extraction-data-augmentation-ipynb

April 6, 2023

0.1 Transfer Learning using Feature Extraction Method – With Data Augmentation

Transfer learning is a machine learning ***technique that allows a model trained on one task to be reused or adapted for a different task***. There are two main types of transfer learning:

- ***Feature Extraction:*** In feature extraction, a pre-trained model is used as a fixed feature extractor. The pre-trained model is typically trained on a large dataset, such as ImageNet, and the final layer(s) that output the class predictions are removed. The remaining layers are then used as a feature extractor for a new dataset. These extracted features can be used as input to train a new model on the new dataset.
- ***Fine-tuning:*** In fine-tuning, a pre-trained model is used as an initialization for a new model. The pre-trained model is typically trained on a large dataset, such as ImageNet, and the final layer(s) that output the class predictions are replaced with new ones for the new task. The new model is then trained on the new dataset while keeping the weights of the pre-trained layers fixed or with a small learning rate.

0.1.1 *Import Necessary Libraries*

[ ]:
```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Flatten
from keras.applications.vgg16 import VGG16
```

[ ]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

0.1.2 *Downloading & Loading the dataset*

[ ]:
```
!kaggle datasets download -d salader/dogs-vs-cats
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading dogs-vs-cats.zip to /content
```

```
100% 1.06G/1.06G [00:28<00:00, 56.7MB/s]
100% 1.06G/1.06G [00:28<00:00, 40.1MB/s]
```

[ ]:
```python
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

0.1.3 *Defining the Convolutional Base & Freezing it & discarding the top layer (Dense Layer to train again)*

[ ]:
```python
conv_base = VGG16(
    weights='imagenet',
    include_top = False,
    input_shape=(150,150,3)
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

0.1.4 *Defining the sequential model & adding the conv_base*

[ ]:
```python
model = Sequential()

model.add(conv_base)
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

[ ]:
```python
conv_base.trainable = False
```

0.1.5 *Data Augmentation*

0.1.6 *Defining Training & Validation set for model training*

[ ]:
```python
from keras.preprocessing.image import ImageDataGenerator, array_to_img,␣
↪img_to_array, load_img
```

[ ]:
```python
batch_size = 32

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        '/content/train',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        '/content/test',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
```

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

0.1.7 *Model Training*

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

[ ]:
```
history = model.fit_generator(
        train_generator,
        epochs=10,
        validation_data=validation_generator)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  after removing the cwd from sys.path.
```

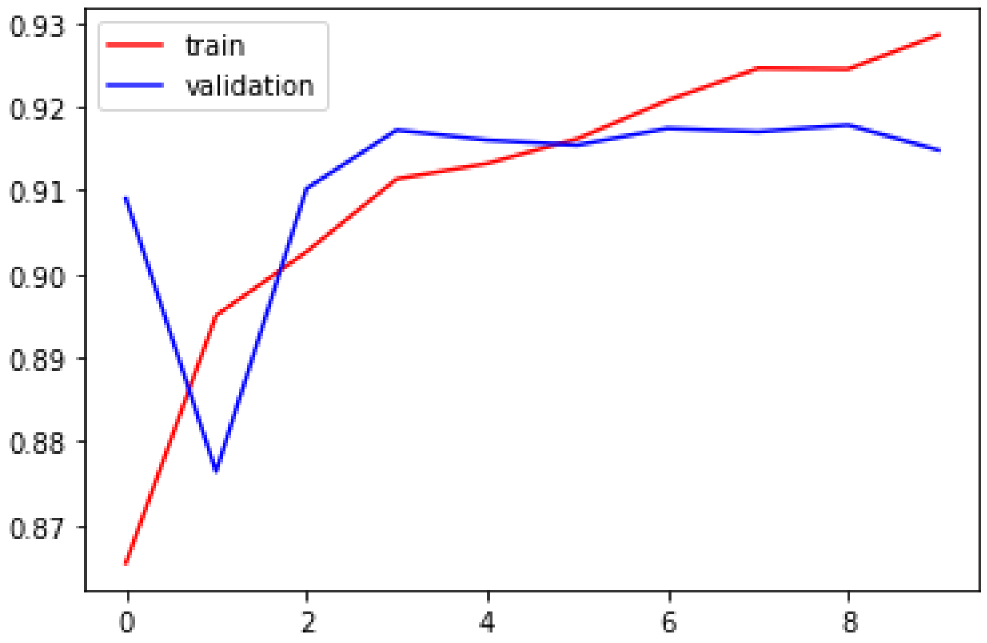
```
Epoch 1/10
625/625 [==============================] - 163s 259ms/step - loss: 0.3170 -
accuracy: 0.8655 - val_loss: 0.2139 - val_accuracy: 0.9090
Epoch 2/10
625/625 [==============================] - 160s 255ms/step - loss: 0.2446 -
accuracy: 0.8950 - val_loss: 0.2698 - val_accuracy: 0.8764
Epoch 3/10
625/625 [==============================] - 159s 255ms/step - loss: 0.2250 -
accuracy: 0.9026 - val_loss: 0.2066 - val_accuracy: 0.9102
Epoch 4/10
625/625 [==============================] - 159s 254ms/step - loss: 0.2113 -
accuracy: 0.9114 - val_loss: 0.1968 - val_accuracy: 0.9172
Epoch 5/10
625/625 [==============================] - 159s 254ms/step - loss: 0.2039 -
accuracy: 0.9132 - val_loss: 0.1975 - val_accuracy: 0.9160
Epoch 6/10
```

```
625/625 [==============================] - 159s 254ms/step - loss: 0.1971 -
accuracy: 0.9161 - val_loss: 0.2001 - val_accuracy: 0.9154
Epoch 7/10
625/625 [==============================] - 159s 254ms/step - loss: 0.1876 -
accuracy: 0.9208 - val_loss: 0.1950 - val_accuracy: 0.9174
Epoch 8/10
625/625 [==============================] - 159s 254ms/step - loss: 0.1789 -
accuracy: 0.9245 - val_loss: 0.2114 - val_accuracy: 0.9170
Epoch 9/10
625/625 [==============================] - 159s 254ms/step - loss: 0.1765 -
accuracy: 0.9244 - val_loss: 0.1883 - val_accuracy: 0.9178
Epoch 10/10
625/625 [==============================] - 159s 254ms/step - loss: 0.1697 -
accuracy: 0.9286 - val_loss: 0.1993 - val_accuracy: 0.9148
```

0.1.8 *Loss curve for Accuracy*

[ ]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```

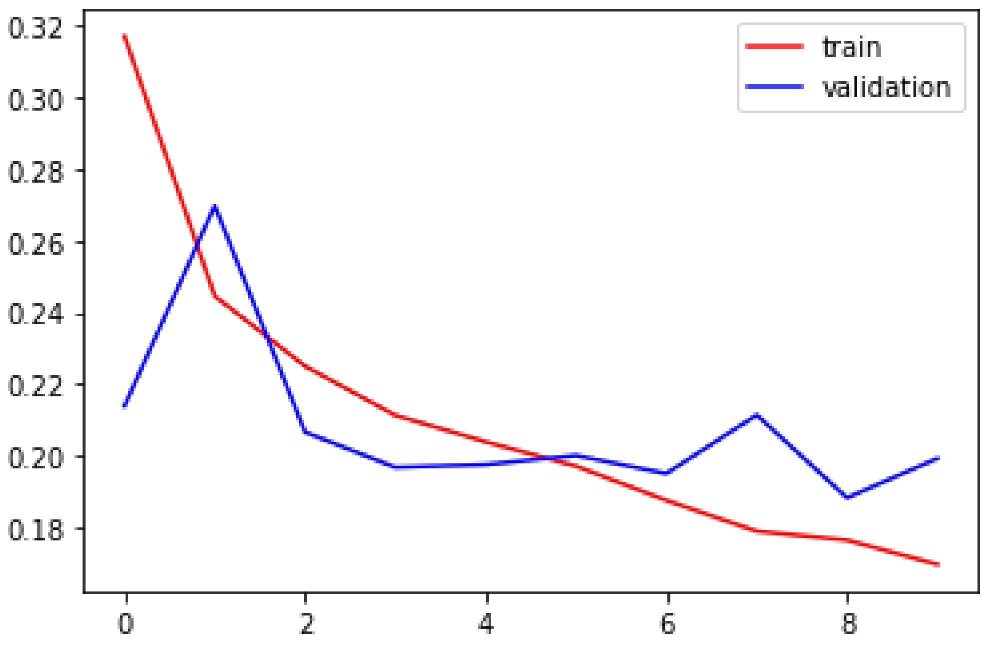


Epoch 10/10 625/625 [==============================] - 159s 254ms/step - loss: 0.1697 - accuracy: 0.9286 - val_loss: 0.1993 - val_accuracy: 0.9148

Here, we have used the Data Augmentation Technique to enhance our model training, to improve the accuracy, by providing the model different versions of a same image. The gap(overfitting case) we saw in case of the Feature Extraction Method without the data augmentation part, is reduced to a good amount, that means our purpose has performed as expected.

0.1.9 *Loss curve for training loss*

```
[ ]: plt.plot(history.history['loss'],color='red',label='train')
     plt.plot(history.history['val_loss'],color='blue',label='validation')
     plt.legend()
     plt.show()
```



Epoch 10/10 625/625 [==============================] - 159s 254ms/step - loss: 0.1697 - accuracy: 0.9286 - val_loss: 0.1993 - val_accuracy: 0.9148

The difference between the acc & val_acc is very less, we have overcome overfitting case.

Name: Shobhandeb Paul **Linkedin:** https://www.linkedin.com/in/shobhandeb-paul/ **Github:** https://github.com/herbert0419