Summary DAY 2

1 Data Handling

- **DataLoaders**: PyTorch provides Dataset and DataLoader classes to handle large datasets efficiently.
- **Custom Dataset**: Subclass `torch.utils.data.Dataset` and implement `__len__()` and `__getitem__()`.
- **DataLoader**: Used to load data in mini-batches, shuffle data, and utilize multiprocessing.
- **Built-in Datasets**: PyTorch provides datasets in `torchvision.datasets` and `torchtext.datasets`.

2 Data Augmentation

- **Purpose**: Create virtual training samples to improve model performance, especially when data is limited.
- **Techniques**: Horizontal flip, random crop, color casting, geometric distortion, translation, etc.
- Tools like `Albumentations` simplify data augmentation.
- **Error Analysis**: Identify model weaknesses (e.g., failure with small objects, rotations, or blurry images) and apply relevant augmentations.
- **Test-Time Augmentation (TTA)**: Generate multiple augmented versions of test images, pass them through the model, and average predictions to improve accuracy.

3 Transfer Learning

- **Fine-Tuning**: Adapt a pre-trained model to a new task.
- **When to Fine-Tune**:
  - Small dataset + similar distribution: Freeze feature extraction layers and fine-tune the classifier.

- Small dataset + different distribution: Use the pre-trained network as a feature extractor and train a light classifier (e.g., SVM).
- Large dataset: Fine-tune the entire network.

- **Example**: Transfer learning from ImageNet (1000 classes) to a binary classification task.

4 Ensembling

- **Purpose**: Combine predictions from multiple models to reduce errors and improve accuracy.
- **Strategies**:
  - **Bagging (Bootstrap Aggregating)**: Train multiple models on different subsets of data (e.g., Random Forest).
  - **Boosting**: Train models sequentially, focusing on errors from previous models (e.g., AdaBoost, Gradient Boosting).
- **Combining Predictions**:
  - Hard Voting: Majority vote from models.
  - Soft Voting: Average predicted probabilities from models.
  - Regression: Average predictions from all models.
- **Error Analysis**: The probability of error for an ensemble of M models is given by:

$$\rho(e) = 1 - (1-e)^M - \binom{M}{2}(1-e)^{M-1}e$$

- **Example**: For $M = 3$ and $e = 0.01$, $\rho(e) = 0.0003$.

5 Dropout

- **Purpose**: Prevent overfitting by randomly dropping neurons during training.
- **How It Works**:
  - During training, neurons are dropped with probability p.
  - During inference, all neurons are used, and activations are scaled by p to maintain consistency.
- **Example**: If $p = 0.1$, activations during inference are scaled by 0.1.
- **PyTorch Implementation**:

```
model.train()  # Enables dropout during training
model.eval()   # Disables dropout during inference
```

6 Batch Normalization

- **Purpose**: Normalize intermediate layers to improve training stability and convergence.
- **Normalization Formula**:
$$\hat{x} = \frac{x - \mu}{\sigma^2 + \epsilon}$$
where μ is the mean across the batch, σ^2 is the variance across the batch, and ϵ is a small constant for numerical stability.
- **Learnable Parameters**:
$$y = \gamma \hat{x} + \beta$$
where γ is the scale parameter and β is the shift parameter (both learnable).
- **Training vs. Inference**:
  - During training, use batch statistics.
  - During inference, use running averages of mean and variance.
- **Advantages**: Improves gradient flow, allows higher learning rates, and acts as regularization.
- **Disadvantages**: Behaves differently during training and testing, which can cause bugs.

7 Full Training Workflow

- **Initial Setup**:
  - Start with a pre-trained model if possible.
  - Define an initial architecture without regularization or augmentations.
  - Set up validation strategy and choose evaluation metrics.
  - Train the model to get a baseline score.
- **Improvement Process**:
  - Apply regularization techniques (e.g., dropout, batch normalization).
  - Perform error analysis to identify weaknesses and apply relevant augmentations.
  - Tune hyperparameters (e.g., layers, epochs, learning rate, batch size).
  - Optionally, use ensembling to boost performance.
- **Finalization**:
  - Save the optimized model for deployment.
  - Use the model for inference in real-world applications.

8 Key Formulas

- **Batch Normalization Normalization**:

$$\hat{x} = \frac{x - \mu}{\sigma^2 + \epsilon}$$

- **Learnable Parameters**:

$$y = \gamma\hat{x} + \beta$$

- **Ensembling Error Probability**:

$$\rho(e) = 1 - (1-e)^M - \binom{M}{2}(1-e)^{M-1}e$$

9 Examples

- **Data Augmentation**:

$$\text{Original Image: } \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\text{Augmented Image (Random Crop): } \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

- **Dropout**:

During Training: Drop neurons with $p = 0.5$.

During Inference: Scale activations by $p = 0.5$.

- **Batch Normalization Example**:

$$x = [1, 2, 3, 4, 5], \quad \mu = 3, \quad \sigma^2 = 2$$

$$\hat{x} = \frac{x - 3}{2 + \epsilon}$$

$$y = \gamma\hat{x} + \beta$$