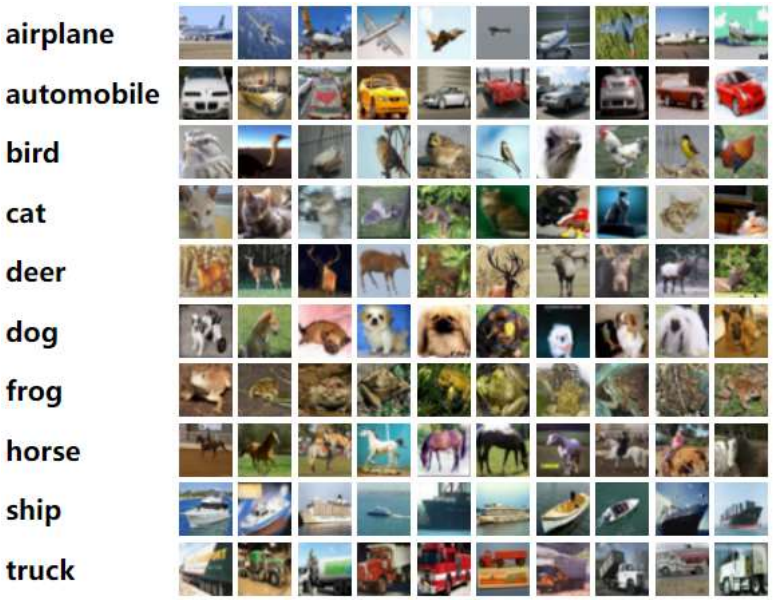
Image Classification Using Convolutional Neural Network (CNN)

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
```

Load the dataset




```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
(50000, 32, 32, 3)
```

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

Here we see there are 50000 training images and 1000 test images

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

```
y_train = y_train.reshape(-1,)
y_train[:5]
```

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

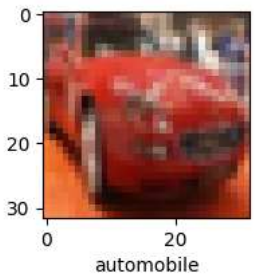
```
y_test = y_test.reshape(-1,)

classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

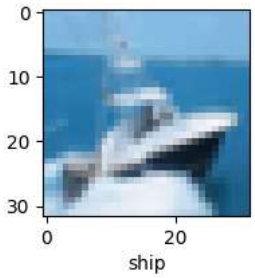
Let's plot some images to see what they are

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])

plot_sample(X_train, y_train, 5)
```




```
plot_sample(X_train, y_train, 8)
```



Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0-->1 range, we need to divide it by 255

Normalizing the training data

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 144s 91ms/step - loss: 1.8096 - accuracy: 0.3547
Epoch 2/5
1563/1563 [==============================] - 133s 85ms/step - loss: 1.6205 - accuracy: 0.4276
Epoch 3/5
```

```
1563/1563 [==============================] - 134s 86ms/step - loss: 1.5399 - accuracy: 0.4575
Epoch 4/5
1563/1563 [==============================] - 133s 85ms/step - loss: 1.4795 - accuracy: 0.4775
Epoch 5/5
1563/1563 [==============================] - 139s 89ms/step - loss: 1.4324 - accuracy: 0.4968
<keras.src.callbacks.History at 0x7a83510728f0>
```

You can see that at the end of 5 epochs, accuracy is at around 49%

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
313/313 [==============================] - 9s 28ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.65      0.51      1000
           1       0.67      0.54      0.60      1000
           2       0.51      0.13      0.21      1000
           3       0.32      0.41      0.36      1000
           4       0.41      0.46      0.43      1000
           5       0.44      0.32      0.37      1000
           6       0.46      0.65      0.53      1000
           7       0.68      0.43      0.53      1000
           8       0.53      0.70      0.60      1000
           9       0.56      0.51      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

