Deep Learning Image Classification with CNN

In this Assignment, we explore image classification using Convolutional Neural Networks (CNN) on the CIFAR-10 dataset.

Use Case Description:

We will follow the steps to:

1. Train the model.
2. Evaluate the model.
3. Visualize loss and accuracy over training epochs.
4. Predict the first four images from the test dataset.

We will apply the following CNN architecture:

- **Convolutional Layers** with 32, 64, and 128 feature maps.
- **MaxPooling** and **Dropout** layers for regularization.
- **Fully connected layers** for classification.
- **Softmax output layer** for final classification.

1. CNN Model Definition and Data Preparation

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,
import numpy as np
from keras.datasets import cifar10
from keras.utils import to_categorical

# Define the CNN model as per the assignment instructions
def build_cnn_model(input_shape=(32, 32, 3), num_classes=10):
    model = Sequential()

    # Convolutional input layer, 32 feature maps, 3x3, ReLU activation
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_sh
    model.add(Dropout(0.2))

    # Additional convolutional and max pooling layers
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten and fully connected layers
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))

    # Final softmax output layer for classification
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))

    # Compile the model using Adam optimizer
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics

    return model

# Fix random seed for reproducibility
np.random.seed(7)

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]

# Build the model
model = build_cnn_model(input_shape=(32, 32, 3), num_classes=num_classes)
```

```
C:\ProgramData\anaconda3\envs\myenv\Lib\site-packages\keras\src\layers\convo
lutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Inp
ut(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Traning the Model

