Install and Load Dependencies

In [ ]:
```
! pip install carbontracker
```

In [ ]:
```
import carbontracker
from carbontracker.tracker import CarbonTracker
```

Define Carbontracker function for Machine Learning

In [ ]:
```
def track_ML_carbon(model, X, y, max_epochs, validation_data="optional"):
    # Track the carbon footprint of a machine learning model

    ct = CarbonTracker(epochs=max_epochs)

    # Training loop.
    for epoch in range(max_epochs):
        # start epoch tracker
        ct.epoch_start()

        # Your model training.
        if validation_data != "optional":
          model.fit(X, y, validation_data)
        else:
          model.fit(X, y)

        # end epoch tracker
        ct.epoch_end()

    # Optional: Add a stop in case of early termination before all monito
    # been monitored to ensure that actual consumption is reported.
    ct.stop()
    return ct
```

Test a simple Perceptron model (linear)

In [ ]:
```
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
clf = Perceptron(tol=1e-3, random_state=0)
```

In [ ]:
```
# train model for insights
clf.fit(X_train, y_train)
```

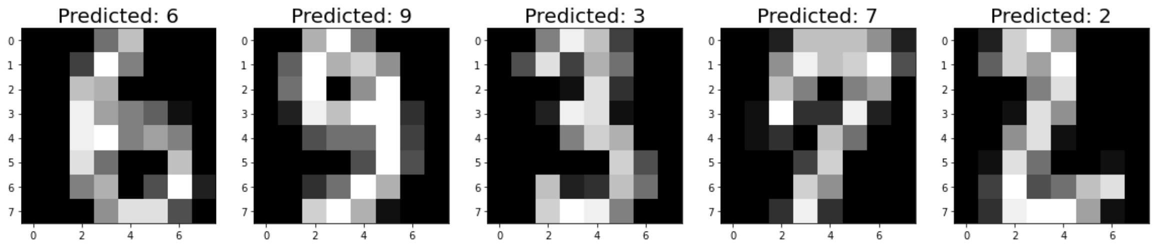
Out[ ]:
```
Perceptron()
```

In [ ]:
```python
# try predicting test data
pred = clf.predict(X_test)

# show accuracy
from sklearn.metrics import accuracy_score
test_score = accuracy_score(pred, y_test)
print("score on test data: ", test_score, "\n")

# visualize the predictions
# compare the predicted values with the actual values
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(20, 4))
for index, (image, prediction) in enumerate(zip(X_test[0:5], pred[0:5])):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Predicted: %i' % prediction, fontsize = 20)
```

```
score on test data:  0.9579124579124579
```



In [ ]:
```python
# track carbon for perceptron
Perceptron_tracker = track_ML_carbon(clf, X, y, 100)
```

```
CarbonTracker: The following components were found: GPU with device(s) Te
sla T4.
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:00:00
        Energy: 0.000000 kWh
        CO2eq:  0.000060 g
        This is equivalent to:
        0.000001 km travelled by car
CarbonTracker:
Predicted consumption for 100 epoch(s):
        Time:   0:00:05
        Energy: 0.000021 kWh
        CO2eq:  0.006033 g
        This is equivalent to:
        0.000050 km travelled by car
CarbonTracker: Finished monitoring.
```

