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

```
pip install keras_tuner
```

```
# 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

# 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', 'truck']

# 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, 3),
            activation = 'relu',
            input_shape = (32, 32, 3)
        ))
      else:
        model.add(layers.Conv2D(
          filters = hp.Int('filters_' + str(i), min_value=32, max_value=128, step=16),
          kernel_size = (3, 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('units_' + str(i), min_value=32, max_value=128, step=16),
          activation = 'relu'))

      # Tune the dropout rate
      model.add(layers.Dropout(rate=hp.Float('dropout_' + str(i), min_value=0.0, max_value=0.5, step=0.1)))

    # The last dense layer with 10 output units (for 10 classes of CIFAR-10 dataset)
    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, 1e-4]))

    model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

    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'
)

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

```
# Step 9: Plotting training & validation accuracy and loss values
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
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, 1])
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.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()
```

```
Trial 10 Complete [00h 01m 14s]
val_accuracy: 0.6549000144004822

Best val_accuracy So Far: 0.6758999824523926
Total elapsed time: 00h 10m 42s
Epoch 1/10
1563/1563 [==============================] - 15s 8ms/step - loss: 1.8427 - accuracy:
Epoch 2/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3892 - accuracy:
Epoch 3/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.2268 - accuracy:
Epoch 4/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.1221 - accuracy:
Epoch 5/10
1563/1563 [==============================] - 13s 8ms/step - loss: 1.0406 - accuracy:
Epoch 6/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.9794 - accuracy:
Epoch 7/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.9192 - accuracy:
Epoch 8/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.8817 - accuracy:
Epoch 9/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.8445 - accuracy:
Epoch 10/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.8044 - accuracy:
```