CSC3831 Coursework Part 3

Dataset: CIFAR-10 imported from https://keras.io/api/datasets/cifar10/

Modules: PyTorch, numpy, keras, matplotlib

Section 1 – Early stopping

CNN:

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, 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.fc2 = nn.Linear(256, 128)
        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc3 = nn.Linear(128, 10)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

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

        x = x.view(-1, 128 * 4 * 4) # adjust dimensions to match fc1 input
        x = F.relu(self.fc1(x)) # fc1 -> relu
        x = F.relu(self.fc2(x)) # fc2 -> relu
        x = self.fc3(x) # fc3 -> output

        return x
```

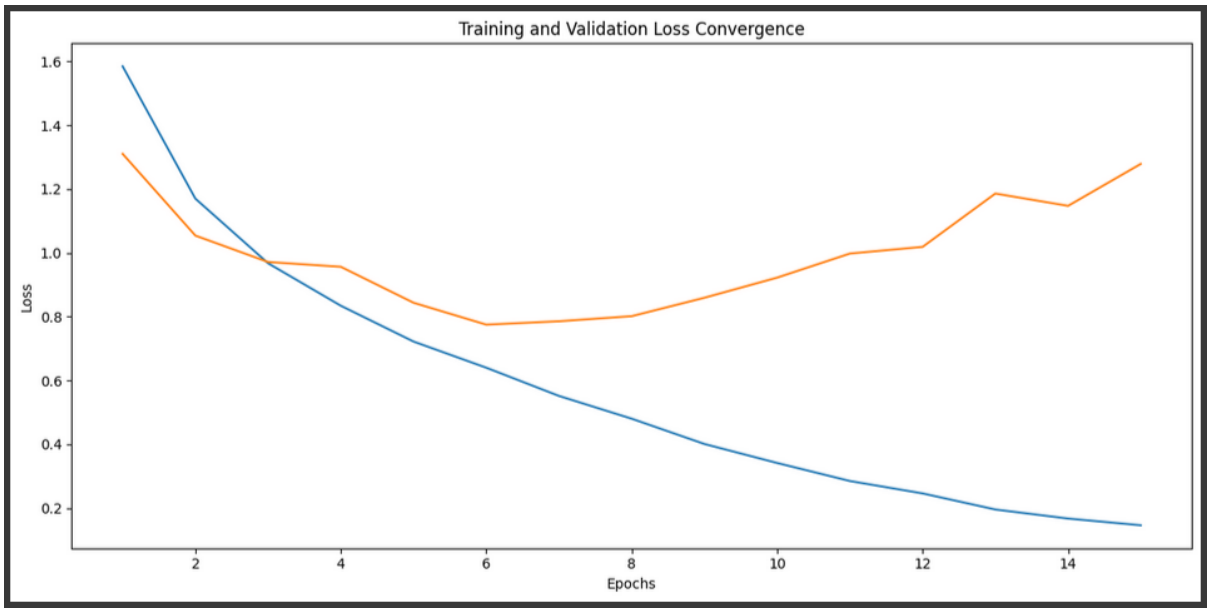
I have set the hyperparameters to be as follows:

- Learning rate = 0.001
- Max epochs = 30
- Minimum epochs = 5
- Training stops after 5 epochs without improvement
- Batch size = 64
- 80% of training data is used for training, 20% used for validation

Early stopping was used in training this model. The model will save the most accurate but keep training new models five epochs after the model stopped improving to see if it continues to improve.

Below I have plotted the training and validation loss for each epoch over a preset size of 20 epochs. You can clearly see a divergence of the validation loss (orange) from the training loss (blue) at around 7 epochs due to the model overfitting to the training data. This, paired with my implementation of early stopping saving the model at 7 epochs, shows that after a certain

number of epochs, the model is at it's most accurate and training needs to be stopped before the model gets less accurate.



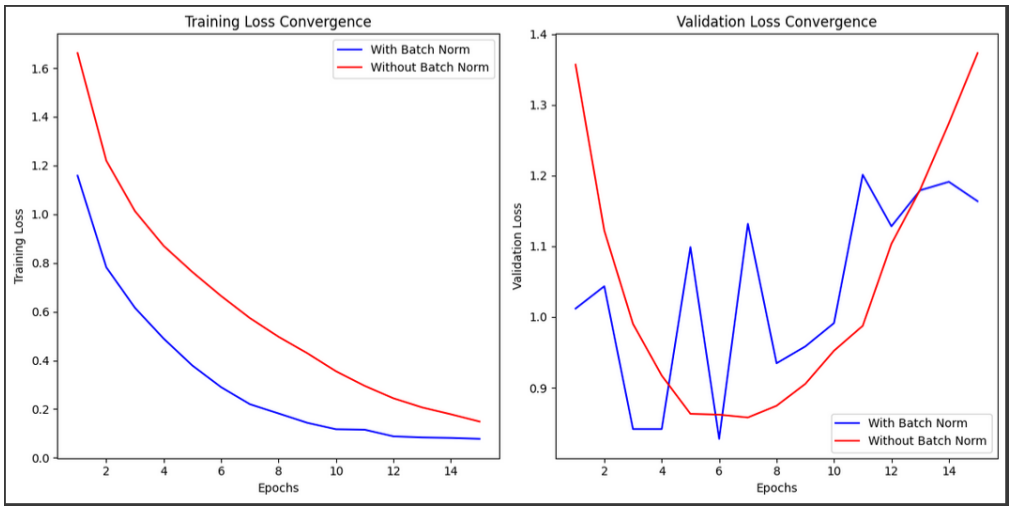
Section 2 – Batch Normalisation

CNN with batch normalisation:

```python
class CNNWithBatchNorm(nn.Module):
    def __init__(self):
        super(CNNWithBatchNorm, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)

        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.bn_fc1 = nn.BatchNorm1d(256)
        self.fc2 = nn.Linear(256, 128)
        self.bn_fc2 = nn.BatchNorm1d(128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x)))) # conv1 -> bn -> relu -> pool
        x = self.pool(F.relu(self.bn2(self.conv2(x)))) # conv2 -> bn -> relu -> pool
        x = self.pool(F.relu(self.bn3(self.conv3(x)))) # conv3 -> bn -> relu -> pool

        x = x.view(-1, 128 * 4 * 4)

        x = F.relu(self.bn_fc1(self.fc1(x))) # fc1 -> bn -> relu
        x = F.relu(self.bn_fc2(self.fc2(x))) # fc2 -> bn -> relu
        x = self.fc3(x) # fc3 -> output

        return x
```

I kept the same hyperparameters from the first section (learning rate = 0.001, 30 max epochs for training, and a batch size of 64), using 80% of the training data for actual training, and using the remaining 20% for validation.

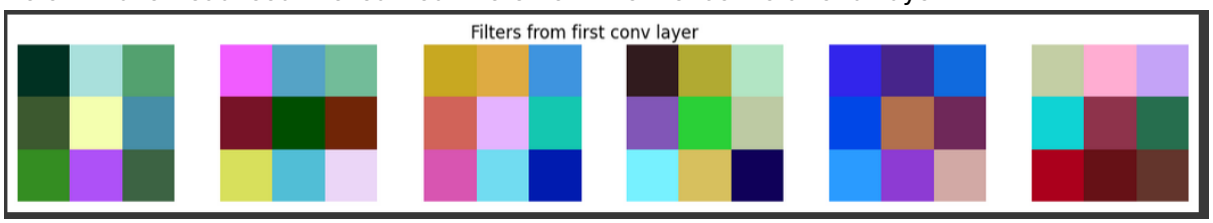
Finally, the convergence graph shows with batch normalisation there is a much greater improvement in training loss and it also hits minimum validation loss much faster than without batch normalisation.



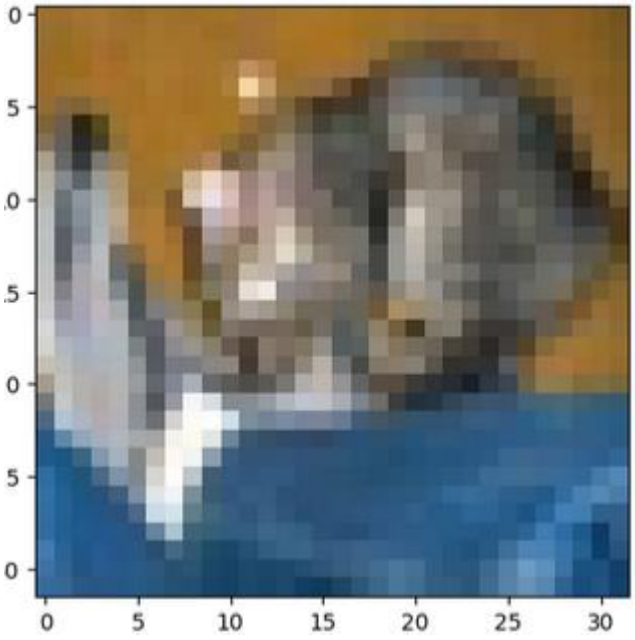
Section 3 – Visualising Filters

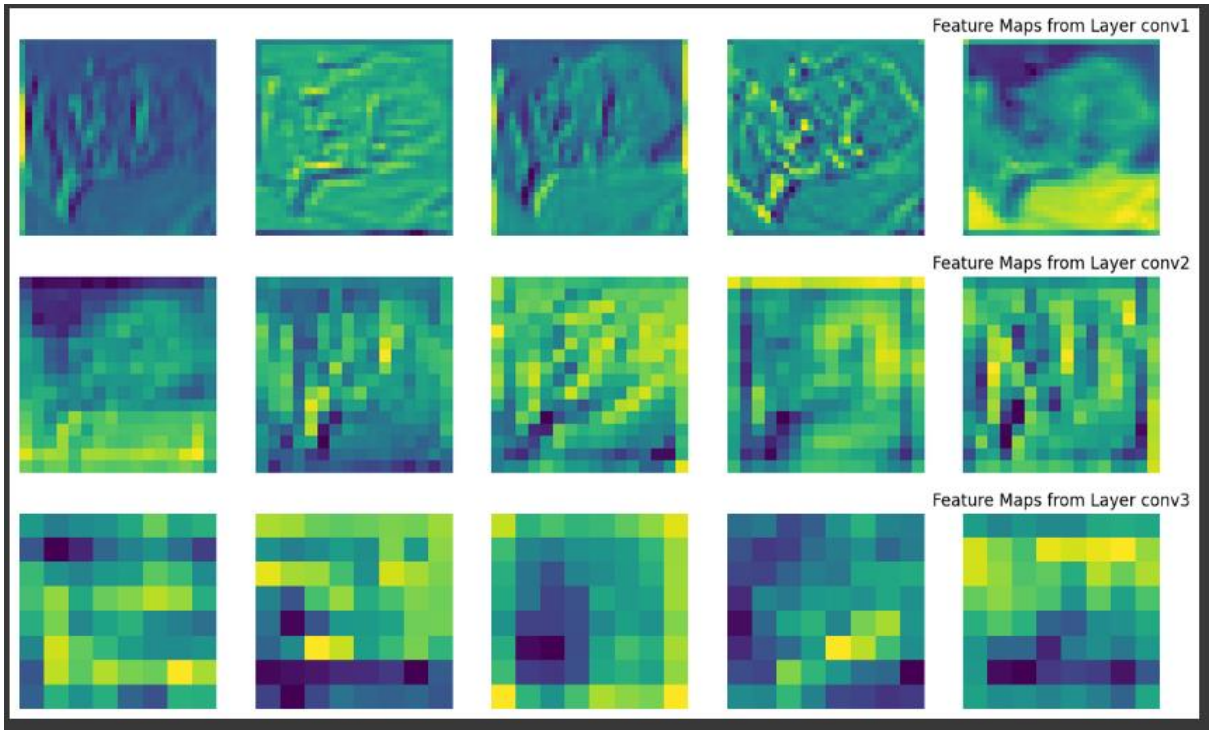
For this section I have used the same model trained in section two, I chose the model trained without batch normalisation.

Below I have visualised five learned filters from the first convolutional layer.



Below I have shown the result of applying the filters from each convolutional layer to the displayed test image.




Feature Maps from Layer conv1
Feature Maps from Layer conv2
Feature Maps from Layer conv3