensor**: `tensor.view(-1)`
- **Squeeze a Tensor (Remove Single-Dimensional Entries)**: `tensor.squeeze()`
- **Unsqueeze (Add a Dimension)**: `tensor.unsqueeze(dim=0)`
- **Tensor to Numpy Array**: `tensor.numpy()`
- **Numpy Array to Tensor**: `torch.from_numpy(numpy_array)`
- **Tensor Data Type**: `tensor.dtype`
- **Tensor to a Specified Device**: `tensor.to(device)`
- **Copying a Tensor**: `tensor.clone()`
- **Concatenating Tensors**: `torch.cat([tensor1, tensor2], dim=0)`
- **Stacking Tensors**: `torch.stack([tensor1, tensor2])`
- **Chunking a Tensor**: `torch.chunk(tensor, chunks=3, dim=0)`
- **Splitting a Tensor**: `torch.split(tensor, split_size_or_sections=3)`
- **Tiling a Tensor**: `tensor.repeat(3,2, 1)`
- **Permute Tensor Dimensions:** `tensor.permute(2, 0, 1)`
- **Transpose a Tensor**: `tensor.t()`
- **Specific Element Indexing**: `tensor[0, 1]`
- **Slicing a Tensor**: `tensor[:, 1]`
- **Conditional Selection in Tensor**: `tensor[tensor > 0]`
- **Applying a Function Elementwise**: `torch.apply_along_axis(func, dim,tensor)`
- **Filling Tensor with a Value**: `tensor.fill_(5)`
- **In-Place Operations (Postfix with an Underscore)**: `tensor.add_(5)`

2. Mathematical Operations

- **Addition**: `torch.add(tensor1, tensor2)`

- **Subtraction**: `tensor1 - tensor2`
- **Elementwise Multiplication**: `torch.mul(tensor1, tensor2)`
- **Elementwise Division**: `torch.div(tensor1, tensor2)`
- **Matrix Multiplication**: `torch.matmul(tensor1, tensor2)`
- **Dot Product**: `torch.dot(tensor1, tensor2)`
- **Power**: `torch.pow(tensor, exponent)`
- **Square Root**: `torch.sqrt(tensor)`
- **Logarithm**: `torch.log(tensor)`
- **Exponential**: `torch.exp(tensor)`
- **Summation**: `torch.sum(tensor)`
- **Product**: `torch.prod(tensor)`
- **Mean**: `torch.mean(tensor)`
- **Median**: `torch.median(tensor)`
- **Standard Deviation**: `torch.std(tensor)`
- **Variance**: `torch.var(tensor)`
- **Max**: `torch.max(tensor)`
- **Min**: `torch.min(tensor)`
- **Absolute Value**: `torch.abs(tensor)`
- **Clamping Values**: `torch.clamp(tensor, min=-1, max=1)`

3. Linear Algebra

- **Eigenvalues and Eigenvectors**: `torch.eig(tensor, eigenvectors=True)`
- **Determinant**: `torch.det(tensor)`
- **Inverse of Matrix**: `torch.inverse(tensor)`
- **Singular Value Decomposition**: `torch.svd(tensor)`
- **QR Decomposition**: `torch.qr(tensor)`
- **Cholesky Decomposition**: `torch.cholesky(tensor)`
- **Cross Product**: `torch.cross(tensor1, tensor2)`

4. Advanced Tensor Operations

- **Diagonal Elements**: `torch.diag(tensor)`
- **Trace (Sum of Diagonal Elements)**: `torch.trace(tensor)`
- **Rank of a Tensor**: `torch.matrix_rank(tensor)`
- **Kronecker Product**: `torch.kron(tensor1, tensor2)`
- **Flatten a Tensor**: `tensor.flatten()`
- **Tensor Unfolding (Matricization)**: `tensor.unfold(dimension, size, step)`
- **Tensor Normalization**: `torch.nn.functional.normalize(tensor, p=2, dim=1)`

- **Finding Unique Elements**: torch.unique(tensor)
- **Sorting a Tensor**: torch.sort(tensor)
- **Top-k Elements**: torch.topk(tensor, k=5)
- **Comparing Two Tensors**: torch.equal(tensor1, tensor2)
- **Where Operation**: torch.where(condition, tensor1, tensor2)
- **Index Select**: torch.index_select(tensor, dim, index)

5. Random and Probability

- **Setting the Seed for Randomness**: torch.manual_seed(42)
- **Random Sampling from Uniform Distribution**: torch.rand(size)
- **Random Sampling from Normal Distribution**: torch.randn(size)
- **Random Integer Generation**: torch.randint(low=0, high=10, size=size)
- **Shuffling Elements**: tensor[torch.randperm(tensor.size(0))]
- **Bernoulli Sampling**: torch.bernoulli(tensor)

6. Gradients and Autograd

- **Enabling Gradient Tracking**: tensor.requires_grad_(True)
- **Computing Gradients**: tensor.backward()
- **Accessing the Gradient**: tensor.grad
- **Disabling Gradient Tracking**: with torch.no_grad():
- **Detaching Gradients**: tensor.detach()
- **Zeroing Gradients**: optimizer.zero_grad()

