Practical 6 :

Commands + Code + Text Run all Connect

```
# Import all the necessary modules
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
import cv2  # OpenCV for drawing boxes

# Check for GPU availability (Crucial for speed!)
print("TensorFlow version:", tf.__version__)
device_name = tf.test.gpu_device_name()
if device_name != '/device/GPU:0':
  print('WARNING: GPU device not found. This will be much slower.')
else:
  print('Found GPU at: {}'.format(device_name))
```

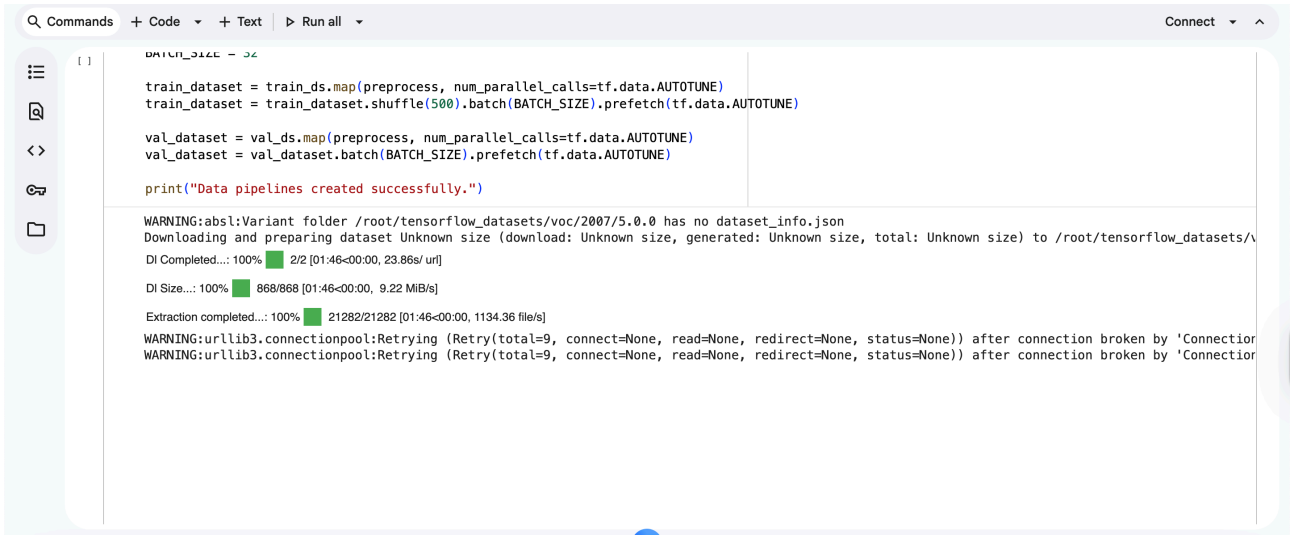
```
TensorFlow version: 2.19.0
WARNING: GPU device not found. This will be much slower.
```

Commands + Code + Text Run all Connect

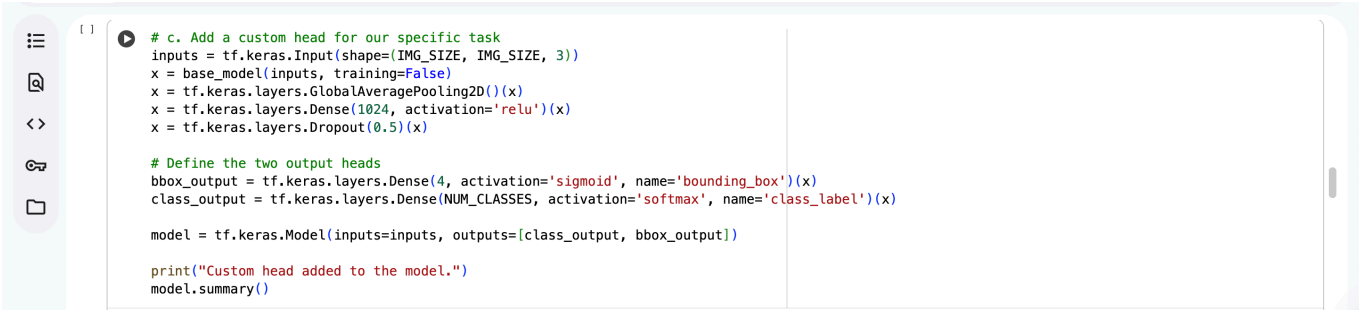
```
# --- 1. Load the Dataset ---
(train_ds_full, val_ds_full), ds_info = tfds.load(
    'voc/2007',
    split=['train', 'validation'],
    with_info=True,
    shuffle_files=True
)

# --- 2. IMPORTANT: Take a small subset to run faster ---
NUM_TRAIN_SAMPLES = 800
NUM_VAL_SAMPLES = 200

train_ds = train_ds_full.take(NUM_TRAIN_SAMPLES)
val_ds = val_ds_full.take(NUM_VAL_SAMPLES)

print(f"Using {NUM_TRAIN_SAMPLES} training samples and {NUM_VAL_SAMPLES} validation samples.")

# --- 3. Define Preprocessing ---
IMG_SIZE = 224
NUM_CLASSES = ds_info.features['objects']['label'].num_classes

def preprocess(sample):
    """Resizes image, normalizes bounding box, and one-hot encodes the label."""
    image = sample['image']
    objects = sample['objects']

    original_h = tf.cast(tf.shape(image)[0], tf.float32)
```

Commands + Code + Text Run all Connect

```
    original_h = tf.cast(tf.shape(image)[0], tf.float32)
    original_w = tf.cast(tf.shape(image)[1], tf.float32)

    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = tf.cast(image, tf.float32) / 255.0

    # Use the first object in the image for this simple example
    bbox = objects['bbox'][0]
    label = objects['label'][0]

    # Normalize bounding box coords to [x_min, y_min, width, height]
    ymin, xmin, ymax, xmax = bbox[0], bbox[1], bbox[2], bbox[3]
    normalized_bbox = [xmin / original_w, ymin / original_h, (xmax - xmin) / original_w, (ymax - ymin) / original_h]

    one_hot_label = tf.one_hot(label, depth=NUM_CLASSES)

    return image, {'class_label': one_hot_label, 'bounding_box': normalized_bbox}

# --- 4. Create Data Pipelines ---
BATCH_SIZE = 32

train_dataset = train_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(500).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

val_dataset = val_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

print("Data pipelines created successfully.")
```


Commands
+ Code
+ Text
Run all
Connect
BATCH_SIZE = 32
train_dataset = train_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(500).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
val_dataset = val_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
print("Data pipelines created successfully.")
WARNING:absl:Variant folder /root/tensorflow_datasets/voc/2007/5.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensorflow_datasets/v
Dl Completed...: 100% 2/2 [01:46<00:00, 23.86s/ url]
Dl Size...: 100% 868/868 [01:46<00:00, 9.22 MiB/s]
Extraction completed...: 100% 21282/21282 [01:46<00:00, 1134.36 file/s]
WARNING:urllib3.connectionpool:Retrying (Retry(total=9, connect=None, read=None, redirect=None, status=None)) after connection broken by 'Connectior
WARNING:urllib3.connectionpool:Retrying (Retry(total=9, connect=None, read=None, redirect=None, status=None)) after connection broken by 'Connectior


Dataset voc downloaded and prepared to /root/tensorflow_datasets/voc/2007/5.0.0. Subsequent calls will reuse this data.
Using 800 training samples and 200 validation samples.
Data pipelines created successfully.
# a. Load the pre-trained model (MobileNetV2 is lightweight and fast)
# b. Freeze its layers
base_model = tf.keras.applications.MobileNetV2(
input_shape=(IMG_SIZE, IMG_SIZE, 3),
include_top=False,
weights='imagenet'
)
base_model.trainable = False
print("Base model loaded and frozen.")
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_
9406464/9406464 1s 0us/step
Base model loaded and frozen.


# c. Add a custom head for our specific task
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
# Define the two output heads
bbox_output = tf.keras.layers.Dense(4, activation='sigmoid', name='bounding_box')(x)
class_output = tf.keras.layers.Dense(NUM_CLASSES, activation='softmax', name='class_label')(x)
model = tf.keras.Model(inputs=inputs, outputs=[class_output, bbox_output])
print("Custom head added to the model.")
model.summary()

```
model.summary()
```

Custom head added to the model.
Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-
mobilenetv2_1.00_2… (Functional)	(None, 7, 7, 1280)	2,257,984	input_layer_1[0]…
global_average_poo… (GlobalAveragePool…	(None, 1280)	0	mobilenetv2_1.00…
dense (Dense)	(None, 1024)	1,311,744	global_average_p…
dropout (Dropout)	(None, 1024)	0	dense[0][0]
class_label (Dense)	(None, 20)	20,500	dropout[0][0]
bounding_box (Dense)	(None, 4)	4,100	dropout[0][0]

Total params: 3,594,328 (13.71 MB)
Trainable params: 1,336,344 (5.10 MB)
Non-trainable params: 2,257,984 (8.61 MB)

```
# d. Train the new layers
losses = {
    "class_label": tf.keras.losses.CategoricalCrossentropy(),
    "bounding_box": tf.keras.losses.MeanSquaredError()
}

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss=losses,
    metrics={"class_label": "accuracy"}
)

print("\n--- Training the head ---")
# Reduced epochs for a faster experiment
initial_epochs = 5
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=initial_epochs
)
```

```
    epochs=initial_epochs
)
```

```
--- Training the head ---
Epoch 1/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 51s 2s/step - bounding_box_loss: 0.0819 - class_label_accuracy: 0.2668 - class_label_loss: 2.8625 - loss: 2.9443 - val_bo
Epoch 2/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 2s/step - bounding_box_loss: 8.4799e-04 - class_label_accuracy: 0.7032 - class_label_loss: 0.9974 - loss: 0.9982 - va
Epoch 3/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 2s/step - bounding_box_loss: 3.2473e-04 - class_label_accuracy: 0.8030 - class_label_loss: 0.5712 - loss: 0.5715 - va
Epoch 4/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - bounding_box_loss: 2.1146e-04 - class_label_accuracy: 0.8681 - class_label_loss: 0.4050 - loss: 0.4052 - va
Epoch 5/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - bounding_box_loss: 2.1524e-04 - class_label_accuracy: 0.9415 - class_label_loss: 0.2360 - loss: 0.2362 - va
```

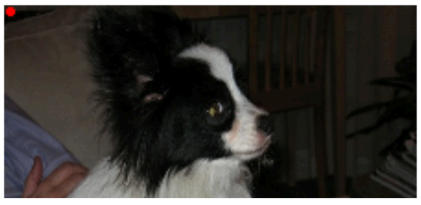
```
# e. Fine-tune the model
base_model.trainable = True

# Let's unfreeze from the 100th layer onwards
for layer in base_model.layers[:100]:
    layer.trainable = False

# Re-compile with a very low learning rate for fine-tuning
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss=losses,
    metrics={"class_label": "accuracy"}
)

print("\n--- Starting Fine-Tuning ---")
# Reduced epochs for a faster experiment
fine_tune_epochs = 5
total_epochs = initial_epochs + fine_tune_epochs

# *** THIS IS THE CORRECTED LINE ***
history_fine_tune = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=total_epochs,
    initial_epoch=initial_epochs # Correctly continue from the end of the last training
)
```

```
    initial_epoch=initial_epochs # Correctly continue from the end of the last training
)
```

```
--- Starting Fine-Tuning ---
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 80s 3s/step - bounding_box_loss: 9.4575e-05 - class_label_accuracy: 0.8264 - class_label_loss: 0.5773 - loss: 0.5774 - va
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - bounding_box_loss: 9.0003e-05 - class_label_accuracy: 0.8683 - class_label_loss: 0.4425 - loss: 0.4426 - va
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - bounding_box_loss: 1.0105e-04 - class_label_accuracy: 0.9008 - class_label_loss: 0.3427 - loss: 0.3428 - va
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 61s 2s/step - bounding_box_loss: 7.4575e-05 - class_label_accuracy: 0.9125 - class_label_loss: 0.3070 - loss: 0.3071 - va
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - bounding_box_loss: 1.1746e-04 - class_label_accuracy: 0.9414 - class_label_loss: 0.2703 - loss: 0.2705 - va
```

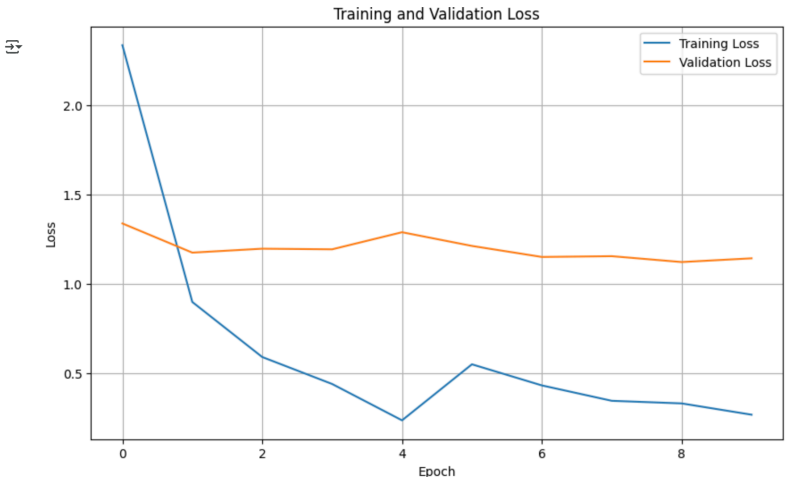
```
# --- Visualize Predictions ---

def predict_and_visualize(dataset, num_samples=5):
    class_names = ds_info.features['objects']['label'].names

    for images, labels in dataset.take(1):
        predictions = model.predict(images)
        pred_labels, pred_boxes = predictions[0], predictions[1]

        for i in range(num_samples):
            image = (images[i].numpy() * 255).astype(np.uint8)
            true_box = labels['bounding_box'][i].numpy()

            pred_label_idx = np.argmax(pred_labels[i])
            pred_class = class_names[pred_label_idx]
            pred_confidence = np.max(pred_labels[i])
            pred_box = pred_boxes[i]

            # Draw Predicted Box (Red)
            img_h, img_w, _ = image.shape
            x_p, y_p, w_p, h_p = pred_box
            xp, yp = int(x_p * img_w), int(y_p * img_h)
            wp, hp = int(w_p * img_w), int(h_p * img_h)

            # Create a copy to avoid drawing over the original
            vis_image = image.copy()
            cv2.rectangle(vis_image, (xp, yp), (xp + wp, yp + hp), (255, 0, 0), 2)
            pred_text = f"Pred: {pred_class} ({pred_confidence:.2f})"
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```



```
# --- Plot the training and validation loss ---

# Get the loss history from the initial training
acc = history.history['loss']
val_acc = history.history['val_loss']

# Get the loss history from the fine-tuning phase and append it
acc += history_fine_tune.history['loss']
val_acc += history_fine_tune.history['val_loss']

# Get the total number of epochs
epochs_range = range(total_epochs)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(epochs_range, acc, label='Training Loss')
plt.plot(epochs_range, val_acc, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
```

Variables Terminal



Variables Terminal