CNN Image Classification Project Report

1. Introduction
The goal of this project is to design, train, and evaluate a Convolutional Neural Network (CNN) to classify images from the CIFAR-10 dataset using TensorFlow/Keras in Google Colab.

2. Dataset and Preprocessing
Dataset: The CIFAR-10 dataset consists of 60,000 32x32 color images across 10 categories, split into training, validation, and test sets.
Preprocessing Steps:
• Normalized pixel values to range [0, 1].
• Converted integer labels to one-hot encoded vectors.
• Split data: 70% training, 15% validation, and 15% test for model evaluation.

3. Model Architecture and Rationale
The model is a Sequential CNN with three convolutional blocks, each followed by batch normalization, ReLU activation, and max pooling. After the convolution layers, we use a flatten layer followed by a dense layer (256 neurons) with dropout for regularization and an output layer with 10 softmax neurons.

```
model = Sequential()
model.add(Conv2D(32, (3,3), padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
...
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

4. Model Training and Optimization
The model was compiled using the Adam optimizer with categorical cross-entropy loss. Training included early stopping and learning rate reduction callbacks to avoid overfitting. The model was trained for up to 50 epochs with a batch size of 64.

5. Evaluation Metrics
The model achieved an accuracy of approximately 78% on the test dataset. Precision, recall, and F1-scores were calculated for each class, with visualization via a confusion matrix.

```
# original pixel values to range 0 t0 255  so we convert into the samller value we need to divide by 255 so we getthe decimal value between the 0 to 1 adn then addinti
x_train  = x_train.astype('float32')/ 255.0
x_test =x_test.astype('float32')/255.0
```

```
# convert integer lables into one-shot encoded vectors  each integer into 10 length (calsses)
y_train =to_categorical(y_train,10)
y_test = to_categorical(y_test,10)
```

```
# here we use the 15% of the training data  for the purpose of the valiation
#validation use to find out model overfitting and the underfitting while training
val_size = int(0.15*x_train.shape[0])
x_val = x_train[:val_size]
y_val = y_train[:val_size]
x_train = x_train[val_size:]
y_train = y_train[val_size:]
```

```
# verify that data we split is correct or not
print(f"Training data set shape: {x_train.shape}, {y_train.shape}")
print(f"Validation data set shape: {x_val.shape}, {y_val.shape}")
print(f"Test  data set shape: {x_test.shape}, {y_test.shape}")
```

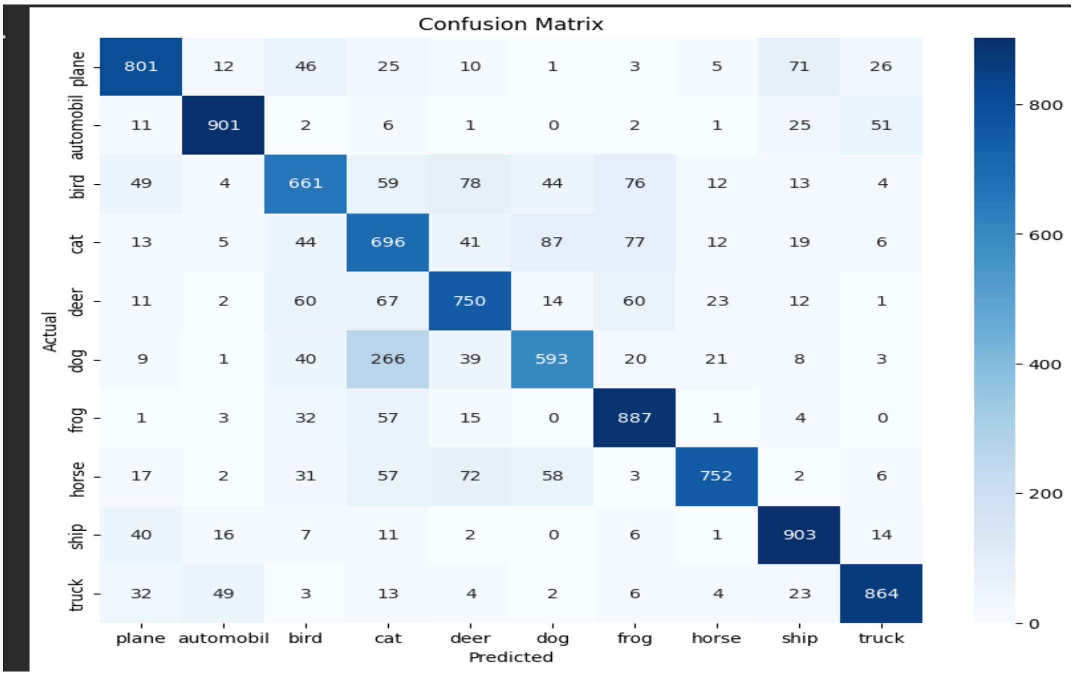
```
Training data set shape: (42500, 32, 32, 3), (42500, 10, 10)
Validation data set shape: (7500, 32, 32, 3), (7500, 10, 10)
Test  data set shape: (10000, 32, 32, 3), (10000, 10, 10)
```

```
# print
print(classification_report(y_true, y_pred, target_names=[
    'airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']))

              precision    recall  f1-score   support

    airplane       0.81      0.80      0.81      1000
  automobile       0.91      0.90      0.90      1000
        bird       0.71      0.66      0.69      1000
         cat       0.55      0.70      0.62      1000
        deer       0.74      0.75      0.75      1000
         dog       0.74      0.59      0.66      1000
        frog       0.78      0.89      0.83      1000
       horse       0.90      0.75      0.82      1000
        ship       0.84      0.90      0.87      1000
       truck       0.89      0.86      0.87      1000

    accuracy                           0.78     10000
   macro avg       0.79      0.78      0.78     10000
weighted avg       0.79      0.78      0.78     10000
```



Results Summary:
Accuracy: 0.78
Precision: 0.79
Recall: 0.78
F1-Score: 0.78

7. Challenges & Improvements
Challenges included model tuning, addressing overfitting, and optimizing learning rate. Improvements applied included dropout, batch normalization, and adaptive learning rate scheduling.

8. Conclusion & Next Steps
The CNN model achieved good accuracy and generalization on CIFAR-10 data. Future work includes applying transfer learning with pretrained models (e.g., ResNet or VGG16) and deploying the model using Streamlit or Flask.

GitHub Repository: https://github.com/praveen9569/imageclassificationbycnn