image-object-recognition-version2

May 24, 2024

[1]:
```
import os
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,␣
 ↪MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# 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()

# Function to split dataset into training and validation sets
def representative_split(X, y, test_size=0.2, random_state=42):
    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y)
    return X_train, X_val, y_train, y_val

# Split the training data into training and validation sets
X_train, X_val, y_train, y_val = representative_split(train_images,␣
 ↪train_labels)

# Print the shape of the datasets
print(f"Training set shape: {X_train.shape}")
print(f"Validation set shape: {X_val.shape}")
print(f"Test set shape: {test_images.shape}")

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

# Data augmentation
datagen = ImageDataGenerator(
```

```
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
datagen.fit(X_train)

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2,␣
 ↪min_lr=0.001)

# Train the model
history = model.fit(datagen.flow(X_train, y_train, batch_size=32),
                    epochs=20,
                    validation_data=(X_val, y_val),
                    callbacks=[early_stopping, reduce_lr])

# 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 22:52:46.524143: 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 set shape: (40000, 32, 32, 3)
Validation set shape: (10000, 32, 32, 3)
Test set shape: (10000, 32, 32, 3)
Training samples: 40000
Validation samples: 10000
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)
```

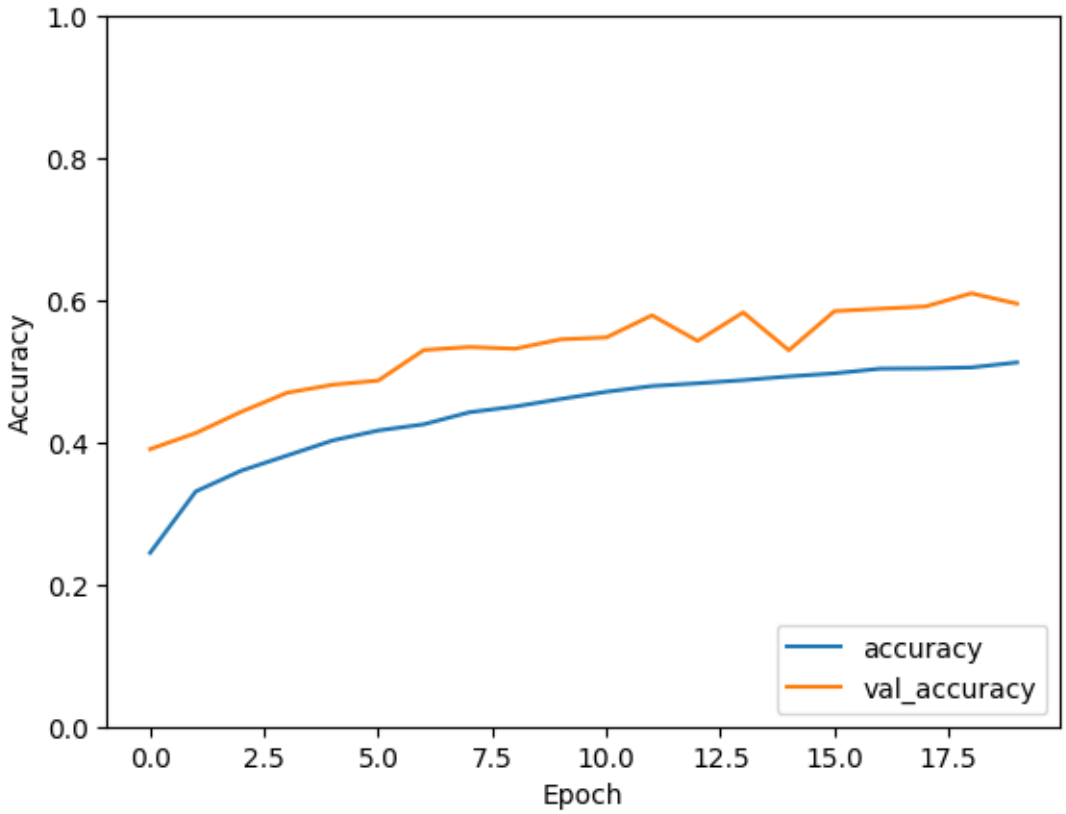
```
Epoch 1/20
```

```
/Users/williambolton/myenv/lib/python3.12/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
1250/1250            46s 35ms/step -
accuracy: 0.1885 - loss: 2.2856 - val_accuracy: 0.3905 - val_loss: 1.6899 -
learning_rate: 0.0010
Epoch 2/20
1250/1250            47s 38ms/step -
accuracy: 0.3211 - loss: 1.8469 - val_accuracy: 0.4131 - val_loss: 1.5915 -
learning_rate: 0.0010
```

```
Epoch 3/20
1250/1250            47s 37ms/step -
accuracy: 0.3523 - loss: 1.7537 - val_accuracy: 0.4431 - val_loss: 1.5439 -
learning_rate: 0.0010
Epoch 4/20
1250/1250            46s 36ms/step -
accuracy: 0.3775 - loss: 1.7080 - val_accuracy: 0.4698 - val_loss: 1.4979 -
learning_rate: 0.0010
Epoch 5/20
1250/1250            42s 33ms/step -
