Experiment: 6 Transfer Learning with CNN using MobileNetV2

Tanmay Rathod

1 Aim

The aim of this practical is to implement transfer learning with Convolutional Neural Networks (CNN) using the MobileNetV2 architecture. Transfer learning allows us to leverage pre-trained models to solve new, similar tasks more efficiently by using the knowledge gained from a previous related task.

2 Objective

The objective of this practical is to understand and implement the following steps:

1. Preparing the dataset of cats and dogs images.
2. Building a CNN using MobileNetV2 as a base model.
3. Freezing the base model's layers and adding custom layers for the specific classification task.
4. Training the model on the dataset and evaluating its performance.
5. Analyzing the training and validation accuracy/loss.

3 Introduction

Transfer learning is a technique in machine learning where a model developed for a task is reused as the starting point for a model on a second task. MobileNetV2 is a lightweight deep learning model that is well-suited for mobile and edge devices due to its efficient architecture. In this practical, we will use transfer learning with MobileNetV2 to classify images of cats and dogs.

4 Theory

Transfer learning with CNNs involves using a pre-trained CNN model, such as MobileNetV2, and adapting it to a new task. This adaptation usually involves

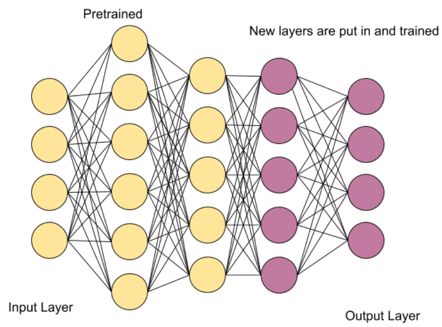

Figure 1: Architecture
https://www.danrose.ai/blog/transfer-learning-from-a-business-perspective

freezing the early layers of the network (which capture general features) and adding new layers on top to learn task-specific features.

5 Transfer Learning Code

```



import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf


_URL = 'https://storage.googleapis.com/mledu-datasets/
    cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.
    zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), '
    cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = tf.keras.utils.
    image_dataset_from_directory(train_dir,
```

```
          shuffle=True,
          batch_size=BATCH_SIZE,
          image_size=IMG_SIZE)


validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
               shuffle=True,
               batch_size=BATCH_SIZE,
               image_size=IMG_SIZE)


class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")



val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)


print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))


AUTOTUNE = tf.data.AUTOTUNE

```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)


data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])


for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')



preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input



rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)


IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')


image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
```

```
print(feature_batch.shape)


base_model.trainable = False


base_model.summary()



global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)


prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)


inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)


model.summary()


len(model.trainable_variables)


pip install pydot



tf.keras.utils.plot_model(model, show_shapes=True)

```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0.5, name='accuracy')])


initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)


print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))


history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)


acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
```

```
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

6 Results

The model was trained using transfer learning with MobileNetV2 on the cats and dogs dataset. Here are the results:

- Final Validation Accuracy: (0.9765)
- Final Training Accuracy: (0.9015)
- Number of validation batches: 26
- Number of test batches: 6

The training and validation accuracy/loss plots show the model's learning progress over the 10 epochs.

Epoch	Time/Epoch	Step	Loss	Accuracy	Val Loss	Val Accuracy
1	54s	63/63	0.7688	0.5565	0.5334	0.7413
2	46s	63/63	0.5763	0.7020	0.3961	0.8614
3	47s	63/63	0.4505	0.8085	0.3022	0.9109
4	48s	63/63	0.3855	0.8390	0.2430	0.9394
5	53s	63/63	0.3452	0.8505	0.2130	0.9493
6	55s	63/63	0.3091	0.8735	0.1897	0.9505
7	51s	63/63	0.2860	0.8890	0.1646	0.9616
8	53s	63/63	0.2608	0.8980	0.1516	0.9653
9	53s	63/63	0.2380	0.9140	0.1410	0.9666
10	48s	63/63	0.2405	0.9015	0.1251	0.9765

Table 1: Training and Validation Metrics

Figure 2: Dataset Of Cat & Dogs

Figure 3: Data Augmentation

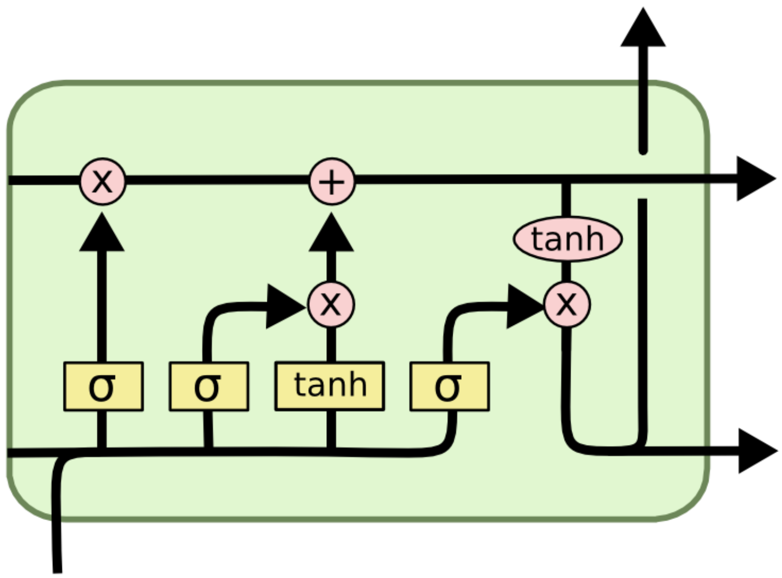

Figure 4: Enter Caption

7 Conclusion

In conclusion, this practical demonstrated the process of transfer learning with Convolutional Neural Networks using MobileNetV2. By leveraging a pre-trained model like MobileNetV2, we were able to achieve good accuracy for the classification of cats and dogs images. Transfer learning allows us to benefit from existing knowledge in deep learning models and adapt them to new tasks with less data and computation.

8 Reference

https://www.tensorflow.org/tutorials/images/transfer_learning