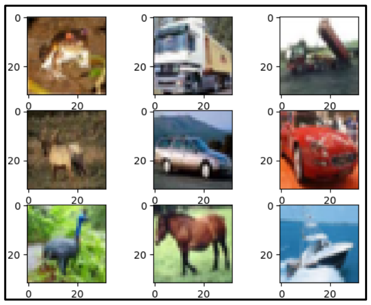
# CIFAR-10 CNN Design for Object ID

Erik Barraza Cordova

## System Design



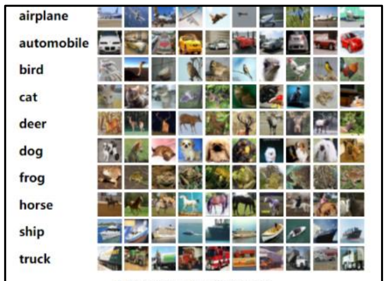


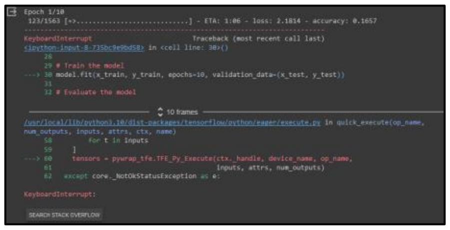
Figure 1-Cifar-10 dataset in a snapshot

Convolutional Neural Networks (CNNs) are intricate designs that enable the efficient development of models for image classification. CNNs have provide a powerful framework for understanding images. These models are meticulously designed for processing visual data.

The key architectural components of a CNN include convolutional layers, pooling layers, fully connected layers, and activation functions. Convolutional layers are responsible for feature extraction, detecting patterns in images. Pooling layers reduce spatial dimensions and enhance invariance to translations. Fully connected layers make predictions, and activation functions introduce non-linearity to those predictions. This design enables CNNs to automatically learn features from images through associated learning in ensemble networks.

## Experiment & Algorithm

Phase 1:



```
Epoch 1/10
 123/1563 [=>...........................] - ETA: 1:06 - loss: 2.1814 - accuracy: 0.1657
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-735bc9e9bd58> in <cell line: 30>()
     28 
     29 # Train the model
---> 30 model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
     31 
     32 # Evaluate the model

                              10 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58         for t in inputs
     59     ]
---> 60     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     61                                         inputs, attrs, num_outputs)
     62   except core._NotOkStatusException as e:

KeyboardInterrupt: 
```

These results come from two CNN networks that draw in 2Dimensional Convolutional layers that will enable us to compute the features of each kernel and communicate them across for the classification of the 10 classes of images for all 60k images. About 50k images are used in training whilst 10k images are used in the training and validation datasets.

In Phase 1 of this project, the primary goal is to construct a Convolutional Neural Network (CNN) architecture for image classification on the CIFAR-10 dataset. The initial design includes a series of convolutional layers, followed by a few fully connected layers.

CNN is built to process images categorized into 10 distinct classes. During this phase, we are establishing the architecture, and setting up the loss function & optimizer. The design extracts feature from the input images and lays the groundwork convolutional layers for learning outputs.

Phase 2: We build on these skills with information from the pooling layers & dropout methods.

Phase 3:

ShuffleNet is a convolutional neural network architecture designed for efficient classification. It introduces the concept of channel shuffling & pointwise group convolution to reduce the computational exertion of training while maintaining accuracy & performance. It is well-suited for mobile & embedded devices, as it reduces the number of operations and parameters while maintaining good accuracy. “ShuffleNetV2 employs a bottleneck architecture, which reduces the computational cost by using 1x1 and 3x3 convolutions to capture spatials and channels.” (Tensorflow). Transfer learning is used for the development of the lower layers to help build generic features to make the process accelerated and reduces the knowledge in large datasets into smaller ones to deploy on embedded systems with high accuracy.

*Dropout*

Dropout is a regularization technique commonly used in neural networks, including Convolutional Neural Networks (CNNs), to mitigate the risk of overfitting. Dropout selects neurons randomly and nullifies their outputs to the next layer. This leads to training on only a fraction of the information in the forward and backward pass of features and information. Doing so trains subnetworks instead of maintaining a larger network, which enables accuracy increases and help create generalizable features through subnetwork learning. Overfitting often becomes apparent when the training accuracy is significantly higher than the validation (or test) accuracy, in our model post-Phase 2, we see an accuracy of 0.8 and a validation of 0.7; this gap is an indication of overfitting.

