Reporte de Taller 4

Instalación de TENSORFLOW

```
!pip install tensorflow-gpu==2.0.0
!pip install tensorflow_hub
```

```
Collecting tensorflow-gpu==2.0.0
  Downloading tensorflow_gpu-2.0.0-cp37-cp37m-manylinux2010_x86_64.whl (380.8 MB)
     |████████████████████████████████| 380.8 MB 43 kB/s
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (3.3.0)
Collecting tensorboard<2.1.0,>=2.0.0
  Downloading tensorboard-2.0.2-py3-none-any.whl (3.8 MB)
     |████████████████████████████████| 3.8 MB 34.5 MB/s
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.15.0)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (3.17.3)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (0.37.0)
Collecting gast==0.2.2
  Downloading gast-0.2.2.tar.gz (10 kB)
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (0.12.0)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.41.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.1.2)
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (0.2.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.19.5)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (0.8.1)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.12.1)
Collecting tensorflow-estimator<2.1.0,>=2.0.0
  Downloading tensorflow_estimator-2.0.1-py2.py3-none-any.whl (449 kB)
     |████████████████████████████████| 449 kB 31.4 MB/s
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu==2.0.0) (1.1.0)
Collecting keras-applications>=1.0.8
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 6.5 MB/s
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from keras-applications>=1.0.8->tensorflow-gpu==2.0.0) (3.1.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (1.35.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (1.0.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (0.4.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (2.23.0)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (57.4.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (3.3.4)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (4.7.2)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (4.2.4)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (1.3.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (4.8.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<2,>=1.6.3->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (0.4.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (2021.5.30)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (3.0.4)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (3.1.1)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py->keras-applications>=1.0.8->tensorflow-gpu==2.0.0) (1.5.2)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->markdown>=2.6.8->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (3.7.4.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->markdown>=2.6.8->tensorboard<2.1.0,>=2.0.0->tensorflow-gpu==2.0.0) (3.6.0)
Building wheels for collected packages: gast
  Building wheel for gast (setup.py) ... done
  Created wheel for gast: filename=gast-0.2.2-py3-none-any.whl size=7554 sha256=97d663f67150232bf147416ab3a34991722dc0e4e2ed51c535704b9684f6bd12
  Stored in directory: /root/.cache/pip/wheels/21/7f/02/420f32a803f7d0967b48dd823da3f558c5166991bfd204eef3
Successfully built gast
Installing collected packages: tensorflow-estimator, tensorboard, keras-applications, gast, tensorflow-gpu
  Attempting uninstall: tensorflow-estimator
    Found existing installation: tensorflow-estimator 2.6.0
    Uninstalling tensorflow-estimator-2.6.0:
      Successfully uninstalled tensorflow-estimator-2.6.0
```

Instalación de las librerías que necesita Tensorflow

```python
from __future__ import absolute_import, division, print_function, unicode_literals

import matplotlib.pylab as plt
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
```

Instalación de pandas que es una biblioteca para una mejor visualización

```python
import pandas as pd

# Increase precision of presented data for better side-by-side comparison
pd.set_option("display.precision", 8)
```

```python
print("Version: ", tf.__version__)
print("Hub version: ", hub.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

```
Version:  2.0.0
Hub version:  0.12.0
Eager mode:  True
GPU is available
```

Cargar un paquete de imágenes de referencia o enlace

```python
data_root = tf.keras.utils.get_file(
  'flower_photos','https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
   untar=True)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228818944/228813984 [==============================] - 3s 0us/step
```

Se procede a cambia la cantidad de pixeles de las imágenes,
Se indica en donde se va a llevar acabo el procedimiento de entrenamiento
Se pasa la imagen a blanco y negro
Se re escala dividendo la un 20% para entrenar y validar
Se crean los generadores de validación y entrenamiento

```python
# Create data generator for training and validation

IMAGE_SHAPE = (224, 224)
TRAINING_DATA_DIR = str(data_root)

datagen_kwargs = dict(rescale=1./255, validation_split=.20)
valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwargs)
valid_generator = valid_datagen.flow_from_directory(
    TRAINING_DATA_DIR,
    subset="validation",
    shuffle=True,
    target_size=IMAGE_SHAPE
)

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwargs)
train_generator = train_datagen.flow_from_directory(
    TRAINING_DATA_DIR,
    subset="training",
    shuffle=True,
    target_size=IMAGE_SHAPE)
```

```
Found 731 images belonging to 5 classes.
Found 2939 images belonging to 5 classes.
```

Se presenta la informacion que contiene el generador de entrenamiento .
Indicando que existen 32 imagenes de 224X224, el numero 3 que esta al final nos indica los tres colores primarios usados RGB
La etiquete o laber1, indica que existe 32 etiquetas y cinco clases

```python
# Learn more about data batches

image_batch_train, label_batch_train = next(iter(train_generator))
print("Image batch shape: ", image_batch_train.shape)
print("Label batch shape: ", label_batch_train.shape)
```

```
Image batch shape:  (32, 224, 224, 3)
Label batch shape:  (32, 5)
```

Se proceden a ordenar las etiquetas en orden alfabetico

```python
# Learn about dataset labels

dataset_labels = sorted(train_generator.class_indices.items(), key=lambda pair:pair[1])
dataset_labels = np.array([key.title() for key, value in dataset_labels])
print(dataset_labels)
```

```
['Daisy' 'Dandelion' 'Roses' 'Sunflowers' 'Tulips']
```

Se procede a cargar el modelo pre entrenado, con determinadas características, dicho proceso funciona aun si se tiene un numero diferentes de clases

```
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/4",
                 output_shape=[1280],
                 trainable=False),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
model.build([None, 224, 224, 3])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     multiple                  2257984
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

cuando se genera el modelo , se debe compilar. Utilizando del algoritmo de ADAM para optimizacion

```
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss='categorical_crossentropy',
  metrics=['acc'])
```

Para el entrenamiento se usa el algoritmo con 10 epoch o ciclos de entrenamiento, entre mas ciclos de entrenamiento tenga , sera mas eficiente en su búsqueda pero va a tardar mas

```python
# Run model training

steps_per_epoch = np.ceil(train_generator.samples/train_generator.batch_size)
val_steps_per_epoch = np.ceil(valid_generator.samples/valid_generator.batch_size)

hist = model.fit(
    train_generator,
    epochs=10,
    verbose=1,
    steps_per_epoch=steps_per_epoch,
    validation_data=valid_generator,
    validation_steps=val_steps_per_epoch).history
```

```
Train for 92.0 steps, validate for 23.0 steps
Epoch 1/10
92/92 [==============================] - 31s 341ms/step - loss: 0.8961 - acc: 0.6679 - val_loss: 0.4984 - val_acc: 0.8167
Epoch 2/10
92/92 [==============================] - 20s 219ms/step - loss: 0.4682 - acc: 0.8261 - val_loss: 0.4088 - val_acc: 0.8550
Epoch 3/10
92/92 [==============================] - 20s 213ms/step - loss: 0.3717 - acc: 0.8649 - val_loss: 0.3730 - val_acc: 0.8700
Epoch 4/10
92/92 [==============================] - 20s 216ms/step - loss: 0.3194 - acc: 0.8857 - val_loss: 0.3582 - val_acc: 0.8714
Epoch 5/10
92/92 [==============================] - 20s 217ms/step - loss: 0.2919 - acc: 0.8986 - val_loss: 0.3404 - val_acc: 0.8769
Epoch 6/10
92/92 [==============================] - 20s 216ms/step - loss: 0.2804 - acc: 0.9030 - val_loss: 0.3401 - val_acc: 0.8810
Epoch 7/10
92/92 [==============================] - 20s 219ms/step - loss: 0.2588 - acc: 0.9088 - val_loss: 0.3335 - val_acc: 0.8810
Epoch 8/10
92/92 [==============================] - 20s 218ms/step - loss: 0.2436 - acc: 0.9153 - val_loss: 0.3320 - val_acc: 0.8782
Epoch 9/10
92/92 [==============================] - 20s 219ms/step - loss: 0.2218 - acc: 0.9224 - val_loss: 0.3343 - val_acc: 0.8810
Epoch 10/10
92/92 [==============================] - 20s 219ms/step - loss: 0.2150 - acc: 0.9319 - val_loss: 0.3296 - val_acc: 0.8851
```

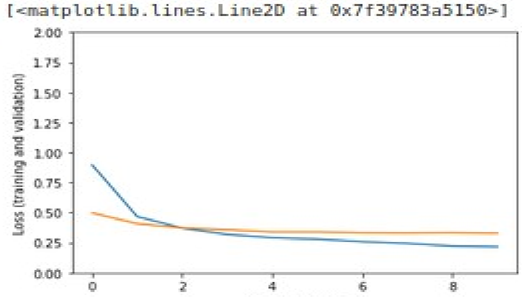
```python
# Measure accuracy and loss after training

final_loss, final_accuracy = model.evaluate(valid_generator, steps = val_steps_per_epoch)
```

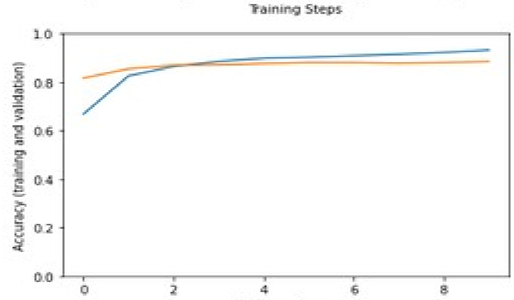
```
23/23 [==============================] - 5s 203ms/step - loss: 0.3296 - acc: 0.8851
```

Después del entrenamiento se grafica o plotea los datos de perdidas y precisión para el entrenamiento y la validación

```python
# Visualize training process

plt.figure()
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(hist["loss"])
plt.plot(hist["val_loss"])

plt.figure()
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(hist["acc"])
plt.plot(hist["val_acc"])
```

```
[<matplotlib.lines.Line2D at 0x7f39783a5150>]
```





Exporta tu modelo

Guardaremos nuestro modelo como formato de modelo guardado de TensorFlow. Después de eso, haremos inferencia en el modelo recargado, por lo que si viene con un modelo ya entrenado, será más fácil inspeccionarlo.

```
FLOWERS_SAVED_MODEL = "saved_models/flowers3"
tf.saved_model.save(model, FLOWERS_SAVED_MODEL)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1781: calling BaseResourceVariable.__init__ (from tens
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1781: calling BaseResourceVariable.__init__ (from tens
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: saved_models/flowers3/assets
INFO:tensorflow:Assets written to: saved_models/flowers3/assets
```

Cargar modelo guardado de TensorFlow

Carguemos el modelo de TensorFlow desde el formato SavedModel. Debido a que usamos la capa personalizada de TensorFlow Hub, debemos señalar explícitamente la implementación con el parámetro `custom_obiects`.

```
# Load SavedModel

flowers_model = hub.load(FLOWERS_SAVED_MODEL)
print(flowers_model)
```

```
<tensorflow.python.saved_model.load.Loader._recreate_base_user_object.<locals>._UserObject object at 0x7f3970516850>
```

Se procede a consultar las predicciones en el modelo cargado

```
# Get images and labels batch from validation dataset generator

val_image_batch, val_label_batch = next(iter(valid_generator))
true_label_ids = np.argmax(val_label_batch, axis=-1)

print("Validation batch shape:", val_image_batch.shape)
```

```
Validation batch shape: (32, 224, 224, 3)
```

La forma del lote de validación nos dice que tenemos un lote de 32 imágenes, con tamaño y canales: 224x224x3.
Calculemos predicciones para todo el lote.

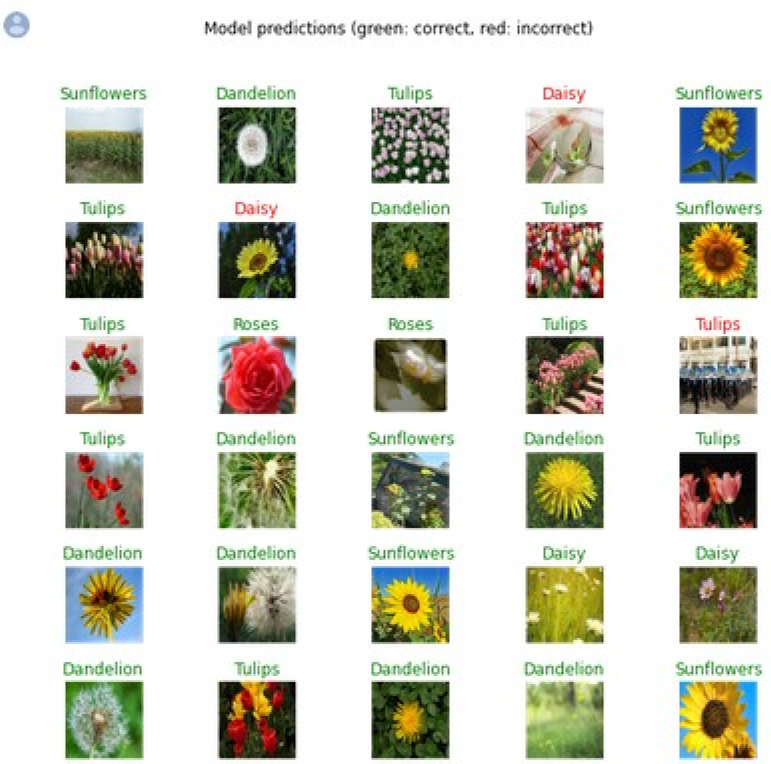
```
tf_model_predictions = flowers_model(val_image_batch)
print("Prediction results shape:", tf_model_predictions.shape)
```

```
Prediction results shape: (32, 5)
```

```
# Convert prediction results to Pandas dataframe, for better visualization

tf_pred_dataframe = pd.DataFrame(tf_model_predictions.numpy())
tf_pred_dataframe.columns = dataset_labels

print("Prediction results for the first elements")
```

```
[21] predicted_ids = np.argmax(tf_model_predictions, axis=-1)
     predicted_labels = dataset_labels[predicted_ids]
```

```
# Print images batch and labels predictions

plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(val_image_batch[n])
  color = "green" if predicted_ids[n] == true_label_ids[n] else "red"
  plt.title(predicted_labels[n].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("Model predictions (green: correct, red: incorrect)")
```



```
[24] !mkdir "tflite_models"
```

```
[25] TFLITE_MODEL = "tflite_models/flowers.tflite"
     TFLITE_QUANT_MODEL = "tflite_models/flowers_quant.tflite"
```

```
# Get the concrete function from the Keras model.
run_model = tf.function(lambda x : flowers_model(x))

# Save the concrete function.
concrete_func = run_model.get_concrete_function(
    tf.TensorSpec(model.inputs[0].shape, model.inputs[0].dtype)
)

# Convert the model
converter = tf.lite.TFLiteConverter.from_concrete_functions([concrete_func])
converted_tflite_model = converter.convert()
open(TFLITE_MODEL, "wb").write(converted_tflite_model)

# Convert the model to quantized version with post-training quantization
converter = tf.lite.TFLiteConverter.from_concrete_functions([concrete_func])
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_quant_model = converter.convert()
open(TFLITE_QUANT_MODEL, "wb").write(tflite_quant_model)

print("TFLite models and their sizes:")
!ls "tflite_models" -lh
```

```
TFLite models and their sizes:
total 11M
-rw-r--r-- 1 root root 2.3M Oct 17 01:54 flowers_quant.tflite
-rw-r--r-- 1 root root 8.5M Oct 17 01:54 flowers.tflite
```

Convertir modelo a TFLite

Convierta el modelo cargado recientemente en modelos de TensorFlow Lite (estándar y cuantificado con una [cuantificación posterior al entrenamiento] (https://www.tensorflow.org/lite/performance/post_training_quantization)).

Debido a la naturaleza de TensorFlow 2.0, necesitaremos convertir el modelo de TensorFlow en una función concreta y luego realizar la conversión a TFLite. Más información la obtenemos en el siguiente enlace (https://www.tensorflow.org/lite/r2/convert/concrete_function).

Cargar modelo TFLite

Cargue el modelo TensorFlow lite con interfaz de intérprete.

Load TFLite model

Load TensorFlow lite model with interpreter interface.

```
# Load TFLite model and see some details about input/output

tflite_interpreter = tf.lite.Interpreter(model_path=TFLITE_MODEL)

input_details = tflite_interpreter.get_input_details()
output_details = tflite_interpreter.get_output_details()

print("== Input details ==")
print("name:", input_details[0]['name'])
print("shape:", input_details[0]['shape'])
print("type:", input_details[0]['dtype'])

print("\n== Output details ==")
print("name:", output_details[0]['name'])
print("shape:", output_details[0]['shape'])
print("type:", output_details[0]['dtype'])
```

```
== Input details ==
name: x
shape: [  1 224 224   3]
type: <class 'numpy.float32'>

== Output details ==
name: Identity
shape: [1 5]
type: <class 'numpy.float32'>
```

Cambiar el tamaño de las formas de los tensores de entrada y salida

La forma de entrada del modelo TFLite cargado es 1x224x224x3, lo que significa que podemos hacer predicciones para una sola imagen.

Cambiemos el tamaño de los tensores de entrada y salida, para que podamos hacer predicciones para lotes de 32 imágenes.

[28]

```python
tflite_interpreter.resize_tensor_input(input_details[0]['index'], (32, 224, 224, 3))
tflite_interpreter.resize_tensor_input(output_details[0]['index'], (32, 5))
tflite_interpreter.allocate_tensors()

input_details = tflite_interpreter.get_input_details()
output_details = tflite_interpreter.get_output_details()

print("== Input details ==")
print("name:", input_details[0]['name'])
print("shape:", input_details[0]['shape'])
print("type:", input_details[0]['dtype'])

print("\n== Output details ==")
print("name:", output_details[0]['name'])
print("shape:", output_details[0]['shape'])
print("type:", output_details[0]['dtype'])
```

```
== Input details ==
name: x
shape: [ 32 224 224   3]
type: <class 'numpy.float32'>

== Output details ==
name: Identity
shape: [32  5]
type: <class 'numpy.float32'>
```

[29]

```python
tflite_interpreter.set_tensor(input_details[0]['index'], val_image_batch)

tflite_interpreter.invoke()

tflite_model_predictions = tflite_interpreter.get_tensor(output_details[0]['index'])
print("Prediction results shape:", tflite_model_predictions.shape)
```

```
Prediction results shape: (32, 5)
```

```python
# Convert prediction results to Pandas dataframe, for better visualization

tflite_pred_dataframe = pd.DataFrame(tflite_model_predictions)
tflite_pred_dataframe.columns = dataset_labels

print("TFLite prediction results for the first elements")
tflite_pred_dataframe.head()
```

TFLite prediction results for the first elements

	Daisy	Dandelion	Roses	Sunflowers	Tulips
0	0.00000239	0.99999690	0.00000018	0.00000028	0.00000019
1	0.00005266	0.99991584	0.00000355	0.00000981	0.00001816
2	0.00596758	0.00023825	0.24203724	0.00325616	0.74850076
3	0.97424465	0.00677155	0.00073572	0.01760287	0.00064516
4	0.90023369	0.06073571	0.00863129	0.01207192	0.01832741

Ahora hagamos lo mismo para el modelo cuantificado TFLite:
- Modelo de carga,
- Cambiar la forma de la entrada para manejar lotes de imágenes,
- Ejecutar predicción

[31]
```
# Load quantized TFLite model
tflite_interpreter_quant = tf.lite.Interpreter(model_path=TFLITE_QUANT_MODEL)

# Learn about its input and output details
input_details = tflite_interpreter_quant.get_input_details()
output_details = tflite_interpreter_quant.get_output_details()

# Resize input and output tensors to handle batch of 32 images
tflite_interpreter_quant.resize_tensor_input(input_details[0]['index'], (32, 224, 224, 3))
tflite_interpreter_quant.resize_tensor_input(output_details[0]['index'], (32, 5))
tflite_interpreter_quant.allocate_tensors()

input_details = tflite_interpreter_quant.get_input_details()
output_details = tflite_interpreter_quant.get_output_details()

print("== Input details ==")
print("name:", input_details[0]['name'])
print("shape:", input_details[0]['shape'])
print("type:", input_details[0]['dtype'])

print("\n== Output details ==")
print("name:", output_details[0]['name'])
print("shape:", output_details[0]['shape'])
print("type:", output_details[0]['dtype'])

# Run inference
tflite_interpreter_quant.set_tensor(input_details[0]['index'], val_image_batch)

tflite_interpreter_quant.invoke()

tflite_q_model_predictions = tflite_interpreter_quant.get_tensor(output_details[0]['index'])
print("\nPrediction results shape:", tflite_q_model_predictions.shape)
```

```
== Input details ==
name: x
shape: [ 32 224 224   3]
type: <class 'numpy.float32'>

== Output details ==
name: Identity
shape: [32  5]
type: <class 'numpy.float32'>
```

```
# Convert prediction results to Pandas dataframe, for better visualization

tflite_q_pred_dataframe = pd.DataFrame(tflite_q_model_predictions)
tflite_q_pred_dataframe.columns = dataset_labels

print("Quantized TFLite model prediction results for the first elements")
tflite_q_pred_dataframe.head()
```

