**Background and Method Introduction:**

**Overview of CNN:**

Convolutional Neural Networks (CNNs) are a type of deep learning architecture specifically designed for processing data with a grid-like structure, such as images. Inspired by the structure of the animal visual cortex, CNNs excel at automatically extracting features and patterns from images for various tasks, including image classification.

**Key Components:**

1. **Convolutional Layers:** These layers apply filters to the input image, identifying edges, shapes, and other low-level features. The filters slide across the image, extracting these features at different locations.
2. **Pooling Layers:** These layers downsample the data, reducing its dimensionality while preserving important features. This helps control overfitting and computational cost.
3. **Fully Connected Layers:** These layers function similarly to traditional neural networks, taking the extracted features and using them to learn higher-level, more abstract concepts.
4. **Activation Functions:** These functions introduce non-linearity into the network, allowing it to learn complex relationships between features.

**How CNNs work for Image Classification:**

1. **Preprocessing:** Images are often resized and normalised before feeding them into the CNN.
2. **Feature Extraction:** The convolutional layers extract features from the image, progressively building more complex representations from edges and shapes to objects and parts of objects.
3. **Classification:** The fully connected layers take the extracted features and learn to classify the image into a specific category (e.g., cat, dog, car).

**Applications of CNNs in Image Classification:**

1. **Self-driving cars:** CNNs are crucial for real-time object detection and classification, enabling cars to identify pedestrians, vehicles, and traffic signs.
2. **Facial recognition:** CNNs power facial recognition systems used for security, social media tagging, and photo organisation.
3. **Medical image analysis:** CNNs aid in analysing medical images like X-rays and MRIs for disease detection and classification.
4. **Content moderation:** CNNs help identify inappropriate content on social media platforms by classifying images.
5. **Product recommendation systems:** CNNs can analyse product images to recommend similar or complementary items to users.

**Dataset and Tasks Description:**

**About CIFAR-10 Dataset:**

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks in machine learning. It consists of 60,000 colour images, evenly distributed across 10 different object classes i.e., aeroplane, automobile (excluding trucks), bird, cat, deer, dog, frog, horse, ship, truck. Each image is 32x32 pixels in size and represented in RGB format. The dataset is further split into 50,000 training images and 10,000 test images. Additionally, there are 10,000 labelled images from the same distribution used for validation.

**Tasks:**

1. Simpler CNN Architecture:

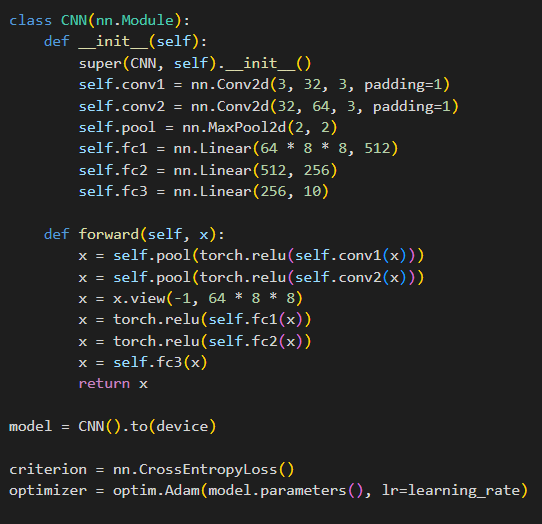
* Data Loading and Preprocessing:Loads CIFAR-10 dataset and applies transformations (ToTensor, Normalise).
* Model Architecture: Two convolutional layers with ReLU activation and max pooling layer for downsampling. Two fully-connected layers with ReLU activation (except final layer).
* Training Process: Used cross-entropy loss and Adam optimizer. Trains for a set number of epochs (10).
* Evaluation: Calculates accuracy on the test set.

2: Improved CNN Architecture:

* This is built upon the first model with enhancements.
* Data Splitting: Splits training data into training and validation sets.
* Model Architecture: Similar to the first model, but includes BatchNorm layers after convolutions.
* Training Process: Uses lower learning rate and L2 weight decay for regularisation. Implements early stopping based on validation loss.
* Evaluation:Calculates accuracy on the test set.

