Neural Network Construction:

1. Input Layer (Convolutional):

    - Input channels: 3 (RGB channels)
    - Output channels: 32
    - Kernel size: 3x3
    - Padding: 1 (to preserve spatial dimensions)
    - Activation: ReLU
    - This layer extracts 32 feature maps from the input image.

2. Hidden Layer (Convolutional):

    - Input channels: 32
    - Output channels: 64
    - Kernel size: 3x3
    - Stride: 1
    - Padding: 1
    - Activation: ReLU
    - Max Pooling: 2x2 with a stride of 2
    - Dropout: 50% dropout rate
    - This layer increases the depth of feature maps and reduces spatial dimensions through max pooling, promoting feature extraction and spatial hierarchy.

3. Hidden Layer (Convolutional):

    - Input channels: 64
    - Output channels: 128
    - Kernel size: 3x3
    - Stride: 1
    - Padding: 1
    - Activation: ReLU
    - Max Pooling: 2x2 with a stride of 2
    - Dropout: 50% dropout rate
    - Similar to the previous layer, this layer further increases feature depth and reduces spatial dimensions.

4. Hidden Layer (Convolutional):

    - Input channels: 128
    - Output channels: 256
    - Kernel size: 3x3
    - Stride: 1
    - Padding: 1
    - Activation: ReLU
    - Max Pooling: 2x2 with a stride of 2

- Dropout: 50% dropout rate
- This layer further deepens the feature representation and reduces spatial dimensions to capture high-level patterns.

5. Fully Connected Layers:

    - The output from the convolutional layers is flattened into a 1D tensor.
    - Fully connected layers with ReLU activation are used for feature aggregation and classification.
    - Dropout layers with a dropout rate of 50% are employed to prevent overfitting.
    - The final fully connected layer has 10 output units corresponding to the 10 classes in the CIFAR-10 dataset.

6. Model Output:

    - The model outputs the class probabilities for each image using softmax activation.

```python
class Cifar10CnnModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.device = device  # Store the device (CPU or GPU) on which the model will be trained
        self.network = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),  # First convolutional layer: input channels=3, output channels=32, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),  # Second convolutional layer: input channels=32, output channels=64, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.MaxPool2d(2, 2),  # Max pooling layer: kernel size=2x2, stride=2
            nn.Dropout(0.5),  # Dropout layer with dropout rate of 0.5 (to prevent overfitting)

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),  # Third convolutional layer: input channels=64, output channels=128, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),  # Fourth convolutional layer: input channels=128, output channels=128, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.MaxPool2d(2, 2),  # Max pooling layer: kernel size=2x2, stride=2
            nn.Dropout(0.5),  # Dropout layer with dropout rate of 0.5

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),  # Fifth convolutional layer: input channels=128, output channels=256, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),  # Sixth convolutional layer: input channels=256, output channels=256, kernel size=3x3, padding=1
            nn.ReLU(),  # ReLU activation function
            nn.MaxPool2d(2, 2),  # Max pooling layer: kernel size=2x2, stride=2
            nn.Dropout(0.5),  # Dropout layer with dropout rate of 0.5

            nn.Flatten(),  # Flatten the output from convolutional layers into a 1D tensor
            nn.Linear(256*4*4, 1024),  # Fully connected layer: input features=256*4*4, output features=1024
            nn.ReLU(),  # ReLU activation function
            nn.Dropout(0.5),  # Dropout layer with dropout rate of 0.5

            nn.Linear(1024, 512),  # Fully connected layer: input features=1024, output features=512
            nn.ReLU(),  # ReLU activation function
            nn.Linear(512, 10)  # Output layer: fully connected layer with 10 output units
        )
```

The architecture is designed to balance model complexity and generalization capability, with convolutional layers extracting hierarchical features and fully connected layers performing classification based on the learned features. Dropout layers are utilized to mitigate overfitting, ensuring better generalization to unseen data. Overall, the architecture aims to achieve high classification accuracy on the CIFAR-10 dataset while maintaining computational efficiency.

Calculation of the number of weights for each layer of the CNN:

First Convolutional Layer (Conv2d):
Input channels: 3
Output channels: 32
Kernel size: 3x3
Padding: 1
Number of weights: (3 * 3 * 3 + 1) * 32 = 896

Second Convolutional Layer (Conv2d):
Input channels: 32
Output channels: 64
Kernel size: 3x3
Padding: 1
Number of weights: (32 * 3 * 3 + 1) * 64 = 18,496

Third Convolutional Layer (Conv2d):
Input channels: 64
Output channels: 128
Kernel size: 3x3
Padding: 1
Number of weights: (64 * 3 * 3 + 1) * 128 = 73,856

Fourth Convolutional Layer (Conv2d):
Input channels: 128
Output channels: 128
Kernel size: 3x3
Padding: 1
Number of weights: (128 * 3 * 3 + 1) * 128 = 147,584

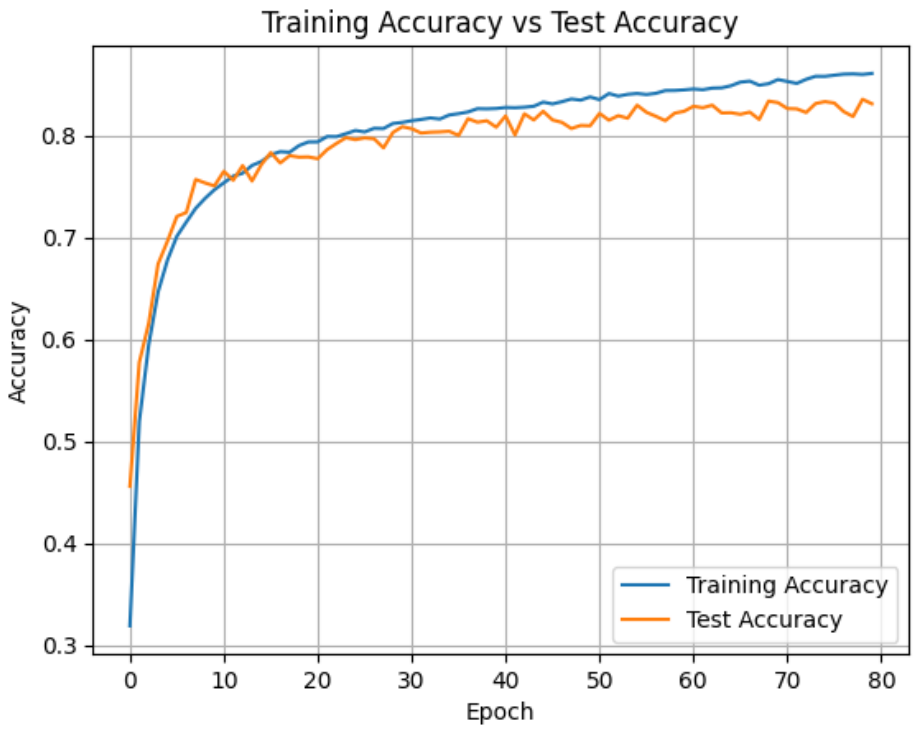
Fifth Convolutional Layer (Conv2d):
Input channels: 128
Output channels: 256
Kernel size: 3x3
Padding: 1
Number of weights: (128 * 3 * 3 + 1) * 256 = 295,168

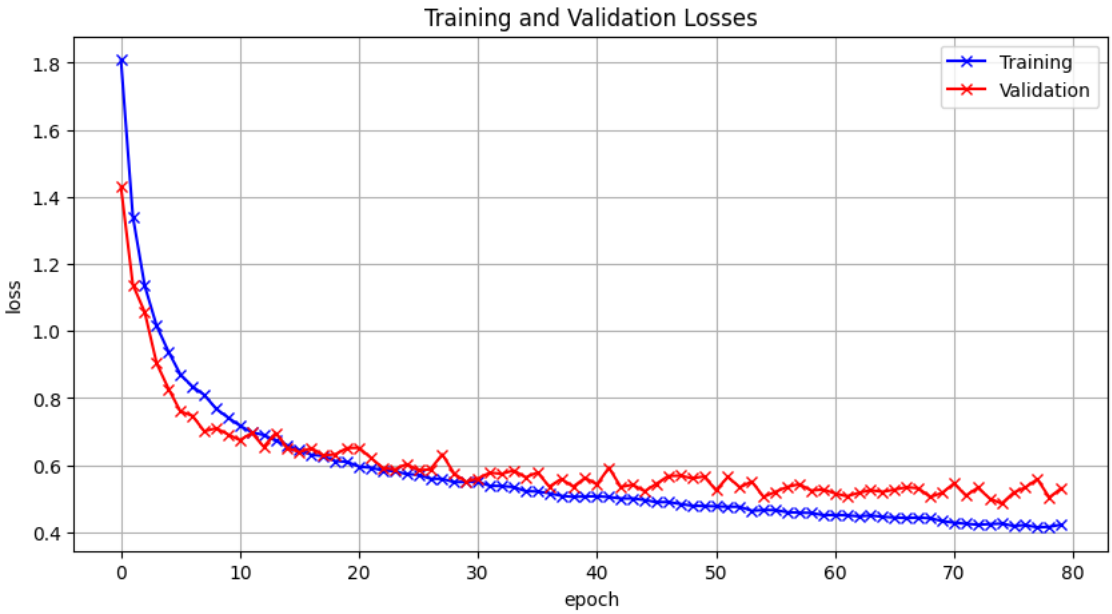
Sixth Convolutional Layer (Conv2d):
Input channels: 256
Output channels: 256
Kernel size: 3x3
Padding: 1
Number of weights: (256 * 3 * 3 + 1) * 256 = 590,080

First Fully Connected Layer (Linear):
Input features: 256 * 4 * 4
Output features: 1024
Number of weights: (256 * 4 * 4 + 1) * 1024 = 4,195,328

Second Fully Connected Layer (Linear):
Input features: 1024
Output features: 512
Number of weights: (1024 + 1) * 512 = 524,800

Output Layer (Linear):
Input features: 512
Output features: 10
Number of weights: (512 + 1) * 10 = 5,130





The graph illustrates the learning progress of my model during training and its predictive performance on new, unseen data. As I trained the model over multiple epochs, it gradually improved its understanding of the training images, leading to a rise in training accuracy. Simultaneously, we evaluated its performance on previously unseen images, reflected in the test accuracy.

Ideally, we aim for both training and test accuracies to increase and converge, indicating effective learning without over-reliance on the training data. In this case, the final test accuracy stands at approximately 83.12%, indicating that the model adeptly classifies unseen images while avoiding overfitting, even with the extended training duration.

Overfitting prevention:

Here ill show the implemented techniques to prevent overfitting in my CNN model. Overfitting occurs when a model learns to memorize the training data instead of learning to generalize from it, resulting in poor performance on unseen data. To prevent overfitting, I employed the following techniques:

Dropout: Dropout is a regularization technique commonly used to prevent overfitting in neural networks. It works by randomly dropping a fraction of the neurons during training, forcing the network to learn redundant representations and making it more robust. In my implementation, I added nn.Dropout layers with a dropout rate of 0.5 after each convolutional layer.

```
nn.Dropout(0.5),  # Dropout layer with dropout rate of 0.5
```

