2306AML117 MAHESH BABU M ASSIGNMENT 12

In [3]:
```
import tensorflow as tf  # IMPORTING THE REQUIRED LIBRARIES
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt # plotting library
%matplotlib inline

from keras.models import Sequential
from keras.layers import Dense , Activation, Dropout
from keras.optimizers import Adam ,RMSprop
from keras import  backend as K
```

CIFAR10 DATA SET

The CIFAR-10 dataset is a widely used benchmark dataset in the field of machine learning and computer vision. It comprises 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 test images, making it suitable for training and evaluating machine learning models, particularly for image classification tasks.

Here are some key points about the CIFAR-10 dataset:

1. Classes: The dataset consists of 10 classes, each representing a specific object or category: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

In [4]:
```
# import dataset
from keras.datasets import mnist

# load dataset
(x_train, y_train),(x_test, y_test) = cifar10.load_data()

# count the number of unique train labels
unique, counts = np.unique(y_train, return_counts=True)
print("Train labels: ", dict(zip(unique, counts)))

# count the number of unique test labels
unique, counts = np.unique(y_test, return_counts=True)
print("\nTest labels: ", dict(zip(unique, counts)))
```

```
Train labels:  {0: 5000, 1: 5000, 2: 5000, 3: 5000, 4: 5000, 5: 5000, 6: 5000, 7: 5000, 8: 5000, 9: 5000}

Test labels:  {0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000, 5: 1000, 6: 1000, 7: 1000, 8: 1000, 9: 1000}
```

In [5]:
```
y_train
```

Out[5]:
```
array([[6],
       [9],
       [9],
       ...,
       [9],
       [1],
       [1]], dtype=uint8)
```

In [6]:
```
y_test
```

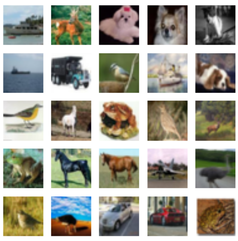
Out[6]:
```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

3. Data visualization

In [7]:
```
# sample 25 mnist digits from train dataset
indexes = np.random.randint(0, x_train.shape[0], size=25)
images = x_train[indexes]
labels = y_train[indexes]

plt.figure(figsize=(5,5))
for i in range(len(indexes)):
    plt.subplot(5, 5, i + 1)
    image = images[i]
    plt.imshow(image, cmap='gray')
    plt.axis('off')

plt.show()
plt.savefig("cifar10-samples.png")
plt.close('all')
```



4. Designing model architecture using the keras

4. 1 Compute the number of labels

In [8]:
```
# compute the number of labels
num_labels = len(np.unique(y_train))
```

4.2 Data Preprocessing

In [9]:
```
# Preprocess the data
x_train, x_test = x_train / 255.0, x_test / 255.0
```

4.3 Designing the model architecture

In [10]:
```
# Build the model
model = Sequential([
    Flatten(input_shape=(32, 32, 3)),  # Flatten the input images
    Dense(512, activation='relu'),     # Fully connected layer with ReLU activation
    Dense(256, activation='relu'),     # Fully connected layer with ReLU activation
    Dense(10, activation='softmax')   # Output layer with 10 classes using softmax activation
])
```

view model summery

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 512)               1573376

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 1707274 (6.51 MB)
Trainable params: 1707274 (6.51 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The above listing shows the model summary of the proposed network. It requires a total of 1707274 parameters.

This is substantial considering that we have a simple task of classifying cifar10 digits. So, MLPs are not parameter efficient.

The total number of parameters required can be computed as follows:

From input to Dense layer: 6145 × 256 + 256 = 1573376.

From first Dense to second Dense: (256 × 256)2 + 256 = 131328

From second Dense to the output layer: 10 × 256 + 10 = 2,570.

The total is 1573376 + 131328 + 2,570 = 1707274.

5. Implement MLP model using Keras

5.1 Compile the model with compile() method

In [16]:
```
from tensorflow.keras.optimizers import SGD  # Ensure to import SGD optimizer
from tensorflow.keras.optimizers import Adagrad  # Make sure to import Adagrad optimizer
```

In [17]:
```
# Compile the model
# Compile the model with different optimizers
optimizers = {
    'SGD': SGD(),
    'Adam': Adam(),
    'RMSprop': RMSprop(),
    'Adagrad': Adagrad()
}
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Optimization (optimizer adam)

With optimization, the objective is to minimize the loss function. The idea is that if the loss is reduced to an acceptable level, the model has indirectly learned the function mapping input to output.

In Keras, there are several choices for optimizers. The most commonly used optimizers are; Stochastic Gradient Descent (SGD), Adaptive Moments (Adam) and Root Mean Squared Propagation (RMSprop).

Each optimizer features tunable parameters like learning rate, momentum, and decay.

Adam and RMSprop are variations of SGD with adaptive learning rates. In the proposed classifier network, Adam is used since it has the highest test accuracy.

Metrics (accuracy)

Performance metrics are used to determine if a model has learned the underlying data distribution. The default metric in Keras is loss.

During training, validation, and testing, other metrics such as accuracy can also be included.

Accuracy is the percent, or fraction, of correct predictions based on ground truth.

5.2 Train the model with fit() method

In [20]:
```
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

```
Epoch 1/5
782/782 [==============================] - 57s 73ms/step - loss: 1.4518 - accuracy: 0.4831 - val_loss: 1.4947 - val_accuracy: 0.4611
Epoch 2/5
782/782 [==============================] - 61s 78ms/step - loss: 1.4205 - accuracy: 0.4911 - val_loss: 1.4731 - val_accuracy: 0.4742
Epoch 3/5
782/782 [==============================] - 63s 81ms/step - loss: 1.4045 - accuracy: 0.4978 - val_loss: 1.4373 - val_accuracy: 0.4830
Epoch 4/5
782/782 [==============================] - 67s 86ms/step - loss: 1.3788 - accuracy: 0.5073 - val_loss: 1.4771 - val_accuracy: 0.4796
Epoch 5/5
782/782 [==============================] - 69s 88ms/step - loss: 1.3520 - accuracy: 0.5177 - val_loss: 1.4171 - val_accuracy: 0.4968
```

Out[20]:
```
<keras.src.callbacks.History at 0x27c8c7081f0>
```

5.3 Evaluating model performance with evaluate() method

In [21]:
```
model.metrics_names
```

Out[21]:
```
['loss', 'accuracy']
```

In [23]:
```
loss, acc = model.evaluate(x_test, y_test, batch_size=128)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
```

```
79/79 [==============================] - 2s 23ms/step - loss: 1.4171 - accuracy: 0.4968

Test accuracy: 49.7%
```

In [24]:
```
from sklearn.metrics import classification_report

import numpy as np
y_predict = np.argmax(model.predict(x_test), axis=-1)
y_predict
```

```
313/313 [==============================] - 3s 8ms/step
array([3, 9, 1, ..., 5, 2, 7], dtype=int64)
```

Out[24]:

In [ ]:
```

```