**Training a Convolution Network with the CIFAR-10 dataset using the TensorFlow framework**

CIFAR-10

The acronym CIFAR stands for Canadian Institute For Advanced Research. CIFAR-10 is a set of images and is widely used in the field of machine learning and more specifically, computer vision. The data was collected by pioneers of the machine learning field; Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The variety of pictures and the low resolution of the images it contains makes it ideal to train convolutional neural networks. The dataset consists of a collection of 60,000 images across 10 different classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Each asset per class is a colour image and has a size of 32 pixels x 32 pixels. The color system used is RGB: - Red, Green, Blue. The red, green, and blue use 8 bits each, which have integer values from 0 to 255. This makes 256\*256\*256=16,777,216 possible colours.

**Analysing and preparing the CIFAR-10 dataset**

To prepare the **CIFAR-10 dataset** for training a Convolutional Neural Network (CNN), the dataset needs to be split into three subsets: training set, validation set, and test set.

When the dataset is initially loaded the function calls returns 4 vectors of type numpy.ndarray. The function will return by default the following vectors:

* The features training (x\_train\_all) data has a length of 50000. Its shape is (50000, 32, 32, 3).
* The labels training (y\_train\_all) data has a length of 50000. Its shape is (50000,1).
* The features testing set also contains a vector which we will call x\_test, and it has the following shape: (10000, 32, 32, 3).
* The labels test data which we will call y\_test consists in a vector of 10000 elements, it has the shape (10000, 1)
* The format of the data we receive is neither quite adapted nor optimised for multi-class classification yet. We need to apply a couple of changes on the data in order to comply with the format required by our model.

Normalising data

Firstly, we can scale down the image's pixel values to floating points between 0.00 and 1.00. Numpy arrays allow us to compute this operation by simply dividing the vectors by 255 and it will recursively apply the division to all elements in the vector.

The normalised vectors will enhance the learning performance and speed up the convergence of our model.

Encoding data  
  
We then perform an operation called “One-Hot Encoding” on the label vectors which consists of transforming the categorical labels into vectors of 10 binary elements arrays, keeping only the label category as 1 and the others as 0. The computation converts the y axis categories into a suitable format used for neural networks multi-class classification.

We assign the resulting arrays to the variables y\_cat\_train\_all and y\_cat\_test, respectively.

Splitting data

The validation set is a subset of the training set. It provides unseen data during the training process to perform cross-validation between the training and validation accuracy and concurrently monitor the evolution of the loss function results for the network to learn and adjust its hyperparameters.

We will create validation datasets x\_val and y\_cat\_val which are a subset of x\_train\_all and y\_cat\_train\_all.  
x\_val will contain the 10000 first elements in x\_train\_all whereas y\_cat\_val will contain the 10000 first elements of y\_cat\_train\_all

We can now create the training arrays x\_train and y\_train which contains respectively the last 40000 elements of x\_train\_all and the last 40000 element of y\_cat\_train\_all

In order to evaluate the performance of the model after training, we will use the testing sets x\_test with dimension (10000, 32, 32, 3) and y\_cat\_test with shape (10000, 1). They will provide an unbiased assessment of the predictive performance of the model after the training has completed and our model has converged.

**The architecture of the Artificial Neural Network**

Before delving into the architecture of the model, it is required to understand what operation is performed on each type of layer in the network.

Convolution  
  
Convolution is a mathematical operation which consists of computing the dot product of 2 tensors and results in an output tensor called the feature map. Correlated to the CIFAR-10 image size, 32x32 pixels, in the current case, the input tensor **A** has 32 rows and 32 columns, and the kernel or filter is 4 rows by 4 columns tensor **B**. During the operation, starting on the first row and column of tensor **A**, tensor **B** slides across tensor **A** on axis xusing a stride of 1 in the current model configuration. Once it has reached the width of tensor **A** it will slide down on the Y-axis by 1 stride and move back to X-axis 1 and repeat the sliding pattern. The resulting scalar product per stride is stored in a third matrix of 29 rows by 29 columns when the operation is completed.The layer will use 32 filters allowing it to extract a wide range of features within the image.

Pooling

Pooling is a downsampling method which decreases the size of the input tensor and therefore reduces the dispensable details or noise in the image. There are two types of pooling:

**Average pooling** is a downsampling method which calculates the mean of the addition of all values within a predefined area of the matrix and outputs the result to a third matrix.

For instance, the top right area of the image represented by a 4x4 tensor with values [x1 y2]=5, [x2 y2]=13, [x1 y1]=6, [x2 y1]=8 would yield a value of 8.

**Max Pooling** is also a downsampling method of the input feature map, but rather than calculating the average of a defined area in the input tensor it selects its highest value.  
For instance, the top right area of the image represented by a 4x4 tensor with values [x1 y2]=5, [x2 y2]=13, [x1 y1]=6, [x2 y1]=8 would yield a value of 13, which would be forwarded in the output tensor.

The Networkis implemented using the framework TensorFlow/Keras. This type of deep learning algorithm is designed to extract patterns from images and is used for image classification or recognition tasks, such as classifying images from the **CIFAR-10 dataset** in this instance or similar datasets with 32x32 RGB images and 10 classes.

Architecture

The model is constructed by sequentially adding different types of layers each fulfilling a different purpose within the network.

The first layer is of type Conv2D which corresponds to a convolutional layer used for feature extraction. The output is a tensor with shape (30, 30, 32).

The second layer is of type MaxPooling2D which is used to perform a 2x2 pooling on the input tensor. The downsampled feature map is a tensor with shape (15, 15, 32)

The third layer is of type Dropout. It is used to reduce overfitting by dropping neurons randomly. Dropout is a powerful technique introduced for improving the generalization error of large neural networks [(Dahl, Sainath, and Hinton 2013)](https://paperpile.com/c/3mv9Xj/2ZoOk)

The fourth layer is of type Conv2D which corresponds to a convolutional layer used for feature extraction. The output parameter is a tensor with shape (13, 13, 32).

The fifth layer is of type MaxPooling2D which is used to perform a 2x2 pooling on the input tensor. The downsampled feature map is a tensor with shape (6, 6, 32)

The sixth layer is of type Dropout. It is used to reduce overfitting by dropping neurons randomly.

The seventh layer is of type Flatten which converts the 3-dimensional feature maps received from the previous layer to a 1-dimensional vector.

The eighth layer is of type Dense which is a fully connected layer with 128 neurons. The eight layer is of type Dense which is a fully connected layer. It is the output layer and last in the network with 10 neurons which represents the 10 classes of the CIFAR-10 dataset.

**The activation function**

Activation functions are crucial to the learning process of a neural network. They establish whether neurons should be activated or not. This calculation happens during the feedforward propagation and the backpropagation operations. The activation function will produce an output result based on a set of input received by a node or neuron in the network.

Neural networks can use different activation functions. The most common functions are:  
  
The **Sigmoid** activation function which is formulated as:

The result of the function is a value between 0 and 1 crossing Y at 0.5.

**The Tahn** activation function:

similar to the Sigmoid activation function but will constrain the output of the function between -1 and 1 and it is centered at origin 0. Our convolutional neural network will use the **ReLU** (Rectified Linear Unit) activation function commonly used in deep learning, especially for convolutional neural networks and feedforward neural networks. The function introduces non-linearity to the model, allowing it to learn complex patterns.

The Relu formula is calculated as follow:

The function returns a value of zero for any negative input. If the input value of the function f is greater than 0 it will return the value f(x).

The **Softmax** activation function is used on the last layer output to transform the tensor’s result to a vector of probabilities. The output of the function will be used by the categorical cross entropy loss function and will be back propagated through the CNN using the chain rule to allow the CNN to adjust its weights.

**Training process and parameters used**

The model receives its first arguments; the x\_train and y\_cat\_train tensors which respectively contain the training data and the labels that were previously adapted for the algorithm.

The maximum iteration value through the training dataset called an epoch is set to 80, which means that in theory, the network wilI loop 80 times forward and backward through its 8 layers. However, it may never reach this value since it is set up using a callback function “EarlyStopping”, causing an early stop of the learning process after four epochs when there is no improvement in the loss validation score.

The model learning function “fit” is also set up using a callback to optimise its learning process and minimise overfitting which would prevent the model from generalising with unseen data. The callback used is “ReduceLROnPlateau”. During training, it observes the validation loss value and will trigger a call to the “fit” function to reduce the learning rate when the validation loss stops improving for a specified number of epochs. This specific callback is set to decrease the learning rate by a factor of 10 after 2 epochs if the loss validation score has plateaued. In this case the initial learning rate is set to 0.00020 and it decreases to 0.0001 by the 33rd epochs.

**Model performance analysis and key observations**

We will focus on the key metrics, the training accuracy, training loss, the validation accuracy, and the validation loss in order to determine the performance of the network in relation to the learning rate and the numbers of epochs. The model starts with a training accuracy of **25.23%** and increases to **76.62%** at the end of epoch **47** where the learning process halts. The validation accuracy is slightly lower than the training accuracy which indicates a good adaptation in generalising.

We can observe that the network has converged after 46 epochs, and the learning process stops after 47 epochs. During training, the validation loss function is 0.7982 after epoch 44 and slightly increases to 0.8034 after epoch 45

To summarize, the model has learned well, achieving a score of **76.62%** training accuracy and **72.18%** validation accuracy. We can notice that the learning rate reduction was triggered and applied after epoch 36, helping the model fine-tuning its performance.  
The model reaches convergence and stabilises, with minimal improvements in subsequent epochs.

Overall, a satisfying training result ready with good potential in predicting images on the testing set.

The evaluation method on the testing set reports that the average loss function value is **0.7630** and the accuracy of the model is 0.7184 (**71.84%**). This result means the model can generalise and predict reasonably well with unseen images. There is a subtle difference between the training and testing accuracy values, showing that there is room for improvement.

Classification Performance

**Class 0 and 1** ('airplane', 'automobile') have good performance (Precision, Recall, and F1-Score are all above 0.75).

**Class 2 and 3** ('bird','cat') have relatively lower F1-Scores (~0.61 and ~0.53, respectively), suggesting the model struggles with these classes.

**Class 4 and 5** ( 'deer', 'dog') both show decent recall, with slightly lower precision, which means there may be some false positives for these classes.

**Class 6, 7, 8, and 9** ( 'frog', 'horse', 'ship', 'truck') perform well overall, with F1-Scores close to or above 0.77, suggesting the model handles these classes effectively

Conclusion  
  
It took several iterations in adjusting the model’s hyperparameters as well as introducing an extra call back to dynamically adjust the learning rate during training in order to get to the final result. Further experimentation with the architecture and set up of the network’s parameters will be necessary to grasp the correlation between those variables and attain the desired performance.

**References**

[Bishop, Christopher M., and Hugh Bishop. 2023. *Deep Learning: Foundations and Concepts*. Springer Nature.](http://paperpile.com/b/3mv9Xj/QJdI)

[Dahl, George E., Tara N. Sainath, and Geoffrey E. Hinton. 2013. “Improving Deep Neural Networks for LVCSR Using Rectified Linear Units and Dropout.” In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 8609–13. IEEE.](http://paperpile.com/b/3mv9Xj/2ZoOk)

*Vakalopoulou M, Christodoulidis S, Burgos N, Colliot O, Lepetit V (2023) ‘Deep Learning: Basics and Convolutional Neural Networks (CNNs)’*[*(Bishop and Bishop 2023)*](https://paperpile.com/c/3mv9Xj/QJdI)