Now we take a look at a more advanced Convolutional Neural Network model

Import modules, load data and show insights

In [ ]:
```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

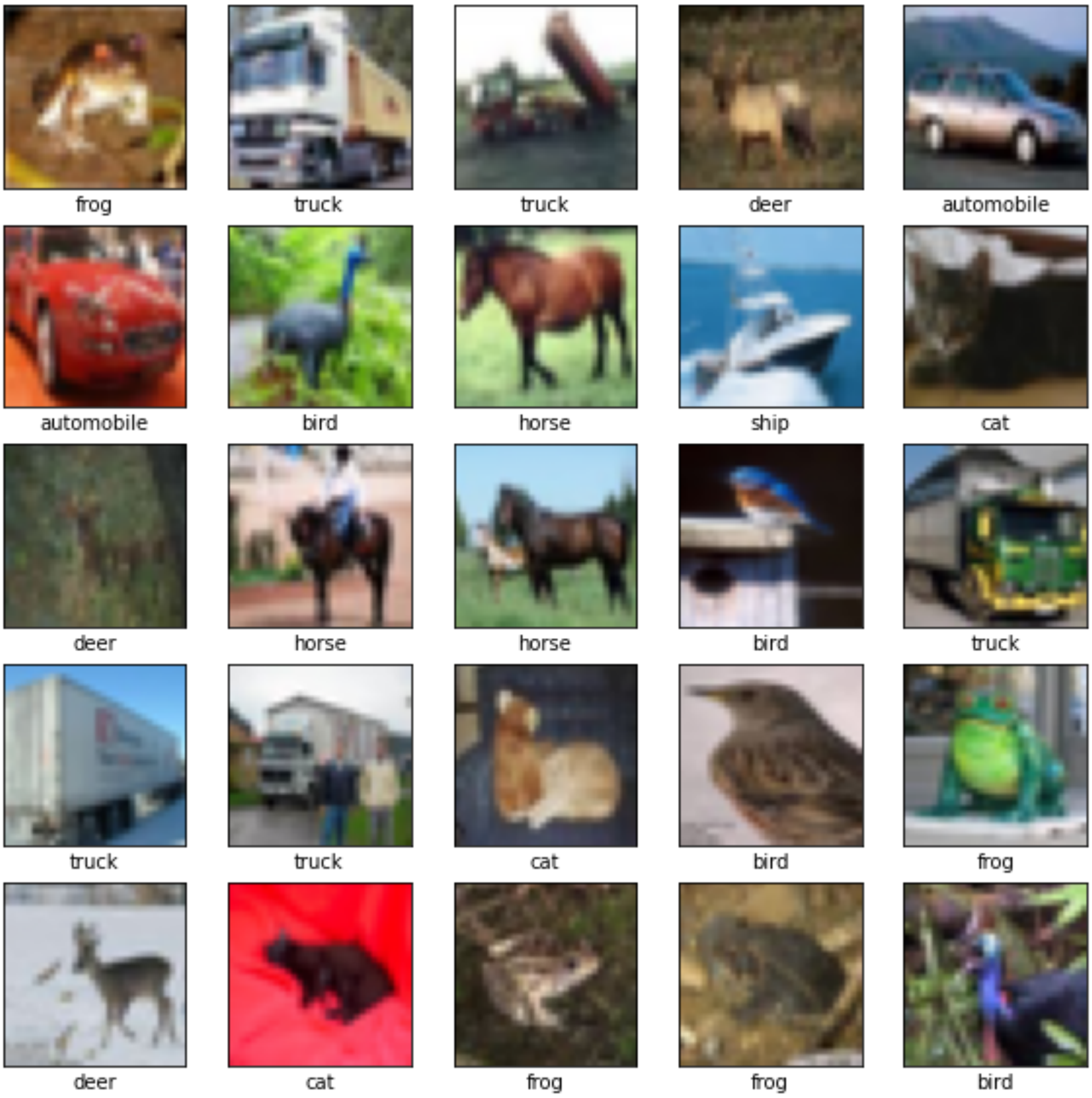
In [ ]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.ta
r.gz
170498071/170498071 [==============================] - 14s 0us/step
```

In [ ]:
```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Create convolutional base

In [ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 3
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

In [ ]:
```
# add dense layers to complete model
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

In [ ]:
```
# summary of complete model architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 64)                65600     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [ ]:

```
# run for 10 epochs
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_log
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

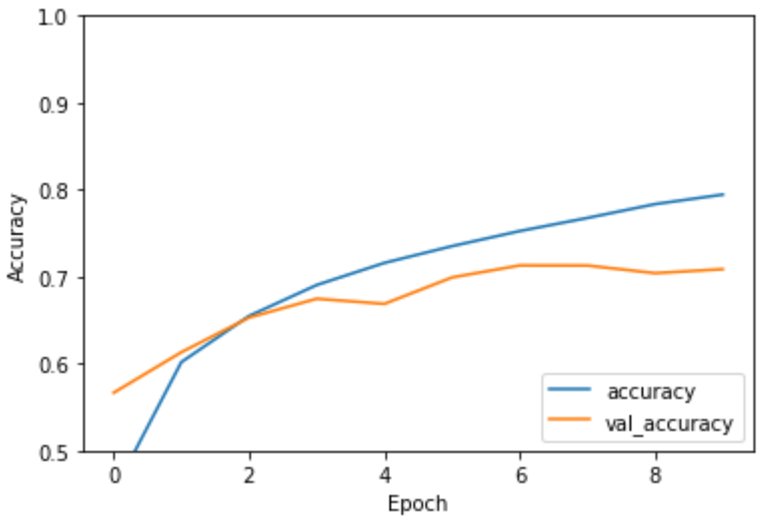
```
Epoch 1/10
1563/1563 [==============================] - 17s 5ms/step - loss: 1.4990
- accuracy: 0.4525 - val_loss: 1.2094 - val_accuracy: 0.5665
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1277 -
accuracy: 0.6017 - val_loss: 1.1083 - val_accuracy: 0.6131
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9793 -
accuracy: 0.6548 - val_loss: 1.0096 - val_accuracy: 0.6529
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8791 -
accuracy: 0.6904 - val_loss: 0.9329 - val_accuracy: 0.6746
Epoch 5/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8086 -
accuracy: 0.7158 - val_loss: 0.9691 - val_accuracy: 0.6687
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7537 -
accuracy: 0.7351 - val_loss: 0.8744 - val_accuracy: 0.6991
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7046 -
accuracy: 0.7524 - val_loss: 0.8506 - val_accuracy: 0.7129
Epoch 8/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.6615 -
accuracy: 0.7675 - val_loss: 0.8701 - val_accuracy: 0.7128
Epoch 9/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6164 -
accuracy: 0.7834 - val_loss: 0.8965 - val_accuracy: 0.7040
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5799 -
accuracy: 0.7943 - val_loss: 0.8893 - val_accuracy: 0.7086
```

Evaluate the model

In [ ]:
```
# visualize results
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8893 - accuracy: 0.7086 - 969ms/epoch - 3ms/step
```



In [ ]:
```
# test accuracy
print(f"The test accuracy of our CNN model on the CIFAR-10 dataset was",
```

```
The test accuracy of our CNN model on the CIFAR-10 dataset was 0.70859998
46458435
```

Finally, track the carbon of our CNN training

In [ ]:
```
# define max_epochs
max_epochs = 1000 # boosted to 1000 epochs

# create tracker instance
cnn_tracker = CarbonTracker(epochs=max_epochs, verbose=2)

# Start tracking
cnn_tracker.epoch_start()

# Implement model training
model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
          metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=1,
                validation_data=(test_images, test_labels))
# End tracking
cnn_tracker.epoch_end()

# Optional: Add a stop in case of early termination before all monitor_ep
# been monitored to ensure that actual consumption is reported.
cnn_tracker.stop()
```

```
CarbonTracker: The following components were found: GPU with device(s) Te
sla T4.
1563/1563 [==============================] - 16s 5ms/step - loss: 1.5184
- accuracy: 0.4475 - val_loss: 1.2976 - val_accuracy: 0.5385
CarbonTracker: Average carbon intensity during training was 294.21 gCO2/k
Wh at detected location: Singapore, Singapore, SG.
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:00:17
        Energy: 0.000243 kWh
        CO2eq:  0.071432 g
        This is equivalent to:
        0.000593 km travelled by car
CarbonTracker: Live carbon intensity could not be fetched at detected loc
ation: Singapore, Singapore, SG. Defaulted to average carbon intensity fo
r EU-28 in 2017 of 294.21 gCO2/kWh.
CarbonTracker:
Predicted consumption for 1000 epoch(s):
        Time:   4:49:08
        Energy: 0.242797 kWh
        CO2eq:  71.432357 g
        This is equivalent to:
        0.593292 km travelled by car
CarbonTracker: Finished monitoring.
```

Garbage collector

In [ ]:
```
import gc
gc.collect()
```

Out[ ]:
```
47854
```

Playground for you to experiment :)

In [ ]:
```
from carbontracker.tracker import CarbonTracker

# define max_epochs
max_epochs = 10 # your choice

# create tracker instance
tracker = CarbonTracker(epochs=max_epochs)

tracker.epoch_start()

####### YOUR CODE STARTS HERE ############################################
# Implement your model training


####### YOUR CODE ENDS HERE ##############################################

tracker.epoch_end()

# Optional: Add a stop in case of early termination before all monitor_ep
# been monitored to ensure that actual consumption is reported.
tracker.stop()
```