CIFAR10

April 11, 2023

[ ]:
```python
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from keras import callbacks
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
2023-04-10 19:39:56.439052: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
Num GPUs Available:  1
```

[ ]:
```python

```

[ ]:
```python
(ds_train, ds_test), ds_info = tfds.load(
    'cifar10',
    split=['train[:20%]', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

[ ]:
```python
#IMG_SIZE = 224

#ds_train = ds_train.map(lambda x, y: (tf.image.resize(x, (IMG_SIZE,␣
↪IMG_SIZE)), y))
#ds_test = ds_test.map(lambda x, y: (tf.image.resize(x, (IMG_SIZE, IMG_SIZE)),␣
↪y))
```

[ ]:
```python
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label
```

```
ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
#ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(32)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
```

[ ]:
```
ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(32)
#ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
```

[ ]:
```
'''
base_model = keras.applications.Xception(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False)  # Do not include the ImageNet classifier at the top.
'''
base_model = keras.applications.ResNet50(
    weights="imagenet",
    input_shape=(128,128,3),
    include_top=False
)
```

[ ]:
```
'''
testing_model = keras.applications.ResNet50(
    weights="imagenet",
    input_shape=(32,32,3),
    include_top=False
)
testing_model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')],
)

testing_model.evaluate(ds_test)
'''
```

[ ]:
```
'\ntesting_model = keras.applications.ResNet50(\n    weights="imagenet",\n
input_shape=(32,32,3),\n    include_top=False\n)\ntesting_model.compile(\n
optimizer=tf.keras.optimizers.Adam(0.001),\n
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),\n    metri
cs=[tf.keras.metrics.SparseCategoricalAccuracy(name=\'accuracy\')],\n)\n\ntestin
g_model.evaluate(ds_test)\n'
```

[ ]:
```
'''
#original
base_model.trainable = False


inputs = keras.Input(shape=(32, 32, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
#x = keras.layers.Dense(128,activation='relu''relu')
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(10)(x)
model = keras.Model(inputs, outputs)
'''
```

[ ]:
```
"\n#original \nbase_model.trainable = False\n\n\ninputs = keras.Input(shape=(32,
32, 3))\n# We make sure that the base_model is running in inference mode
here,\n# by passing `training=False`. This is important for fine-tuning, as you
will\n# learn in a few paragraphs.\nx = base_model(inputs, training=False)\n#
Convert features of shape `base_model.output_shape[1:]` to vectors\nx =
keras.layers.GlobalAveragePooling2D()(x)\n#x =
keras.layers.Dense(128,activation='relu''relu')\n# A Dense classifier with a
single unit (binary classification)\noutputs = keras.layers.Dense(10)(x)\nmodel
= keras.Model(inputs, outputs)\n"
```

[ ]:
```
#sequential
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.UpSampling2D((2,2)))
model.add(tf.keras.layers.UpSampling2D((2,2)))
#model.add(tf.keras.layers.UpSampling2D((2,2)))
model.add(base_model)
model.add(tf.keras.layers.GlobalAveragePooling2D())

model.add(tf.keras.layers.Dense(128, activation='relu'))
#model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10,activation='softmax'))
```

[ ]:
```
earlystopping = callbacks.EarlyStopping(monitor='val_loss',
                                        mode='min',
```

```
                                        patience=10,
                                        restore_best_weights=True)
```

[ ]:
```
#Traditional Training
#model = tf.keras.models.Sequential([
#   tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
#   tf.keras.layers.Dense(128, activation='relu'),
#   tf.keras.layers.Dense(10)
#])

model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')],
)

model.fit(
    ds_train,
    epochs=5,
    #batch_size=512,
    validation_data=ds_test,
    callbacks =[earlystopping],
)
```

```
Epoch 1/5
313/313 [==============================] - 79s 180ms/step - loss: 1.3189 -
accuracy: 0.5631 - val_loss: 2.8050 - val_accuracy: 0.1000
Epoch 2/5
313/313 [==============================] - 46s 139ms/step - loss: 0.8158 -
accuracy: 0.7266 - val_loss: 3.0494 - val_accuracy: 0.1144
Epoch 3/5
313/313 [==============================] - 56s 170ms/step - loss: 0.6207 -
accuracy: 0.7893 - val_loss: 1.4199 - val_accuracy: 0.5133
Epoch 4/5
313/313 [==============================] - 71s 209ms/step - loss: 0.4749 -
accuracy: 0.8387 - val_loss: 0.9142 - val_accuracy: 0.7308
Epoch 5/5
313/313 [==============================] - 52s 157ms/step - loss: 0.3439 -
accuracy: 0.8843 - val_loss: 1.1176 - val_accuracy: 0.6863
```

[ ]:
```
<keras.callbacks.History at 0x7eff0c190d00>
```

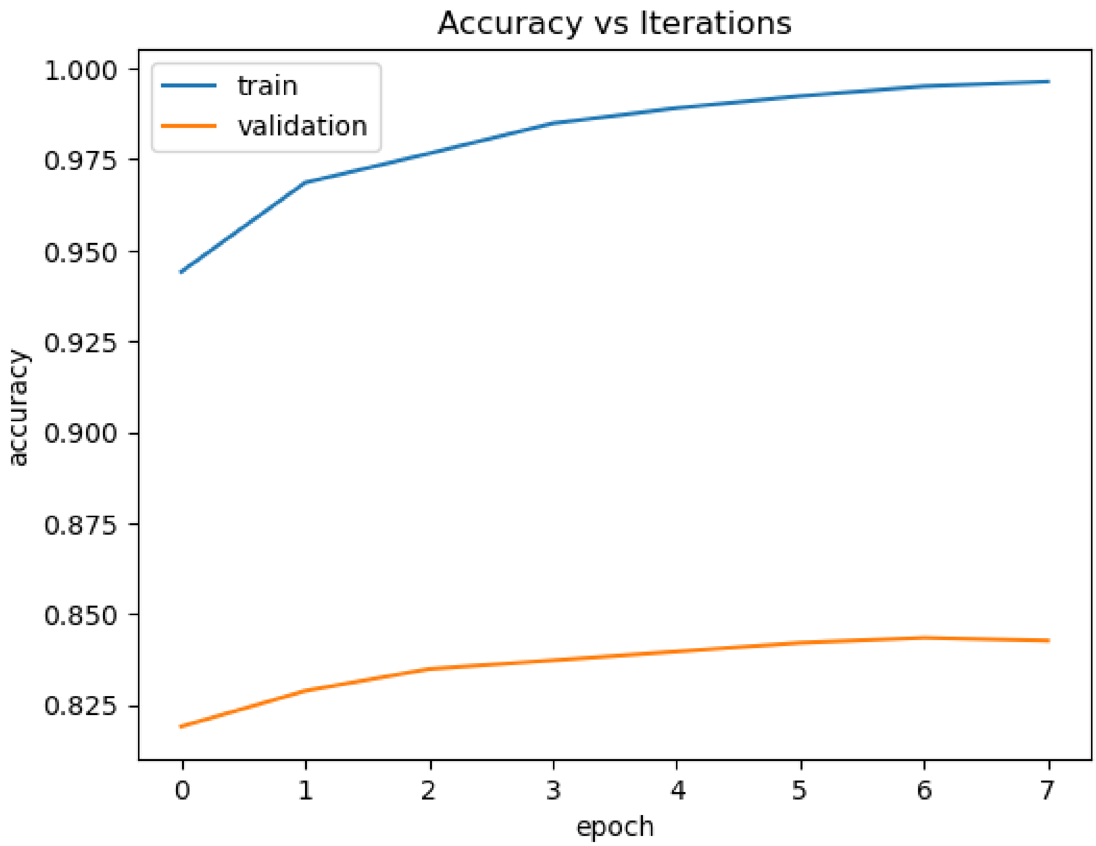
[ ]:
```
# Unfreeze the base model
base_model.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
```

```python
# are take into account
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')],
)
testing = model.fit(
    ds_train,
    epochs=11,
    #batch_size=512,
    validation_data=ds_test,
    verbose=1,
    callbacks =[earlystopping],
)
```

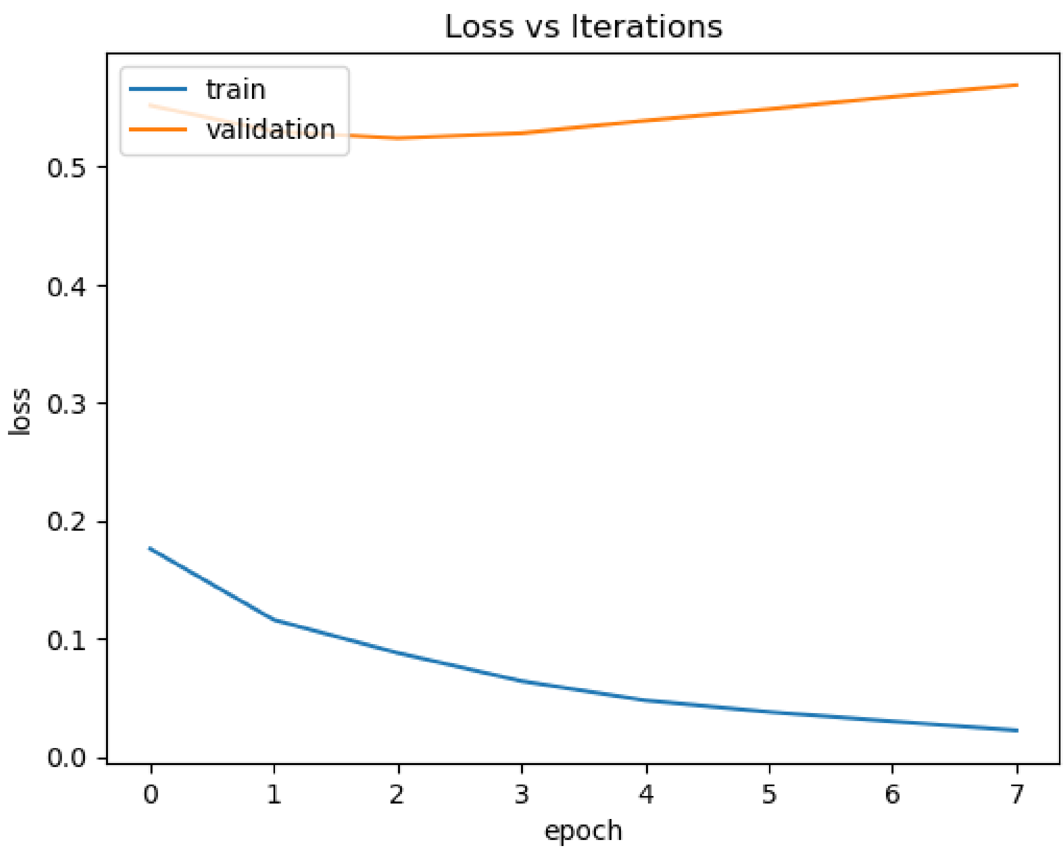
```
Epoch 1/11
313/313 [==============================] - 68s 186ms/step - loss: 0.0621 -
accuracy: 0.9863 - val_loss: 0.5250 - val_accuracy: 0.8407
Epoch 2/11
313/313 [==============================] - 55s 175ms/step - loss: 0.0451 -
accuracy: 0.9909 - val_loss: 0.5310 - val_accuracy: 0.8419
Epoch 3/11
313/313 [==============================] - 60s 182ms/step - loss: 0.0373 -
accuracy: 0.9923 - val_loss: 0.5446 - val_accuracy: 0.8425
Epoch 4/11
313/313 [==============================] - 96s 296ms/step - loss: 0.0302 -
accuracy: 0.9939 - val_loss: 0.5533 - val_accuracy: 0.8437
Epoch 5/11
313/313 [==============================] - 54s 164ms/step - loss: 0.0207 -
accuracy: 0.9968 - val_loss: 0.5613 - val_accuracy: 0.8441
Epoch 6/11
313/313 [==============================] - 68s 209ms/step - loss: 0.0175 -
accuracy: 0.9979 - val_loss: 0.5728 - val_accuracy: 0.8442
Epoch 7/11
313/313 [==============================] - 56s 170ms/step - loss: 0.0164 -
accuracy: 0.9970 - val_loss: 0.5814 - val_accuracy: 0.8447
Epoch 8/11
313/313 [==============================] - 53s 170ms/step - loss: 0.0121 -
accuracy: 0.9986 - val_loss: 0.5913 - val_accuracy: 0.8448
Epoch 9/11
313/313 [==============================] - 67s 205ms/step - loss: 0.0128 -
accuracy: 0.9980 - val_loss: 0.5978 - val_accuracy: 0.8447
Epoch 10/11
313/313 [==============================] - 51s 154ms/step - loss: 0.0087 -
accuracy: 0.9993 - val_loss: 0.6019 - val_accuracy: 0.8449
Epoch 11/11
313/313 [==============================] - 58s 185ms/step - loss: 0.0078 -
```

```
accuracy: 0.9993 - val_loss: 0.6131 - val_accuracy: 0.8448
```

[ ]:
```python
plt.plot(testing.history['accuracy'])
plt.plot(testing.history['val_accuracy'])
plt.title('Accuracy vs Iterations')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



[ ]:
```python
plt.plot(testing.history['loss'])
plt.plot(testing.history['val_loss'])
plt.title('Loss vs Iterations')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



[ ]:
```
model.evaluate(ds_test)
```

```
313/313 [==============================] - 8s 25ms/step - loss: 0.5243 -
accuracy: 0.8348
```

[ ]:
```
[0.5242804288864136, 0.8348000049591064]
```