Lab 2

Deep Learning – B.Sc Data Science and Analytics (Hons.)

Fifth Semester 2023

Lab Date/Time: 12/07/2023

Theory and Lab Faculty: Mr. Kunal Dey

Lab Description

In this lab, you will learn about building and training a deep neural network model using TensorFlow and Keras for image classification on the CIFAR-10 dataset.

Objectives

Lab Tool(s)

https://www.kaggle.com/
and
https://colab.research.google.com/

Lab Deliverables

At the end of the semester, you are required to submit the lab record, before the completion of final CA.

References:

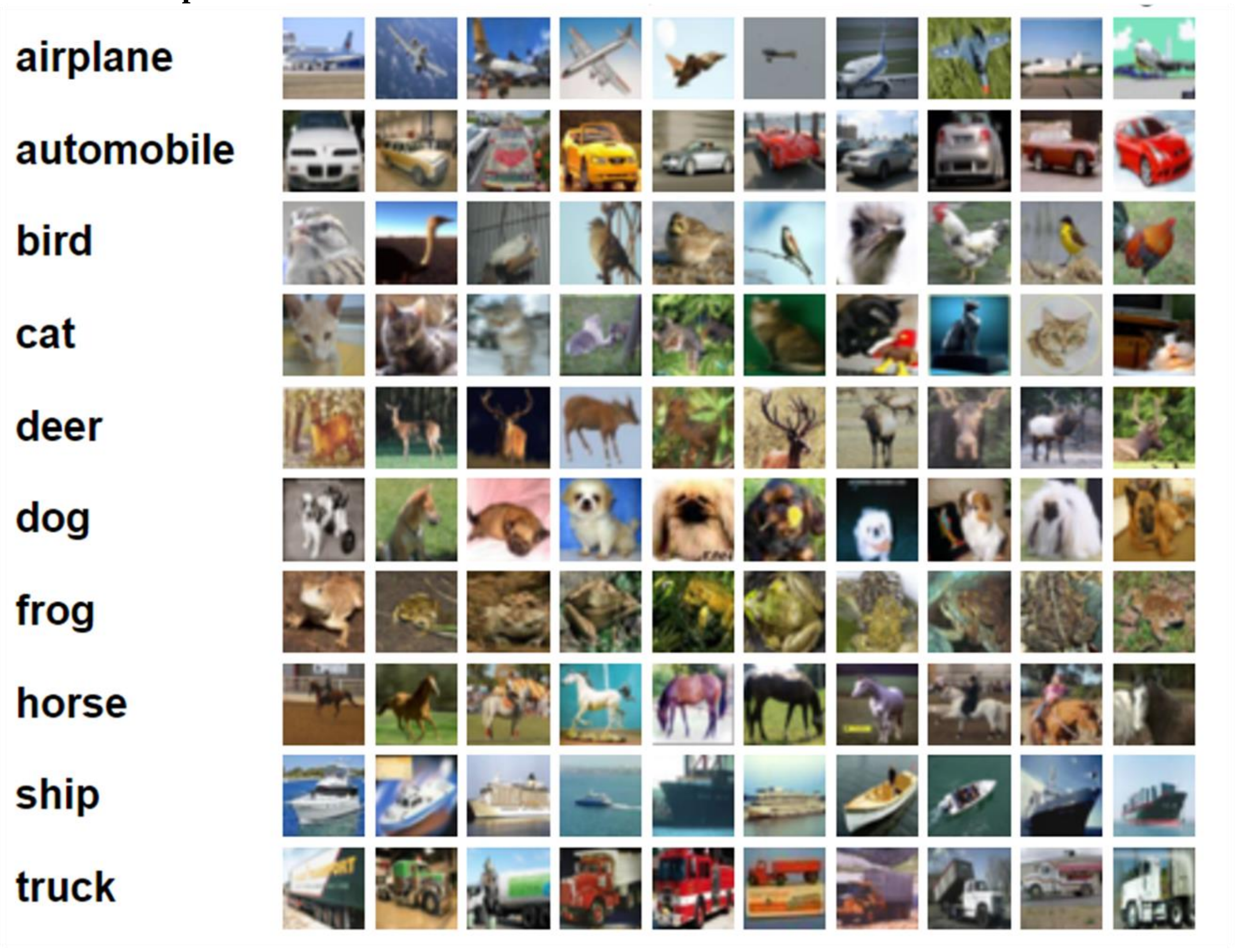
https://github.com/Drakunal/Deep-Learning

Exploratory Data Analysis

2- Image Classification with a Fully Connected Neural Network using the CIFAR10 Dataset

Link - https://www.cs.toronto.edu/~kriz/cifar.html

Dataset sample -



1.1 Using Keras and Tensorflow

Importing the Necessary Libraries

```
import numpy as np
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import cv2
from PIL import Image
import matplotlib.pyplot as plt
```

In this section, necessary libraries and modules are imported. numpy is imported as np for numerical operations. tensorflow and its submodules (keras, datasets, models, layers) are imported. cv2 is imported for image processing operations, PIL (Python Imaging Library) is imported for handling images, and matplotlib.pyplot is imported for plotting images and visualizations.purposes.

Dataset Loading

This code loads the CIFAR-10 dataset, which is a popular benchmark dataset for image classification tasks. The dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The training set (x_train0, y_train0) and the test set (x_test0, y_test0) are loaded using the cifar10.load_data() function:

```
# Load CIFAR-10 dataset
(x_train0, y_train0), (x_test0, y_test0) = cifar10.load_data()
 x_train0.shape
```

This line prints the shape of the x_train0 array, which represents the shape of the training images. It gives an idea of the number of training examples and their dimensions.

Dataset Preprocessing

In this step, the pixel values of the images in x_train0 and x_test0 are preprocessed. The pixel values are converted to floating-point numbers and then normalized by dividing by 255.0. This step scales the pixel values between 0 and 1, which helps in training the neural network:

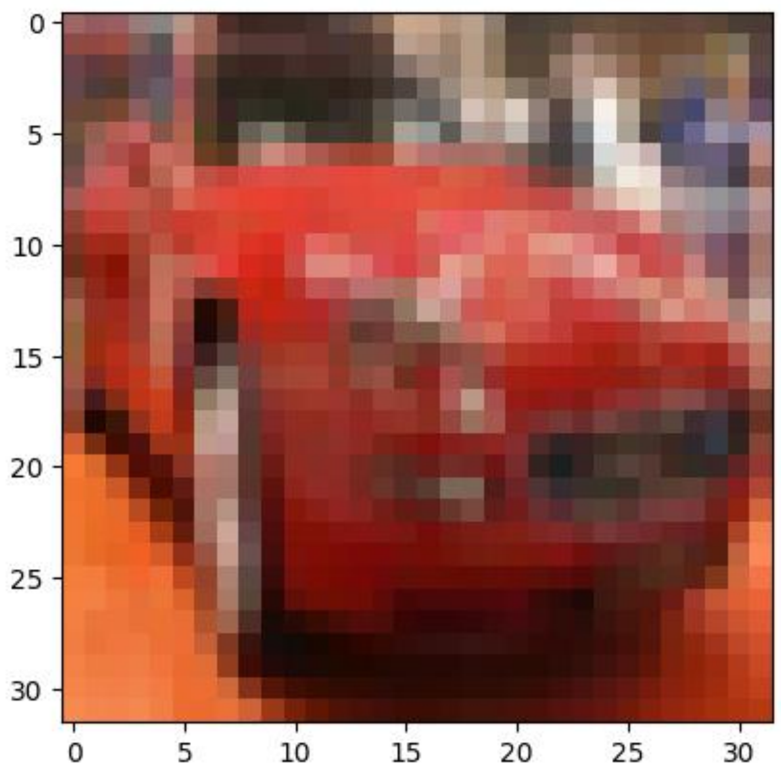
```
# Preprocess the data
x_train = x_train0.astype('float32') / 255.0
x_test = x_test0.astype('float32') / 255.0

# Convert class vectors to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train0, num_classes)
y_test = keras.utils.to_categorical(y_test0, num_classes)
```

In this step, the class labels for the training and test sets, y_train0 and y_test0, are converted from integers to binary class matrices using keras.utils.to_categorical(). This is necessary for training a neural network with categorical cross-entropy loss, as it requires one-hot encoded class labels.

Displaying an image

These lines use PIL to create an image from the array x_train0[5], which represents the 6th training image. The image is then displayed using plt.imshow() from matplotlib.pyplot. This is a visual check to see one of the images from the dataset:

```
img = Image.fromarray(x_train0[5], "RGB")
plt.imshow(img)
```



Model Architecture

A sequential model is a linear stack of layers where each layer has exactly one input tensor and one output tensor. This is a simple and common type of model for deep learning tasks. Here, a flatten layer is added to the model using model.add(). The flatten layer converts the input data, which is a 3D tensor representing the images, into a 1D vector. The input_shape argument is set to x_train.shape[1:], which is the shape of a single image without the batch size:

```
# Create a sequential model
model = Sequential()

# Flatten the input data
model.add(Flatten(input_shape=x_train.shape[1:]))

# Add fully connected layers
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))

# Add output layer
```

```
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Here we also have added fully connected layers to the model using model.add(). Each dense layer has a specified number of units (neurons) and uses the ReLU activation function. The choice of the number of units and the number of layers depends on the specific problem and can be adjusted to improve performance.
Here, the output layer is added to the model using model.add(). The output layer has num_classes units, corresponding to the number of classes in the classification task, and uses the softmax activation function. The softmax function outputs a probability distribution over the classes, indicating the model's confidence for each class.
The last line compiles the model using model.compile(). The loss parameter is set to 'categorical_crossentropy', which is a common loss function used for multi-class classification tasks. The optimizer is set to 'adam', which is a popular optimization algorithm. The metrics parameter is set to ['accuracy'], which specifies that the model's performance will be evaluated based on classification accuracy during training and evaluation.

Model Training

This code trains the model using the fit() function of the model object. The training data (x_train, y_train) is used for training. The batch_size parameter specifies the number of samples per gradient update, and the epochs parameter specifies the number of times the entire training dataset is traversed during training. The validation_data parameter is set to (x_test, y_test) to evaluate the model's performance on the test set after each epoch:

```
# Train the model
batch_size = 128
epochs = 10
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(x_test, y_test))
```

Model Evaluation

These lines evaluate the trained model on the test set using model.evaluate(). The evaluation result, including the test loss and test accuracy, is stored in the scores variable. The test loss is printed using print().:

```
# Evaluate the model
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
print(confusion_mat)

predictions = model.predict(x_test)
```

The last line uses the trained model to make predictions on the test set (x_test). The predict() function returns the predicted class probabilities for each input sample in x_test, and the result is stored in the predictions variable.

Result

In this loop, the code iterates over the first six test samples. For each sample, an image is created using PIL from the x_test0 array. The image is displayed using plt.imshow(). The predicted label for the image is obtained by finding the index of the highest predicted probability using np.argmax(). The predicted label is then added as text to the image plot using plt.text(). Finally, plt.axis('off') removes the axis ticks, and plt.show() displays the image with the predicted label.

```
classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","
truck"]

for i in range(6):
    img = Image.fromarray(x_test0[i], "RGB")
    plt.imshow(img)
    predicted_label = np.argmax(predictions[i])
    #print(classes[predicted_label])
    plt.text(0, -5, f"Predicted: {classes[predicted_label]}",
color='red')  # Add predicted label as text
    plt.axis('off')  # Remove axis ticks
    plt.show()
```

Predicted: cat

Predicted: ship