Quantized TFLite model prediction results for the first elements

	Daisy	Dandelion	Roses	Sunflowers	Tulips
0	0.00000249	0.99999404	0.00000057	0.00000198	0.00000082
1	0.00001438	0.99998367	0.00000010	0.00000067	0.00000103
2	0.54346687	0.04247782	0.05645334	0.02169827	0.33590364
3	0.85414314	0.14497678	0.00000312	0.00086568	0.00001135
4	0.97871244	0.02087993	0.00001474	0.00014146	0.00025156

Compara los resultados de la predicción

Ahora usaremos Pandas para visualizar los resultados de los 3 modelos y encontrar diferencias entre ellos.

```
[33] # Concatenate results from all models

all_models_dataframe = pd.concat([tf_pred_dataframe,
                                 tflite_pred_dataframe,
                                 tflite_q_pred_dataframe],
                                keys=['TF Model', 'TFLite', 'TFLite quantized'],
                                axis='columns')
all_models_dataframe.head()
```

	TF Model					TFLite					TFLite quantized				
	Daisy	Dandelion	Roses	Sunflowers	Tulips	Daisy	Dandelion	Roses	Sunflowers	Tulips	Daisy	Dandelion	Roses	Sunflowers	Tulips
0	0.00000239	0.99999690	0.00000018	0.00000028	0.00000019	0.00000239	0.99999690	0.00000018	0.00000028	0.00000019	0.00000249	0.99999404	0.00000057	0.00000198	0.00000082
1	0.00005266	0.99991584	0.00000355	0.00000981	0.00001816	0.00005266	0.99991584	0.00000355	0.00000981	0.00001816	0.00001438	0.99998367	0.00000010	0.00000067	0.00000103
2	0.00596754	0.00023824	0.24203569	0.00325609	0.74850243	0.00596758	0.00023825	0.24203724	0.00325616	0.74850076	0.54346687	0.04247782	0.05645334	0.02169827	0.33590364
3	0.97424465	0.00677152	0.00073572	0.01760292	0.00064516	0.97424465	0.00677155	0.00073572	0.01760287	0.00064516	0.85414314	0.14497678	0.00000312	0.00086568	0.00001135
4	0.90023416	0.06073533	0.00863124	0.01207189	0.01832737	0.90023369	0.06073571	0.00863129	0.01207192	0.01832741	0.97871244	0.02087993	0.00001474	0.00014146	0.00025156

```
# Swap columns to hava side by side comparison

all_models_dataframe = all_models_dataframe.swaplevel(axis='columns')[tflite_pred_dataframe.columns]
all_models_dataframe.head()
```

	Daisy			Dandelion			Roses			Sunflowers			Tulips		
	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized
0	0.00000239	0.00000239	0.00000249	0.99999690	0.99999690	0.99999404	0.00000018	0.00000018	0.00000057	0.00000028	0.00000028	0.00000198	0.00000019	0.00000019	0.00000082
1	0.00005266	0.00005266	0.00001438	0.99991584	0.99991584	0.99998367	0.00000355	0.00000355	0.00000010	0.00000981	0.00000981	0.00000067	0.00001816	0.00001816	0.00000103
2	0.00596754	0.00596758	0.54346687	0.00023824	0.00023825	0.04247782	0.24203569	0.24203724	0.05645334	0.00325609	0.00325616	0.02169827	0.74850243	0.74850076	0.33590364
3	0.97424465	0.97424465	0.85414314	0.00677152	0.00677155	0.14497678	0.00073572	0.00073572	0.00000312	0.01760292	0.01760287	0.00086568	0.00064516	0.00064516	0.00001135
4	0.90023416	0.90023369	0.97871244	0.06073533	0.06073571	0.02087993	0.00863124	0.00863129	0.00001474	0.01207189	0.01207192	0.00014146	0.01832737	0.01832741	0.00025156

+ Code + Text

```
# Highlight TFLite models predictions that are different from original model

def highlight_diff(data, color='yellow'):
    attr = 'background-color: {}'.format(color)
    other = data.xs('TF Model', axis='columns', level=-1)
    return pd.DataFrame(np.where(data.ne(other, level=0), attr, ''),
                        index=data.index, columns=data.columns)

all_models_dataframe.style.apply(highlight_diff, axis=None)
```

	Daisy			Dandelion			Roses			Sunflowers			Tulips		
	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized
0	0.00000239	0.00000239	0.00000249	0.99999690	0.99999690	0.99999404	0.00000018	0.00000018	0.00000057	0.00000028	0.00000028	0.00000198	0.00000019	0.00000019	0.00000082
1	0.00005266	0.00005266	0.00001438	0.99991584	0.99991584	0.99998367	0.00000355	0.00000355	0.00000010	0.00000981	0.00000981	0.00000067	0.00001816	0.00001816	0.00000103
2	0.00596754	0.00596758	0.54346687	0.00023824	0.00023825	0.04247782	0.24203569	0.24203724	0.05645334	0.00325609	0.00325616	0.02169827	0.74850243	0.74850076	0.33590364
3	0.97424465	0.97424465	0.85414314	0.00677152	0.00677155	0.14497678	0.00073572	0.00073572	0.00000312	0.01760292	0.01760287	0.00086568	0.00064516	0.00064516	0.00001135
4	0.90023416	0.90023369	0.97871244	0.06073533	0.06073571	0.02087993	0.00863124	0.00863129	0.00001474	0.01207189	0.01207192	0.00014146	0.01832737	0.01832741	0.00025156
5	0.00017718	0.00017718	0.00912974	0.00000895	0.00000895	0.00014438	0.00012870	0.00012870	0.00059792	0.00002085	0.00002085	0.00047234	0.99966431	0.99966431	0.98965561
6	0.00942139	0.00942150	0.04061473	0.01718830	0.01718832	0.03465945	0.00484312	0.00484314	0.03651631	0.00193145	0.00193146	0.01651362	0.96661568	0.96661556	0.87169594
7	0.00030787	0.00030787	0.00020450	0.00001660	0.00001660	0.00000478	0.96229154	0.96229196	0.99848467	0.00007082	0.00007082	0.00000140	0.03731311	0.03731269	0.00130472
8	0.83642560	0.83642513	0.47610903	0.02318105	0.02318107	0.42972967	0.00062485	0.00062486	0.00003382	0.13434093	0.13434125	0.09390795	0.00542758	0.00542754	0.00021952
9	0.00032974	0.00032973	0.05728308	0.00001263	0.00001263	0.01719359	0.28348947	0.28349027	0.86991590	0.60503703	0.60503846	0.00726742	0.11113115	0.11112890	0.04833997
10	0.00027485	0.00027485	0.01322673	0.00002672	0.00002672	0.00177448	0.65258217	0.65258402	0.95267034	0.00000045	0.00000045	0.00000679	0.34711581	0.34711391	0.03232165
11	0.00006083	0.00006083	0.00002163	0.99990451	0.99990451	0.99996424	0.00000071	0.00000071	0.00000020	0.00003347	0.00003347	0.00001179	0.00000044	0.00000044	0.00000214
12	0.05072153	0.05072142	0.00851961	0.94440705	0.94440717	0.98921037	0.00039697	0.00039697	0.00016830	0.00307118	0.00307119	0.00014498	0.00140324	0.00140322	0.00195682
13	0.94699538	0.94699556	0.86087441	0.00042071	0.00042071	0.06192913	0.01301802	0.01301802	0.07333661	0.00291894	0.00291892	0.00102119	0.03664685	0.03664680	0.00283870
14	0.00086238	0.00086237	0.00055620	0.99908721	0.99908721	0.99931931	0.00000383	0.00000383	0.00000602	0.00003323	0.00003323	0.00011325	0.00001335	0.00001335	0.00000509
15	0.00000219	0.00000219	0.00000333	0.00000314	0.00000314	0.00001581	0.00178468	0.00178471	0.01832670	0.00000120	0.00000120	0.00000116	0.99820876	0.99820876	0.98165303
16	0.00056937	0.00056938	0.00139181	0.00000215	0.00000215	0.00000578	0.00589823	0.00589828	0.02686730	0.00161214	0.00161214	0.00059562	0.99191815	0.99191803	0.97113949
17	0.94628590	0.94628531	0.75057566	0.00120936	0.00120937	0.01356014	0.00015797	0.00015797	0.00492893	0.05182082	0.05182146	0.23012659	0.00052594	0.00052595	0.00080864
18	0.00025370	0.00025370	0.00003164	0.00001279	0.00001279	0.00000300	0.00749821	0.00749831	0.06241143	0.00022724	0.00022725	0.00001476	0.99200809	0.99200797	0.93753916
19	0.01272865	0.01272847	0.15641145	0.00001218	0.00001218	0.00539323	0.98288417	0.98288453	0.83607125	0.00009990	0.00009990	0.00016072	0.00427501	0.00427494	0.00196322
20	0.97514319	0.97514307	0.99459970	0.00360369	0.00360372	0.00259586	0.00053460	0.00053461	0.00001061	0.01908203	0.01908213	0.00277476	0.00163636	0.00163636	0.00001904
21	0.00043196	0.00043196	0.21351860	0.00127300	0.00127301	0.16375867	0.00005917	0.00005917	0.00064802	0.95042741	0.95042795	0.40986747	0.04780855	0.04780786	0.21220723
22	0.12399055	0.12399056	0.21286239	0.82691061	0.82691067	0.76372111	0.00563731	0.00563733	0.01646654	0.03043750	0.03043752	0.00144626	0.01302398	0.01302388	0.00550367
23	0.00174523	0.00174525	0.00112750	0.00055612	0.00055613	0.00231674	0.00344107	0.00344110	0.12101149	0.00029711	0.00029711	0.00115616	0.99396050	0.99396038	0.87438804
24	0.00183642	0.00183643	0.00276214	0.00133148	0.00133148	0.00528529	0.00523357	0.00523358	0.01438646	0.00682942	0.00682945	0.03324563	0.98476911	0.98476899	0.94432044
25	0.00714532	0.00714532	0.00822989	0.04239199	0.04239194	0.15844436	0.00119575	0.00119573	0.00013650	0.94753909	0.94753909	0.83315617	0.00172790	0.00172788	0.00003306
26	0.01262539	0.01262543	0.11418915	0.00594214	0.00594215	0.03597498	0.00027768	0.00027768	0.00027456	0.98112398	0.98112386	0.84955180	0.00003083	0.00003083	0.00000951
27	0.00000146	0.00000146	0.00000071	0.99999833	0.99999833	0.99999106	0.00000000	0.00000000	0.00000001	0.00000007	0.00000007	0.00000816	0.00000012	0.00000012	0.00000013
28	0.00096367	0.00096368	0.01041956	0.00942104	0.00942103	0.03319456	0.00031937	0.00031937	0.00010456	0.15884982	0.15885146	0.18122704	0.83044612	0.83044440	0.77505428
29	0.99363178	0.99363166	0.99771690	0.00362296	0.00362299	0.00194166	0.00008530	0.00008530	0.00001328	0.00191093	0.00191093	0.00029595	0.00074904	0.00074904	0.00003221
30	0.00540076	0.00540078	0.02913926	0.00006684	0.00006684	0.00001063	0.77516419	0.77516735	0.96824640	0.00280588	0.00280602	0.00002297	0.21656238	0.21655902	0.00258074
31	0.21025416	0.21025336	0.00655279	0.78624076	0.78624171	0.99284571	0.00020892	0.00020892	0.00002239	0.00203225	0.00203222	0.00051215	0.00126388	0.00126385	0.00006694

Como podemos ver, en la mayoría de los casos las predicciones son diferentes entre todos los modelos, generalmente por pequeños factores. Las predicciones de alta confianza entre los modelos de TensorFlow y TensorFlow Lite son muy cercanas entre sí (en algunos casos, incluso hay similares). El modelo cuantificado es el que más destaca, pero este es el costo de las optimizaciones (el modelo pesa entre 3 y 4 veces menos).

```
[36] # Concatenation of argmax and max value for each row
     def max_values_only(data):
       argmax_col = np.argmax(data, axis=1).reshape(-1, 1)
       max_col = np.max(data, axis=1).reshape(-1, 1)
       return np.concatenate([argmax_col, max_col], axis=1)

     # Build simplified prediction tables
     tf_model_pred_simplified = max_values_only(tf_model_predictions)
     tflite_model_pred_simplified = max_values_only(tflite_model_predictions)
     tflite_q_model_pred_simplified = max_values_only(tflite_q_model_predictions)
```

```
[37] # Build DataFrames and present example
     columns_names = ["Label_id", "Confidence"]
     tf_model_simple_dataframe = pd.DataFrame(tf_model_pred_simplified)
     tf_model_simple_dataframe.columns = columns_names

     tflite_model_simple_dataframe = pd.DataFrame(tflite_model_pred_simplified)
     tflite_model_simple_dataframe.columns = columns_names

     tflite_q_model_simple_dataframe = pd.DataFrame(tflite_q_model_pred_simplified)
     tflite_q_model_simple_dataframe.columns = columns_names

     tf_model_simple_dataframe.head()
```

	Label_id	Confidence
0	1.0	0.99999690
1	1.0	0.99991584
2	4.0	0.74850243
3	0.0	0.97424465
4	0.0	0.90023416

```python
# Concatenate results from all models
all_models_simple_dataframe = pd.concat([tf_model_simple_dataframe,
                                          tflite_model_simple_dataframe,
                                          tflite_q_model_simple_dataframe],
                                         keys=['TF Model', 'TFLite', 'TFLite quantized'],
                                         axis='columns')

# Swap columns for side-by-side comparison
all_models_simple_dataframe = all_models_simple_dataframe.swaplevel(axis='columns')[tf_model_simple_dataframe.columns]

# Highlight differences
all_models_simple_dataframe.style.apply(highlight_diff, axis=None)
```

	Label_id			Confidence		
	TF Model	TFLite	TFLite quantized	TF Model	TFLite	TFLite quantized
0	1.00000000	1.00000000	1.00000000	0.99999690	0.99999690	0.99999404
1	1.00000000	1.00000000	1.00000000	0.99991584	0.99991584	0.99998367
2	4.00000000	4.00000000	0.00000000	0.74850243	0.74850076	0.54346687
3	0.00000000	0.00000000	0.00000000	0.97424465	0.97424465	0.85414314
4	0.00000000	0.00000000	0.00000000	0.90023416	0.90023369	0.97871244
5	4.00000000	4.00000000	4.00000000	0.99966431	0.99966431	0.98965561
6	4.00000000	4.00000000	4.00000000	0.96661568	0.96661556	0.87169594
7	2.00000000	2.00000000	2.00000000	0.96229154	0.96229196	0.99848467
8	0.00000000	0.00000000	0.00000000	0.83642560	0.83642513	0.47610903
9	3.00000000	3.00000000	2.00000000	0.60503703	0.60503846	0.86991590
10	2.00000000	2.00000000	2.00000000	0.65258217	0.65258402	0.95267034
11	1.00000000	1.00000000	1.00000000	0.99990451	0.99990451	0.99996424
12	1.00000000	1.00000000	1.00000000	0.94440705	0.94440717	0.98921037
13	0.00000000	0.00000000	0.00000000	0.94699538	0.94699556	0.86087441
14	1.00000000	1.00000000	1.00000000	0.99908721	0.99908721	0.99931931
15	4.00000000	4.00000000	4.00000000	0.99820876	0.99820876	0.98165303
16	4.00000000	4.00000000	4.00000000	0.99191815	0.99191803	0.97113949
17	0.00000000	0.00000000	0.00000000	0.94628590	0.94628531	0.75057566
18	4.00000000	4.00000000	4.00000000	0.99200809	0.99200797	0.93753916
19	2.00000000	2.00000000	2.00000000	0.98288417	0.98288453	0.83607125
20	0.00000000	0.00000000	0.00000000	0.97514319	0.97514307	0.99459970
21	3.00000000	3.00000000	3.00000000	0.95042741	0.95042795	0.40986747
22	1.00000000	1.00000000	1.00000000	0.82691061	0.82691067	0.76372111
23	4.00000000	4.00000000	4.00000000	0.99396050	0.99396038	0.87438804
24	4.00000000	4.00000000	4.00000000	0.98476911	0.98476899	0.94432044
25	3.00000000	3.00000000	3.00000000	0.94753909	0.94753909	0.83315617
26	3.00000000	3.00000000	3.00000000	0.98112398	0.98112386	0.84955180
27	1.00000000	1.00000000	1.00000000	0.99999833	0.99999833	0.99999106
28	4.00000000	4.00000000	4.00000000	0.83044612	0.83044440	0.77505428
29	0.00000000	0.00000000	0.00000000	0.99363178	0.99363166	0.99771690
30	2.00000000	2.00000000	2.00000000	0.77516419	0.77516735	0.96824640
31	1.00000000	1.00000000	1.00000000	0.78624076	0.78624171	0.99284571

Visualiza predicciones de modelos TFLite

Al final, visualicemos las predicciones de TensorFlow Lite y los modelos cuantificados de TensorFlow Lite.

```python
# Print images batch and labels predictions for TFLite Model

tflite_predicted_ids = np.argmax(tflite_model_predictions, axis=-1)
tflite_predicted_labels = dataset_labels[tflite_predicted_ids]
tflite_label_id = np.argmax(val_label_batch, axis=-1)

plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(val_image_batch[n])
  color = "green" if tflite_predicted_ids[n] == true_label_ids[n] else "red"
  plt.title(tflite_predicted_labels[n].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("TFLite model predictions (green: correct, red: incorrect)")
```

```python
# Print images batch and labels predictions for TFLite Model

tflite_q_predicted_ids = np.argmax(tflite_q_model_predictions, axis=-1)
tflite_q_predicted_labels = dataset_labels[tflite_q_predicted_ids]
tflite_q_label_id = np.argmax(val_label_batch, axis=-1)

plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(val_image_batch[n])
  color = "green" if tflite_q_predicted_ids[n] == true_label_ids[n] else "red"
  plt.title(tflite_q_predicted_labels[n].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("Quantized TFLite model predictions (green: correct, red: incorrect)")
```

Exportar lote de validación de imágenes

Exporte el lote de validación para que pueda probarse en el lado del cliente. A continuación, creamos un archivo comprimido que contiene todas las imágenes nombradas con la convención:

donde el primer número es índice, el segundo índice de etiqueta verdadero, el tercer valor predicho por TFLite moder generado en este cuaderno.

Luego, todas las imágenes se colocarán en el código de prueba del lado del cliente (res / assets en las pruebas de Android). Las pruebas de integración ejecutarán un proceso de inferencia en cada imagen y luego compararán los resultados con los guardados en los nombres de los archivos.

```
[41] from PIL import Image
```

```
[42] VAL_BATCH_DIR = "validation_batch"
```

```
[43] !mkdir {VAL_BATCH_DIR}
```

```python
# Export batch to *.jpg files with specific naming convention.
# Make sure they are exported in the full quality, otherwise the inference
# process will return different results.

for n in range(32):
  filename = "n{:0.0f}_true{:0.0f}_pred{:0.0f}.jpg".format(
      n,
      true_label_ids[n],
      tflite_model_pred_simplified[n][0]
  )
  img_arr = np.copy(val_image_batch[n])
  img_arr *= 255
  img_arr = img_arr.astype("uint8")
  img11 = Image.fromarray(img_arr, 'RGB')
  img11.save("{}/{}".format(VAL_BATCH_DIR, filename), "JPEG", quality=100)
```

```
!tar -zcvf {VAL_BATCH_DIR}.tar.gz {VAL_BATCH_DIR}
```

```
validation_batch/
validation_batch/n16_true4_pred4.jpg
validation_batch/n9_true3_pred3.jpg
validation_batch/n11_true1_pred1.jpg
validation_batch/n17_true0_pred0.jpg
validation_batch/n18_true4_pred4.jpg
validation_batch/n26_true3_pred3.jpg
validation_batch/n12_true1_pred1.jpg
validation_batch/n0_true1_pred1.jpg
validation_batch/n2_true4_pred4.jpg
validation_batch/n13_true2_pred0.jpg
validation_batch/n21_true3_pred3.jpg
validation_batch/n20_true0_pred0.jpg
validation_batch/n23_true4_pred4.jpg
validation_batch/n8_true0_pred0.jpg
validation_batch/n25_true3_pred3.jpg
validation_batch/n27_true1_pred1.jpg
validation_batch/n28_true4_pred4.jpg
validation_batch/n29_true0_pred0.jpg
validation_batch/n3_true0_pred0.jpg
validation_batch/n15_true4_pred4.jpg
validation_batch/n31_true0_pred1.jpg
validation_batch/n1_true1_pred1.jpg
validation_batch/n19_true2_pred2.jpg
validation_batch/n14_true1_pred1.jpg
validation_batch/n24_true4_pred4.jpg
validation_batch/n7_true2_pred2.jpg
validation_batch/n22_true1_pred1.jpg
validation_batch/n4_true0_pred0.jpg
validation_batch/n6_true4_pred4.jpg
validation_batch/n5_true4_pred4.jpg
validation_batch/n10_true4_pred2.jpg
validation_batch/n30_true2_pred2.jpg
```