prototype

February 24, 2025

[3]:
```
import os
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

data gen

[4]:
```
train_dir = '../data/train'
val_dir   = '../data/value'

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Just rescaling for validation
val_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 8

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'  # We have 4 classes
)
```

```
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 112 images belonging to 4 classes.
Found 22 images belonging to 4 classes.
Found 22 images belonging to 4 classes.
```

load mobilenetv2

[5]:
```
from tensorflow.keras.applications import MobileNetV2

base_model = MobileNetV2(weights='imagenet',
                         include_top=False,
                         input_shape=(224, 224, 3))

# Freeze the base model so we don't train over ImageNet weights initially
base_model.trainable = False

# Build a simple classifier on top
model = Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(4, activation='softmax')  # <-- 4 classes instead of 3
])

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

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d	(None, 1280)	0

```
 (GlobalAveragePooling2D)

 dense (Dense)                    (None, 128)                   163,968

 dropout (Dropout)                (None, 128)                         0

 dense_1 (Dense)                  (None, 4)                         516


 Total params: 2,422,468 (9.24 MB)

 Trainable params: 164,484 (642.52 KB)

 Non-trainable params: 2,257,984 (8.61 MB)
```

train

[6]:
```
epochs = 10

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,  # 140 // 8 = 17␣
 ↪steps
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size    # 20 // 8 = 2 steps
)
```

```
C:\Users\demet\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\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()

Epoch 1/10
14/14             0s 492ms/step -
accuracy: 0.3124 - loss: 1.6553

C:\Users\demet\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\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()
```

```
14/14             13s 707ms/step -
accuracy: 0.3243 - loss: 1.6290 - val_accuracy: 0.8125 - val_loss: 0.4362
Epoch 2/10
Epoch 2/10
14/14             8s 565ms/step -
accuracy: 0.8259 - loss: 0.4276 - val_accuracy: 1.0000 - val_loss: 0.0816
Epoch 3/10
14/14             8s 559ms/step -
accuracy: 0.9548 - loss: 0.1619 - val_accuracy: 1.0000 - val_loss: 0.0243
Epoch 4/10
14/14             8s 547ms/step -
accuracy: 0.9727 - loss: 0.1065 - val_accuracy: 1.0000 - val_loss: 0.0171
Epoch 5/10
14/14             8s 552ms/step -
accuracy: 0.9530 - loss: 0.1300 - val_accuracy: 1.0000 - val_loss: 0.0166
Epoch 6/10
14/14             8s 547ms/step -
accuracy: 0.9705 - loss: 0.0667 - val_accuracy: 1.0000 - val_loss: 0.0076
Epoch 7/10
14/14             8s 547ms/step -
accuracy: 0.9950 - loss: 0.0382 - val_accuracy: 1.0000 - val_loss: 0.0069
Epoch 8/10
14/14             8s 561ms/step -
accuracy: 0.9913 - loss: 0.0304 - val_accuracy: 1.0000 - val_loss: 0.0048
Epoch 9/10
14/14             8s 548ms/step -
accuracy: 1.0000 - loss: 0.0191 - val_accuracy: 1.0000 - val_loss: 0.0025
Epoch 10/10
14/14             8s 557ms/step -
accuracy: 0.9982 - loss: 0.0393 - val_accuracy: 1.0000 - val_loss: 0.0038
```

fine tuning

[7]:
```
base_model.trainable = True

fine_tune_at = 100  # example
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(
    optimizer=keras.optimizers.Adam(1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
fine_tune_epochs = 5
history_fine = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=fine_tune_epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size
)
```

```
Epoch 1/5
14/14             17s 701ms/step -
accuracy: 0.8058 - loss: 0.6399 - val_accuracy: 1.0000 - val_loss: 0.0034
Epoch 2/5
14/14             8s 565ms/step -
accuracy: 0.8758 - loss: 0.3676 - val_accuracy: 1.0000 - val_loss: 0.0035
Epoch 3/5
14/14             8s 561ms/step -
accuracy: 0.8680 - loss: 0.3071 - val_accuracy: 1.0000 - val_loss: 0.0031
Epoch 4/5
14/14             8s 571ms/step -
accuracy: 0.8944 - loss: 0.3117 - val_accuracy: 1.0000 - val_loss: 0.0027
Epoch 5/5
14/14             8s 567ms/step -
accuracy: 0.9378 - loss: 0.2758 - val_accuracy: 1.0000 - val_loss: 0.0030
```

evaluation

[8]:
```
val_loss, val_acc = model.evaluate(val_generator, steps=val_generator.samples //
  ↪ batch_size)
print(f"Validation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_acc:.4f}")
```

```
2/2             1s 459ms/step -
accuracy: 1.0000 - loss: 0.0021
Validation Loss: 0.0029
Validation Accuracy: 1.0000
```

predictions

[13]:
```
class_indices = train_generator.class_indices
index_to_class = {v: k for k, v in class_indices.items()}

import cv2

def predict_image(model, img_path):
    # Read the image (OpenCV reads BGR)
    img = cv2.imread(img_path)
    # Convert to RGB
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    # Resize to 224x224
    img_resized = cv2.resize(img, (224, 224))
    # Scale pixel values (same as training scale)
    img_array = np.expand_dims(img_resized / 255.0, axis=0)

    # Predict
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    predicted_label = index_to_class[predicted_class_index]

    return predicted_label

# Example usage:
test_path = '../data/truths/product-b.JPG'
prediction = predict_image(model, test_path)
print("Predicted:", prediction)
```

```
1/1             1s 756ms/step
1/1             1s 756ms/step
Predicted: product-b
```

save

[14]:
```
model.save('my_product_classifier.h5')  # saves architecture + weights
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```