# Deep Learning

## Assignment 2

## Report

Saurabh Chavan[B22EE061]

Colab:

https://colab.research.google.com/drive/1XDPFhArequjFDz0xMKO8FxR-L0H8TK8y?usp=sharing

Colab for bonus task:
https://colab.research.google.com/drive/11aMP1OifKtJxeYCBqFclW0TzkPY7vmX3?usp=sharing

## Results:

70:30 split with 25 epochs

```
Training Results:
Class Accuracy: 0.8699
Superclass Accuracy: 0.8891
Group Accuracy: 0.8789

Test Results:
Class Accuracy: 0.6922
Superclass Accuracy: 0.7161
Group Accuracy: 0.6954
```

Since two of test accuracies were just below 70. I ran up to 35 epochs:

```
Training Results:
Class Accuracy: 0.9515
Superclass Accuracy: 0.9666
Group Accuracy: 0.9591

Test Results:
Class Accuracy: 0.7291
Superclass Accuracy: 0.7445
Group Accuracy: 0.7282
```

80:20 on 25 epochs

```
Training Results:
Class Accuracy: 0.8681
Superclass Accuracy: 0.8916
Group Accuracy: 0.8761

Test Results:
Class Accuracy: 0.6063
Superclass Accuracy: 0.6326
Group Accuracy: 0.6053
```

These test accuracies were reached around epoch 17-18, after that the test accuracies kept swinging slightly.

90:10 on 25 epochs

```
Training Results:
Class Accuracy: 0.8854
Superclass Accuracy: 0.9040
Group Accuracy: 0.8913

Test Results:
Class Accuracy: 0.6217
Superclass Accuracy: 0.6458
Group Accuracy: 0.6178
```

## Explanation of the Model

The CIFAR-100 dataset consists of 60,000 images, each of size 32×32×3, belonging to 100 fine-grained classes. These classes are organized into 20 superclasses and 9 broader groups.

**Data Loading and Augmentation**

- The dataset is loaded and merged from the train and test splits to ensure uniform distribution.

- The data is shuffled to avoid biases due to initial dataset splits.
- The images are normalized by scaling pixel values to the range [0, 1].

**Hierarchical Label Mapping**

- A **superclass mapping** is created by grouping each fine class into one of 20 superclasses.
- A **group mapping** assigns each superclass to one of 9 broader groups based on semantic similarity.
- Labels are converted into one-hot encoding for multi-output classification.

## Model Architecture

The model is implemented using TensorFlow and Keras. The CNN backbone is designed with progressively increasing depth and regularization techniques to enhance learning.

**1 Convolutional Backbone**

The backbone consists of:

- Three convolutional blocks with increasing filter sizes (64 → 128 → 256)
- Batch normalization for stabilization
- Max pooling for spatial downsampling
- Dropout layers for regularization
- A fully connected layer with 1024 neurons and L2 regularization

**2 Multi-Head Output**

The final feature representation is fed into three separate fully connected layers for different classification levels:

1. **Class Output (100 classes)** - Softmax activation
2. **Superclass Output (20 superclasses)** - Softmax activation
3. **Group Output (9 groups)** - Softmax activation

## Training and Optimization

**1 Loss Function and Weights**

- The model uses **categorical cross-entropy loss** for all three classification tasks.
- Weighted loss functions (0.4, 0.3, 0.3) ensure balanced learning across hierarchical levels.

**2 Optimizer and Learning Rate Scheduling**

- Adam optimizer with an **Exponential Decay Learning Rate Schedule** is employed.
- The learning rate starts at 0.001 and decays over time.

**3 Early Stopping and Hyperparameters**

- Training runs for **25 epochs**.
- Batch size is set to **128** to balance memory efficiency and training speed.
- **Early stopping** monitors validation loss and restores the best weights.

Bonus Task

```
Epoch 20 Results:
Training - Loss: 0.323, Accuracy: 97.16%, Super-Class Acc: 97.34%, Group Acc: 97.57%
Validation - Loss: 7.616, Accuracy: 47.48%, Super-Class Acc: 49.61%, Group Acc: 53.44%
Training completed!
```

**Loss Function Calculation in the Code**

The code defines a custom loss function, **HierarchicalCrossEntropyLoss**, which extends the standard **CrossEntropyLoss** to incorporate hierarchical relationships between classes. The loss calculation follows these steps:

1. **Compute Base Loss**:
    - Uses nn.CrossEntropyLoss(reduction='none') to compute the standard cross-entropy loss for each sample.
2. **Get Predictions**:
    - The model's output is converted to predicted class indices using torch.max(outputs, 1).
3. **Determine Hierarchical Relationships**:

- The predicted and target labels are mapped to their respective **superclass** and **group** levels using pre-defined mappings.

4. **Apply Multipliers Based on Hierarchy**:
    - If the predicted and actual labels belong to different **superclasses** but the same **group**, the loss is **doubled**.
    - If they belong to different **groups**, the loss is **quadrupled**.
    - Otherwise, the original loss is used.
5. **Compute Final Loss**:
    - The modified loss values are averaged over all samples in the batch.

This approach penalizes misclassifications more severely when they occur across broader hierarchical boundaries, encouraging the model to respect the class structure.