7. Neural Network Building Blocks

- **Defining a Linear Layer**: torch.nn.Linear(in_features, out_features)
- **Convolutional Layer**: torch.nn.Conv2d(in_channels, out_channels, kernel_size)
- **Pooling Layer**: torch.nn.MaxPool2d(kernel_size)
- **Non-linear Activations (ReLU, Sigmoid, etc.)**: torch.nn.ReLU()
- **Batch Normalization**: torch.nn.BatchNorm2d(num_features)
- **Dropout**: torch.nn.Dropout(p=0.5)
- **Sequential Container**: torch.nn.Sequential(torch.nn.Linear(), torch.nn.ReLU())

8. Loss Functions

- **Mean Squared Error Loss**: torch.nn.MSELoss()

- **Cross Entropy Loss**: torch.nn.CrossEntropyLoss()
- **Binary Cross Entropy Loss**: torch.nn.BCELoss()
- **L1 Loss (Absolute Error)**: torch.nn.L1Loss()
- **Negative Log Likelihood Loss**: torch.nn.NLLLoss()

9. Optimizers

- **Stochastic Gradient Descent**: torch.optim.SGD(model.parameters(), lr=0.01)
- **Adam Optimizer**: torch.optim.Adam(model.parameters(), lr=0.001)
- **Adagrad Optimizer**: torch.optim.Adagrad(model.parameters(), lr=0.01)
- **RMSprop Optimizer**: torch.optim.RMSprop(model.parameters(), lr=0.01)
- **Learning Rate Scheduling:** torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

10. Data Preprocessing and Loaders

- **Tensor Dataset**: torch.utils.data.TensorDataset(features, targets)
- **Data Loader**: torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
- **Transforms for Data Augmentation**: torchvision.transforms.RandomRotation(degrees=30)
- **Normalizing Data**: torchvision.transforms.Normalize(mean, std)

11. Model Training and Evaluation

- **Training Loop**: Loop through DataLoader and update weights with optimizer
- **Evaluation Loop**: Loop through DataLoader for validation or testing
- **Saving a Model**: torch.save(model.state_dict(), 'model.pth')
- **Loading a Model**: model.load_state_dict(torch.load('model.pth'))
- **Setting Model to Train Mode**: model.train()
- **Setting Model to Evaluation Mode**: model.eval()

12. GPU Utilization

- **Check GPU Availability**: torch.cuda.is_available()
- **Send Model to GPU**: model.to('cuda')
- **Send Data to GPU**: tensor.to('cuda')
- **Copying Data Back to CPU**: tensor.to('cpu')

13. Advanced Operations

- **Custom Autograd Function**: Define forward and backward functions
- **Using Multiple GPUs**: torch.nn.DataParallel(model)
- **Gradient Clipping**: torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)
- **Weight Initialization**: torch.nn.init.xavier_uniform_(tensor)

14. PyTorch Extensions

- **Using PyTorch Lightning for Training Abstraction**: Utilize pytorch_lightning.LightningModule
- **Using Torchvision for Pretrained Models**: Load models with torchvision.models
- **TorchText for NLP**: Preprocessing and loading text data
- **TorchAudio for Audio Processing**: Working with audio data

15. Debugging and Profiling

- **PyTorch Profiler**: with torch.profiler.profile() as prof:
- **Inspecting Tensor Values**: Print or log tensor values during debugging
- **Checking Model Summary**: Use torchsummary for model architecture and parameters

16. Visualization and Interpretation

- **TensorBoard Integration**: from torch.utils.tensorboard import SummaryWriter
- **Visualizing Model Graphs**: Use `torchviz` for visualizing computational graphs
- **Feature Map Visualization**: Extract and visualize intermediate layers
- **Activation Visualization**: Plotting activations of specific layers
- **Weights and Gradients Visualization**: Monitor weights and gradients during training

17. Working with Sequences and Time Series

- **Recurrent Neural Networks (RNN)**: torch.nn.RNN(input_size, hidden_size)
- **Long Short-Term Memory (LSTM)**: torch.nn.LSTM(input_size, hidden_size)
- **Gated Recurrent Units (GRU)**: torch.nn.GRU(input_size, hidden_size)
- **Sequence Padding**: torch.nn.utils.rnn.pad_sequence(sequences)

- **Packing Padded Sequences**: torch.nn.utils.rnn.pack_padded_sequence(input, lengths)

18. Custom Layers and Models

- **Defining Custom Layer**: Create a class inheriting from torch.nn.Module
- **Writing Forward Pass**: Define forward method for custom layers
- **Composite Model Construction**: Combining multiple layers into a single model

19. Advanced Techniques

- **Attention Mechanisms**: Implement attention in sequence models
- **Generative Adversarial Networks (GANs)**: Building generator and discriminator models
- **Transfer Learning**: Fine-tuning pretrained models
- **Multi-Task Learning**: Shared representations for multiple tasks

20. Best Practices and Tips

- **Weight Decay for Regularization**: Using weight_decay in optimizers
- **Batch Normalization Tuning**: Properly placing BatchNorm layers
- **Avoiding Overfitting**: Techniques like dropout, data augmentation
- **Hyperparameter Tuning**: Experimenting with learning rates, batch sizes, etc.

21. Miscellaneous Operations

- **Image to Tensor Conversion**: transforms.ToTensor()
- **Gradient Accumulation**: Summing gradients over multiple mini-batches
- **TorchScript for Deployment**: torch.jit.script(model)
- **Quantization for Model Compression**: torch.quantization.quantize_dynamic(model, {torch.nn.Linear}, dtype=torch.qint8)