fc_tune_vgg16_2class

January 21, 2020

[1]:

```
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import applications
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
```

[24]:

```
# dimensions of our images.
img_width, img_height = 224, 224

top_model_weights_path = '/home/user/models/top_tuned/
 ↪bottleneck_fc_model_2class.h5'
train_data_dir = '/home/user/     /convnets/transfer-learning-keras/dataset/
 ↪training'
validation_data_dir = '/home/user/     /convnets/transfer-learning-keras/
 ↪dataset/validation'
evaluation_data_dir = '/home/user/     /convnets/transfer-learning-keras/
 ↪dataset/evaluation'
nb_train_samples = 3000
nb_validation_samples = 1000
nb_evaluation_samples = 1000
epochs = 20
batch_size = 20
```

[3]:

```
def save_bottleneck_features():
    datagen = ImageDataGenerator(rescale=1. / 255)

    # build the VGG16 network
    model = applications.VGG16(include_top=False, weights='imagenet')

    generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
    bottleneck_features_train = model.predict_generator(
        generator, nb_train_samples // batch_size)
    np.save(open('bottleneck_features_train.npy', 'wb'),
            bottleneck_features_train)

    generator = datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
    bottleneck_features_validation = model.predict_generator(
        generator, nb_validation_samples // batch_size)
    np.save(open('bottleneck_features_validation.npy', 'wb'),
            bottleneck_features_validation)
```

[19]:

```
def train_top_model():
    train_data = np.load(open('bottleneck_features_train.npy', 'rb'))
    train_labels = np.array(
        [0] * (nb_train_samples // 2) + [1] * (nb_train_samples // 2))

    validation_data = np.load(open('bottleneck_features_validation.npy', 'rb'))
    validation_labels = np.array(
        [0] * (nb_validation_samples // 2) + [1] * (nb_validation_samples // 2))

    model = Sequential()
    model.add(Flatten(input_shape=train_data.shape[1:]))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

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

    history = model.fit(train_data, train_labels,
              epochs=epochs,
              batch_size=batch_size,
              validation_data=(validation_data, validation_labels))
    model.save(top_model_weights_path)
    return model, history
```

[11]:

```
def plot(model, history):
    plot_model(model, to_file='model.png')
    # Plot training & validation accuracy values
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    # Plot training & validation loss values
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

[6]:

```
save_bottleneck_features()
```

```
Found 3000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

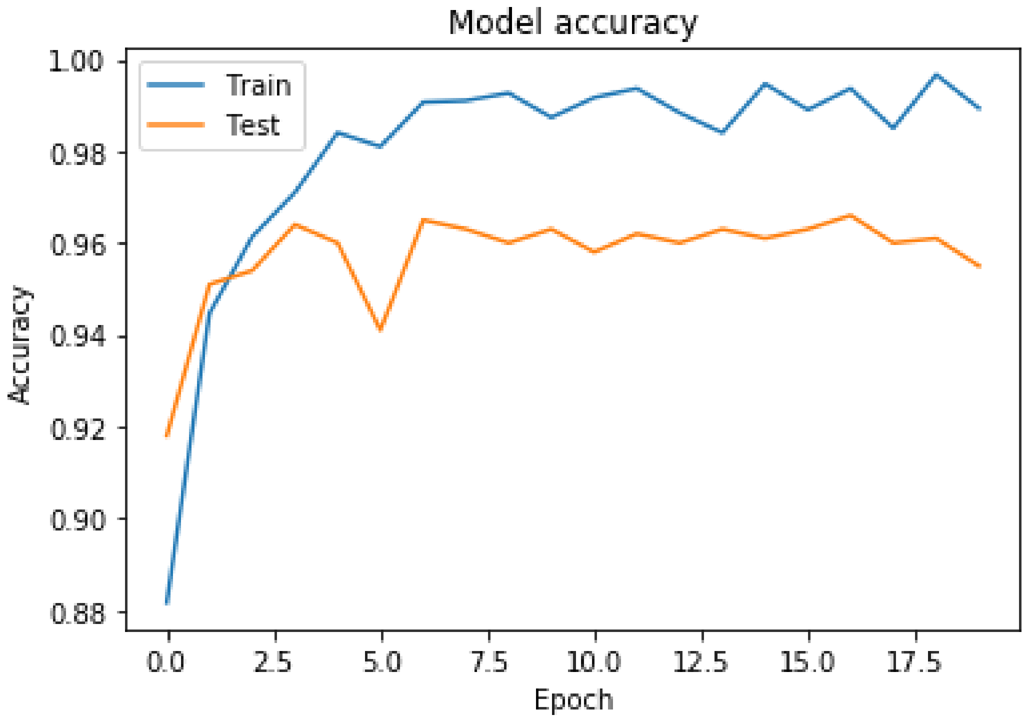
[20]:

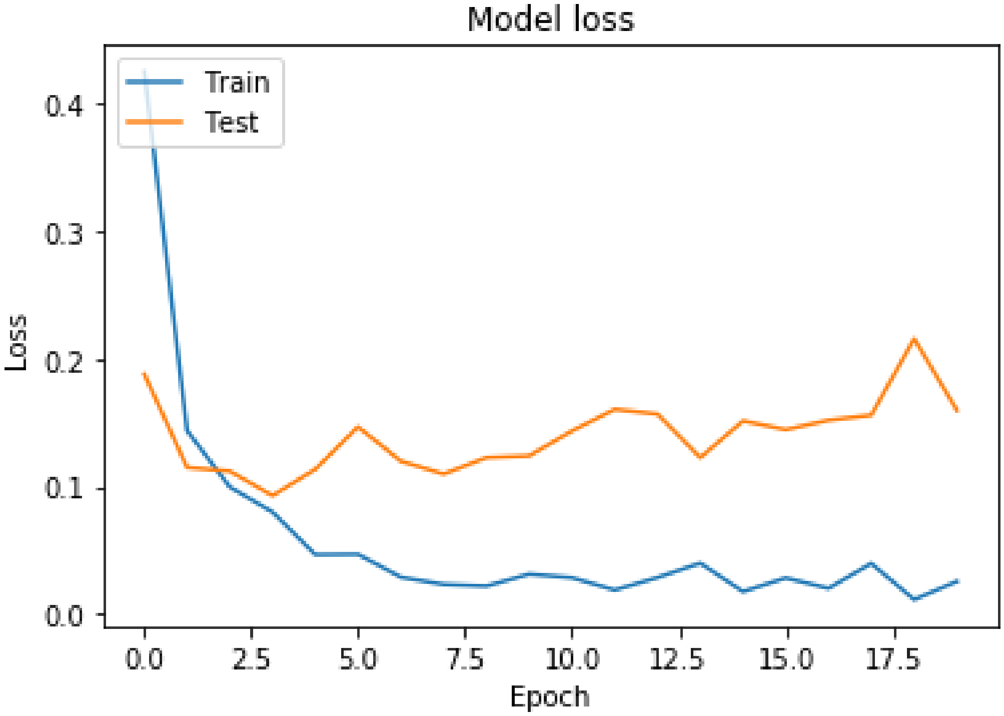
```
model, history = train_top_model()
```

```
Train on 3000 samples, validate on 1000 samples
Epoch 1/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.4256 -
accuracy: 0.8817 - val_loss: 0.1881 - val_accuracy: 0.9180
Epoch 2/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.1437 -
accuracy: 0.9447 - val_loss: 0.1149 - val_accuracy: 0.9510
Epoch 3/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0995 -
accuracy: 0.9613 - val_loss: 0.1120 - val_accuracy: 0.9540
Epoch 4/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0799 -
accuracy: 0.9710 - val_loss: 0.0927 - val_accuracy: 0.9640
Epoch 5/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0464 -
accuracy: 0.9840 - val_loss: 0.1135 - val_accuracy: 0.9600
Epoch 6/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0467 -
accuracy: 0.9810 - val_loss: 0.1467 - val_accuracy: 0.9410
Epoch 7/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0286 -
accuracy: 0.9907 - val_loss: 0.1198 - val_accuracy: 0.9650
Epoch 8/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0230 -
```

```
accuracy: 0.9910 - val_loss: 0.1097 - val_accuracy: 0.9630
Epoch 9/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0217 -
accuracy: 0.9927 - val_loss: 0.1227 - val_accuracy: 0.9600
Epoch 10/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0311 -
accuracy: 0.9873 - val_loss: 0.1240 - val_accuracy: 0.9630
Epoch 11/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0283 -
accuracy: 0.9917 - val_loss: 0.1434 - val_accuracy: 0.9580
Epoch 12/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0186 -
accuracy: 0.9937 - val_loss: 0.1604 - val_accuracy: 0.9620
Epoch 13/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0284 -
accuracy: 0.9883 - val_loss: 0.1567 - val_accuracy: 0.9600
Epoch 14/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0397 -
accuracy: 0.9840 - val_loss: 0.1226 - val_accuracy: 0.9630
Epoch 15/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0174 -
accuracy: 0.9947 - val_loss: 0.1513 - val_accuracy: 0.9610
Epoch 16/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0281 -
accuracy: 0.9890 - val_loss: 0.1444 - val_accuracy: 0.9630
Epoch 17/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0199 -
accuracy: 0.9937 - val_loss: 0.1518 - val_accuracy: 0.9660
Epoch 18/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0395 -
accuracy: 0.9850 - val_loss: 0.1558 - val_accuracy: 0.9600
Epoch 19/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0111 -
accuracy: 0.9967 - val_loss: 0.2160 - val_accuracy: 0.9610
Epoch 20/20
3000/3000 [==============================] - 4s 1ms/sample - loss: 0.0253 -
accuracy: 0.9893 - val_loss: 0.1597 - val_accuracy: 0.9550
```

```
[21]: plot(model, history)
```


Model accuracy
Train
Test
1.00
0.98
0.96
0.94
0.92
0.90
0.88
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model loss
Train
Test
0.4
0.3
0.2
0.1
0.0
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

[32]:

```
def evaluation(model):
    datagen = ImageDataGenerator(rescale=1. / 255)
    generator = datagen.flow_from_directory(
        evaluation_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
    model1 = applications.VGG16(include_top=False, weights='imagenet')
    features = model1.predict_generator(
        generator, nb_evaluation_samples // batch_size)
    return features
```

[33]:

```
features = evaluation(model)
```

```
Found 1000 images belonging to 2 classes.
```

[38]:

```
evaluation_labels = np.array(
        [0] * (nb_evaluation_samples // 2) + [1] * (nb_evaluation_samples // 2))
model.test_on_batch(features, evaluation_labels)
```

[38]:

```
[0.20799212, 0.952]
```