

**Implementation of CIFAR10 with
CNNs Using TensorFlow**

Javier A. Diaz Velazquez

Colorado State University Global

CSC580: Applying Machine Learning and
Neural Networks

Dr. Issa

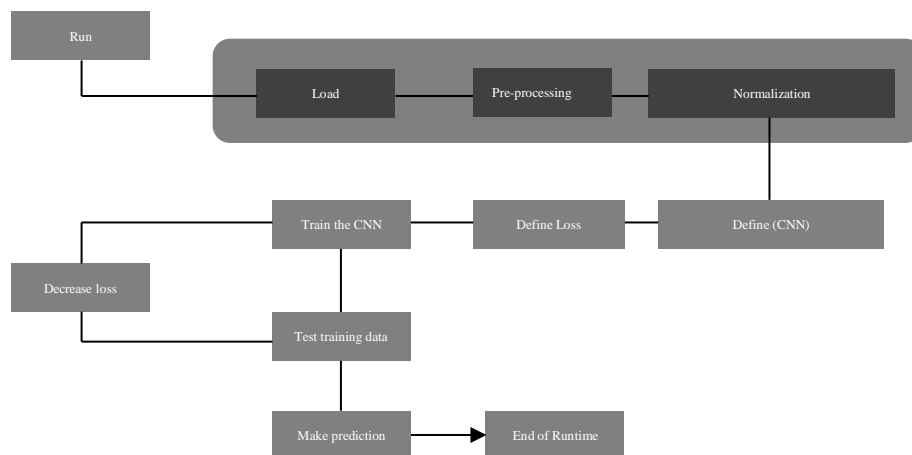
August 27, 2023

CIFAR10 Dataset

The CIFAR-10 is employed to benchmark artificial intelligence, machine learning, and computer vision algorithms. The Canadian Institute for Advanced Research (CIFAR) developed the dataset. The dataset contains over 60,000 (32x32) color images, with ten different classes or categories, each containing 6,000 images. These images are divided into a training set of 50,000 and a test set of 10,000. The dataset is commonly used for tasks like image classification, object recognition, and machine learning algorithm evaluation. Furthermore, the dataset has been utilized in various research and development to gauge the performance of various image classification artificial neural network architectures.

The Research

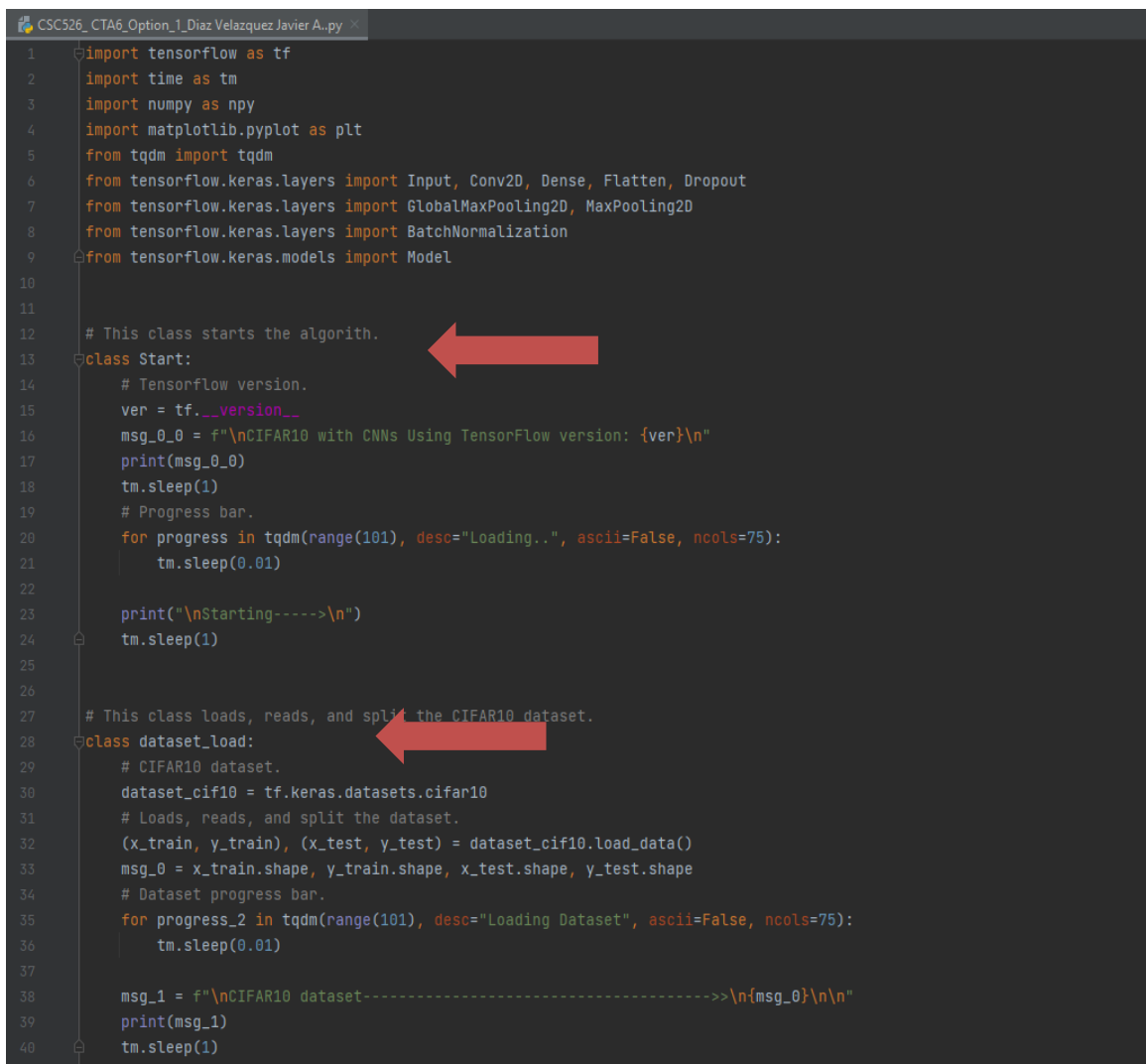
The primary purpose of the research was to develop a convolutional neural network (CNN) in which the CIFAR10 dataset was employed to benchmark and evaluate the performance during training, testing, and classification of the convolutional neural network. Moreover, to evaluate the implementation of the data pipeline used to load, read, split, normalize, train, test, and predict the implementation. Additionally, the following flowchart must be utilized as an implementation map;



Algorithm development flowchart.

CIFAR10 Convolutional Neural Network Implementation

Based on the proposed flowchart used as a guideline for the development of the data pipeline and the convolutional neural network, the architecture of the algorithm is built with the following parameters: the algorithm has been built using classes, which makes the code more organized, easy to read and understand, and to debug when need it. In addition, the use of classes also makes the code modular, meaning that the implementation within such class shall be adaptable and reusable. Figure 1 – 1 and 1 - 2 shows the overall architecture of the algorithm.



```

1  import tensorflow as tf
2  import time as tm
3  import numpy as npy
4  import matplotlib.pyplot as plt
5  from tqdm import tqdm
6  from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
7  from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
8  from tensorflow.keras.layers import BatchNormalization
9  from tensorflow.keras.models import Model
10
11
12  # This class starts the algorithm.
13  class Start:
14      # Tensorflow version.
15      ver = tf.__version__
16      msg_0_0 = f"\nCIFAR10 with CNNs Using TensorFlow version: {ver}\n"
17      print(msg_0_0)
18      tm.sleep(1)
19      # Progress bar.
20      for progress in tqdm(range(101), desc="Loading..", ascii=False, ncols=75):
21          tm.sleep(0.01)
22
23      print("\nStarting----->\n")
24      tm.sleep(1)
25
26
27  # This class loads, reads, and split the CIFAR10 dataset.
28  class dataset_load:
29      # CIFAR10 dataset.
30      dataset_cif10 = tf.keras.datasets.cifar10
31      # Loads, reads, and split the dataset.
32      (x_train, y_train), (x_test, y_test) = dataset_cif10.load_data()
33      msg_0 = x_train.shape, y_train.shape, x_test.shape, y_test.shape
34      # Dataset progress bar.
35      for progress_2 in tqdm(range(101), desc="Loading Dataset", ascii=False, ncols=75):
36          tm.sleep(0.01)
37
38      msg_1 = f"\nCIFAR10 dataset----->\n{msg_0}\n\n"
39      print(msg_1)
40      tm.sleep(1)

```

Figure 1-1 shows that the architecture of the algorithm has been developed following the dataflow demonstrated on the flowchart—Additionally, the use of class provides modularity, flexibility, and adaptability of the code.

```

43 # This class normalizes the test data.
44 class pre_processing:
45     # Splits and normalises the dataset.
46     dataset_load.x_train, dataset_load.x_test = dataset_load.x_train / 255.0, dataset_load.x_test / 255.0
47     dataset_load.y_train, dataset_load.y_test = dataset_load.y_train.flatten(), dataset_load.y_test.flatten()
48
49
50 # This class verifies the dataset classification.
51 class data_verification_Val:
52     dataset_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
53
54     plt.figure(figsize=(10, 10))
55     for labels in range(25):
56         plt.subplot(5, 5, labels + 1)
57         plt.xticks([])
58         plt.yticks([])
59         plt.grid(False)
60         plt.imshow(dataset_load.x_train[labels])
61         plt.xlabel(dataset_labels[dataset_load.y_train[labels]])
62     plt.show()
63
64
65 # This class houses the Convolution Neural Network (CNN).
66 class CNN:
67     # CNN defined.
68     CIFAR10_Model = tf.keras.Sequential()
69     CIFAR10_Model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
70     CIFAR10_Model.add(tf.keras.layers.MaxPooling2D((2, 2)))
71     CIFAR10_Model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
72     CIFAR10_Model.add(tf.keras.layers.MaxPooling2D((2, 2)))
73     CIFAR10_Model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
74     CIFAR10_Model.add(tf.keras.layers.MaxPooling2D((2, 2)))
75     CIFAR10_Model.add(tf.keras.layers.Flatten())
76     CIFAR10_Model.add(tf.keras.layers.Dropout(0.2))
77     CIFAR10_Model.add(tf.keras.layers.Dense(512, activation='relu'))
78     CIFAR10_Model.add(tf.keras.layers.Dropout(0.2))
79     CIFAR10_Model.add(tf.keras.layers.Dense(10))
80
81     # CNN model summary.
82     CIFAR10_Model.summary()

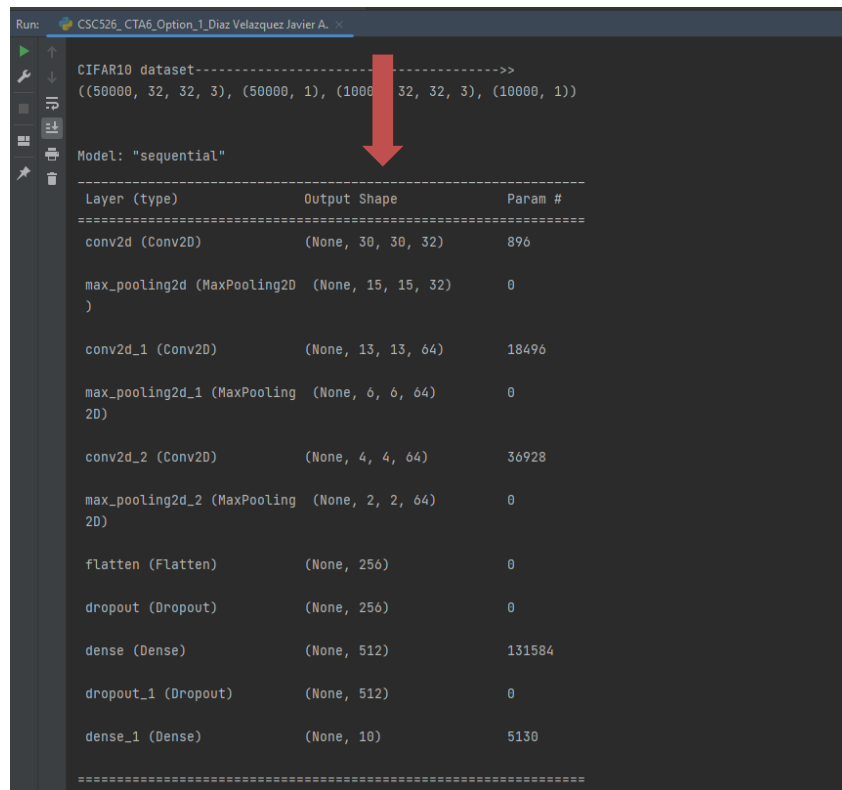
```

Figure 1 – 2 shows the convolutional neural network architecture implemented using classes.

Now, the convolutional neural network architecture implemented for this particular type of task consisted of the following parameters: the input layer with 32 filters or neurons with a size of 3x3 and ReLU activation function, the input shape refers to the size of the image's pixels with three color channels usually referring to the RGB channels. Each convolutional layer has a max pooling layer that performs a max pooling of 2x2, reducing the dimensions. Two more convolutional layers have been added with 64 neurons with the same ReLU activation function; the flattening layer reshapes the 3D shape into a one-dimensional vector.

A dropout layer has also been added after the flattening layer and the dense layers with a dropout rate of 0.2; this prevents the neural network from overfitting. Furthermore, fully connected layers were also added. The last dense layer has ten units, the number of classes in the CIFAR-10 dataset. No activation function is specified for the last layer, as it is utilized for generating raw scores/logits.

The CNN architecture follows the pattern of convolutional layers followed by max-pooling layers, leading to a sequence of hierarchical feature extraction. The flattened features are then passed through fully connected layers for classification. Dropout layers are used for regularization; the final layer provides the raw class scores. Figure 1 – 3 shows the CNN parameters summary.



Run: CSC526_CTA6_Option_1_Diaz Velazquez Javier A.

CIFAR10 dataset----->>>
 ((50000, 32, 32, 3), (50000, 1), (10000, 32, 32, 3), (10000, 1))

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten (Flatten)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 512)	131584
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130

Figure 1 – 3 shows the parameters of the convolutional neural network during runtime execution.

CIFAR10 Model's Accuracy with CPU

After the code completion per the flowchart parameters, two different tests were performed to gauge the algorithm's accuracy and improvement "if it is any!". The first test was performed using the computers' CPU, which yielded 71% accuracy with 83% loss during the first run of ten epochs. The code repeated the process to reduce the loss but this time with different hyperparameters, this time with 32 batch sizes, an image data generator, 50 epochs, and re-fit the training data to the CNN model for training, which yielded a new accuracy of 78% a gaining of 7%, decreasing the loss at 64% reducing the loss by 19%. Despite that, the accuracy only increased by 7%, which could indicate balanced training with no underfitting.

CIFAR10 Model's Accuracy with GPU

The algorithm was also subjected to a training session utilizing the computer's GPU with the same hyperparameters to gauge and benchmark if the model's accuracy increased higher than the yielded 7% increase. The results yielded by the training performed with the GPU were surprisingly closer to the results performed by the CPU execution test; the first training pass yielded a 71% accuracy with an 84% loss. The second pass performed with the same hyperparameters yielded a 76% accuracy with a 72% loss, gaining only 6% accuracy and reducing the loss by 12%. Judging by those results, it could mean that more configuration is required to perform better when utilizing the GPU. However, based on the prediction, it could be concluded that the model can classify the right image with its respecting label.

Conclusion

Although the model has been subjected to a training session with the CPU and the GPU, the results have demonstrated a well-balanced training in which the algorithm can identify and classify the images with their respective labels. Finally, despite different hyperparameters being tested in this implementation, the results were close to each other in every test, and others error out.