Milestone Report (Intermediate)

Springboard Capstone Project 2

Classification of Images

(Deep Learning)

1 Introduction

This project is about classifying images. If an image is fed into a machine, it will try to guess what the image is about. For example if the machine has been trained to classify

among cats, dogs and birds and when an image is fed to the machine it will correctly say what the image was fed to it.

This project can be used for a variety of purposes. It can be used by govts, local authorities and private companies to see if people are ***wearing masks or not***. For a company dealing with superstore or vending machines, the project could help them maintain the ***count of the stock*** of different brands of the same product, say for example a superstore has 3 brands of ice-cream BR, Amul and Vadilal. Each time an ice cream of a brand is picked its stock count would change real time and this will help the client maintain the stock properly.This project could also help in self driving cars in ***classifying the road sign*** and help them take proper action. This project could also help in the ***medical industry*** by helping to classify between *healthy and unhealthy cells*. The potential of image classification is immense.

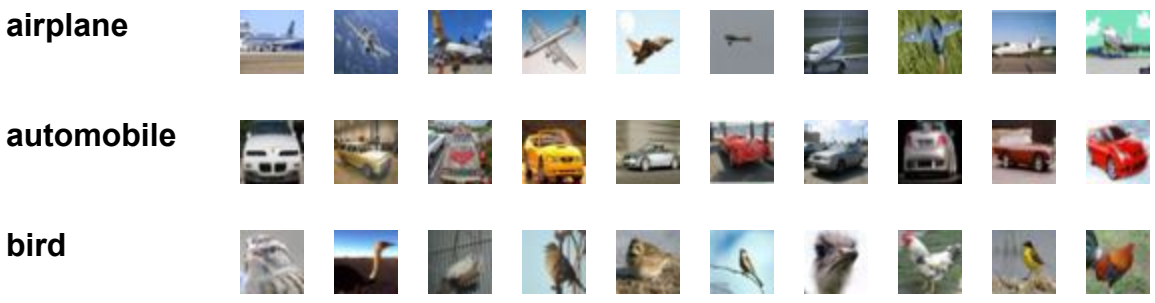
2 Data Acquisition and cleaning

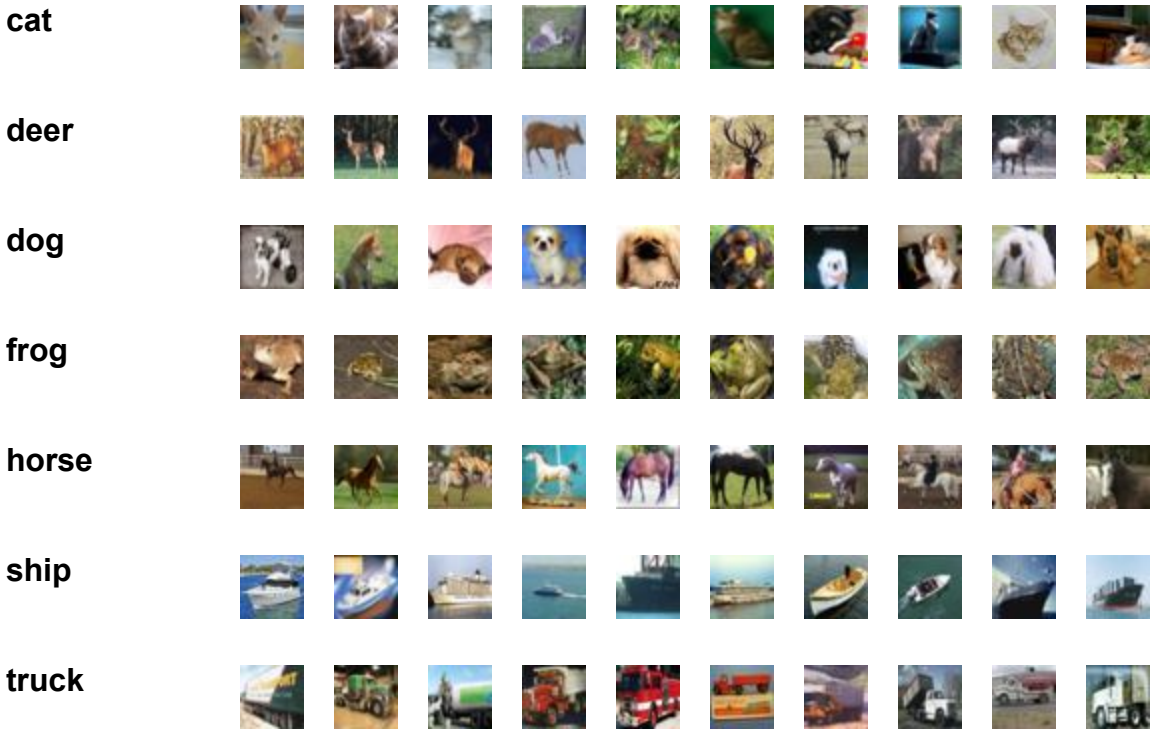
The data set was collected from here. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton (credit).

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



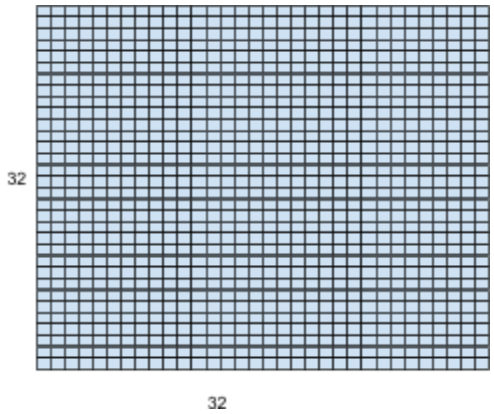


The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

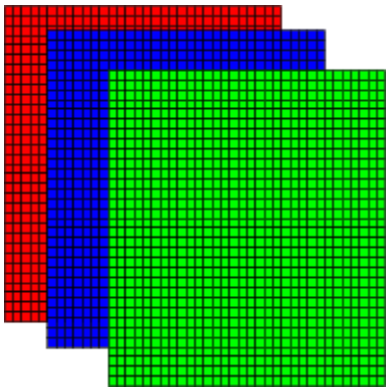
The dataset is clean and not much cleaning is required.