**Algorithms Used:**

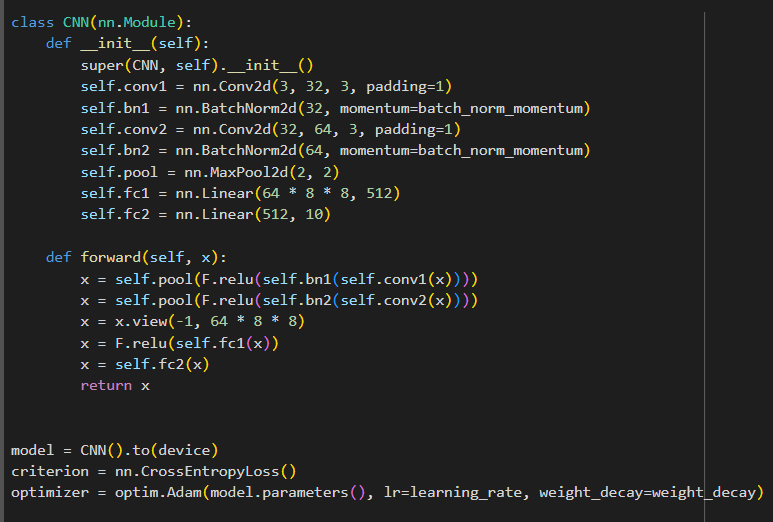
**Simpler CNN Architecture:**

****

**Fig.1. Simpler CNN Architecture**

* Data Loading and Preprocessing:
  + torchvision.datasets.CIFAR10 class loads the dataset.
  + transforms.Compose combines preprocessing steps:
    - transforms.ToTensor(): Converts images from NumPy arrays to PyTorch tensors.
    - transforms.Normalize(): Normalizes pixel values between -1 and 1 for better training.
* Model Architecture (CNN class):
  + Convolutional Layers:
    - Two convolutional layers with 3x3 kernels:
    - First layer: 3 input channels (RGB) -> 32 output channels.
    - Second layer: 32 input channels -> 64 output channels.
    - Padding of 1 ensures the output remains the same size as the input.
    - ReLU activation function introduces non-linearity.
  + Pooling Layer:
    - nn.MaxPool2d layer with a kernel size of 2x2 performs downsampling, reducing image dimensions.
  + Fully-Connected Layers:
    - Flattens the output from the convolutional layers (64 channels, 8x8 spatial dimensions).
    - Two fully-connected layers:
      * First layer: 64 \* 8 \* 8 -> 512 units.
      * Second layer: 512 units -> 10 output units (one for each class).
    - ReLU activation function applied for non-linearity (except for the final output layer).
* Training Process:
  + Uses nn.CrossEntropyLoss for classification tasks with multiple classes.
  + Employs optim.Adam optimizer for efficient parameter updates.
  + Trains for a specified number of epochs (iterations over the entire dataset).
* Evaluation:
  + Calculates accuracy on the test set to assess the model's generalisation ability.

**Improved CNN Architecture:**

****

**Fig. 2. Improved CNN Architecture**

**Enhancements:**

* **Batch Normalisation:**
  + Introduces nn.BatchNorm2d layers after each convolutional layer.
  + Batch normalisation helps stabilise the training process and potentially improve performance.
* **Early Stopping:**
  + Monitors validation loss during training.
  + Stops training if validation loss doesn't improve for a certain number of epochs (patience).
  + Prevents overfitting and improves generalisation.
* **Learning Rate and Weight Decay:**
  + Uses a lower learning rate (0.0001) compared to the first model (0.001) as parameter tuning.
  + Introduces L2 weight decay (weight\_decay) to penalise large weight values and avoid overfitting.
* **Data Splitting:**
  + Splits the training data into training and validation sets (80%/20%).
  + Uses the validation set to monitor performance during training and implement early stopping.
* **Model Architecture (CNN class):**
  + The architecture remains similar to the first model, with the addition of BatchNorm layers after each convolutional layer.
  + Overall, the second CNN architecture incorporates techniques to improve training stability, reduce overfitting, and enhance model performance.

**Classification Results:**

|  | **SImple CNN Architecture** | **Enhanced CNN Architecture** |
| --- | --- | --- |
| **Test Accuracy** | 65.28% | 67.27% |

From the above classification results, the initial, simpler CNN architecture achieved a test accuracy of 65.28%, indicating it learned to classify the images in the CIFAR-10 dataset with moderate success. By incorporating improvements like Batch Normalisation, early stopping, and regularisation, the enhanced CNN architecture reached a test accuracy of 67.27%. This increase suggests that the enhancements were effective in improving the model's ability to learn and generalise better.

**Methods of Improvement:**

**Batch Normalisation:**

* Introduced nn.BatchNorm2d layers after each convolutional layer.
* This technique helps normalise the activations of neurons in each layer, stabilising the training process and potentially improving the speed of convergence. By ensuring the data distribution remains consistent across layers, it allows for a higher learning rate and faster training.

**Early Stopping:**

* Monitored validation loss during training.
* Early stopping prevents overfitting by stopping training if the model's performance on unseen validation data doesn't improve for a predefined number of epochs (patience). This technique helps to avoid the model becoming overly specific to the training data and allows it to generalise better to unseen test data.

**Learning Rate and Weight Decay:**

* Reduced the learning rate (0.0001) compared to the first model (0.001).
* A lower learning rate can help the model converge to a better minimum by taking smaller steps during optimization. This can be particularly beneficial for complex models or when dealing with sensitive parameters.
* Introduced L2 weight decay (weight\_decay) to penalise large weight values during training.
* Weight decay acts as a form of regularisation, helping to prevent overfitting by discouraging the model from relying too heavily on a small number of features or weights.