# PSO_HPO_1_Code

May 1, 2023

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pyswarms as ps
from functools import partial
from sklearn.metrics import classification_report, confusion_matrix,␣
  ↪roc_auc_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
  ↪Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import time

# Load and preprocess the CIFAR-10 dataset
print("Loading and preprocessing CIFAR-10 dataset")
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.
  ↪to_categorical(y_test)

# Split the training data into train and validation sets
print("Splitting the training data into train and validation sets")
validation_split = 0.1
split_index = int(len(x_train) * validation_split)
x_val, y_val = x_train[:split_index], y_train[:split_index]
x_train, y_train = x_train[split_index:], y_train[split_index:]

sample_size = 5000  # Adjust this value as needed
sample_indices = np.random.choice(np.arange(x_train.shape[0]), sample_size,␣
  ↪replace=False)

x_train_small = x_train[sample_indices]
y_train_small = y_train[sample_indices]

# Define a fitness function to be optimized using PSO
def fitness_function(hparams, x_train, y_train, x_val, y_val):
    fitness_values = []
    for hparam in hparams:
        num_filters1, num_filters2, dense_units, learning_rate = hparam
        num_filters1 = int(num_filters1)
        num_filters2 = int(num_filters2)
        dense_units = int(dense_units)

        print(f"Hyperparameters: num_filters1={num_filters1}, "
              f"num_filters2={num_filters2}, dense_units={dense_units}, "
              f"learning_rate={learning_rate}")

        model = Sequential([
            Conv2D(num_filters1, (3, 3), activation='relu', padding='same',␣
↪input_shape=(32, 32, 3)),
            BatchNormalization(),
            Conv2D(num_filters1, (3, 3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2, 2)),
            Dropout(0.25),

            Conv2D(num_filters2, (3, 3), activation='relu', padding='same'),
            BatchNormalization(),
            Conv2D(num_filters2, (3, 3), activation='relu', padding='same'),
            BatchNormalization(),
            MaxPooling2D((2, 2)),
            Dropout(0.25),

            Flatten(),
            Dense(dense_units, activation='relu'),
            BatchNormalization(),
            Dropout(0.5),
            Dense(10, activation='softmax')
        ])

        model.compile(optimizer=Adam(learning_rate=learning_rate),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

        early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)

        history = model.fit(x_train, y_train, epochs=5, batch_size=256,
                            validation_data=(x_val, y_val),
                            callbacks=[early_stopping],
                            verbose=0)

        best_val_acc = max(history.history['val_accuracy'])
        fitness_values.append(1 - best_val_acc)  # Minimize the fitness␣
↪function (1 - val_accuracy)

    return np.array(fitness_values)

# Define the PSO search space for hyperparameters
print("Defining the PSO search space for hyperparameters")
search_space_bounds = (np.array([16, 16, 128, 1e-5]),
                       np.array([128, 128, 1024, 1e-2]))

# Define the fitness function with fixed data arguments
print("Defining the fitness function with fixed data arguments")
fitness_function_data = partial(fitness_function,
                                x_train=x_train_small, y_train=y_train_small,
                                x_val=x_val, y_val=y_val)

# Initialize the PSO optimizer
print("Initializing the PSO optimizer")
options = {'c1': 1.5, 'c2': 1.5, 'w': 0.9}
optimizer = ps.single.GlobalBestPSO(n_particles=20, dimensions=4,␣
↪options=options,
                                    bounds=search_space_bounds)

# Run the PSO optimizer
print("Running the PSO optimizer")
cost, best_hyperparams = optimizer.optimize(fitness_function_data, iters=20)

# Extract the best hyperparameters
best_num_filters1, best_num_filters2, best_dense_units, best_learning_rate =␣
↪best_hyperparams
best_num_filters1 = int(best_num_filters1)
best_num_filters2 = int(best_num_filters2)
best_dense_units = int(best_dense_units)

print(f"Best hyperparameters found by PSO: num_filters1={best_num_filters1}, "
      f"num_filters2={best_num_filters2}, dense_units={best_dense_units}, "
      f"learning_rate={best_learning_rate}")

# Train the model with the best hyperparameters
print("Training the model with the best hyperparameters")
model = Sequential([
    Conv2D(best_num_filters1, (3, 3), activation='relu', padding='same',␣
↪input_shape=(32, 32, 3)),
    BatchNormalization(),
    Conv2D(best_num_filters1, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(best_num_filters2, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(best_num_filters2, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(best_dense_units, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=best_learning_rate),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)
start_time = time.time()

history = model.fit(x_train, y_train, epochs=50, batch_size=64,
                    validation_data=(x_val, y_val),
                    callbacks=[early_stopping],
                    verbose=1)
end_time = time.time()

training_time = end_time - start_time
print(f'Total training time: {training_time:.2f} seconds')

# Evaluate the model on the test dataset
print("Evaluating the model on the test dataset")
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

print(f'Test accuracy: {test_acc}')

print("Classification Report:")
print(classification_report(y_test_classes, y_pred_classes))

print("Confusion Matrix:")
print(confusion_matrix(y_test_classes, y_pred_classes))
```

```
# Calculate ROC-AUC for multi-class classification
roc_auc = roc_auc_score(y_test, y_pred, multi_class='ovr')
print(f'ROC-AUC Score: {roc_auc}')

# Plot the training and validation accuracies
print("Plotting the training and validation accuracies")
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
[ ]: # Using these hardcoded values as obtained from confusion matrix above inorder␣
      ↪to make a plot
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     # Confusion matrix data
     confusion_matrix = np.array([
         [833, 8, 23, 20, 17, 4, 5, 6, 60, 24],
         [8, 905, 4, 4, 1, 3, 4, 0, 24, 47],
         [68, 0, 632, 94, 63, 70, 48, 10, 12, 3],
         [9, 5, 25, 694, 33, 158, 46, 13, 12, 5],
         [9, 1, 37, 66, 810, 31, 31, 9, 6, 0],
         [8, 0, 12, 144, 25, 778, 13, 16, 4, 0],
         [4, 3, 15, 38, 27, 27, 877, 2, 7, 0],
         [13, 1, 10, 53, 58, 54, 3, 799, 2, 7],
         [32, 11, 2, 9, 4, 4, 4, 3, 920, 11],
         [10, 50, 4, 9, 5, 3, 0, 9, 26, 884]
     ])

     # Class names
     class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
      ↪'horse', 'ship', 'truck']

     # Plot confusion matrix
     plt.figure(figsize=(10, 8))
     sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues',␣
      ↪xticklabels=class_names, yticklabels=class_names)
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.title('Confusion Matrix')
     plt.show()
```

```
[ ]:
```