# **Imcla**

Image classification using a neural network (CNN)

### Project Statement

* Choose a dataset: The first step in this project is to choose a dataset to train your neural network. Here, I used the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.
* Preprocess the dataset: Before training the neural network, you need to preprocess the dataset. This includes normalizing the pixel values, splitting the dataset into training and validation sets, and converting the labels into one-hot vectors.
* Build the neural network: The next step is to build the neural network architecture. You can start with a simple architecture like a single hidden layer with 100 neurons and a softmax output layer.
* Train the neural network: Once you have built the neural network architecture, you can start training it on the preprocessed dataset. You can use backpropagation and stochastic gradient descent to update the weights and biases of the neural network.
* Evaluate the neural network: After training the neural network, you can evaluate its performance on the validation set. You can use metrics like accuracy, precision, and recall to measure the performance of the neural network.
* Fine-tune the neural network: Based on the evaluation results, you can fine-tune the neural network architecture by adding more hidden layers, increasing the number of neurons in the hidden layers, or changing the activation functions.
* Test the neural network: Finally, you can test the neural network on the test set to see how well it generalizes to new data.

### Solution

This code trains a convolutional neural network (CNN) to classify images from the CIFAR-10 dataset. Let's go through the code line by line to understand it in detail:

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.utils import to\_categorical

Here, we import the necessary libraries and modules. We use TensorFlow 2.0, Keras API, and load the CIFAR-10 dataset from Keras datasets. We also import different types of layers for our neural network architecture.

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

We load the CIFAR-10 dataset and split it into training and test sets.

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

The pixel values in the dataset range from 0 to 255. We normalize the pixel values by dividing them by 255.0, so they are in the range 0 to 1.

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

We convert the labels to one-hot encoded vectors using the `to\_categorical` function. For example, a label of 2 is represented as [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))

We define our neural network architecture using the Sequential model from Keras. We add three convolutional layers with 32, 64, and 64 filters, respectively, using 3x3 kernel size and ReLU activation function. We also add two MaxPooling2D layers with 2x2 pool size to downsample the output of the convolutional layers. We then flatten the output of the last convolutional layer and add two dense layers with 64 and 10 units, respectively. The last dense layer has a softmax activation function, which gives us the probability distribution over the 10 classes.

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

We compile the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric.

model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_data=(x\_test, y\_test))

We train the model on the training data for 10 epochs with a batch size of 64. We also use the validation set to evaluate the model after each epoch.

loss, accuracy = model.evaluate(x\_test, y\_test)

print('Test loss:', loss)

print('Test accuracy:', accuracy)

Finally, we evaluate the performance of our model on the test set by computing the loss and accuracy. We print the results on the console.