kavaskar
2347230

Lab 3

1. Data Preprocessing:

In [9]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, utils, layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode class labels
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

# Data Augmentation (optional)
datagen = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1
)
datagen.fit(x_train)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 55s 0us/step
```

2. Network Architecture Design:

In [10]:
```
# Define the feedforward neural network model
model = models.Sequential()

# Input Layer (32x32x3 for CIFAR-10 images)
model.add(layers.InputLayer(input_shape=(32, 32, 3)))

# Hidden Layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Flatten())
```

```
# Dense layer before output
model.add(layers.Dense(128, activation='relu'))

# Output Layer with Softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur

# Summary of the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 591274 (2.26 MB)
Trainable params: 591274 (2.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Justification:

- **Convolutional Layers**: Used for extracting features from images (edges, textures, patterns). They capture spatial hierarchies in images.
- **MaxPooling Layers**: Reduces spatial dimensions, helping prevent overfitting and reducing computational complexity.
- **Dense Layer**: Acts as the final classifier after flattening the feature maps.
- **Softmax Output Layer**: Ideal for multi-class classification (10 output classes).

3. **Activation Functions:**

We use **ReLU** for hidden layers and **Softmax** for the output layer.

- **ReLU**: Effective in avoiding vanishing gradient issues and accelerating convergence during training. It activates neurons selectively by outputting zero for negative inputs, which helps in sparse representations.
- **Softmax**: Converts raw output scores into probabilities for multi-class classification.

4. Loss Function and Optimizer:

In [11]:
```
# Loss function comparison
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur
```

- **Categorical Cross Entropy**: Suitable for multi-class classification by comparing predicted probabilities with true labels.
- **Adam Optimizer**: Combines the advantages of momentum and RMSprop, adjusting learning rates dynamically. This results in faster and more stable convergence.

Effect of Learning Rate: A high learning rate may lead to unstable training and overshooting the minimum. A low rate can slow down convergence. Adjust the learning rate dynamically if the model does not converge (e.g., using learning rate schedulers).

5. Training the Model:

In [12]:
```
# Train the model
history = model.fit(datagen.flow(x_train, y_train, batch_size=64),
                    epochs=50,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/50
782/782 [==============================] - 112s 141ms/step - loss: 1.5246 - accur
acy: 0.4413 - val_loss: 1.1981 - val_accuracy: 0.5722
Epoch 2/50
782/782 [==============================] - 95s 121ms/step - loss: 1.1290 - accura
cy: 0.5975 - val_loss: 1.0090 - val_accuracy: 0.6414
Epoch 3/50
782/782 [==============================] - 99s 126ms/step - loss: 0.9787 - accura
cy: 0.6569 - val_loss: 0.8212 - val_accuracy: 0.7113
Epoch 4/50
782/782 [==============================] - 88s 113ms/step - loss: 0.8905 - accura
cy: 0.6872 - val_loss: 0.8078 - val_accuracy: 0.7192
Epoch 5/50
782/782 [==============================] - 58s 74ms/step - loss: 0.8221 - accurac
y: 0.7129 - val_loss: 0.7506 - val_accuracy: 0.7417
Epoch 6/50
782/782 [==============================] - 58s 74ms/step - loss: 0.7778 - accurac
y: 0.7277 - val_loss: 0.7919 - val_accuracy: 0.7266
Epoch 7/50
782/782 [==============================] - 59s 75ms/step - loss: 0.7412 - accurac
y: 0.7403 - val_loss: 0.7785 - val_accuracy: 0.7383
Epoch 8/50
782/782 [==============================] - 60s 77ms/step - loss: 0.7040 - accurac
y: 0.7506 - val_loss: 0.6992 - val_accuracy: 0.7698
Epoch 9/50
782/782 [==============================] - 65s 83ms/step - loss: 0.6824 - accurac
y: 0.7625 - val_loss: 0.6806 - val_accuracy: 0.7690
Epoch 10/50
782/782 [==============================] - 62s 79ms/step - loss: 0.6575 - accurac
y: 0.7683 - val_loss: 0.6780 - val_accuracy: 0.7769
Epoch 11/50
782/782 [==============================] - 59s 76ms/step - loss: 0.6380 - accurac
y: 0.7776 - val_loss: 0.6859 - val_accuracy: 0.7660
Epoch 12/50
782/782 [==============================] - 60s 77ms/step - loss: 0.6261 - accurac
y: 0.7804 - val_loss: 0.6662 - val_accuracy: 0.7814
Epoch 13/50
782/782 [==============================] - 62s 79ms/step - loss: 0.6077 - accurac
y: 0.7860 - val_loss: 0.6281 - val_accuracy: 0.7830
Epoch 14/50
782/782 [==============================] - 62s 79ms/step - loss: 0.5984 - accurac
y: 0.7905 - val_loss: 0.6572 - val_accuracy: 0.7826
Epoch 15/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5802 - accurac
y: 0.7961 - val_loss: 0.6583 - val_accuracy: 0.7867
Epoch 16/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5745 - accurac
y: 0.7989 - val_loss: 0.6501 - val_accuracy: 0.7861
Epoch 17/50
782/782 [==============================] - 62s 79ms/step - loss: 0.5592 - accurac
y: 0.8030 - val_loss: 0.6500 - val_accuracy: 0.7849
Epoch 18/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5521 - accurac
y: 0.8081 - val_loss: 0.6049 - val_accuracy: 0.8010
Epoch 19/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5365 - accurac
y: 0.8119 - val_loss: 0.6358 - val_accuracy: 0.7954
Epoch 20/50
782/782 [==============================] - 62s 79ms/step - loss: 0.5359 - accurac
y: 0.8134 - val_loss: 0.5642 - val_accuracy: 0.8106
```

```
Epoch 21/50
782/782 [==============================] - 62s 79ms/step - loss: 0.5254 - accurac
y: 0.8159 - val_loss: 0.6118 - val_accuracy: 0.8001
Epoch 22/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5170 - accurac
y: 0.8193 - val_loss: 0.5803 - val_accuracy: 0.8088
Epoch 23/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5119 - accurac
y: 0.8202 - val_loss: 0.5747 - val_accuracy: 0.8125
Epoch 24/50
782/782 [==============================] - 61s 79ms/step - loss: 0.5090 - accurac
y: 0.8241 - val_loss: 0.5896 - val_accuracy: 0.8071
Epoch 25/50
782/782 [==============================] - 61s 78ms/step - loss: 0.5051 - accurac
y: 0.8239 - val_loss: 0.5744 - val_accuracy: 0.8099
Epoch 26/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4967 - accurac
y: 0.8251 - val_loss: 0.5748 - val_accuracy: 0.8090
Epoch 27/50
782/782 [==============================] - 72s 92ms/step - loss: 0.4881 - accurac
y: 0.8293 - val_loss: 0.5895 - val_accuracy: 0.8087
Epoch 28/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4823 - accurac
y: 0.8315 - val_loss: 0.5843 - val_accuracy: 0.8096
Epoch 29/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4755 - accurac
y: 0.8343 - val_loss: 0.6243 - val_accuracy: 0.8065
Epoch 30/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4768 - accurac
y: 0.8317 - val_loss: 0.5667 - val_accuracy: 0.8127
Epoch 31/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4709 - accurac
y: 0.8339 - val_loss: 0.6211 - val_accuracy: 0.8092
Epoch 32/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4663 - accurac
y: 0.8366 - val_loss: 0.6184 - val_accuracy: 0.8008
Epoch 33/50
782/782 [==============================] - 61s 77ms/step - loss: 0.4637 - accurac
y: 0.8365 - val_loss: 0.5548 - val_accuracy: 0.8201
Epoch 34/50
782/782 [==============================] - 64s 82ms/step - loss: 0.4564 - accurac
y: 0.8384 - val_loss: 0.5650 - val_accuracy: 0.8174
Epoch 35/50
782/782 [==============================] - 76s 97ms/step - loss: 0.4520 - accurac
y: 0.8418 - val_loss: 0.6222 - val_accuracy: 0.8065
Epoch 36/50
782/782 [==============================] - 75s 96ms/step - loss: 0.4507 - accurac
y: 0.8415 - val_loss: 0.6510 - val_accuracy: 0.7940
Epoch 37/50
782/782 [==============================] - 63s 81ms/step - loss: 0.4470 - accurac
y: 0.8427 - val_loss: 0.5701 - val_accuracy: 0.8197
Epoch 38/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4411 - accurac
y: 0.8445 - val_loss: 0.5847 - val_accuracy: 0.8171
Epoch 39/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4373 - accurac
y: 0.8490 - val_loss: 0.6096 - val_accuracy: 0.8090
Epoch 40/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4445 - accurac
y: 0.8425 - val_loss: 0.5912 - val_accuracy: 0.8168
```

```
Epoch 41/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4351 - accurac
y: 0.8469 - val_loss: 0.6207 - val_accuracy: 0.8115
Epoch 42/50
782/782 [==============================] - 62s 80ms/step - loss: 0.4301 - accurac
y: 0.8482 - val_loss: 0.6001 - val_accuracy: 0.8139
Epoch 43/50
782/782 [==============================] - 63s 81ms/step - loss: 0.4261 - accurac
y: 0.8510 - val_loss: 0.5845 - val_accuracy: 0.8182
Epoch 44/50
782/782 [==============================] - 61s 78ms/step - loss: 0.4258 - accurac
y: 0.8530 - val_loss: 0.5808 - val_accuracy: 0.8143
Epoch 45/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4210 - accurac
y: 0.8514 - val_loss: 0.5544 - val_accuracy: 0.8269
Epoch 46/50
782/782 [==============================] - 64s 82ms/step - loss: 0.4182 - accurac
y: 0.8524 - val_loss: 0.5788 - val_accuracy: 0.8179
Epoch 47/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4194 - accurac
y: 0.8518 - val_loss: 0.5496 - val_accuracy: 0.8257
Epoch 48/50
782/782 [==============================] - 63s 80ms/step - loss: 0.4164 - accurac
y: 0.8535 - val_loss: 0.5808 - val_accuracy: 0.8188
Epoch 49/50
782/782 [==============================] - 62s 80ms/step - loss: 0.4098 - accurac
y: 0.8555 - val_loss: 0.5763 - val_accuracy: 0.8173
Epoch 50/50
782/782 [==============================] - 62s 79ms/step - loss: 0.4114 - accurac
y: 0.8544 - val_loss: 0.5887 - val_accuracy: 0.8160
```