3 Data exploration

Each images are 32 * 32 pixels, what it means is that the height and width of the images are of 32 pixels each.



What does each pixel hold?
Each pixel has numbers which depend on the amount of red, green and blue colors (as we know that red blue and green are the universal colors and all the other colors are produced from combination of these three).



So for each pixel we will have three values.

4 Deep learning Models

After cleaning the data and analysing the images, we can proceed with model building. Here we use convolution neural networks (CNN) and in CNN we will use three different deep learning models, we will use keras along with tensorflow. We will use ***basic CNN***, ***keras tuner***, **transfer learning** and ***image generators***.

4.1 Basic CNN

In basic CNN we will use two convolutional layers and then a dense layer. The accuracy of this model is 73.73%

4.2 Keras Tuner

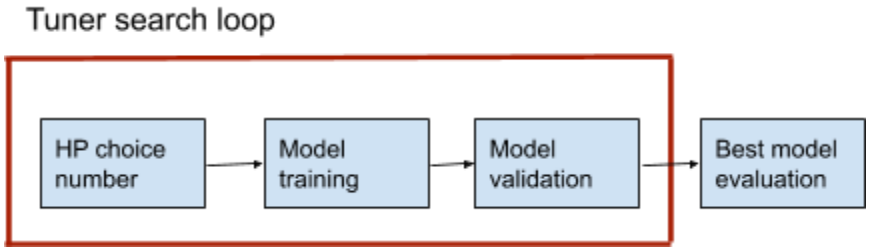
The **Keras Tuner** is a library that helps us pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter *tuning* or hypertuning. It can be comparable to random search.

Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

1. **Model hyperparameters** which influence model selection such as the number and width of hidden layers
2. **Algorithm hyperparameters** which influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent

(SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier

4.2.1 Hyperparameter tuning with Keras Tuner



First, a tuner is defined. Its role is to determine which hyperparameter combinations should be tested. The library search function performs the iteration loop, which evaluates a certain number of hyperparameter combinations. Evaluation is performed by computing the trained model's accuracy on a held-out validation set. Finally, the best hyperparameter combination in terms of validation accuracy can be tested on a held-out test set.

4.2.2 Search Space definition

To perform hyperparameter tuning, we need to define the search space, that is to say which hyperparameters need to be optimized and in what range. Here, there are already 2 hyperparameters that we can tune.

1 The number of filters for the dense layer.
2 The size of the kernel.

```
filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
activation='relu'
```

4.2.3 Architectures comparison

We can tune only one model, but we went for two models to tune to see if the complexity of the models is decreased. Can it give a better result with decreased training time?

A Model with 6 dense layer

```
def build_model(hp):
  model = keras.Sequential([
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(32,32,3)
    ),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
        activation='relu'
    ),
    keras.layers.Dense(10, activation='softmax')
```

B Model with 5 dense layer (less complex)

```
def build_model(hp):
  model = keras.Sequential([
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(32,32,3)
    ),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
        activation='relu'
    ),
    keras.layers.Dense(10, activation='softmax')
```

4.2.4 Summary

Model 1

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 80)	6080
conv2d_1 (Conv2D)	(None, 24, 24, 48)	96048
dropout (Dropout)	(None, 24, 24, 48)	0
batch_normalization (BatchNo	(None, 24, 24, 48)	192
conv2d_2 (Conv2D)	(None, 20, 20, 48)	57648
dropout_1 (Dropout)	(None, 20, 20, 48)	0
batch_normalization_1 (Batch	(None, 20, 20, 48)	192
conv2d_3 (Conv2D)	(None, 16, 16, 48)	57648
dropout_2 (Dropout)	(None, 16, 16, 48)	0
batch_normalization_2 (Batch	(None, 16, 16, 48)	192
conv2d_4 (Conv2D)	(None, 12, 12, 48)	57648
dropout_3 (Dropout)	(None, 12, 12, 48)	0
batch_normalization_3 (Batch	(None, 12, 12, 48)	192
flatten (Flatten)	(None, 6912)	0
dense (Dense)	(None, 96)	663648
dense_1 (Dense)	(None, 10)	970

Total params: 940,458
Trainable params: 940,074
Non-trainable params: 384

Model 2

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 112)	3136
conv2d_1 (Conv2D)	(None, 26, 26, 64)	179264
dropout (Dropout)	(None, 26, 26, 64)	0
batch_normalization (BatchNo	(None, 26, 26, 64)	256
conv2d_2 (Conv2D)	(None, 22, 22, 64)	102464
dropout_1 (Dropout)	(None, 22, 22, 64)	0
batch_normalization_1 (Batch	(None, 22, 22, 64)	256
conv2d_3 (Conv2D)	(None, 18, 18, 64)	102464
dropout_2 (Dropout)	(None, 18, 18, 64)	0
batch_normalization_2 (Batch	(None, 18, 18, 64)	256
flatten (Flatten)	(None, 20736)	0
dense (Dense)	(None, 96)	1990752
dense_1 (Dense)	(None, 10)	970

Total params: 2,379,818
Trainable params: 2,379,434
Non-trainable params: 384

4.2.5 Accuracy

Model 1 64%
Model 2 76.44%

Here we see that a less complex model gave us a better result with decreased training time.