Now let us build a convolutional neural network to train our images:

```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])


cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 57s 36ms/step - loss: 1.4497 - accuracy: 0.4809
Epoch 2/10
1563/1563 [==============================] - 57s 36ms/step - loss: 1.1009 - accuracy: 0.6142
Epoch 3/10
1563/1563 [==============================] - 56s 36ms/step - loss: 0.9720 - accuracy: 0.6602
Epoch 4/10
1563/1563 [==============================] - 55s 35ms/step - loss: 0.8884 - accuracy: 0.6908
Epoch 5/10
1563/1563 [==============================] - 56s 36ms/step - loss: 0.8230 - accuracy: 0.7147
Epoch 6/10
1563/1563 [==============================] - 55s 35ms/step - loss: 0.7634 - accuracy: 0.7360
Epoch 7/10
1563/1563 [==============================] - 57s 37ms/step - loss: 0.7138 - accuracy: 0.7531
Epoch 8/10
1563/1563 [==============================] - 57s 36ms/step - loss: 0.6729 - accuracy: 0.7652
Epoch 9/10
1563/1563 [==============================] - 55s 35ms/step - loss: 0.6342 - accuracy: 0.7792
Epoch 10/10
1563/1563 [==============================] - 54s 35ms/step - loss: 0.5965 - accuracy: 0.7938
```

```
<keras.src.callbacks.History at 0x7a82ec24c160>
```

With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

```
#step-5:cnn evaluation
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 7s 23ms/step - loss: 0.8957 - accuracy: 0.7064
[0.895720899105072, 0.7063999772071838]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 6s 19ms/step
array([[8.3574578e-06, 4.2657825e-04, 4.5966351e-04, 9.7751141e-01,
        1.3795739e-05, 1.9700771e-02, 4.1644034e-04, 4.7671610e-06,
        1.3527590e-03, 1.0543604e-04],
       [3.6554534e-06, 7.3502590e-05, 1.0865112e-08, 1.2153944e-08,
        2.4109295e-09, 1.3947075e-10, 5.2210705e-12, 1.3439251e-10,
        9.9992234e-01, 3.6613864e-07],
       [8.5550398e-02, 1.6132625e-02, 7.7813998e-04, 8.7313587e-03,
        7.5531877e-03, 4.2225738e-04, 3.9342878e-05, 2.0287510e-03,
        8.2760918e-01, 5.1154688e-02],
       [8.7724519e-01, 1.5800520e-04, 2.1510506e-02, 3.8298389e-03,
        1.6662130e-02, 3.4353216e-05, 3.1303789e-04, 1.3116730e-04,
        8.0113508e-02, 2.2736760e-06],
       [8.1200204e-07, 3.5348829e-05, 1.4139208e-01, 2.4058463e-01,
        2.3091134e-01, 2.4194880e-03, 3.8368815e-01, 7.7744681e-07,
        9.5800031e-04, 9.2872360e-06]], dtype=float32)
```

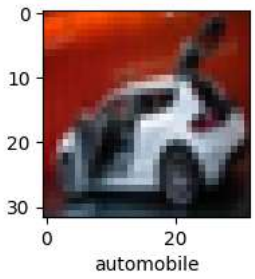
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 8, 8, 0, 6]
```

```
y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,6)
```




```
classes[y_classes[60]]
```

```
'horse'
```

```
from flask import Flask, render_template, request, jsonify, send_file
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import decode_predictions
import numpy as np
import os

app = Flask(__name__)

# Load the pre-trained VGG16 model
model = VGG16(weights='imagenet', include_top=True)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    if request.method == 'POST':
        file = request.files['file']
        img_path = os.path.join('static/uploads/cifar10_cnn.h5')
        file.save(img_path)

        # Preprocess the image for the VGG16 model
        img = load_img(img_path, target_size=(224, 224))
        img_array = img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = preprocess_input(img_array)

        # Make prediction with the VGG16 model
        predictions = model.predict(img_array)
        label = decode_predictions(predictions, top=1)[0][0][1]

        # Return the predicted label
        return jsonify({"label": label})

if __name__ == '__main__':
    app.run(debug=True)
```

```
 Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
 553467096/553467096 [==============================] - 7s 0us/step
  * Serving Flask app '__main__'
  * Debug mode: on
 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
 INFO:werkzeug:Press CTRL+C to quit
 INFO:werkzeug: * Restarting with stat
```

```
import tensorflow as tf
import numpy as np

class Cifar10Classifier:
    def __init__(self):
        self.model = tf.keras.models.load_model('static/models/cifar10_cnn.h5')

    def predict(self, img_path):
        img = tf.keras.preprocessing.image.load_img(
            img_path, target_size=(32, 32)
        )
        img_array = tf.keras.preprocessing.image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255

        predictions = self.model.predict(img_array)
        top3_indices = np.argsort(predictions[0])[-3:]
        top3_classes = [
            ('Class {}: {:.2f}%'.format(i, predictions[0][i] * 100))
            for i in top3_indices
        ]

        return top3_classes


from IPython.display import HTML
HTML(filename='//content//index.html')
```

Image Classifier

Choose File No file chosen
Upload and Classify