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

```
pip install keras_tuner
```

```
Requirement already satisfied: keras_tuner in /usr/local/lib/python3.10/dist-packages (1.4.7)
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)
Requirement already satisfied: kt-legacy in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (1.0.5)
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
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras->
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->keras
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->ker
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->
```

```
# 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

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

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()

    # Tune the number of convolutional layers (1, 2 or 3)
    for i in range(hp.Int('conv_layers', 1, 3)):
        if i == 0:
            model.add(layers.Conv2D(
                filters=hp.Int('filters_' + str(i), min_value=32, max_value=128, step=16),
                kernel_size=3,
                activation='relu',
                input_shape=(32, 32, 3)))
        else:
            model.add(layers.Conv2D(
```

```
                filters=hp.Int('filters_' + str(i), min_value=64, max_value=128, step=16),
                kernel_size=3,
                activation='relu',
                padding='same'))
        model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Flatten())

    # Tune the number of dense layers (1, 2, or 3)
    for i in range(hp.Int('dense_layers', 1, 3)):
        model.add(layers.Dense(
            units=hp.Int(f'units_{i}', min_value=32, max_value=128, step=16),
            activation='relu'))

        # Tune the dropout rate
        model.add(layers.Dropout(rate=hp.Float(f'dropout_{i}', 0.0, 0.5, step=0.1))) #dropout re

    # The last dense layer with 10 output units (for 10 classes)
    model.add(layers.Dense(10, activation='softmax'))

    # Choose an optimizer and learning rate
    optimizer = tf.keras.optimizers.Adam(learning_rate=hp.Choice('learning_rate', values=[1e-2,

    model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l

    return model

# 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'
)
```

```
Reloading Tuner from my_dir/cifar10_tunning/tuner0.json
```

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

# 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, t
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `in
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/nn.py:635: UserWarning: "`sparse_categorical_crosse
  output, from_logits = _get_logits(
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 14s 6ms/step - accuracy: 0.2346 - loss: 2.0180 - val_accuracy: 0.4496 - val_loss: 1.5034
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.4392 - loss: 1.5220 - val_accuracy: 0.5248 - val_loss: 1.3214
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.5026 - loss: 1.3749 - val_accuracy: 0.5300 - val_loss: 1.2878
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.5352 - loss: 1.2777 - val_accuracy: 0.5812 - val_loss: 1.1857
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.5685 - loss: 1.2039 - val_accuracy: 0.5868 - val_loss: 1.1466
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.5879 - loss: 1.1555 - val_accuracy: 0.6087 - val_loss: 1.1113
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6121 - loss: 1.0908 - val_accuracy: 0.6265 - val_loss: 1.0568
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.6321 - loss: 1.0430 - val_accuracy: 0.6319 - val_loss: 1.0372
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6457 - loss: 1.0097 - val_accuracy: 0.6404 - val_loss: 1.0213
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6610 - loss: 0.9682 - val_accuracy: 0.6472 - val_loss: 1.0056
```

```
# 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()
```