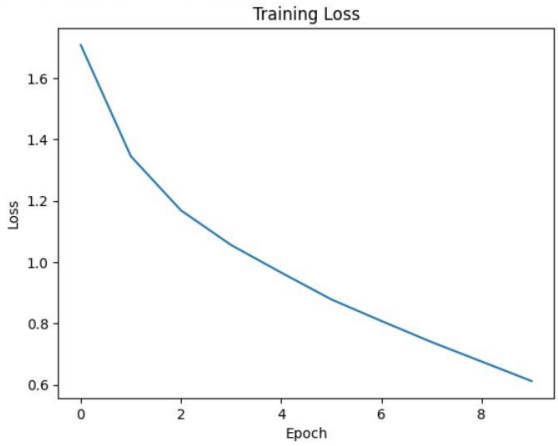
Training a CNN using CIFAR-10 Data

Ans 1: We extracted the following 9 images from the dataset

Ans 2:

Training loss plot as a function of epochs (10 counts):

```
Epoch 1/10
1407/1407 [==============================] - 675s 478ms/step - loss: 1.7087 - accuracy: 0.3813 - val_loss: 1.4404 - val_accuracy: 0.4936
Epoch 2/10
1407/1407 [==============================] - 673s 478ms/step - loss: 1.3455 - accuracy: 0.5217 - val_loss: 1.2256 - val_accuracy: 0.5700
Epoch 3/10
1407/1407 [==============================] - 639s 454ms/step - loss: 1.1689 - accuracy: 0.5873 - val_loss: 1.0908 - val_accuracy: 0.6198
Epoch 4/10
1407/1407 [==============================] - 643s 457ms/step - loss: 1.0559 - accuracy: 0.6286 - val_loss: 1.0808 - val_accuracy: 0.6162
Epoch 5/10
1407/1407 [==============================] - 655s 466ms/step - loss: 0.9659 - accuracy: 0.6639 - val_loss: 1.0235 - val_accuracy: 0.6416
Epoch 6/10
1407/1407 [==============================] - 648s 461ms/step - loss: 0.8784 - accuracy: 0.6937 - val_loss: 1.0165 - val_accuracy: 0.6518
Epoch 7/10
1407/1407 [==============================] - 647s 460ms/step - loss: 0.8079 - accuracy: 0.7207 - val_loss: 1.0123 - val_accuracy: 0.6550
Epoch 8/10
1407/1407 [==============================] - 652s 463ms/step - loss: 0.7392 - accuracy: 0.7419 - val_loss: 1.0234 - val_accuracy: 0.6528
Epoch 9/10
1407/1407 [==============================] - 649s 461ms/step - loss: 0.6756 - accuracy: 0.7650 - val_loss: 1.0362 - val_accuracy: 0.6612
Epoch 10/10
1407/1407 [==============================] - 647s 460ms/step - loss: 0.6117 - accuracy: 0.7865 - val_loss: 1.0399 - val_accuracy: 0.6628
313/313 [==============================] - 42s 133ms/step - loss: 1.0748 - accuracy: 0.6524
Test accuracy: 0.652400016784668
```



Accuracy of the data (test): 65.2 %

Weight Initialisation: Using Glorot uniform initialization for dense layers and a variant of Glorot uniform initialization for convolutional layers.

Architecture:

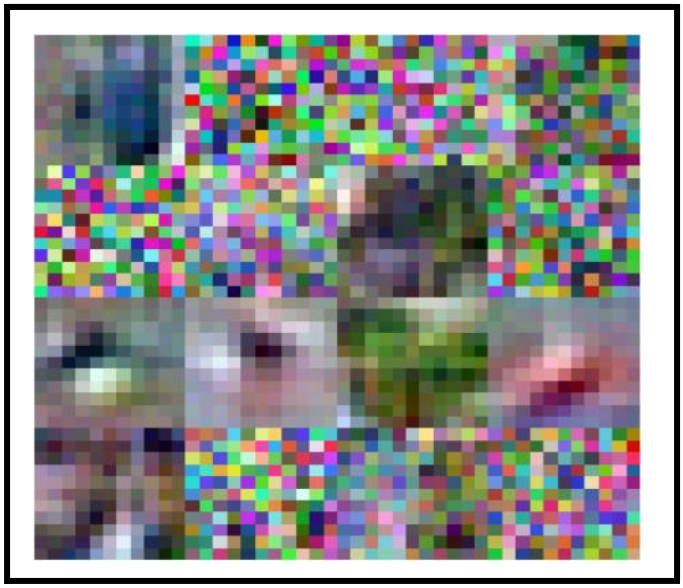
- Input: The input images have a shape of (32, 32, 3), representing 32x32-pixel RGB images.
- Convolutional Layers:
  - First Convolutional Layer: It consists of 64 filters of size 11x11, followed by ReLU activation. The 'same' padding is used to preserve the spatial dimensions of the input.
  - Max Pooling Layer: A 2x2 max pooling layer with a stride of 2 is applied to downsample the feature maps.
  - Second and Third Convolutional Layers: They both have 128 filters of size 3x3, followed by ReLU activation. Again, 'same' padding is used.