Data Augmentation: Data augmentation is a technique used to artificially increase the size of the training dataset by applying random transformations to the input data, such as random rotations, translations, flips, and scaling. This helps the model generalize better by exposing it to more variations of the input data. In my implementation, I applied random horizontal flips and random crops to the training images using the transforms.RandomHorizontalFlip and transforms.RandomCrop transformations, respectively.

```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Impact on Generalization Ability:

- **Dropout:** By randomly dropping neurons during training, dropout prevents the network from relying too heavily on specific features or neurons, thus reducing the likelihood of overfitting. It encourages the network to learn more robust features and prevents the model from memorizing the training data. This regularization technique helps improve the model's generalization ability and makes it more resilient to noise in the input data.
- **Data Augmentation:** By generating additional training examples through random transformations, data augmentation helps expose the model to a wider range of variations in the input data. This allows the model to learn more diverse and invariant features, leading to better generalization performance. Data augmentation acts as a form of regularization by introducing variability into the training data and helps the model generalize better to unseen examples.

Part B:

1.

$$\frac{\partial g}{\partial a} = \begin{cases} 1 & , a > 0 \\ \frac{1}{2} e^{a} & , a \leq 0 \end{cases}$$

2. $$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = (\hat{y} - y) \cdot h$$

3. $$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial h} \cdot \frac{\partial h}{\partial w_1} = (\hat{y} - y) \cdot w_2 \cdot g'(w_1 x + b_1) \cdot x$$

$$\begin{cases} (\hat{y} - y) \cdot w_2 \cdot x & , w_1 x + b_1 > 0 \\ (\hat{y} - y) \cdot w_2 \cdot 0.5 \cdot e^{w_1 \cdot x + b_1} \cdot x & , w_1 x + b_1 \leq 0 \end{cases}$$

4.

$$w_1 = w_1 - a \cdot \sum_{b=1}^{n} \frac{\partial L}{\partial w_1}(x_b)$$

5.

לא, כאילו את לה בחישוב המימדים:

$$x \in R^{d}, \; W_1 \in R^{k \times d}, \; b_1 \in R^{k}, \; h \in R^{k}, \; W_2 \in R^{c \times k}, \; b_2 \in R^{c},$$
$$z \in R^{c}, \; \hat{y} \in R^{c}$$

בא יהיה הגיוני כמות הקלאסים אליהם אנו נסווג, אז יכול להיות מוגדר באופן שרירותי (שכן הוא hyper-parameter) שכאן הוא מסומן ב-K. K אלמו לא חייב להיות באותו מימד של כמות הקלאסים.

ב.

גמישות:

רשתות CNN יכולות להתמודד עם מגוון רחב של משימות, כגון סיווג תמונות, זיהוי עצמים וחילוץ תכונות. לעומת זאת, רשתות FCN מוגבלות יותר ביכולותיהן ומתמקדות בעיקר במשימות של סגמנטציה תמונה.

רשתות CNN מתאימות גם למשימות בהן הפלט הוא תוית יחידה או מחלקה עבור תמונת קלט, בעוד שרשתות FCN מתמקדות בדרך כלל בחיזוי ברמת הפיקסל.

יעילות:

רשתות CNN חוסכות משאבים משמעותיים בהשוואה לרשתות FCN. הן בעלות פחות פרמטרים ודרישות חישוב נמוכות יותר, הן בשלב האימון והן בשלב ההסקה.

יתרון זה הופך את רשתות CNN למועדפות במצבים בהם משאבי המחשוב מוגבלים או כאשר מתמודדים עם מאגרי נתונים גדולים.

קלות יישום:

רשתות CNN בעלות מבנה פשוט יחסית, קל יותר להבין ולתחזק. לעומת זאת, רשתות FCN מורכבות יותר וקשות יותר ליישום.

זמינים מודלים רבים של רשתות CNN מאומנות מראש, המאפשרים העברת למידה והתאמה למשימות ספציפיות תוך מאמץ מינימלי.

.3

א. למה משתמשים בData augmentation?

הגדלת נתונים היא טכניקה המשמשת להגדלת כמות הנתונים הזמינים עבור מערכת למידת מכונה.
במקרה של תמונות, הגדלת נתונים יכולה לשמש למטרות הבאות:

שיפור ביצועי המודל: הגדלת נתונים יכולה לעזור למודל ללמוד דפוסים בצורה יעילה יותר, להיות עמיד יותר לרעש ולשפר את הביצועים שלו על נתונים חדשים.
הפחתת הטיית המודל: הגדלת נתונים יכולה לעזור להבטיח שהמודל ייחשף למגוון רחב של דוגמאות, מה שיכול להפחית את הטיית המודל ולשפר את הכלליות שלו.
הוזלת עלויות איסוף נתונים: הגדלת נתונים יכולה להפחית את הצורך באיסוף נתונים חדשים, מה שיכול לחסוך זמן ועלויות.

ב. שיטות להגדלת נתונים:

היפוך: היפוך תמונה אנכית או אופקית.
סיבוב: סיבוב תמונה בזווית אקראית.
חיתוך: חיתוך חלקים אקראיים מתוך תמונה.

שיטות נוספות להגדלת נתונים:
שינוי קנה מידה , הוספת רעש, שינוי תאורה, שינוי צבע, הוספת טקסט וערבוב תמונות.