image-object-recognition-version3

May 24, 2024

[1]:
```
import os
import pickle
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,␣
 ↪MaxPooling2D
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

# Function to load a batch of CIFAR-10 data
def load_cifar10_batch(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    images = dict[b'data']
    labels = dict[b'labels']
    images = images.reshape(len(images), 3, 32, 32).transpose(0, 2, 3, 1).
 ↪astype("float")
    return images, labels

# Function to load all CIFAR-10 data
def load_cifar10_data(data_dir):
    train_images = []
    train_labels = []
    for i in range(1, 6):
        file = os.path.join(data_dir, f'data_batch_{i}')
        images, labels = load_cifar10_batch(file)
        train_images.append(images)
        train_labels.append(labels)
    train_images = np.concatenate(train_images)
    train_labels = np.concatenate(train_labels)

    test_file = os.path.join(data_dir, 'test_batch')
    test_images, test_labels = load_cifar10_batch(test_file)

    return (train_images, train_labels), (test_images, test_labels)
```

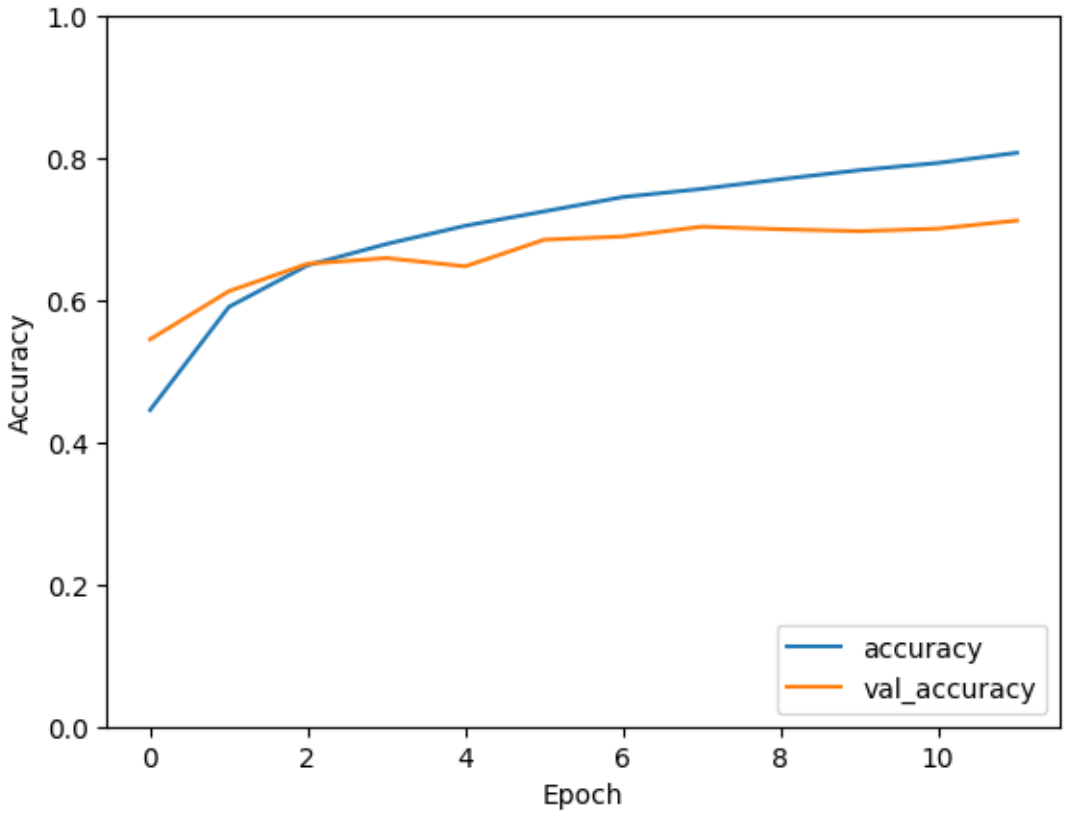
```
# Load CIFAR-10 data
data_dir = '/Users/williambolton/Downloads/cifar-10-batches-py'
(train_images, train_labels), (test_images, test_labels) =␣
 ↪load_cifar10_data(data_dir)

# Normalize the data
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convert labels to numpy arrays
train_labels = np.array(train_labels)
test_labels = np.array(test_labels)

# Display first 5 images from the training dataset
plt.figure(figsize=(10, 2))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(train_images[i])
    plt.title(f"Label: {train_labels[i]}")
    plt.axis('off')
plt.show()

# Ensures representative splitting
def representative_split(X, y, test_size=0.2, random_state=42):
    np.random.seed(random_state)
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    split = int(X.shape[0] * (1 - test_size))
    train_idx, test_idx = indices[:split], indices[split:]
    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]

# Partition the training data to create a validation set using representative␣
 ↪split
train_images, validation_images, train_labels, validation_labels =␣
 ↪representative_split(train_images, train_labels, test_size=0.1)

# Data specifics (metadata)
print(f"Training samples: {train_images.shape[0]}")
print(f"Validation samples: {validation_images.shape[0]}")
print(f"Test samples: {test_images.shape[0]}")

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=12,
                    validation_data=(validation_images, validation_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")

# Plot the training and validation accuracy
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.show()
```

```
2024-05-24 23:17:32.329093: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```


```
Training samples: 45000
Validation samples: 5000
Test samples: 10000
```

```
/Users/williambolton/myenv/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/12
1407/1407            44s 30ms/step -
accuracy: 0.3490 - loss: 1.7554 - val_accuracy: 0.5450 - val_loss: 1.2612
Epoch 2/12
1407/1407            39s 28ms/step -
accuracy: 0.5772 - loss: 1.1948 - val_accuracy: 0.6128 - val_loss: 1.1080
Epoch 3/12
1407/1407            36s 26ms/step -
accuracy: 0.6468 - loss: 1.0192 - val_accuracy: 0.6514 - val_loss: 1.0145
Epoch 4/12
1407/1407            38s 27ms/step -
accuracy: 0.6756 - loss: 0.9271 - val_accuracy: 0.6594 - val_loss: 0.9879
Epoch 5/12
1407/1407            36s 26ms/step -
accuracy: 0.7052 - loss: 0.8460 - val_accuracy: 0.6478 - val_loss: 1.0177
Epoch 6/12
1407/1407            36s 26ms/step -
accuracy: 0.7226 - loss: 0.7934 - val_accuracy: 0.6852 - val_loss: 0.9021
Epoch 7/12
1407/1407            35s 25ms/step -
accuracy: 0.7506 - loss: 0.7182 - val_accuracy: 0.6898 - val_loss: 0.8915
Epoch 8/12
1407/1407            38s 27ms/step -
accuracy: 0.7611 - loss: 0.6746 - val_accuracy: 0.7036 - val_loss: 0.8517
Epoch 9/12
1407/1407            37s 26ms/step -
accuracy: 0.7742 - loss: 0.6455 - val_accuracy: 0.7000 - val_loss: 0.8766
Epoch 10/12
1407/1407            41s 29ms/step -
accuracy: 0.7873 - loss: 0.6069 - val_accuracy: 0.6972 - val_loss: 0.9209
Epoch 11/12
1407/1407            41s 29ms/step -
accuracy: 0.8000 - loss: 0.5586 - val_accuracy: 0.7008 - val_loss: 0.8808
Epoch 12/12
1407/1407            40s 28ms/step -
accuracy: 0.8137 - loss: 0.5241 - val_accuracy: 0.7122 - val_loss: 0.9018
313/313 - 2s - 8ms/step - accuracy: 0.7081 - loss: 0.9220
Test accuracy: 0.7081000208854675
```


1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0
2
4
6
8
10
Epoch
accuracy
val_accuracy

[ ]: