*Experiment No. 4 *

Aim: Design and implement a CNN for Image Classification

a) Select a suitable image classification dataset (medical imaging, agricultural, etc.).

b) Optimized with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np
```

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
# Build CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train model
epochs = 10
history = model.fit(x_train, y_train, epochs=epochs, validation_data=(x_test, y_test), batch_size=64)
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 76s 94ms/step - accuracy: 0.3030 - loss: 1.8829 - val_accuracy: 0.5365 - val_loss: 1.3018
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 95ms/step - accuracy: 0.5233 - loss: 1.3404 - val_accuracy: 0.6167 - val_loss: 1.0810
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 95ms/step - accuracy: 0.5973 - loss: 1.1492 - val_accuracy: 0.6558 - val_loss: 0.9809
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 73s 94ms/step - accuracy: 0.6409 - loss: 1.0255 - val_accuracy: 0.6798 - val_loss: 0.9026
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 83s 95ms/step - accuracy: 0.6749 - loss: 0.9338 - val_accuracy: 0.6959 - val_loss: 0.8750
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 81s 93ms/step - accuracy: 0.6952 - loss: 0.8802 - val_accuracy: 0.6786 - val_loss: 0.9099
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 81s 92ms/step - accuracy: 0.7187 - loss: 0.8119 - val_accuracy: 0.6877 - val_loss: 0.9233
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 92ms/step - accuracy: 0.7405 - loss: 0.7548 - val_accuracy: 0.7228 - val_loss: 0.8046
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 93ms/step - accuracy: 0.7511 - loss: 0.7190 - val_accuracy: 0.7300 - val_loss: 0.7867
```

```
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 74s 95ms/step - accuracy: 0.7659 - loss: 0.6742 - val_accuracy: 0.7364 - val_loss: 0.7807
```

```python
# Evaluate model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.7336 - loss: 0.7835
Test Accuracy: 73.64%
```

```python
# Predictions
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step
```

```python
# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```