*Application of ShuffleNet*

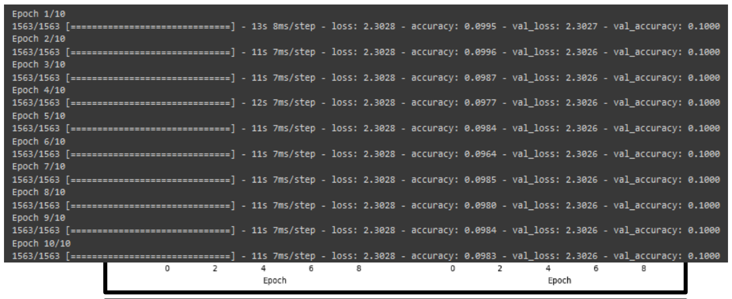
ShuffleNet is a convolutional neural network architecture designed for efficient classification. It introduces the concept of channel shuffling & pointwise group convolution to reduce the computational exertion of training while maintaining accuracy & performance. It is well-suited for mobile & embedded devices, as it reduces the number of operations and parameters while maintaining good accuracy. "ShuffleNetV2 employs a bottleneck architecture, which reduces the computational cost by using 1x1 and 3x3 convolutions to capture spatials and channels." (Tensorflow). Transfer learning is used for the development of the lower layers to help build generic features to make the process accelerated and reduces the knowledge in large datasets into smaller ones to deploy on embedded systems with high accuracy.

## Results

### <u>Phase 1 Results</u>

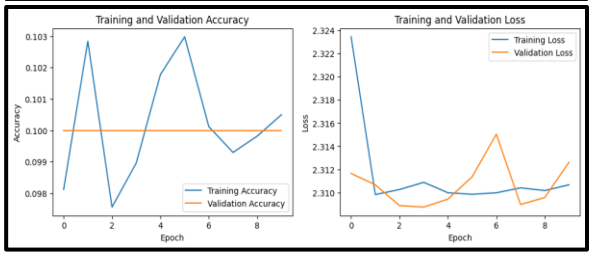
These results come from two CNN networks that draw in 2Dimensional Convolutional layers that will enable us to compute the features of each kernel and communicate them across for the classification of the 10 classes of images for all 60k images. About 50k images are used in training whilst 10k images are used in the training and validation datasets.

In Phase 1 of this project, the primary goal is to construct a Convolutional Neural Network (CNN) architecture for image classification on the CIFAR-10 dataset. The initial design includes a series of convolutional layers, followed by a few fully connected layers.

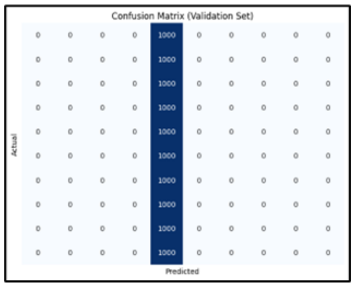


```
Epoch 1/10
1563/1563 [==============================] - 13s 8ms/step - loss: 2.3028 - accuracy: 0.0995 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 2/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 3/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0987 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 4/10
1563/1563 [==============================] - 12s 7ms/step - loss: 2.3028 - accuracy: 0.0977 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 5/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 6/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0964 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 7/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0985 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 8/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0980 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 9/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 10/10
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3028 - accuracy: 0.0983 - val_loss: 2.3026 - val_accuracy: 0.1000
```

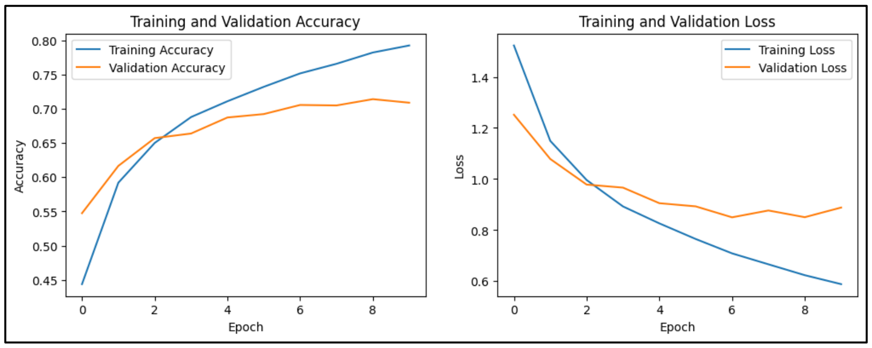
The CNN is built to process images categorized into 10 distinct classes. During this phase, we establishing the architecture, and setting up the loss function & optimizer. The design extracts features from the input images and lays the groundwork convolutional layers for learning outputs.