- Average Pooling Layer: A 2x2 average pooling layer is applied to the pool across the preceding feature map.
- Flatten: The feature maps are flattened into a 1D vector.
- Dense Layer: A fully connected layer with 10 units (corresponding to the 10 classes in CIFAR-10) and a softmax activation function is added as the final output layer.

Hyperparameters:

- Optimizer: The model is compiled with the Adam optimizer, which is a popular choice for training neural networks. Adam adapts the learning rate during training and combines the advantages of both AdaGrad and RMSProp optimizers.
- Loss Function: The sparse categorical cross-entropy loss function is used, as it is suitable for multi-class classification tasks with integer labels (as in CIFAR-10).
- Metrics: The accuracy metric is used to evaluate the model's performance during training and testing.
- Epochs: The model is trained for 10 epochs, meaning it goes through the entire training dataset 10 times.
- Batch Size: The default batch size is used, which is determined by the Keras framework.
- Validation Split: During training, a validation set of 10% of the training data is used to monitor the model's performance and prevent overfitting

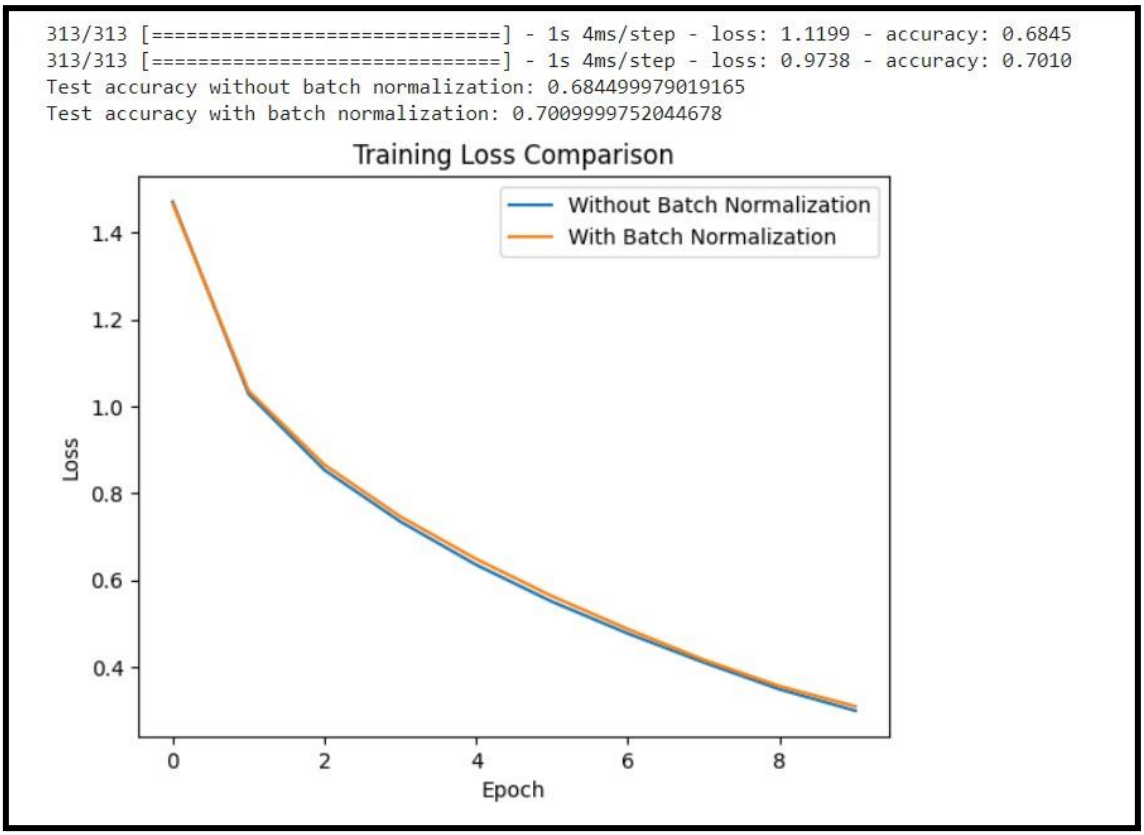
Ans 3: We extract the weights of the first convolutional layer, normalize each filter by contrast stretching, and create a large RGB image array by concatenating the smaller images. The result generated a visualization of the filters learned by the first convolutional layer. Each filter is normalized by contrast stretching to enhance the visualization, and the normalized filters are then concatenated into a large RGB image array. And finally, the RGB image array is displayed.



Ans 4:

We created the below graph to compare the training loss with and without batch normalization as a function of epochs.

Observation: Since the trend of training loss with and without normalization is the same, it means that adding batch normalization between each of the hidden layers did not significantly impact the training loss of the model. Batch normalization improves the convergence speed and generalization performance of the model. However, its effect may vary depending on the specific dataset and model architecture. Since the trend of training loss is the same, it suggests that the model without batch normalization is already able to learn effectively and converge well on the given dataset. We can also check other metrics than training loss, such as validation accuracy, test accuracy, and generalization to unseen data should also be considered to assess the overall effectiveness of the model with and without batch normalization.



Test error with normalization: 70%
Test error without normalization: 68%

Final plot: