ESE 568 Project 4 documentation
Taharina Tasnim (115613595)
November 30, 2023

Tutorial 1: Convolutional Neural Network (CNN) | TensorFlow Core

List of the steps followed:

1. Import TensorFlow
2. Download and prepare the CIFAR10 dataset
3. Verify the data
4. Create the convolutional base
5. Add Dense layers on top
6. Compile and train the model
7. Evaluate the model

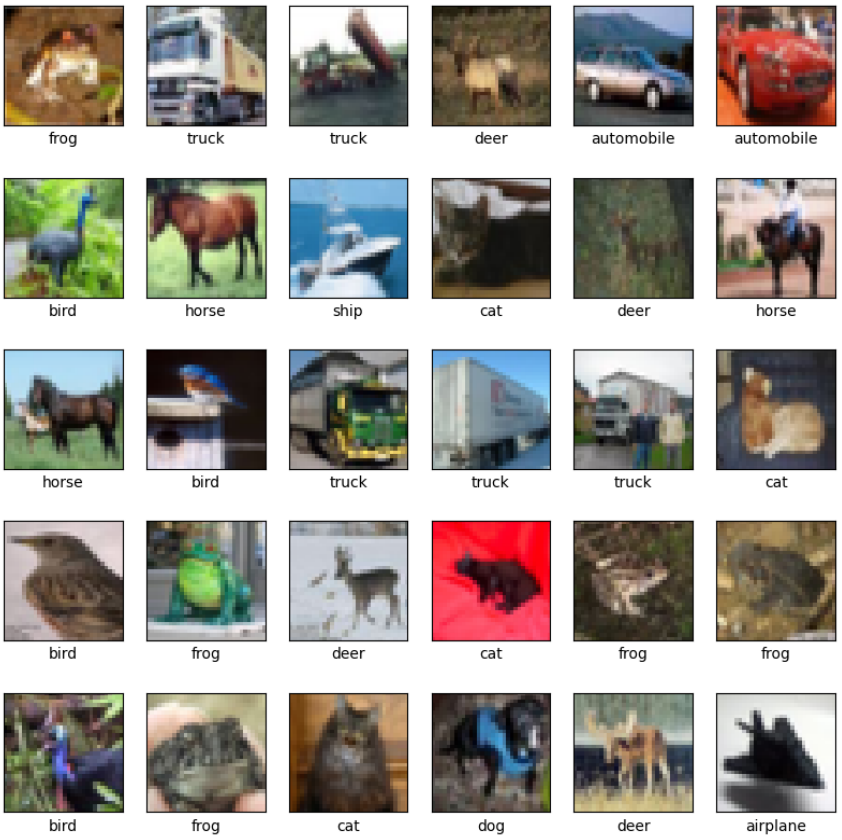
Source code:

```
# Step 1: import tensorflow
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Step 2: download and prepare the CIFAR 10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Step 3: verify the data
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(30):
    plt.subplot(5,6,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

```
# Step 4: create the convolutional base
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()

# Step 5: add dense layers on the top
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()

# Step 6: compile and train the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Step 7: evaluate the model
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)
```

Input: The CIFAR10 dataset

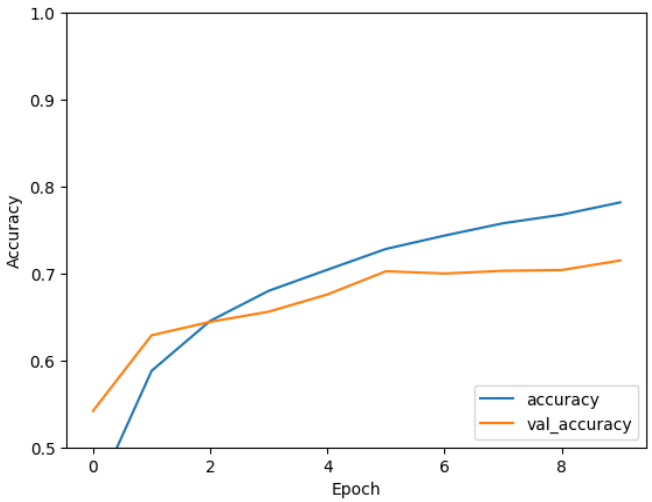
Output:




```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_6 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_7 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_6 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_7 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_11 (Conv2D)          (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 1/10
1563/1563 [==============================] - 70s 44ms/step - loss: 1.5313 - accuracy: 0.4388 - val_loss: 1.2726 - val_accuracy: 0.5419
Epoch 2/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.1565 - accuracy: 0.5881 - val_loss: 1.0559 - val_accuracy: 0.6290
Epoch 3/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.0015 - accuracy: 0.6455 - val_loss: 1.0124 - val_accuracy: 0.6444
Epoch 4/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.9116 - accuracy: 0.6801 - val_loss: 0.9694 - val_accuracy: 0.6561
Epoch 5/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8448 - accuracy: 0.7043 - val_loss: 0.9216 - val_accuracy: 0.6759
Epoch 6/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7826 - accuracy: 0.7283 - val_loss: 0.8456 - val_accuracy: 0.7026
Epoch 7/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7367 - accuracy: 0.7436 - val_loss: 0.8715 - val_accuracy: 0.7000
Epoch 8/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6907 - accuracy: 0.7578 - val_loss: 0.8633 - val_accuracy: 0.7031
Epoch 9/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.6548 - accuracy: 0.7677 - val_loss: 0.8773 - val_accuracy: 0.7040
Epoch 10/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6186 - accuracy: 0.7819 - val_loss: 0.8654 - val_accuracy: 0.7150
313/313 - 3s - loss: 0.8654 - accuracy: 0.7150 - 3s/epoch - 10ms/step
```



0.7149999737739563
[Test Accuracy: 71.5%]

Tutorial 2: [Image classification | TensorFlow Core]

Step 1: Setup

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
import pathlib

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Step 2: Download and explore the dataset

```
dataset_url =
https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
data_dir = tf.keras.utils.get_file('flower_photos.tar', origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')
```

Step 3: Create a dataset and 80% of the images for training and 20% for validation.

```
batch_size = 32
img_height = 180
img_width = 180

train_ds = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size)


val_ds = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size)
```

Step 4: Configure the dataset for performance

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Step 5: Standardize the data

```
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

Step 6: Create, compile the model

```
num_classes = len(class_names)
model = Sequential([
        layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
        layers.Conv2D(16, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes)
      ])

model.compile(optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3989285 (15.22 MB)
Trainable params: 3989285 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

Step 7: Train the model and visualize the result

```
epochs=10
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

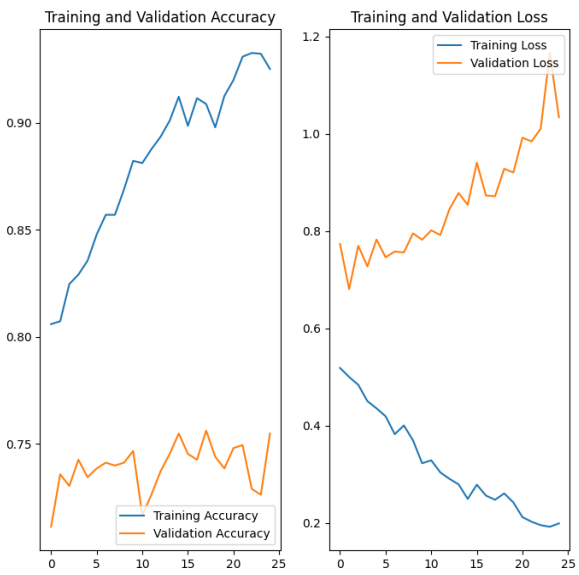
[using 10 epochs]

```
Epoch 1/10
92/92 [==============================] - 15s 48ms/step - loss: 1.3361 - accuracy: 0.4458 - val_loss: 1.0822 - val_accuracy: 0.5232
Epoch 2/10
92/92 [==============================] - 2s 20ms/step - loss: 0.9987 - accuracy: 0.6042 - val_loss: 1.0593 - val_accuracy: 0.5790
Epoch 3/10
92/92 [==============================] - 2s 19ms/step - loss: 0.8142 - accuracy: 0.6884 - val_loss: 0.9003 - val_accuracy: 0.6390
Epoch 4/10
92/92 [==============================] - 2s 22ms/step - loss: 0.5985 - accuracy: 0.7793 - val_loss: 0.9686 - val_accuracy: 0.6444
Epoch 5/10
92/92 [==============================] - 2s 22ms/step - loss: 0.3713 - accuracy: 0.8709 - val_loss: 0.9903 - val_accuracy: 0.6649
Epoch 6/10
92/92 [==============================] - 2s 20ms/step - loss: 0.2314 - accuracy: 0.9217 - val_loss: 1.0376 - val_accuracy: 0.6512
Epoch 7/10
92/92 [==============================] - 2s 19ms/step - loss: 0.1189 - accuracy: 0.9653 - val_loss: 1.5383 - val_accuracy: 0.6362
Epoch 8/10
92/92 [==============================] - 2s 19ms/step - loss: 0.0645 - accuracy: 0.9830 - val_loss: 1.3407 - val_accuracy: 0.6444
Epoch 9/10
92/92 [==============================] - 2s 19ms/step - loss: 0.0485 - accuracy: 0.9891 - val_loss: 1.6125 - val_accuracy: 0.6689
Epoch 10/10
92/92 [==============================] - 2s 19ms/step - loss: 0.0436 - accuracy: 0.9881 - val_loss: 1.6938 - val_accuracy: 0.6390
```

[Using 25 epochs] val_accuracy went up

```
Epoch 11/25
92/92 [==============================] - 3s 28ms/step - loss: 0.3286 - accuracy: 0.8811 - val_loss: 0.8013 - val_accuracy: 0.7166
Epoch 12/25
92/92 [==============================] - 3s 28ms/step - loss: 0.3036 - accuracy: 0.8876 - val_loss: 0.7916 - val_accuracy: 0.7262
Epoch 13/25
92/92 [==============================] - 3s 30ms/step - loss: 0.2905 - accuracy: 0.8934 - val_loss: 0.8450 - val_accuracy: 0.7371
Epoch 14/25
92/92 [==============================] - 3s 29ms/step - loss: 0.2791 - accuracy: 0.9009 - val_loss: 0.8780 - val_accuracy: 0.7452
Epoch 15/25
92/92 [==============================] - 3s 28ms/step - loss: 0.2493 - accuracy: 0.9121 - val_loss: 0.8538 - val_accuracy: 0.7548
Epoch 16/25
92/92 [==============================] - 3s 28ms/step - loss: 0.2784 - accuracy: 0.8985 - val_loss: 0.9404 - val_accuracy: 0.7452
Epoch 17/25
92/92 [==============================] - 3s 28ms/step - loss: 0.2561 - accuracy: 0.9114 - val_loss: 0.8729 - val_accuracy: 0.7425
Epoch 18/25
92/92 [==============================] - 3s 29ms/step - loss: 0.2474 - accuracy: 0.9087 - val_loss: 0.8715 - val_accuracy: 0.7561
Epoch 19/25
92/92 [==============================] - 3s 29ms/step - loss: 0.2606 - accuracy: 0.8978 - val_loss: 0.9278 - val_accuracy: 0.7439
Epoch 20/25
92/92 [==============================] - 3s 28ms/step - loss: 0.2421 - accuracy: 0.9125 - val_loss: 0.9201 - val_accuracy: 0.7384
Epoch 21/25
92/92 [==============================] - 3s 28ms/step - loss: 0.2118 - accuracy: 0.9200 - val_loss: 0.9917 - val_accuracy: 0.7480
Epoch 22/25
92/92 [==============================] - 3s 30ms/step - loss: 0.2023 - accuracy: 0.9309 - val_loss: 0.9841 - val_accuracy: 0.7493
Epoch 23/25
92/92 [==============================] - 3s 30ms/step - loss: 0.1952 - accuracy: 0.9326 - val_loss: 1.0102 - val_accuracy: 0.7289
Epoch 24/25
92/92 [==============================] - 3s 28ms/step - loss: 0.1920 - accuracy: 0.9322 - val_loss: 1.1658 - val_accuracy: 0.7262
Epoch 25/25
92/92 [==============================] - 3s 28ms/step - loss: 0.1988 - accuracy: 0.9251 - val_loss: 1.0336 - val_accuracy: 0.7548
```

Epoch = 10



Epoch = 25

Step 8: Data augmentation

```
data_augmentation = keras.Sequential([
        layers.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),])
```

Step 9: Dropout

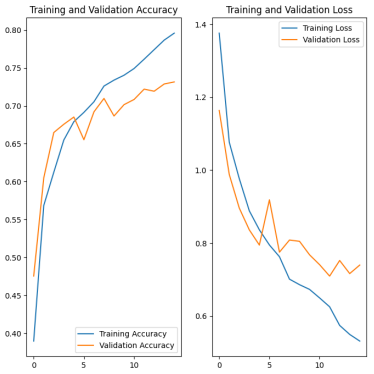
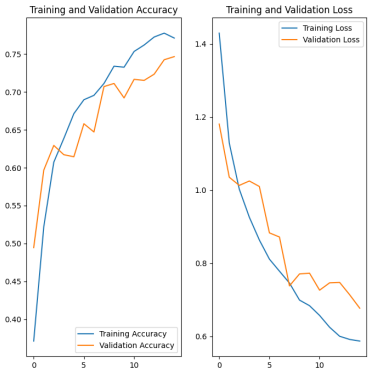
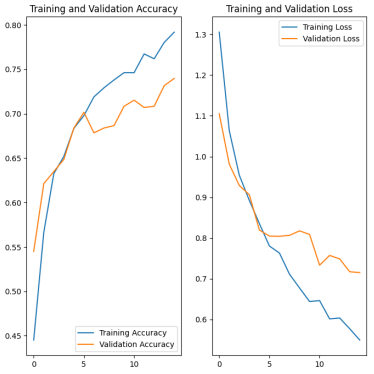
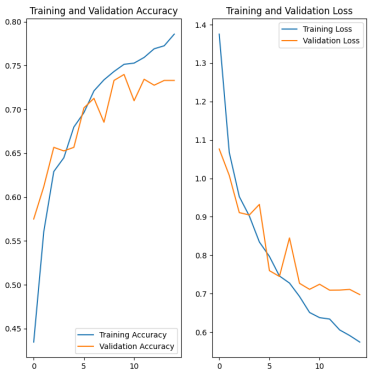
```
model = Sequential([
        data_augmentation,
        layers.Rescaling(1./255),
```

```
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, name="outputs")])
```

Step 10: Compile and train the model, visualise training results

```
model.compile(optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])
model.summary()
epochs = 15
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Applying dropout = 0.1, 0.2, 0.3, 0.4

Dropout = 0.1	- loss: 0.5313 - accuracy: 0.7960 - val_loss: 0.7397 - val_accuracy: 0.7316	
Dropout = 0.2	- loss: 0.5255 - accuracy: 0.8028 - val_loss: 0.7162 - val_accuracy: 0.7275	
Dropout = 0.3	- loss: 0.5494 - accuracy: 0.7919 - val_loss: 0.7150 - val_accuracy: 0.7398	
Dropout = 0.4	- loss: 0.5739 - accuracy: 0.7858 - val_loss: 0.6975 - val_accuracy: 0.7330	

Dropout rate of **0.3** gives us the best accuracy and validation results.

Step 11: Predict on new data

```
# Classify image
# Note: Data augmentation and dropout layers are inactive at inference time.
        sunflower_url =
        "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflo
        wer.jpg"
        sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)
        img = tf.keras.utils.load_img(sunflower_path,
                target_size=(img_height, img_width))
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)
        predictions = model.predict(img_array)
        score = tf.nn.softmax(predictions[0])
        print("This image most likely belongs to {} with a {:.2f} percent confidence."
            .format(class_names[np.argmax(score)], 100 * np.max(score)))
```

This model is giving us the output like below:

```
Downloading data from
https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jp
g
117948/117948 [==============================] - 0s 3us/step
1/1 [==============================] - 0s 164ms/step
This image most likely belongs to sunflowers with a 97.98 percent confidence.
```

Tutorial 3: [Transfer learning and fine-tuning | TensorFlow Core]

Two ways to customize a pretrained model:

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful features from new samples.
2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model.

Step 1: Set up

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

Step 2: Data preprocessing

2.i) Data download

```
_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)
```

2.ii) Create test data[20% of validation set data to a test set]

```
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))
```

2.iii) Configure the dataset for performance

```
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)
```

2.iv) Use data augmentation

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

Step 3: Create a base model from an existing pre-trained conv net

```
# Base model =  tf.keras.applications.MobileNetV2 for use
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)

# Create the base model from the pre-trained model MobileNet V2
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)
```

Step 4: Feature Extraction

4.i) Freeze the convolutional base

```
base_model.trainable = False
base_model.summary()
```

```
ormalization)

block_16_expand_relu (ReLU  (None, 5, 5, 960)           0         ['block_16_expand_BN[0][0]']
)

block_16_depthwise (Depthw  (None, 5, 5, 960)           8640      ['block_16_expand_relu[0][0]']
iseConv2D)

block_16_depthwise_BN (Bat  (None, 5, 5, 960)           3840      ['block_16_depthwise[0][0]']
chNormalization)

block_16_depthwise_relu (R  (None, 5, 5, 960)           0         ['block_16_depthwise_BN[0][0]'
eLU)                                                              ]

block_16_project (Conv2D)   (None, 5, 5, 320)           307200    ['block_16_depthwise_relu[0][0
                                                                  ]']

block_16_project_BN (Batch  (None, 5, 5, 320)           1280      ['block_16_project[0][0]']
Normalization)

Conv_1 (Conv2D)             (None, 5, 5, 1280)          409600    ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalizat  (None, 5, 5, 1280)          5120      ['Conv_1[0][0]']
ion)

out_relu (ReLU)             (None, 5, 5, 1280)          0         ['Conv_1_bn[0][0]']

==================================================================================================
Total params: 2257984 (8.61 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 2257984 (8.61 MB)
__________________________________________________________________________________________________
```

4.ii) Add classification head

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

Output: (32, 1280)

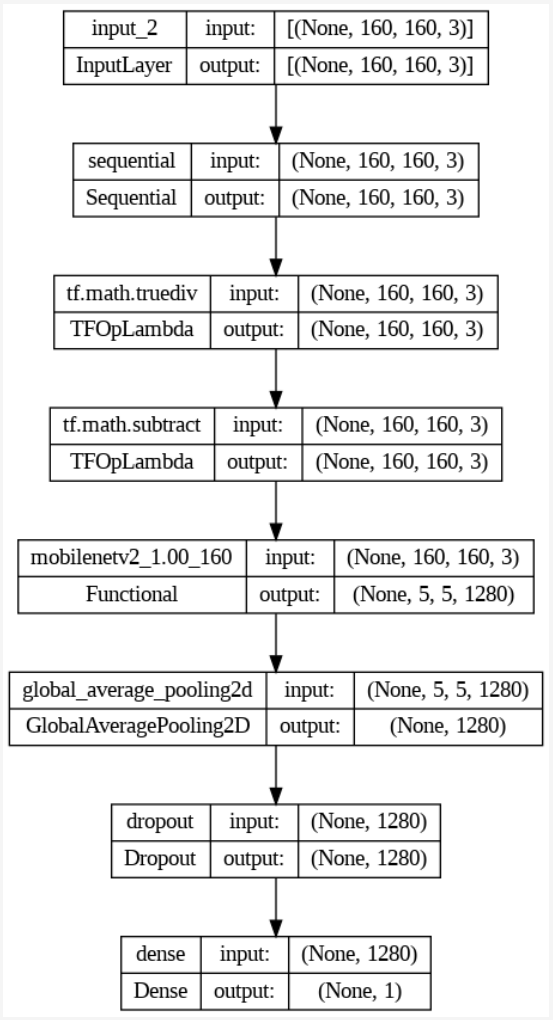
```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

Output: (32, 1)

```
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()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambd  (None, 160, 160, 3)       0
 a)

 tf.math.subtract (TFOpLamb  (None, 160, 160, 3)       0
 da)

 mobilenetv2_1.00_160 (Func  (None, 5, 5, 1280)        2257984
 tional)

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2259265 (8.62 MB)
Trainable params: 1281 (5.00 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

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



Step 5: Compile the model

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.Binary_Crossentropy(from_logits=True),
              metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0, name='accuracy')])
```

Step 6: Train the model

```
initial_epochs = 10
loss0, accuracy0 = model.evaluate(validation_dataset)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

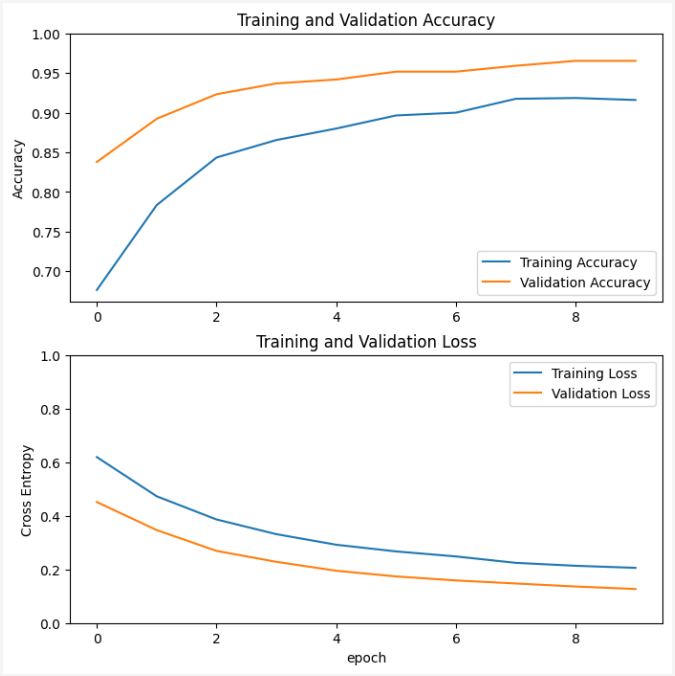
Output : initial loss: 0.68, initial accuracy: 0.61

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

```
Epoch 1/10
63/63 [==============================] - 62s 914ms/step - loss: 0.6190 - accuracy: 0.6765 - val_loss: 0.4513 - val_accuracy: 0.8379
Epoch 2/10
63/63 [==============================] - 58s 914ms/step - loss: 0.4728 - accuracy: 0.7835 - val_loss: 0.3467 - val_accuracy: 0.8923
Epoch 3/10
63/63 [==============================] - 58s 920ms/step - loss: 0.3862 - accuracy: 0.8435 - val_loss: 0.2690 - val_accuracy: 0.9233
Epoch 4/10
63/63 [==============================] - 58s 918ms/step - loss: 0.3314 - accuracy: 0.8655 - val_loss: 0.2282 - val_accuracy: 0.9369
Epoch 5/10
63/63 [==============================] - 60s 947ms/step - loss: 0.2920 - accuracy: 0.8800 - val_loss: 0.1949 - val_accuracy: 0.9418
Epoch 6/10
63/63 [==============================] - 58s 928ms/step - loss: 0.2671 - accuracy: 0.8965 - val_loss: 0.1737 - val_accuracy: 0.9517
Epoch 7/10
63/63 [==============================] - 59s 931ms/step - loss: 0.2484 - accuracy: 0.9000 - val_loss: 0.1588 - val_accuracy: 0.9517
Epoch 8/10
63/63 [==============================] - 60s 948ms/step - loss: 0.2244 - accuracy: 0.9175 - val_loss: 0.1474 - val_accuracy: 0.9592
Epoch 9/10
63/63 [==============================] - 61s 962ms/step - loss: 0.2134 - accuracy: 0.9185 - val_loss: 0.1361 - val_accuracy: 0.9653
Epoch 10/10
63/63 [==============================] - 60s 951ms/step - loss: 0.2058 - accuracy: 0.9160 - val_loss: 0.1268 - val_accuracy: 0.9653
```

Step 7: Learning curves

```
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()
```



Step 8: Fine tuning

8.i) Unfreeze the top layers of the model

```
base_model.trainable = True
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))
```

Output: Number of layers in the base model: 154

```
# Fine-tune from this layer onwards
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
        optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
        metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0, name='accuracy')])
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambd  (None, 160, 160, 3)       0
 a)

 tf.math.subtract (TFOpLamb  (None, 160, 160, 3)       0
 da)

 mobilenetv2_1.00_160 (Func  (None, 5, 5, 1280)        2257984
 tional)

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2259265 (8.62 MB)
Trainable params: 1862721 (7.11 MB)
Non-trainable params: 396544 (1.51 MB)
_________________________________________________________________
```

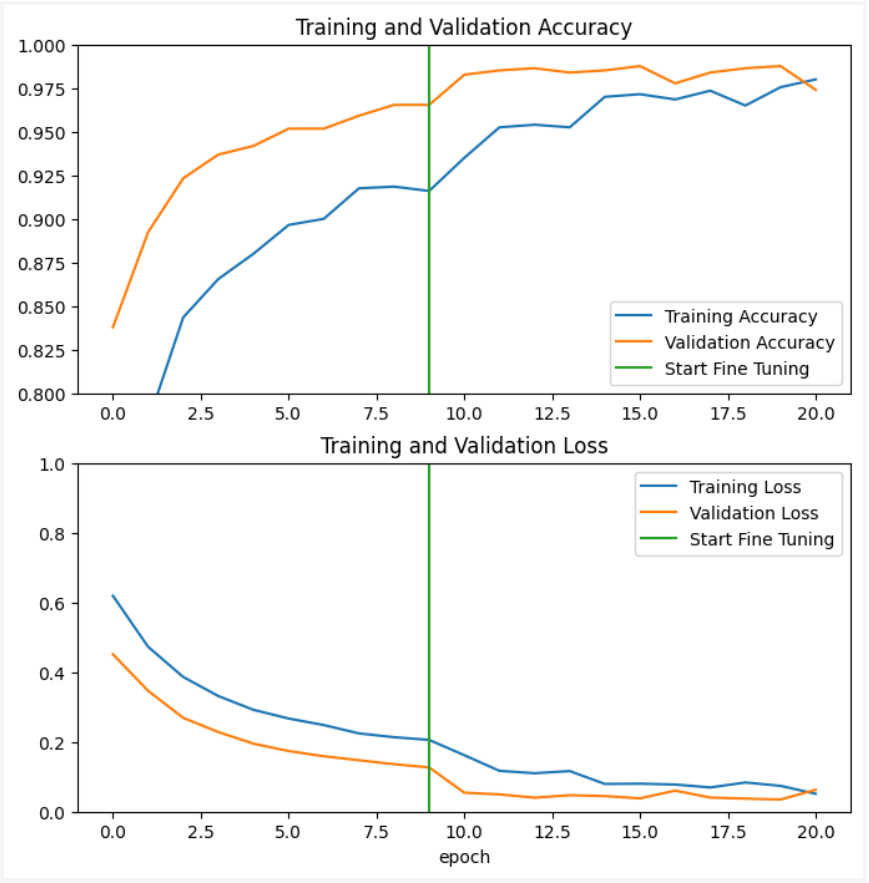
8.ii) Continue training the model

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs
history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

```
Epoch 10/20
63/63 [==============================] - 79s 1s/step - loss: 0.1619 - accuracy: 0.9350 - val_loss: 0.0543 - val_accuracy: 0.9827
Epoch 11/20
63/63 [==============================] - 80s 1s/step - loss: 0.1170 - accuracy: 0.9525 - val_loss: 0.0495 - val_accuracy: 0.9851
Epoch 12/20
63/63 [==============================] - 79s 1s/step - loss: 0.1102 - accuracy: 0.9540 - val_loss: 0.0400 - val_accuracy: 0.9864
Epoch 13/20
63/63 [==============================] - 85s 1s/step - loss: 0.1164 - accuracy: 0.9525 - val_loss: 0.0471 - val_accuracy: 0.9839
Epoch 14/20
63/63 [==============================] - 79s 1s/step - loss: 0.0794 - accuracy: 0.9700 - val_loss: 0.0447 - val_accuracy: 0.9851
Epoch 15/20
63/63 [==============================] - 77s 1s/step - loss: 0.0802 - accuracy: 0.9715 - val_loss: 0.0381 - val_accuracy: 0.9876
Epoch 16/20
63/63 [==============================] - 78s 1s/step - loss: 0.0778 - accuracy: 0.9685 - val_loss: 0.0603 - val_accuracy: 0.9777
Epoch 17/20
63/63 [==============================] - 78s 1s/step - loss: 0.0694 - accuracy: 0.9735 - val_loss: 0.0404 - val_accuracy: 0.9839
Epoch 18/20
63/63 [==============================] - 78s 1s/step - loss: 0.0836 - accuracy: 0.9650 - val_loss: 0.0374 - val_accuracy: 0.9864
Epoch 19/20
63/63 [==============================] - 78s 1s/step - loss: 0.0740 - accuracy: 0.9755 - val_loss: 0.0348 - val_accuracy: 0.9876
Epoch 20/20
63/63 [==============================] - 78s 1s/step - loss: 0.0512 - accuracy: 0.9800 - val_loss: 0.0626 - val_accuracy: 0.9740
```

Validation accuracy : **97.40%**

```
cc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']
loss += history_fine.history['loss']
val_loss += history_fine.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.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
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.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Step 9: Evaluation

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
6/6 [==============================] - 4s 659 ms/step - loss: 0.0450 - accuracy: 0.9844
Test accuracy : 0.984375
```

Step 10: Prediction

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()
# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)
print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)
plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

Predictions:	[0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0]
Labels:	[0 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0]

Analysis

: Unfreeze some of the top layers of the frozen model library, and jointly train the newly added classifier layer and the last few layers of the base model.

: Use a relatively small learning rate (base_learning_rate/10) when fine tuning the base model to reduce overfitting.

Tutorial 4: [TensorFlow Hub Object Detection Colab]

Step 1: Imports and Setup

```
!pip install numpy==1.24.3
!pip install protobuf==3.20.3
import os
import pathlib
import matplotlib
import matplotlib.pyplot as plt
import io
import scipy.misc
import numpy as np
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont
from six.moves.urllib.request import urlopen
import tensorflow as tf
import tensorflow_hub as hub
tf.get_logger().setLevel('ERROR')
```

Step 2: Some utils

```
def load_image_into_numpy_array(path):
  """Load an image from file into a numpy array.

  Puts image into numpy array to feed into tensorflow graph.
  Note that by convention we put it into a numpy array with shape
  (height, width, channels), where channels=3 for RGB.

  Args:
    path: the file path to the image

  Returns:
    uint8 numpy array with shape (img_height, img_width, 3)
  """
  image = None
  if(path.startswith('http')):
    response = urlopen(path)
    image_data = response.read()
    image_data = BytesIO(image_data)
    image = Image.open(image_data)
  else:
    image_data = tf.io.gfile.GFile(path, 'rb').read()
    image = Image.open(BytesIO(image_data))

  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (1, im_height, im_width, 3)).astype(np.uint8)
```

```
ALL_MODELS = {
'CenterNet HourGlass104 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512/1',
'CenterNet HourGlass104 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512_kpts/1',
'CenterNet HourGlass104 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024/1',
'CenterNet HourGlass104 Keypoints 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024_kpts/1',
'CenterNet Resnet50 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512/1',
'CenterNet Resnet50 V1 FPN Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512_kpts/1',
'CenterNet Resnet101 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet101v1_fpn_512x512/1',
'CenterNet Resnet50 V2 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512/1',
'CenterNet Resnet50 V2 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512_kpts/1',
'EfficientDet D0 512x512' : 'https://tfhub.dev/tensorflow/efficientdet/d0/1',
'EfficientDet D1 640x640' : 'https://tfhub.dev/tensorflow/efficientdet/d1/1',
'EfficientDet D2 768x768' : 'https://tfhub.dev/tensorflow/efficientdet/d2/1',
'EfficientDet D3 896x896' : 'https://tfhub.dev/tensorflow/efficientdet/d3/1',
'EfficientDet D4 1024x1024' : 'https://tfhub.dev/tensorflow/efficientdet/d4/1',
'EfficientDet D5 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d5/1',
'EfficientDet D6 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d6/1',
'EfficientDet D7 1536x1536' : 'https://tfhub.dev/tensorflow/efficientdet/d7/1',
'SSD MobileNet v2 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2',
'SSD MobileNet V1 FPN 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v1/fpn_640x640/1',
'SSD MobileNet V2 FPNLite 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_320x320/1',
'SSD MobileNet V2 FPNLite 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_640x640/1',
'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_640x640/1',
'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_1024x1024/1',
'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_640x640/1',
'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_1024x1024/1',
'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_640x640/1',
'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_1024x1024/1',
'Faster R-CNN ResNet50 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1',
'Faster R-CNN ResNet50 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_1024x1024/1',
'Faster R-CNN ResNet50 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_800x1333/1',
'Faster R-CNN ResNet101 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_640x640/1',
'Faster R-CNN ResNet101 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_1024x1024/1',
'Faster R-CNN ResNet101 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_800x1333/1',
'Faster R-CNN ResNet152 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_640x640/1',
'Faster R-CNN ResNet152 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_1024x1024/1',
'Faster R-CNN ResNet152 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_800x1333/1',
'Faster R-CNN Inception ResNet V2 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_640x640/1',
'Faster R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_1024x1024/1',
'Mask R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1'
}
```

```
IMAGES_FOR_TEST = {
  'Beach' : 'models/research/object_detection/test_images/image2.jpg',
  'Dogs' : 'models/research/object_detection/test_images/image1.jpg',
  # By Heiko Gorski, Source: https://commons.wikimedia.org/wiki/File:Naxos_Taverna.jpg
  'Naxos Taverna' : 'https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos_Taverna.jpg',
  # Source: https://commons.wikimedia.org/wiki/File:The_Coleoptera_of_the_British_islands_(Plate_125)_(8592917784).jpg
  'Beatles' : 'https://upload.wikimedia.org/wikipedia/commons/1/1b/The_Coleoptera_of_the_British_islands_%28Plate_125%29_%
  # By Américo Toledano, Source: https://commons.wikimedia.org/wiki/File:Biblioteca_Maim%C3%B3nides,_Campus_Universitario_
  'Phones' : 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Biblioteca_Maim%C3%B3nides%2C_Campus_Universitario_
  # Source: https://commons.wikimedia.org/wiki/File:The_smaller_British_birds_(8053836633).jpg
  'Birds' : 'https://upload.wikimedia.org/wikipedia/commons/0/09/The_smaller_British_birds_%288053836633%29.jpg',
}
```

```
COCO17_HUMAN_POSE_KEYPOINTS = [(0, 1), (0, 2), (1, 3), (2, 4), (0, 5), (0, 6), (5, 7), (7, 9), (6, 8), (8, 10), (5, 6), (5, 11), (6, 12), (11, 12), (11, 13), (13, 15), (12, 14), (14, 16)]
```

Step 3: Install Visualization tools [TensorFlow Object Detection API]

```
# Clone the tensorflow models repository
!git clone --depth 1 https://github.com/tensorflow/models
%%bash
sudo apt install -y protobuf-compiler
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
python -m pip install .
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.utils import ops as utils_ops
%matplotlib inline
```

Step 4: Load label map data (for plotting)

```
PATH_TO_LABELS = './models/research/object_detection/data/mscoco_label_map.pbtxt'
category_index =
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)
```

Step 5: Build a detection model and load pre-trained model weights

```
model_display_name = 'CenterNet HourGlass104 Keypoints 512x512'
model_handle = ALL_MODELS[model_display_name]
```

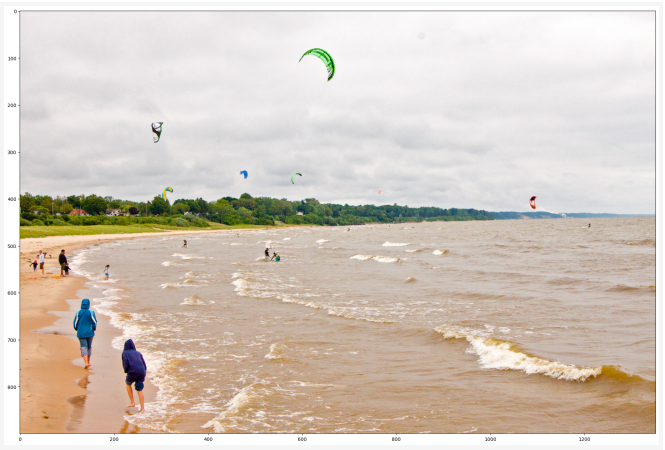
Step 6: Loading the selected model from TensorFlow Hub

```
hub_model = hub.load(model_handle)
```

Step 7: Loading an image

```
selected_image = 'Beach'
flip_image_horizontally = False
convert_image_to_grayscale = False
image_path = IMAGES_FOR_TEST[selected_image]
image_np = load_image_into_numpy_array(image_path)
# Flip horizontally
if(flip_image_horizontally):
  image_np[0] = np.fliplr(image_np[0]).copy()
# Convert image to grayscale
if (convert_image_to_grayscale):
  image_np[0] = np.tile(np.mean(image_np[0], 2, keepdims=True), (1, 1, 3)).astype(np.uint8)
```

```
plt.figure(figsize=(24,32))
plt.imshow(image_np[0])
plt.show()
```



Step 8: Doing the inference

```
results = hub_model(image_np)
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

Output: dict_keys(['detection_classes', 'detection_keypoints', 'detection_keypoint_scores', 'detection_scores', 'detection_boxes', 'num_detections'])

Step 9: Visualizing the results

```
label_id_offset = 0
image_np_with_detections = image_np.copy()
keypoints, keypoint_scores = None, None
if 'detection_keypoints' in result:
  keypoints = result['detection_keypoints'][0]
  keypoint_scores = result['detection_keypoint_scores'][0]

viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections[0],
    result['detection_boxes'][0],
    (result['detection_classes'][0] + label_id_offset).astype(int),
    result['detection_scores'][0],
    category_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=200,
    min_score_thresh=.30,
    agnostic_mode=False,
    keypoints=keypoints,
    keypoint_scores=keypoint_scores,
```

```
      keypoint_edges=COCO17_HUMAN_POSE_KEYPOINTS)
plt.figure(figsize=(24,32))
plt.imshow(image_np_with_detections[0])
plt.show()
```

Step 10: Instance Segmentation

```
label_id_offset = 0
image_np_with_detections = image_np.copy()
keypoints, keypoint_scores = None, None
if 'detection_keypoints' in result:
  keypoints = result['detection_keypoints'][0]
  keypoint_scores = result['detection_keypoint_scores'][0]

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections[0],
      result['detection_boxes'][0],
      (result['detection_classes'][0] + label_id_offset).astype(int),
      result['detection_scores'][0],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.30,
      agnostic_mode=False,
      keypoints=keypoints,
      keypoint_scores=keypoint_scores,
      keypoint_edges=COCO17_HUMAN_POSE_KEYPOINTS)

plt.figure(figsize=(24,32))
plt.imshow(image_np_with_detections[0])
plt.show()
```

Step 9

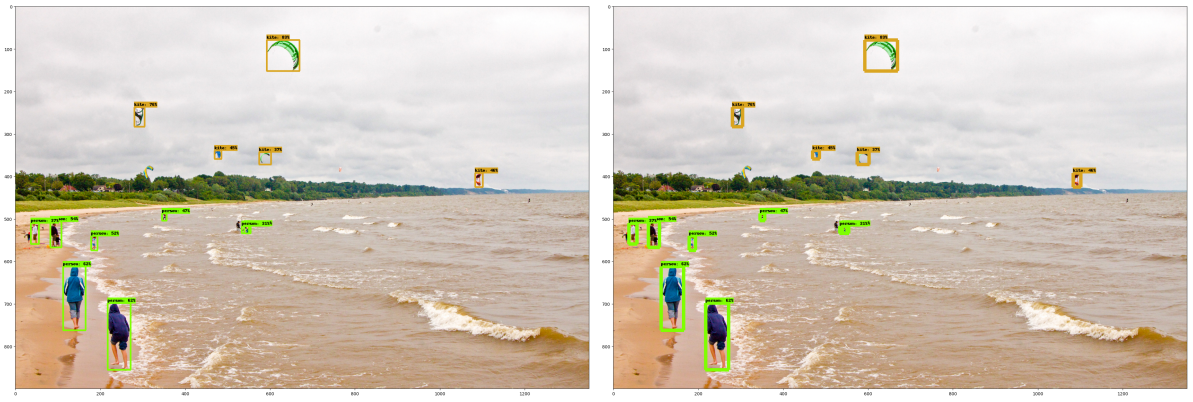
Step 10

Analysis

- Compared random three model outputs

1. Model: CenterNet HourGlass104 Keypoints 512x512(Normal VS Instance Segmentation)

2. Model: SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)(Normal VS Instance Segmentation)



3. Model: Mask R-CNN Inception ResNet V2 1024x1024(Normal VS Instance Segmentation)

- Modify some of the input images and see if detection still works.

1st row: Regular Image, 2nd row: flipped image, 3rd row: gray scaled image

- Printed other keys/values of result[]
    1. print(result['detection_classes'])
       containing detection class index from the label file

```
[[38.  1.  1. 38. 38. 38.  1.  1. 38.  1.  1. 38.  1.  1.  1. 38. 42. 42.
   1. 38. 42. 38.  1.  1.  1. 31. 38. 38. 42.  1. 31. 27.  1.  1. 38.  1.
  38. 38.  1. 38. 37. 38. 38. 27. 38. 38. 38. 44. 38. 38. 27.  1.  1. 38.
  18.  1. 38. 42. 16. 38.  9.  1.  1. 27. 27. 38. 38.  1. 38. 38.  1.  1.
  38. 38. 31. 38.  1. 38. 38. 31. 42.  9. 38. 34. 40. 38. 38.  1. 38. 38.
   1.  3. 18. 27. 38. 18.  1. 31. 42. 38.]]
```

2. print(result['detection_boxes'])
   shape [N, 4] containing bounding box coordinates in the following order: [ymin, xmin, ymax, xmax].

```
[[[0.26432082 0.20752867 0.31309542 0.2260658 ]
  [0.77713764 0.1609514  0.94884294 0.20045316]
  [0.67923814 0.08259597 0.84991753 0.12335501]
  [0.37808904 0.34561563 0.40276065 0.35999358]
  [0.43852508 0.80208844 0.47138757 0.8146835 ]
  [0.42254236 0.56203616 0.43731672 0.5697473 ]
  [0.6016265  0.13113071 0.6359133  0.14298652]
  [0.56748664 0.02776559 0.6233568  0.04252449]
  [0.38130382 0.4262706  0.41372663 0.44455338]
  [0.54203176 0.25657332 0.56274015 0.26342946]
  [0.51386476 0.51544577 0.5242671  0.52009803]
```

3. print(result[detection_keypoint_scores])
   containing the scores of the detected keypoints

```
[[[0.      0.      0.      ... 0.      0.      0.     ]
  [0.1     0.1     0.1     ... 0.68494534 0.65866584 0.72573555]
  [0.1     0.1     0.1     ... 0.54757327 0.5877377  0.53938365]
  ...
  [0.      0.      0.      ... 0.      0.      0.     ]
  [0.      0.      0.      ... 0.      0.      0.     ]
  [0.      0.      0.      ... 0.      0.      0.     ]]]
```

4. print(result[detection_keypoints])
   containing the detected keypoints

```
[[[[0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]
   ...
   [0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]]

  [[0.79360217 0.1743773 ]
   [0.7911957  0.17398714]
   [0.78952396 0.17618614]
   ...
   [0.8975664  0.1891281 ]
   [0.93033284 0.17792487]
   [0.93803436 0.19330634]]

  [[0.6959763  0.10606581]
   [0.6925977  0.10491037]
   [0.6916311  0.10701141]
   ...
   [0.7957555  0.10822457]
   [0.83013135 0.1033422 ]
   [0.83323944 0.10772318]]

  ...

  [[0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]
   ...
   [0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]]

  [[0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]
   ...
   [0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]]

  [[0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]
   ...
   [0.        0.       ]
   [0.        0.       ]
   [0.        0.       ]]]]
```

5. print(result[num_detections])
   the number of detections [N]
   [100.]

6. print(result[detection_scores])
   shape [N] containing detection scores

```
[[0.7874129  0.75997096 0.71207833 0.70352083 0.6951335  0.6714657
  0.65866023 0.65863436 0.64609224 0.592553   0.57132316 0.5659301
  0.53217715 0.4980022  0.48648158 0.46749735 0.44454393 0.43180284
  0.42613152 0.41494724 0.34783825 0.33208862 0.29186383 0.27491355
  0.22426191 0.2154774  0.19558646 0.1693214  0.14862265 0.14858493
  0.12984638 0.12905034 0.12587646 0.11661585 0.11654806 0.10967631
  0.10795615 0.10436504 0.10283968 0.10018196 0.09955008 0.09835976
  0.09754375 0.0945785  0.09373897 0.09105832 0.08917156 0.08771159
  0.08680478 0.08467623 0.08258423 0.08221053 0.07959869 0.07888587
  0.07827885 0.0763692  0.07599562 0.07498261 0.07431222 0.07423439
  0.07394202 0.07364845 0.07344545 0.0724974  0.07139292 0.0703356
  0.06988402 0.06961608 0.06830767 0.06818408 0.06608088 0.06569937
  0.06430262 0.06413528 0.06399935 0.06367771 0.06236419 0.06056397
  0.05969112 0.05917878 0.05894096 0.05811541 0.05703297 0.05675089
  0.05667497 0.05463363 0.05457538 0.05366581 0.05360183 0.0535081
  0.0520446  0.05122811 0.04971898 0.04889282 0.04796573 0.0476618
  0.04746943 0.04737025 0.04684095 0.0467912 ]]
```

---------- next page ----------

Tutorial 5: [High-performance image generation using Stable Diffusion in KerasCV | TensorFlow Core]

Goal: Generate novel images based on a text prompt using the KerasCV's Stable Diffusion.

- Text prompt is encoded into a latent vector.
- Latent vector is concatenated with a noise patch.
- Diffusion model denoises the patch over multiple steps.
- The final latent image is decoded into a high-resolution image.

Step 1: Installation and Imports.

```
import time
import keras_cv
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 475.2/475.2 MB 2.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 803.1/803.1 kB 35.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 80.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 442.0/442.0 kB 32.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 58.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 950.8/950.8 kB 43.6 MB/s eta 0:00:00
```

Step 2: Model Construction

```
model = keras_cv.models.StableDiffusion(img_width=512, img_height=512)
```

Step 3: Giving the model a text prompt

```
images = model.text_to_image("photograph of an astronaut riding a horse", batch_size=3)
def plot_images(images):
    plt.figure(figsize=(20, 20))
    for i in range(len(images)):
        ax = plt.subplot(1, len(images), i + 1)
        plt.imshow(images[i])
        plt.axis("off")
plot_images(images)
```

Output: (Using colab system RAM)

```
Downloading data from
https://github.com/openai/CLIP/blob/main/clip/bpe_simple_vocab_16e6.txt.gz?raw=true
1356917/1356917 [==============================] - 0s 0us/step
Downloading data from
https://huggingface.co/fchollet/stable-diffusion/resolve/main/kcv_encoder.h5
492466864/492466864 [==============================] - 6s 0us/step
Downloading data from
https://huggingface.co/fchollet/stable-diffusion/resolve/main/kcv_diffusion_model.h5
3439090152/3439090152 [==============================] - 36s 0us/step
50/50 [==============================] - 7137s 133s/step
Downloading data from
https://huggingface.co/fchollet/stable-diffusion/resolve/main/kcv_decoder.h5
198180272/198180272 [==============================] - 3s 0us/step
```



```
50/50 [==============================] - 6531s 130s/step
```



Steps inside the model:

1. Specified text prompt gets projected into a latent vector space by the text encoder, which is simply a pretrained, frozen language model.
2. The prompt vector is concatenated to a randomly generated noise patch, which is repeatedly "denoised" by the diffusion model over a series of "steps" (like 50 steps).
3. The 64x64 latent image is sent through the decoder to properly render it in high resolution.

Benchmarking

1. Unoptimized model:

```
benchmark_result = []
start = time.time()
images = model.text_to_image(
        "A cute otter in a rainbow whirlpool holding shells, watercolor",
        batch_size=3,)
end = time.time()
benchmark_result.append(["Standard", end - start])
plot_images(images)
print(f"Standard model: {(end - start):.2f} seconds")
keras.backend.clear_session()  # Clear session to preserve memory.
```

2. Mixed precision model:

```
start = time.time()
images = model.text_to_image(
        "a cute magical flying dog, fantasy art, "
        "golden color, high quality, highly detailed, elegant, sharp focus, "
        "concept art, character concepts, digital painting, mystery, adventure",
        batch_size=3,)
end = time.time()
benchmark_result.append(["Mixed Precision", end - start])
plot_images(images)
```

3. XLA Compilation

```
start = time.time()
images = model.text_to_image(
        "A cute otter in a rainbow whirlpool holding shells, watercolor",
        batch_size=3,)
end = time.time()
benchmark_result.append(["XLA", end - start])
plot_images(images)
print(f"With XLA: {(end - start):.2f} seconds")
```

4. Combined model

```
keras.mixed_precision.set_global_policy("mixed_float16")
model = keras_cv.models.StableDiffusion(jit_compile=True)
 images = model.text_to_image(
            "Teddy bears conducting machine learning research",
            batch_size=3,)
plot_images(images)

start = time.time()
images = model.text_to_image(
            "A mysterious dark stranger visits the great pyramids of egypt, "
            "high quality, highly detailed, elegant, sharp focus, "
            "concept art, character concepts, digital painting",
            batch_size=3,)
end = time.time()

benchmark_result.append(["XLA + Mixed Precision", end - start])
plot_images(images)
print(f"XLA + mixed precision: {(end - start):.2f} seconds")
```

Output:

Using CPU:

Model	Runtime	Output
Unoptimized model	6553.60 seconds	
Mixed precision model	6539.49 seconds	
XLA Compilation	6488.01 seconds	
Combined model	6147.62 seconds	

Using V100 GPU:

Model	Runtime	Output
Unoptimized model	2231.19 seconds	
Mixed precision model	2219.49 seconds	
XLA Compilation	2166.54 seconds	
Combined model	1142.30 seconds