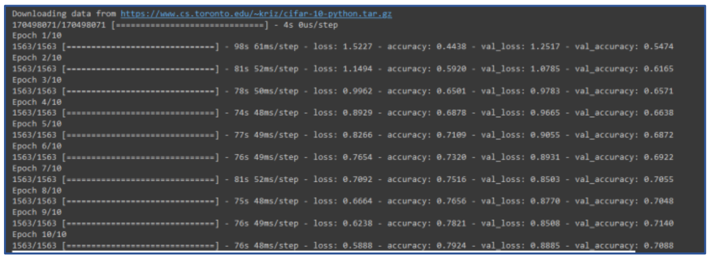


## Phase 2 Results

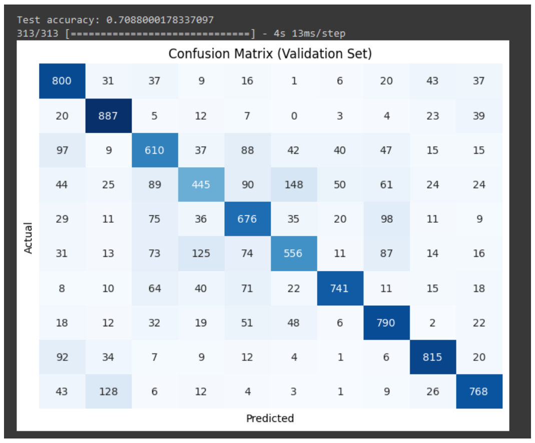


| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1000 | 0 | 0 | 0 | 0 | 0 |





```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
Epoch 1/10
1563/1563 [==============================] - 98s 61ms/step - loss: 1.5227 - accuracy: 0.4438 - val_loss: 1.2517 - val_accuracy: 0.5474
Epoch 2/10
1563/1563 [==============================] - 81s 52ms/step - loss: 1.1494 - accuracy: 0.5920 - val_loss: 1.0785 - val_accuracy: 0.6165
Epoch 3/10
1563/1563 [==============================] - 78s 50ms/step - loss: 0.9962 - accuracy: 0.6501 - val_loss: 0.9783 - val_accuracy: 0.6571
Epoch 4/10
1563/1563 [==============================] - 74s 48ms/step - loss: 0.8929 - accuracy: 0.6878 - val_loss: 0.9665 - val_accuracy: 0.6638
Epoch 5/10
1563/1563 [==============================] - 77s 49ms/step - loss: 0.8266 - accuracy: 0.7109 - val_loss: 0.9055 - val_accuracy: 0.6872
Epoch 6/10
1563/1563 [==============================] - 76s 49ms/step - loss: 0.7654 - accuracy: 0.7320 - val_loss: 0.8931 - val_accuracy: 0.6922
Epoch 7/10
1563/1563 [==============================] - 81s 52ms/step - loss: 0.7092 - accuracy: 0.7516 - val_loss: 0.8503 - val_accuracy: 0.7055
Epoch 8/10
1563/1563 [==============================] - 75s 48ms/step - loss: 0.6664 - accuracy: 0.7656 - val_loss: 0.8770 - val_accuracy: 0.7048
Epoch 9/10
1563/1563 [==============================] - 76s 49ms/step - loss: 0.6238 - accuracy: 0.7821 - val_loss: 0.8508 - val_accuracy: 0.7140
Epoch 10/10
1563/1563 [==============================] - 76s 48ms/step - loss: 0.5888 - accuracy: 0.7924 - val_loss: 0.8885 - val_accuracy: 0.7088
```



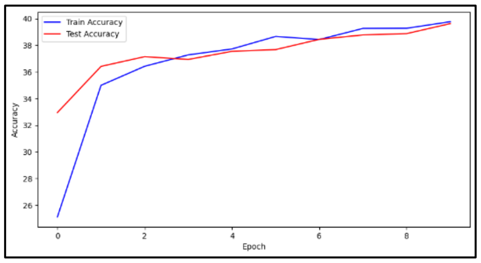
```
Test accuracy: 0.7088000178337097
313/313 [==============================] - 4s 13ms/step
```

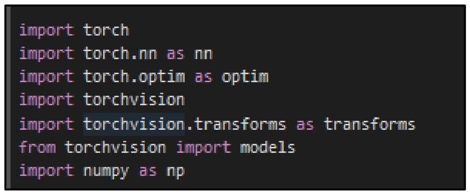
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 800 | 31 | 37 | 9 | 16 | 1 | 6 | 20 | 43 | 37 |
| 20 | 887 | 5 | 12 | 7 | 0 | 3 | 4 | 23 | 39 |
| 97 | 9 | 610 | 37 | 88 | 42 | 40 | 47 | 15 | 15 |
| 44 | 25 | 89 | 445 | 90 | 148 | 50 | 61 | 24 | 24 |
| 29 | 11 | 75 | 36 | 676 | 35 | 20 | 98 | 11 | 9 |
| 31 | 13 | 73 | 125 | 74 | 556 | 11 | 87 | 14 | 16 |
| 8 | 10 | 64 | 40 | 71 | 22 | 741 | 11 | 15 | 18 |
| 18 | 12 | 32 | 19 | 51 | 48 | 6 | 790 | 2 | 22 |
| 92 | 34 | 7 | 9 | 12 | 4 | 1 | 6 | 815 | 20 |
| 43 | 128 | 6 | 12 | 4 | 3 | 1 | 9 | 26 | 768 |

## Phase 3 Results

Using the machine learning modelling framework of PyTorch for the implementation of ShuffleNet, I take the structure of the Tensorflow connected layers and implement it into the structure of the provided PyTorch ShuffleNet framework. By doing this, I have changed the framework I have been using while maintaining the same Pseudocode logic in my machine learning model. By doing so, the accuracy is maintained & improved so as to not cause overfitting & enable better results in my model. Using the ShuffleNet code, we load a pre-trained ResNet-18 model, replaces the final classification layer, and fine-tunes it on the CIFAR-10 dataset. It includes dropout for, and learning rate increases to help achieve an accuracy of 90% or higher.

My initial attempt with two regularizing dropout layers in my fully connected layers yielded an overall loss to my accuracy, as it scaled down to 40%. In my second attempt, I removed the second layer & implemented the ShuffleNET machine framework for most of the system without connecting my convolutional layers. With only one dropout layer, we hope that it will satisfy an accuracy of 90% or higher without overfitting. Overfitting would lead to our validation test have a significant dip from the training accuracy, therefore yielding a 90% accurate test useless due to it not being able to adapt properly





```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models
import numpy as np
```

## Discussion

My initial attempt with two regularizing dropout layers in my fully connected layers yielded an overall loss to my accuracy, as it scaled down to 40%. In my second attempt, I removed the second layer & implemented the ShuffleNET machine framework for most of the system without connecting my convolutional layers. With only one dropout layer, we hope that it will satisfy an accuracy of 90% or higher without overfitting. Overfitting would lead to our validation test have a significant dip from the training accuracy, therefore yielding a 90% accurate test useless due to it not being able to adapt properlyWe can resolve the issues by including more epochs, most issues are hard to nurture to health as it is going through the steps without breaks.

Resources:

- https://www.youtube.com/watch?v=zadvMgHaTfA
- https://www.youtube.com/watch?v=7HPwo4wnJeA
- https://www.youtube.com/watch?v=jztwpsIzEGc

- https://www.youtube.com/watch?v=gHrrCkMcSTo
- https://www.youtube.com/watch?v=pBOfQTfDMVg
- https://www.tensorflow.org/api_docs/python/tf/keras/datasets/cifar10
- https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
- https://arxiv.org/abs/1807.11164#
- https://colab.research.google.com/github/pytorch/pytorch.github.io/blob/master/assets/hub/pytorch_vision_shufflenet_v2.ipynb
- https://colab.research.google.com/drive/1p8lWqa2q0xxMccFGgGExoZpvnAeTFDNb
- https://www.tutorialspoint.com/google_colab/google_colab_using_free_gpu.htm
- https://www.tensorflow.org/api_docs/python/tf/random/shuffle
- https://github.com/Zhengtq/shufflenetv2-tensorflow2.0