Image Classification Using CNN with the CIFAR-10 Dataset using **basic Hyperparameter Tunning**

```
pip install keras_tuner
```

```
Collecting keras_tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (3.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (2.32.3)
Collecting kt-legacy (from keras_tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (3.12.1)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (0.13.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras->keras_tuner) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tun
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras->kera
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->keras_tun
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->keras_t
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->kera
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 4.4 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras_tuner
Successfully installed keras_tuner-1.4.7 kt-legacy-1.0.5
```

```
# Step 1: Importing necessary libraries

import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras_tuner.tuners import RandomSearch
import matplotlib.pyplot as plt
```

RandomSearch explores the hyperparameter space by randomly sampling combinations of hyperparameters. This can be effective for finding a good set of hyperparameters without the need to exhaustively evaluate every possible combination.

```
# Step 2: Loading and preprocessing the CIFAR-10 dataset

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
# Step 3: Defining the class names for CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
```

Optemizer is responsible for updating the values of models weight, during training, in order to minimize the loss function.

```
# Step 4: Define a function to build the model

def build_model(hp):
  model = models.Sequential()
  model.add(layers.Conv2D(hp.Int('conv1_units', min_value=32, max_value=128, step=16), (3,3), activa
  model.add(layers.MaxPooling2D((2,2)))
  model.add(layers.Conv2D(hp.Int('conv2_units', min_value=32, max_value=128, step=16), (3,3), activa
  model.add(layers.MaxPooling2D((2,2)))
  model.add(layers.Conv2D(hp.Int('conv3_units', min_value=32, max_value=128, step=16), (3,3), activa
  model.add(layers.Flatten())
  model.add(layers.Dense(hp.Int('dense_units', min_value=32, max_value=128, step=16), activation='re
  model.add(layers.Dropout(hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)))
  model.add(layers.Dense(10))

  # Choose an optimizer and learning rate
  optimizer = tf.keras.optimizers.Adam(learning_rate=hp.Choice('learning_rate', values=[1e-2, 1e-3,
  model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
  return model
```

Optemizer is responsible for updating the values of models weight, during training, in order to minimize the loss function.

```
# Step 5: Define the Tuner
tuner = RandomSearch(
  build_model,
  objective='val_accuracy',
  max_trials=10,
  executions_per_trial=1,
  directory='my_dir',
  project_name='cifar10_tunning'
)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Step 6: Perform the Hyperparameter search
tuner.search(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))
```

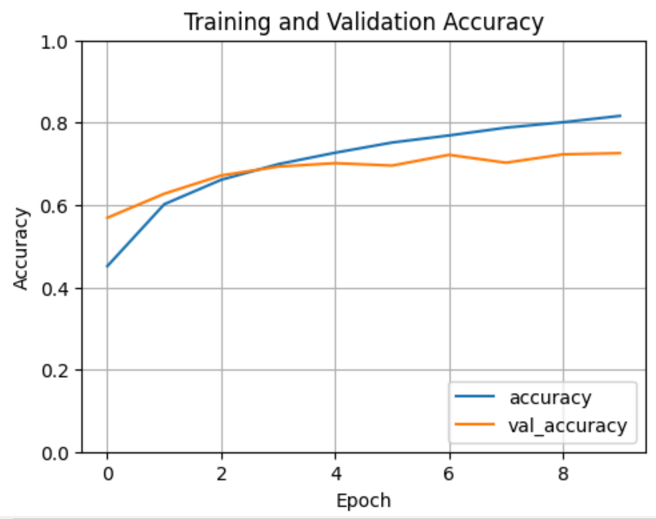
```
Trial 10 Complete [00h 00m 56s]
val_accuracy: 0.5982999801635742

Best val_accuracy So Far: 0.694100022315979
Total elapsed time: 00h 07m 42s
```

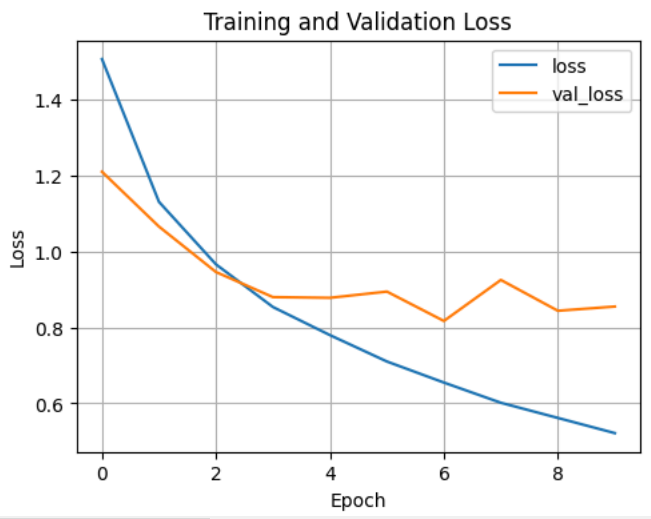
```
# Step 7: Get the best Hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```
# Step 8: Build the model with the best Hyperparameters and train it
model = tuner.hypermodel.build(best_hps)
trained_model = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3526 - loss: 1.7445 - val_accuracy: 0.5688 - val_loss: 1.2101
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.5833 - loss: 1.1769 - val_accuracy: 0.6273 - val_loss: 1.0657
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.6523 - loss: 0.9933 - val_accuracy: 0.6716 - val_loss: 0.9460
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.7007 - loss: 0.8543 - val_accuracy: 0.6933 - val_loss: 0.8799
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.7262 - loss: 0.7790 - val_accuracy: 0.7015 - val_loss: 0.8780
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7557 - loss: 0.6980 - val_accuracy: 0.6957 - val_loss: 0.8944
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7713 - loss: 0.6452 - val_accuracy: 0.7218 - val_loss: 0.8170
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7917 - loss: 0.5956 - val_accuracy: 0.7027 - val_loss: 0.9251
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8057 - loss: 0.5475 - val_accuracy: 0.7230 - val_loss: 0.8439
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8239 - loss: 0.4998 - val_accuracy: 0.7259 - val_loss: 0.8550
```

```
# Step 9: Plotting training & validation accuracy and loss values

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(trained_model.history['accuracy'], label='accuracy')
plt.plot(trained_model.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(trained_model.history['loss'], label='loss')
plt.plot(trained_model.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.grid()

plt.show()
```


Training and Validation Accuracy
Accuracy
Epoch
accuracy
val_accuracy


Training and Validation Loss
Loss
Epoch
loss
val_loss