Backpropagation:

- During backpropagation, weights are updated using gradients from the loss function. The learning rate controls how much to adjust weights. If learning is too slow, increase the rate; if oscillating, decrease it.

6. Model Evaluation:

In [13]:
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc}")

# Confusion Matrix and classification report
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

print(confusion_matrix(y_true, y_pred_classes))
print(classification_report(y_true, y_pred_classes))
```

```
313/313 [==============================] - 3s 11ms/step - loss: 0.5887 - accurac
y: 0.8160
Test accuracy: 0.8159999847412109
313/313 [==============================] - 3s 10ms/step
[[847  15  37   3  13   5   5   9  26  40]
 [  7 928   0   1   1   1   4   1   5  52]
 [ 48   2 737  18  47  40  69  23   4  12]
 [ 20  11  47 562  54 136  90  33  13  34]
 [ 13   2  39  22 781  18  44  71   5   5]
 [  7   5  31  78  34 749  40  44   2  10]
 [  6   4  17   8  17  11 915   6   5  11]
 [ 11   5  19  17  27  32   5 868   3  13]
 [ 81  23   4   2   1   1   6   3 849  30]
 [ 20  36   3   2   2   0   3   3   7 924]]
              precision    recall  f1-score   support

           0       0.80      0.85      0.82      1000
           1       0.90      0.93      0.91      1000
           2       0.79      0.74      0.76      1000
           3       0.79      0.56      0.66      1000
           4       0.80      0.78      0.79      1000
           5       0.75      0.75      0.75      1000
           6       0.77      0.92      0.84      1000
           7       0.82      0.87      0.84      1000
           8       0.92      0.85      0.88      1000
           9       0.82      0.92      0.87      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.81     10000
weighted avg       0.82      0.82      0.81     10000
```

- **Precision, Recall, F1-score**: Help identify the model's effectiveness in distinguishing between classes.
- **Confusion Matrix**: Highlights misclassification between classes.

Improving Accuracy: Consider deeper architectures (more layers), data augmentation, or tuning hyperparameters (batch size, learning rate).

7. **Optimization Strategies:**

- **Early Stopping**: Monitors validation loss to stop training if it stops improving.
- **Learning Rate Scheduling**: Reduces the learning rate gradually to ensure smooth convergence.
- **Weight Initialization**: Proper initialization (e.g., He initialization) avoids vanishing/exploding gradients.

Importance of Weight Initialization: Poor initialization can lead to slow or unstable convergence. Proper initialization helps the network converge faster.

8. **Report:**

- **Model Architecture**: Detailed explanation and rationale for choices.
- **Training/Test Accuracy**: Relevant plots for loss/accuracy over epochs.

- **Hyperparameters**: Learning rate, batch size, number of epochs, etc.
- **Challenges**: E.g., managing overfitting, tuning hyperparameters.