accuracy: 0.3997 - loss: 1.6694 - val_accuracy: 0.4811 - val_loss: 1.4421 -
learning_rate: 0.0010
Epoch 6/20
1250/1250            43s 34ms/step -
accuracy: 0.4154 - loss: 1.6563 - val_accuracy: 0.4870 - val_loss: 1.4541 -
learning_rate: 0.0010
Epoch 7/20
1250/1250            45s 36ms/step -
accuracy: 0.4197 - loss: 1.6289 - val_accuracy: 0.5300 - val_loss: 1.3208 -
learning_rate: 0.0010
Epoch 8/20
1250/1250            44s 35ms/step -
accuracy: 0.4397 - loss: 1.6067 - val_accuracy: 0.5342 - val_loss: 1.3639 -
learning_rate: 0.0010
Epoch 9/20
1250/1250            45s 36ms/step -
accuracy: 0.4544 - loss: 1.5734 - val_accuracy: 0.5318 - val_loss: 1.3603 -
learning_rate: 0.0010
Epoch 10/20
1250/1250            43s 34ms/step -
accuracy: 0.4592 - loss: 1.5619 - val_accuracy: 0.5450 - val_loss: 1.3085 -
learning_rate: 0.0010
Epoch 11/20
1250/1250            42s 33ms/step -
accuracy: 0.4676 - loss: 1.5488 - val_accuracy: 0.5477 - val_loss: 1.2920 -
learning_rate: 0.0010
Epoch 12/20
1250/1250            44s 35ms/step -
accuracy: 0.4790 - loss: 1.5320 - val_accuracy: 0.5788 - val_loss: 1.2401 -
learning_rate: 0.0010
Epoch 13/20
1250/1250            44s 35ms/step -
accuracy: 0.4839 - loss: 1.5218 - val_accuracy: 0.5428 - val_loss: 1.2909 -
learning_rate: 0.0010
Epoch 14/20
1250/1250            42s 33ms/step -
accuracy: 0.4854 - loss: 1.5083 - val_accuracy: 0.5829 - val_loss: 1.2134 -
learning_rate: 0.0010
```

```
Epoch 15/20
1250/1250            42s 34ms/step -
accuracy: 0.4883 - loss: 1.4954 - val_accuracy: 0.5296 - val_loss: 1.3484 -
learning_rate: 0.0010
Epoch 16/20
1250/1250            42s 34ms/step -
accuracy: 0.5011 - loss: 1.4804 - val_accuracy: 0.5847 - val_loss: 1.2095 -
learning_rate: 0.0010
Epoch 17/20
1250/1250            42s 34ms/step -
accuracy: 0.5030 - loss: 1.4790 - val_accuracy: 0.5881 - val_loss: 1.2303 -
learning_rate: 0.0010
Epoch 18/20
1250/1250            43s 35ms/step -
accuracy: 0.5047 - loss: 1.4675 - val_accuracy: 0.5913 - val_loss: 1.2100 -
learning_rate: 0.0010
Epoch 19/20
1250/1250            42s 34ms/step -
accuracy: 0.5048 - loss: 1.4610 - val_accuracy: 0.6098 - val_loss: 1.1697 -
learning_rate: 0.0010
Epoch 20/20
1250/1250            43s 35ms/step -
accuracy: 0.5094 - loss: 1.4647 - val_accuracy: 0.5952 - val_loss: 1.2017 -
learning_rate: 0.0010
313/313 - 3s - 10ms/step - accuracy: 0.5921 - loss: 1.2076
Test accuracy: 0.5921000242233276
```


1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
accuracy
val_accuracy

[ ]: