Computer Vision and Artificial Intelligence Coursework Part 3 Report

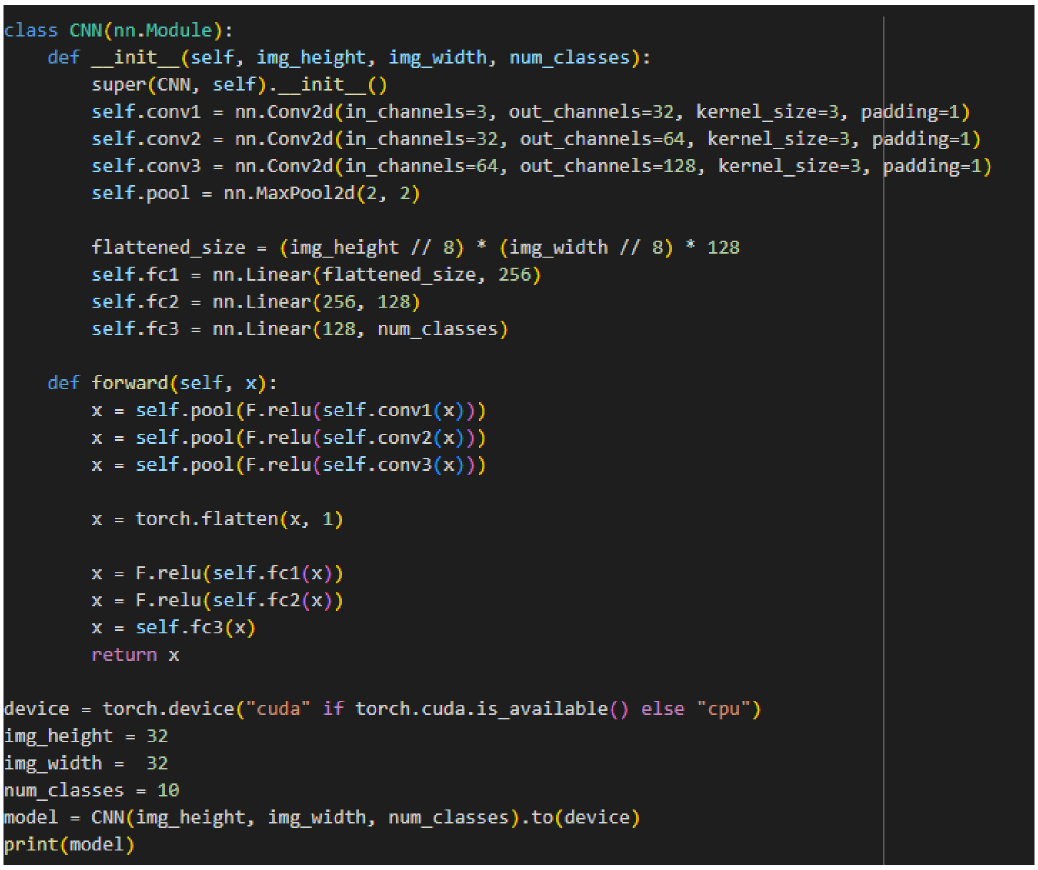
Task 1: Early Stopping a CNN on CIFAR-10 Dataset

Objective of task:

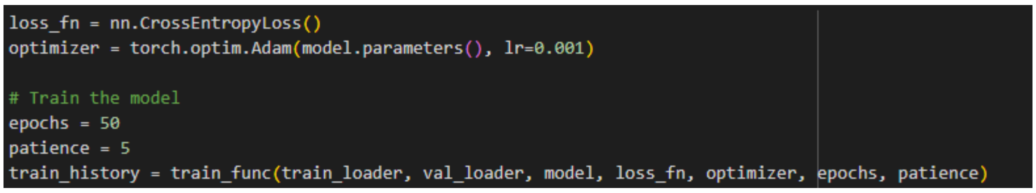
To train a Convolutional Neural Network (CNN) on the CIFAR-10 dataset using early stopping to avoid overfitting.

Model Architecture:

This model uses three convolutional layers and three fully connected layers. The following class definition was used to create the model in Pytorch:



```python
class CNN(nn.Module):
    def __init__(self, img_height, img_width, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        flattened_size = (img_height // 8) * (img_width // 8) * 128
        self.fc1 = nn.Linear(flattened_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        x = torch.flatten(x, 1)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
img_height = 32
img_width =  32
num_classes = 10
model = CNN(img_height, img_width, num_classes).to(device)
print(model)
```



```python
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train the model
epochs = 50
patience = 5
train_history = train_func(train_loader, val_loader, model, loss_fn, optimizer, epochs, patience)
```

The hyperparameters used include:

- Learning Rate: 0.001
- Optimizer: Adam
- Loss Function: CrossEntropyLoss

- Epochs: 50
- Patience: 5

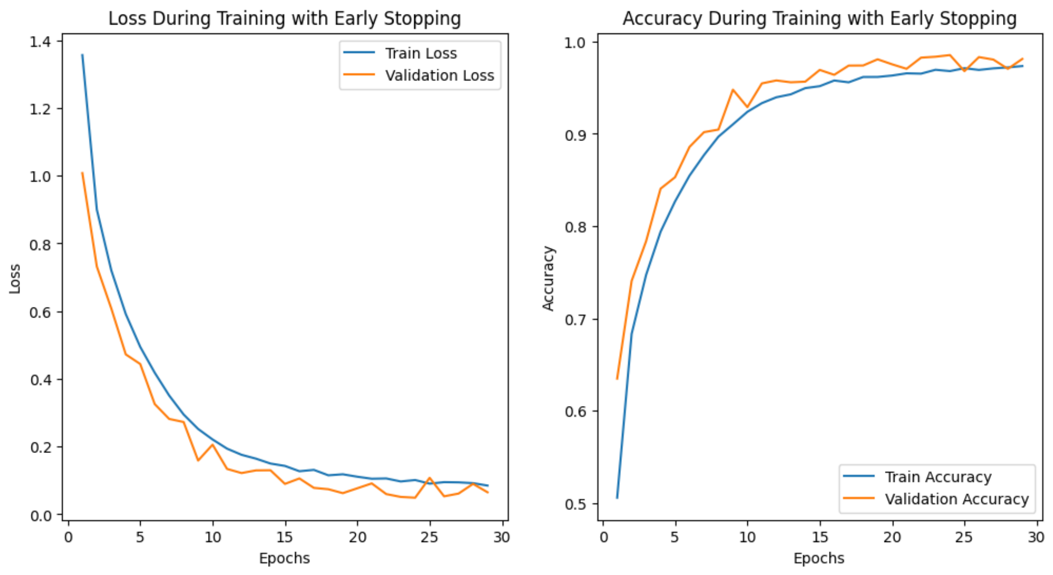
We use the Early Stopping technique to terminate or end the training when the performance of validation does not improve to avoid overfitting.

The model can track the training loss and accuracy, and validation loss and accuracy. Validation loss is the main cause for early stoppage.

After each epoch, the model saves state if the value of the validation loss is improving compared to the previous best value. If it does not improve, for a specified number of epochs, the training stops and restores the best performing model.

Visualizations:

Given below is the convergence graph of the model training and validation losses and accuracy during Early Stopping:



Observations:

Training loss decreases consistently, showing that the model is learning from the data. Validation loss also decreases initially and then starts to fluctuate and then, in this case, stops after reaching epoch 29 where validation loss no longer improves.

The training and validation accuracy follows the reverse of the loss training by increasing over the epochs instead.

There is no significant divergence between training and validation loss or accuracy, proving that the model does not overfit.

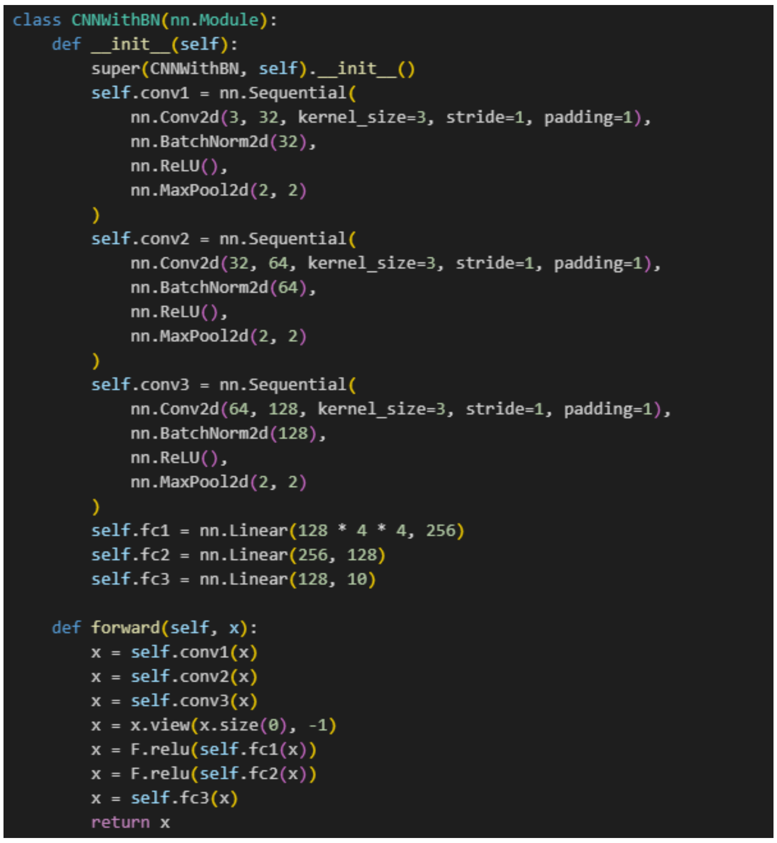
Task 2: Performance Analysis of CNN with and without Batch Normalisation

Objective:

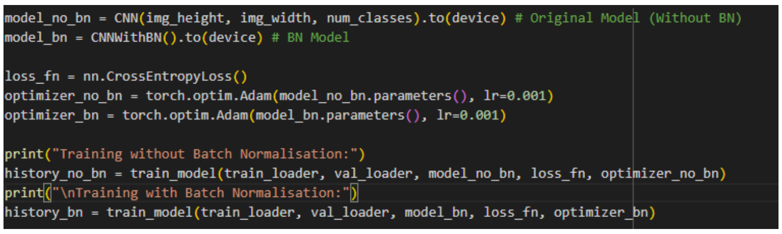
To train a CNN on the CIFAR-10 dataset with Batch Normalisation.

Model Architecture:

This model also uses three convolutional layers followed by Batch Normalisation and three fully connected layers.



```python
class CNNWithBN(nn.Module):
    def __init__(self):
        super(CNNWithBN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



```python
model_no_bn = CNN(img_height, img_width, num_classes).to(device) # Original Model (Without BN)
model_bn = CNNWithBN().to(device) # BN Model

loss_fn = nn.CrossEntropyLoss()
optimizer_no_bn = torch.optim.Adam(model_no_bn.parameters(), lr=0.001)
optimizer_bn = torch.optim.Adam(model_bn.parameters(), lr=0.001)

print("Training without Batch Normalisation:")
history_no_bn = train_model(train_loader, val_loader, model_no_bn, loss_fn, optimizer_no_bn)
print("\nTraining with Batch Normalisation:")
history_bn = train_model(train_loader, val_loader, model_bn, loss_fn, optimizer_bn)
```

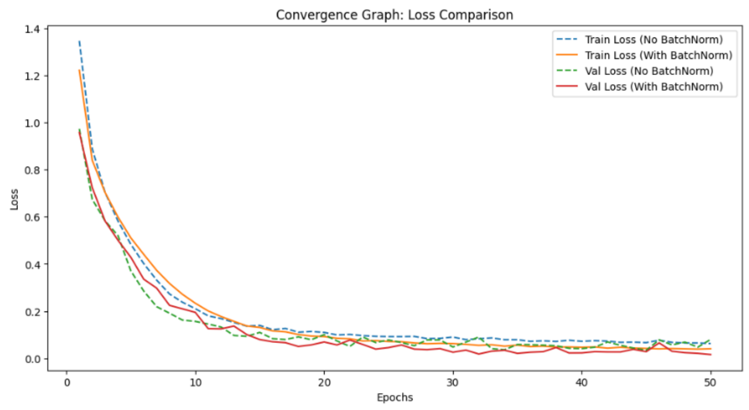
Hyperparameters used:

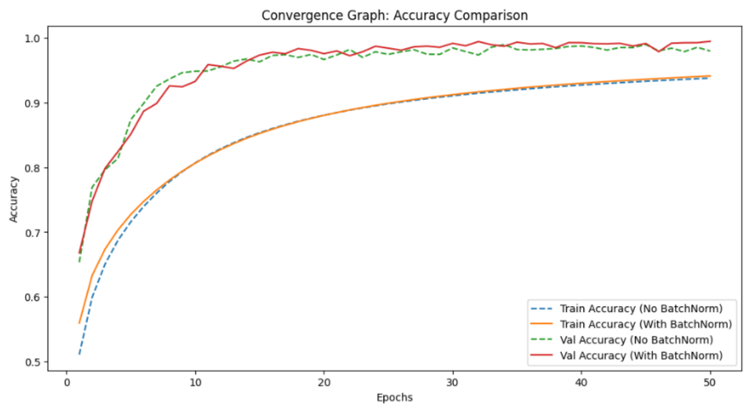
- Epochs: 50
- Learning rate: 0.001
- Optimizer: Adam
- Kernel Size: 3x3
- Padding: 1

We use the Batch Normalisation technique to improve the training of the CNN by normalising the inputs to each layer. In contrast to the model used in task 1 which does not use Batch Normalisation, this model improves the stability of the training and accelerates convergence, helping the model's learning process and improving performance.

Visualisations:

Given below is the convergence graph of both the models' training and validation losses and accuracy with and without Batch Normalisation:





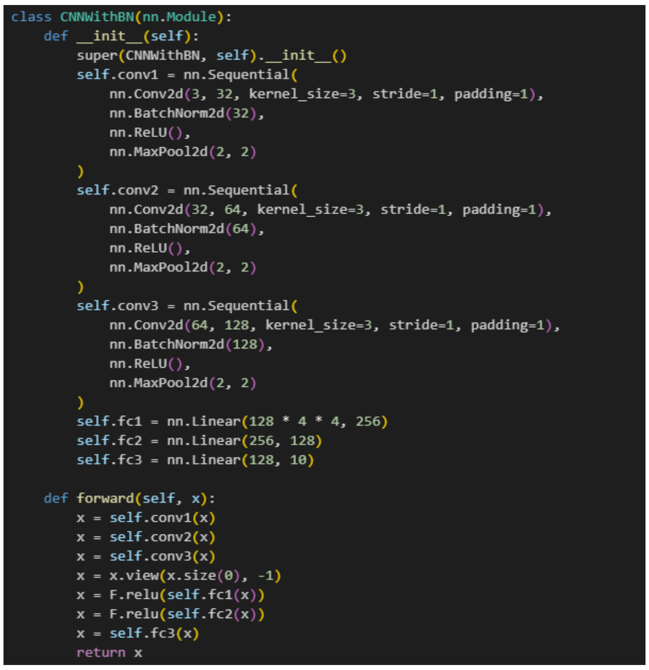
Observations:
Batch Normalization (BN) improves both train and validation accuracy more significantly than the model without BN.

Additionally, the loss curve with BN decreases more quickly, showing that the model is learning more effectively.

<u>Task 3: Visualizing Convolutional Features / Filters</u>

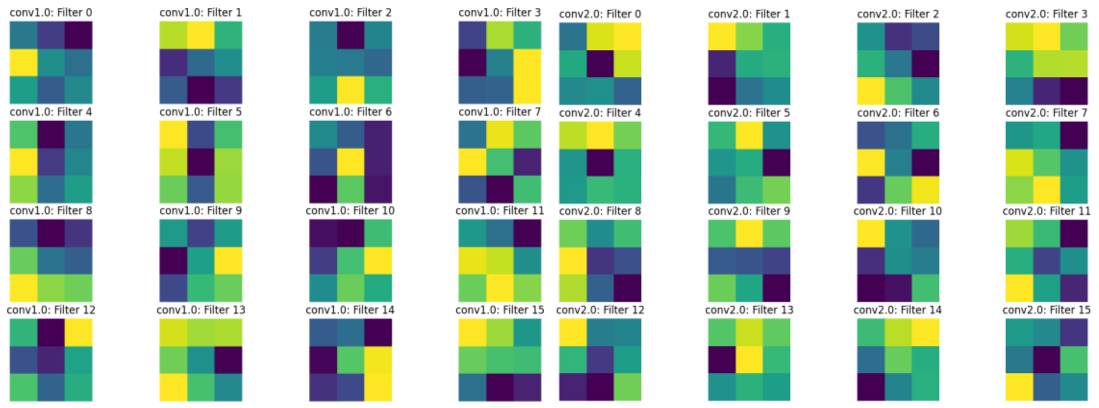
Objective:
To visualize the convolutional features and feature maps generated at different layers of the CNN model.

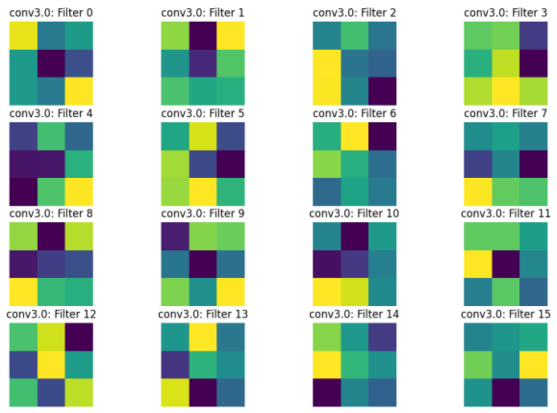
We will be using the CNN model that uses Batch Normalisation for the visualizations of features and filters.



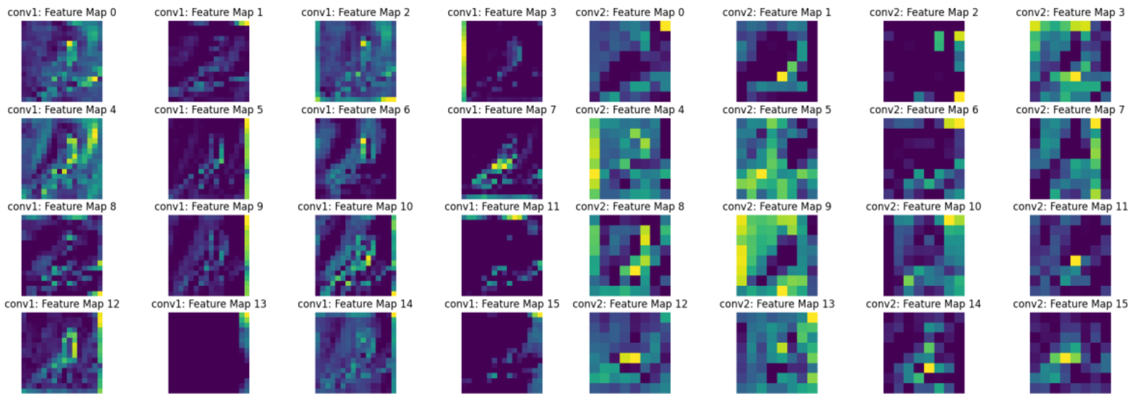
```python
class CNNWithBN(nn.Module):
    def __init__(self):
        super(CNNWithBN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

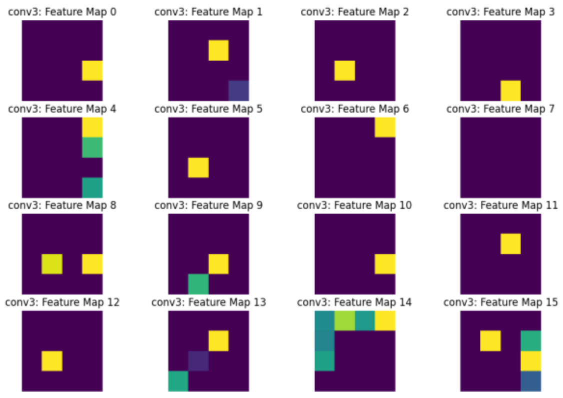
The visualize_filters function iterates the convolutional layers of the model and represents the filters applied at each layer. These filters represent the patterns that the model learns to detect at each layer.





The visualize_feature_maps function passes a test image through the model and visualizes the feature maps produced at each layer. Each feature map is the result of applying a filter from the corresponding convolutional layer to the image. As the image passes through deeper layers, the feature maps become vaguer.





The test image used to generate these filters:

The visualizations produced give valuable insights into the network's architecture and learning behaviour.