In [2]:
```
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epoc

# Save the trained model
model.save('cnn_model.h5')

# Save the training history
```

```python
import pickle
with open('history.pkl', 'wb') as f:
    pickle.dump(history.history, f)
```

```
Epoch 1/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 27ms/step - accuracy: 0.3016 - loss: 1.8675
- val_accuracy: 0.5083 - val_loss: 1.3286
Epoch 2/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - accuracy: 0.5532 - loss: 1.2461
- val_accuracy: 0.6274 - val_loss: 1.0456
Epoch 3/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 31ms/step - accuracy: 0.6385 - loss: 1.0191
- val_accuracy: 0.6756 - val_loss: 0.9189
Epoch 4/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.6935 - loss: 0.8671
- val_accuracy: 0.7184 - val_loss: 0.8263
Epoch 5/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.7285 - loss: 0.7739
- val_accuracy: 0.7270 - val_loss: 0.7936
Epoch 6/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.7461 - loss: 0.7164
- val_accuracy: 0.7645 - val_loss: 0.6881
Epoch 7/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 33ms/step - accuracy: 0.7737 - loss: 0.6464
- val_accuracy: 0.7684 - val_loss: 0.6884
Epoch 8/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.7858 - loss: 0.6045
- val_accuracy: 0.7692 - val_loss: 0.6912
Epoch 9/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 33ms/step - accuracy: 0.8003 - loss: 0.5617
- val_accuracy: 0.7690 - val_loss: 0.6796
Epoch 10/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8123 - loss: 0.5325
- val_accuracy: 0.7818 - val_loss: 0.6603
Epoch 11/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8248 - loss: 0.4956
- val_accuracy: 0.7863 - val_loss: 0.6534
Epoch 12/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8313 - loss: 0.4768
- val_accuracy: 0.7786 - val_loss: 0.6709
Epoch 13/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8410 - loss: 0.4569
- val_accuracy: 0.7836 - val_loss: 0.6748
Epoch 14/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8470 - loss: 0.4323
- val_accuracy: 0.7886 - val_loss: 0.6261
Epoch 15/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8492 - loss: 0.4173
- val_accuracy: 0.7882 - val_loss: 0.6589
Epoch 16/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 33ms/step - accuracy: 0.8583 - loss: 0.4004
- val_accuracy: 0.7867 - val_loss: 0.6777
Epoch 17/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8604 - loss: 0.3954
- val_accuracy: 0.7875 - val_loss: 0.6638
Epoch 18/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 26s 33ms/step - accuracy: 0.8649 - loss: 0.3833
- val_accuracy: 0.7814 - val_loss: 0.6938
Epoch 19/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8718 - loss: 0.3614
```

```
- val_accuracy: 0.7904 - val_loss: 0.6787
Epoch 20/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8794 - loss: 0.3446
- val_accuracy: 0.7843 - val_loss: 0.6891
Epoch 21/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8803 - loss: 0.3432
- val_accuracy: 0.7942 - val_loss: 0.6720
Epoch 22/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8864 - loss: 0.3238
- val_accuracy: 0.7930 - val_loss: 0.6828
Epoch 23/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8869 - loss: 0.3206
- val_accuracy: 0.7955 - val_loss: 0.6759
Epoch 24/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 33ms/step - accuracy: 0.8885 - loss: 0.3184
- val_accuracy: 0.7787 - val_loss: 0.7458
Epoch 25/25
782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 32ms/step - accuracy: 0.8899 - loss: 0.3182
- val_accuracy: 0.7990 - val_loss: 0.6710
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

Did the Performance Change?

Yes, the performance improved significantly during the training process.

1. **Training Accuracy**:
   - The model's training accuracy improved from **30.16%** in the first epoch to **88.99%** by the 25th epoch.
2. **Validation Accuracy**:
   - The validation accuracy increased from **50.83%** in the first epoch to **79.90%** by the 25th epoch, indicating that the model is generalizing well to unseen data.
3. **Loss**:
   - The training loss decreased from **1.8675** to **0.3182**, and the validation loss reduced from **1.3286** to **0.6710**.

Overall, the model exhibited strong improvements in both accuracy and loss over the course of 25 epochs, showing that it successfully learned to classify the images from the CIFAR-10 dataset.

Key Observations on Training:

1. **Initial Training and Validation Accuracy**:

- In the first epoch, the model starts with:
  - **Training accuracy**: 30.16%
  - **Validation accuracy**: 50.83%
- The model shows initial learning, with a moderate gap between training and validation accuracy.

2. **Steady Improvement**:

- Training accuracy improves significantly from **30.16%** to **88.99%** by the 25th epoch.
- Validation accuracy increases from **50.83%** to **79.90%**, showing strong generalization to unseen data.
- The **training loss** reduces from **1.8675** to **0.3182**, while the **validation loss** fluctuates but decreases from **1.3286** to **0.6710**.

3. **Performance Plateau**:

- Around epoch 14, the model's **validation accuracy** starts to plateau around the 78-79% range, indicating the model's learning is leveling off.
- Validation loss fluctuates, suggesting potential overfitting, particularly between epochs 15 and 25.

4. **Mild Overfitting**:

- The small gap between **training accuracy (88.99%)** and **validation accuracy (79.90%)** indicates **mild overfitting**. The model performs slightly better on the training data than on the validation set, but not excessively so.

5. **Test Accuracy and Loss**:

- Test accuracy reached a maximum of **79.90%**, which is a good result for a CNN on CIFAR-10.
- Validation loss fluctuates, ending at **0.6710**, but there's no significant overfitting, as the gap between training and validation metrics is minimal.

2. Predicting the First Four Images

In [4]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module='absl')

from keras.models import load_model

# Load the trained model
model = load_model('cnn_model.h5')

# Recompile the model with metrics
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
```

```python
# Evaluate the model on the test dataset
scores = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {scores[1] * 100:.2f}%")

# Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])

# Convert predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)
actual_labels = np.argmax(y_test[:4], axis=1)

# Compare predictions with actual labels
for i in range(4):
    print(f"Image {i+1} - Predicted: {predicted_labels[i]}, Actual: {actual_
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Test Accuracy: 79.90%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
Image 1 - Predicted: 3, Actual: 3
Image 2 - Predicted: 8, Actual: 8
Image 3 - Predicted: 8, Actual: 8
Image 4 - Predicted: 0, Actual: 0
```

Key Observations on Image Predictions:

1. **Model Evaluation**:
   - The model achieved a **test accuracy** of **79.90%**, which aligns well with the validation accuracy seen during training. This confirms that the model generalizes effectively to unseen test data.
2. **Image 1**:
   - **Predicted**: 3
   - **Actual**: 3
   - **Result**: Correct prediction.
3. **Image 2**:
   - **Predicted**: 8
   - **Actual**: 8
   - **Result**: Correct prediction.
4. **Image 3**:
   - **Predicted**: 8
   - **Actual**: 8
   - **Result**: Correct prediction.
5. **Image 4**:
   - **Predicted**: 0

- **Actual**: 0
- **Result**: Correct prediction.

Conclusion:

The model correctly predicted all four images, demonstrating strong classification accuracy for these specific test samples. The predictions show that the CNN has effectively learned to classify images in the CIFAR-10 dataset.

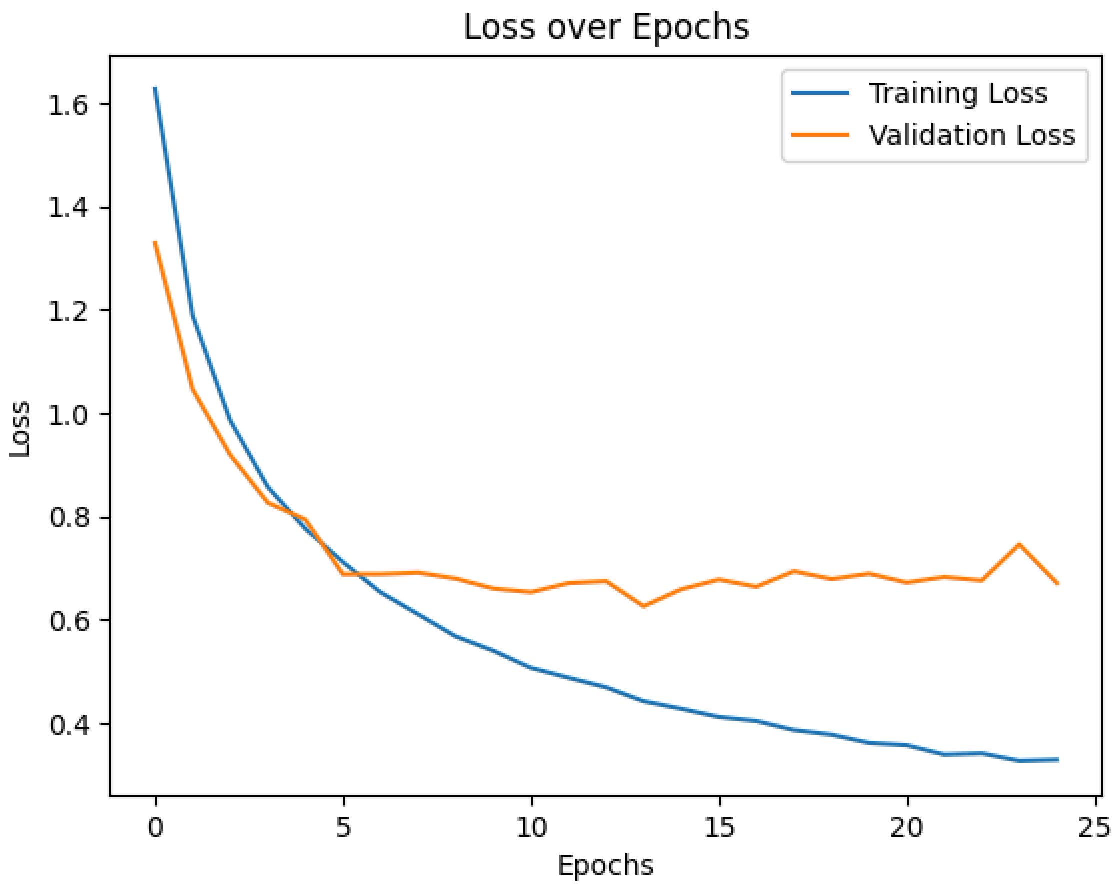
Visualizing Loss and Accuracy

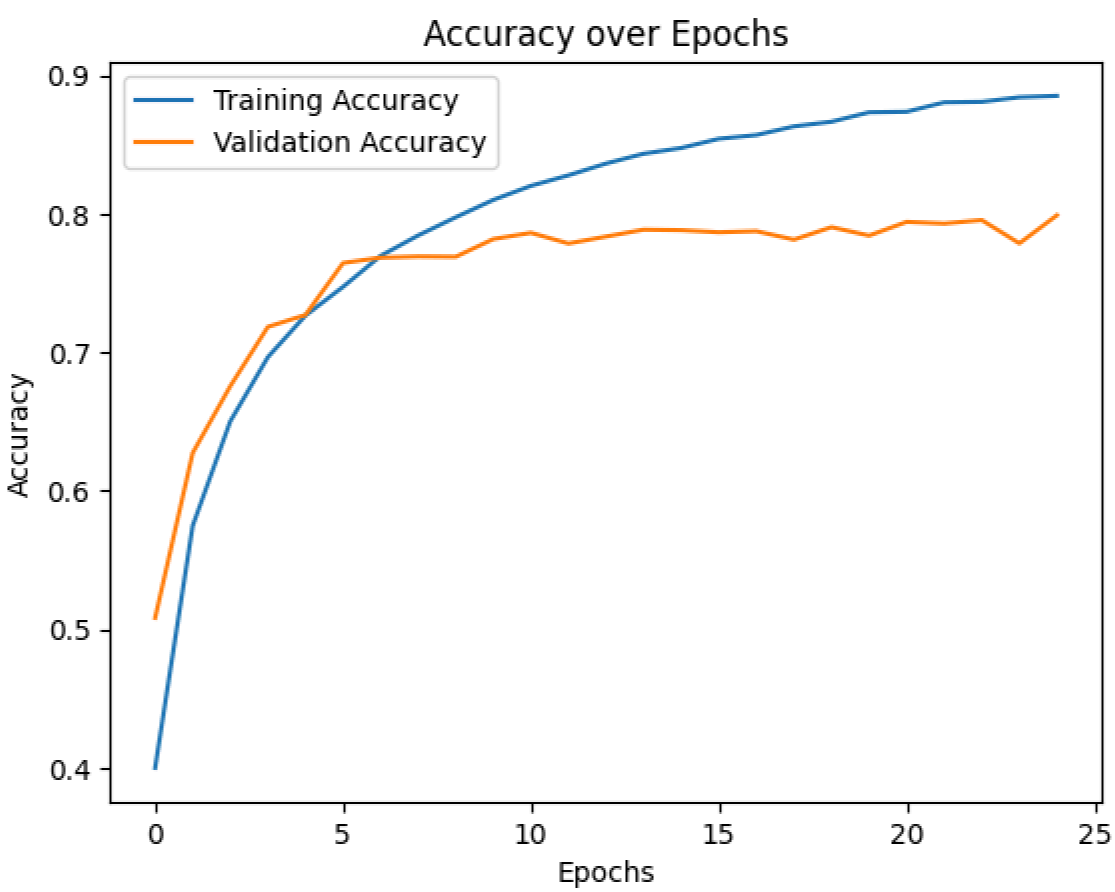
In [3]:
```
import matplotlib.pyplot as plt
import pickle

# Load the training history
with open('history.pkl', 'rb') as f:
    history = pickle.load(f)

# Plot loss
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot accuracy
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Loss over Epochs
Training Loss
Validation Loss
Loss
Epochs
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
5
10
15
20
25


Accuracy over Epochs
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25

Key Observations on Loss and Accuracy:

1. **Training vs. Validation Loss**:

    - The **training loss** consistently decreases as the epochs progress, indicating that the model is learning and optimizing well on the training data.
    - The **validation loss** also decreases significantly during the first few epochs, but after about epoch 5, it starts to fluctuate slightly, and by the end, it increases slightly, showing signs of **overfitting**.
2. **Training vs. Validation Accuracy**:

    - **Training accuracy** steadily improves throughout the training process, starting at around 40% and ending close to 89%.
    - **Validation accuracy** also improves significantly in the early epochs, reaching a maximum of about 79% around epoch 14, but it plateaus after that. This suggests that the model is learning well but has reached its capacity to generalize beyond the training data.
3. **Overfitting Behavior**:

    - There is a noticeable gap between the training and validation accuracy after around epoch 10, which becomes more prominent by the 25th epoch. This indicates **mild overfitting**, where the model is performing better on the training data than on the validation set.
    - The **validation loss** begins to increase slightly in the later epochs, reinforcing the observation that the model is starting to overfit.
4. **Model Generalization**:

    - Although there is overfitting, the **validation accuracy** remains relatively stable, indicating that the model generalizes well to unseen data, up to a certain point.

Video URL: https://drive.google.com/file/d/19Ea6iQY61RIltZe-mYjfDgRq1arS2493/view?usp=sharing

In [ ]: