Machine Learning with TensorFlow Project

Istanbul or Taipei?

Aslihan Demirkaya & Jerry Hsu

Project Objective:

This project is aimed to train the machine to differentiate between the cities "Istanbul" and "Taipei" when a picture of those cities is given as an input.

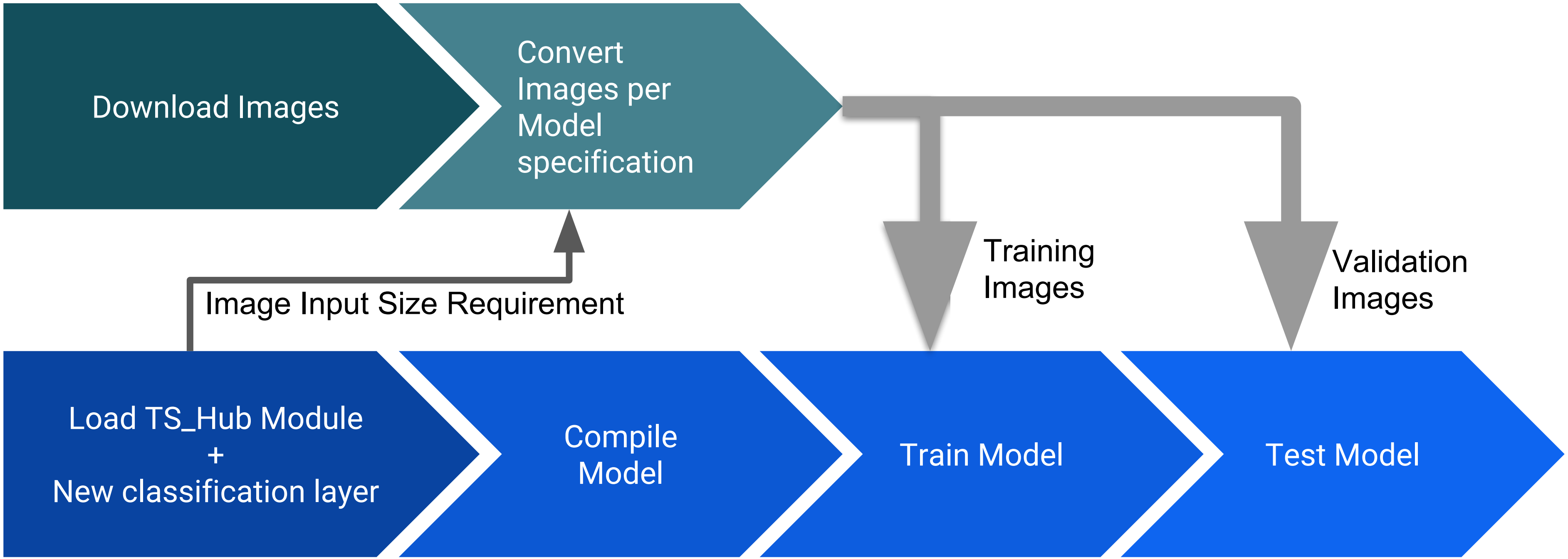
Ex: Input:

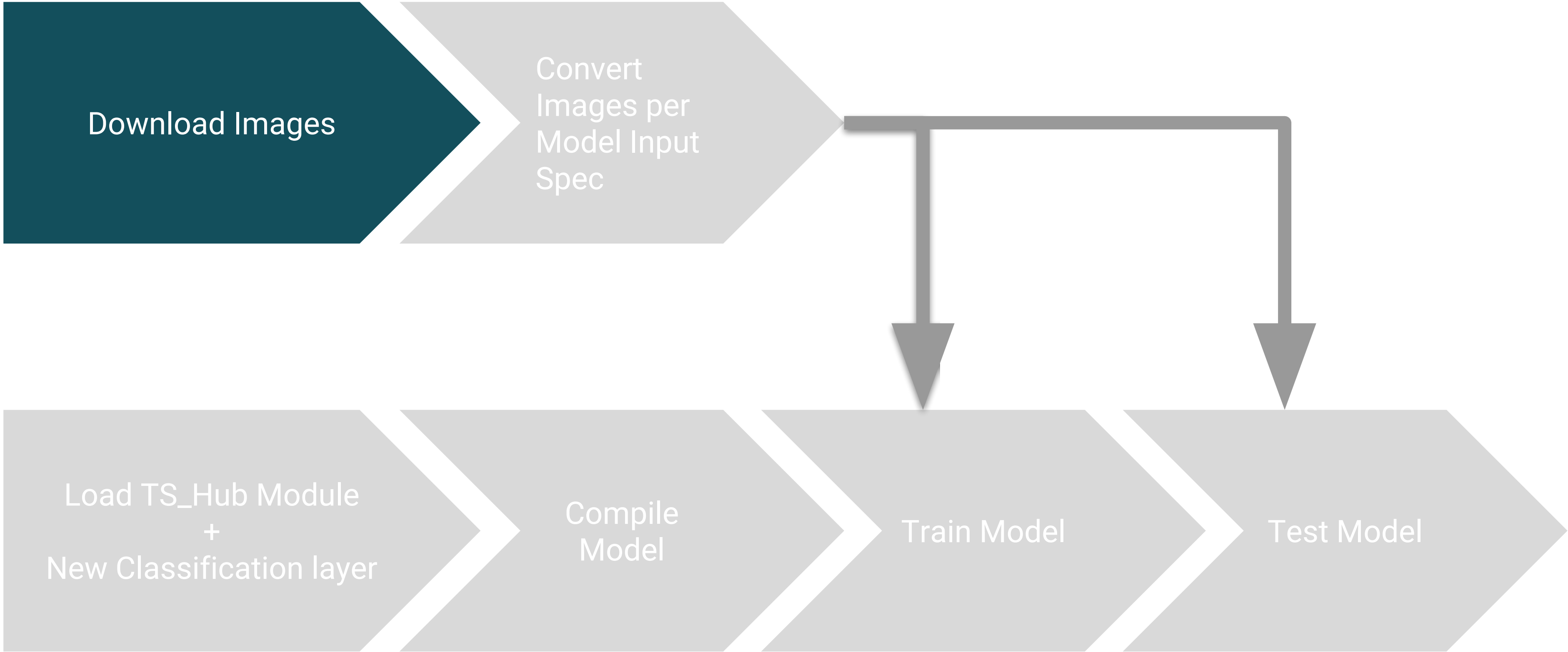
Can you guess the output?

Key Technology Components

- Use Flickr API and Images to acquire data
- Use Tensorflow Keras High Level API for rapid development
- Build network model by combining ‘softmax’ and MobileNetV2 module from Tensorflow Hub

Data Flow Overview




Download Images
Convert Images per Model Input Spec
Load TS_Hub Module + New Classification layer
Compile Model
Train Model
Test Model

Preparing the Data Set:

- We created an equally balanced data set consisting of images from both cities: Istanbul and Taipei. (3391 Istanbul + 3391 Taipei pictures)

- We used Flickr (an image hosting service) as the source of our data set.

- Steps to download the images:

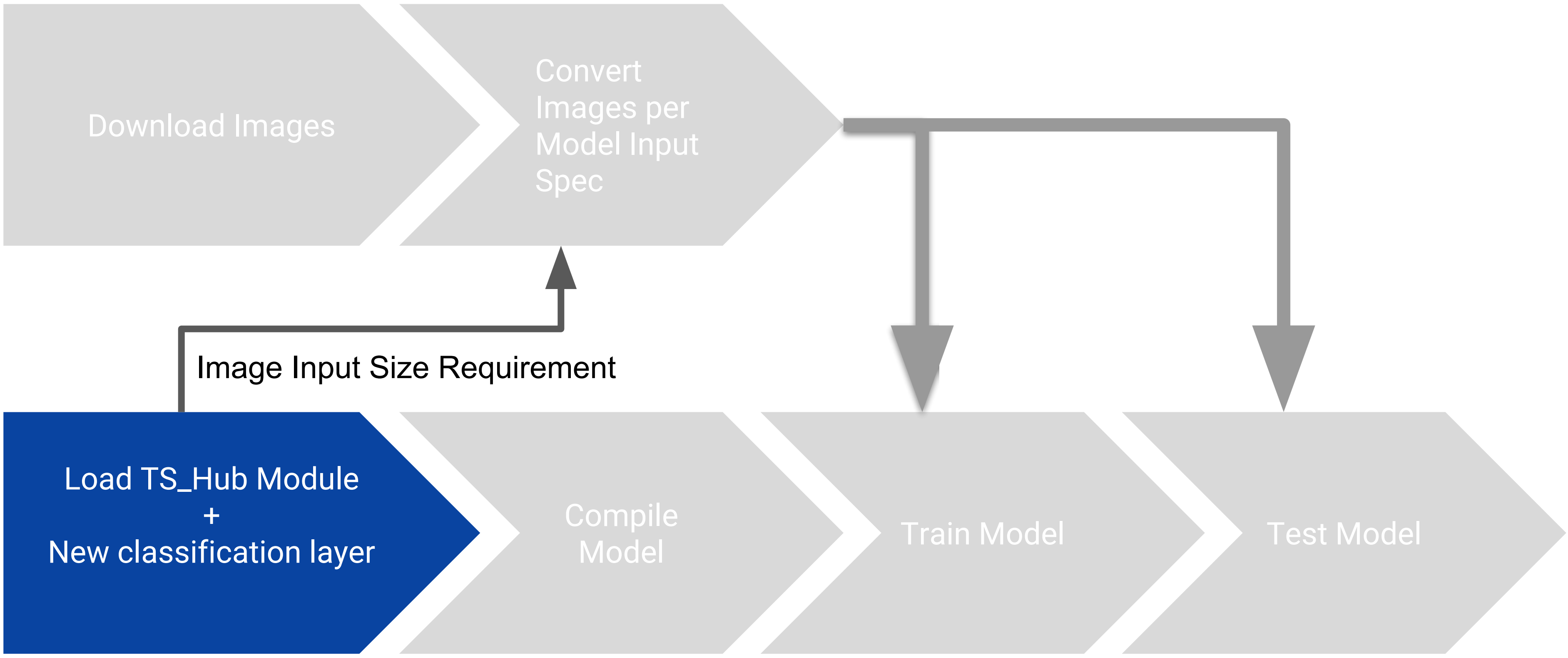
Step 1: We install Python FlickrAPI from the Python Package Index: https://stuvel.eu/flickrapi-doc/

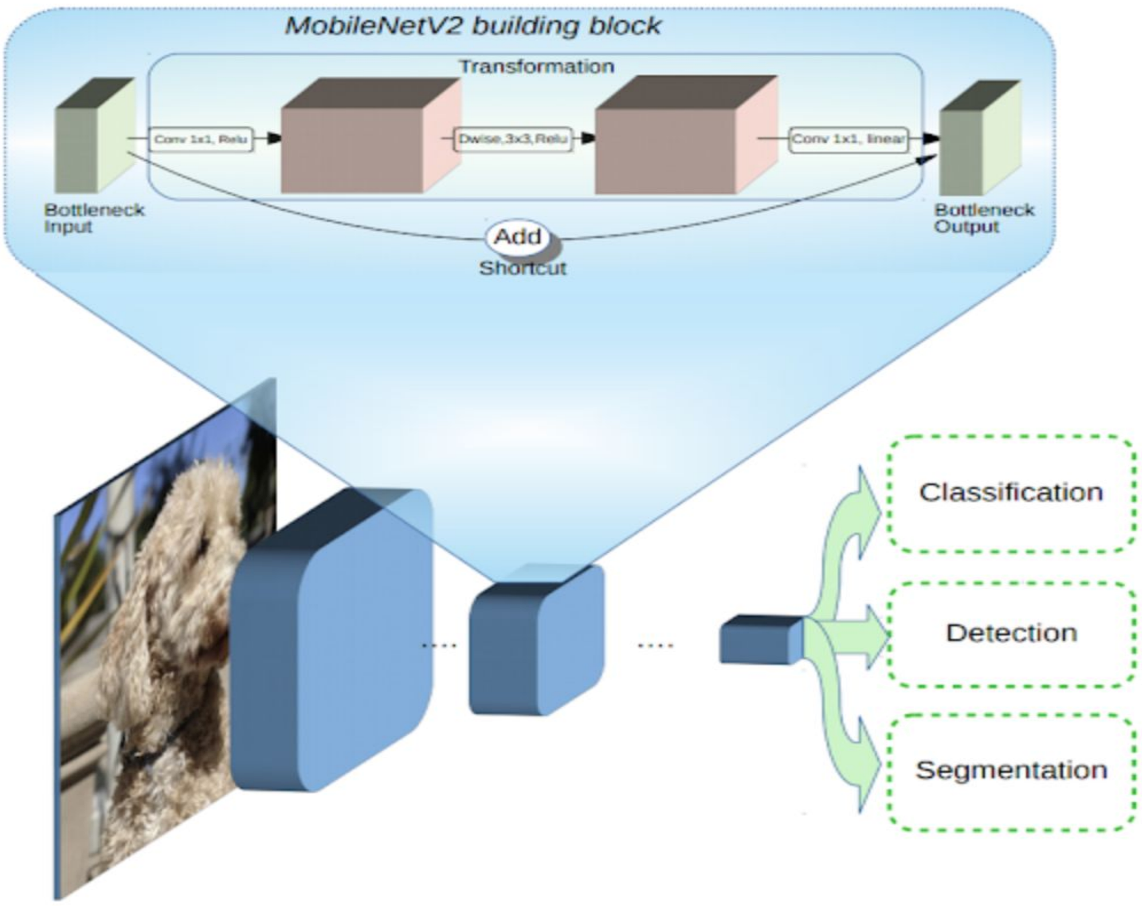
Step 2: We created an app from this link: https://www.flickr.com/services/apps/create/noncommercial/ and obtained Flickr API key. The steps are explained in this video: https://www.youtube.com/watch?v=Lq1XRx6dsDU

Step 3: We download images of Istanbul and Taipei using the code below. The source for the code is found here: https://gist.github.com/yunjey/14e3a069ad2aa3adf72dee93a53117d6

```python
#In the first part, we obtain the urls of the images:
import flickrapi
import urllib
from PIL import Image

api_key = u'53a6e9e4eb5daab8bffbad47f0f4e4e8' # Flickr api access key is obtained in Step 2.
api_secret = u'dfb5411b99361e87'

flickr=flickrapi.FlickrAPI(api_key, api_secret, cache=True)

keyword =  'istanbul' #'taipei'

photos = flickr.walk(text=keyword,
                     tag_mode='all',
                     tags=keyword,
                     extras='url_c',
                     per_page=100,  # Number of photos to return per page, the default number is 100.
                     sort='relevance')

urls = []
for i, photo in enumerate(photos):

    url = photo.get('url_c')
    urls.append(url)

    if i > 4292:  # get 4292 urls
        break
```

```python
#In the second part we download images from the urls and resize them.
import pickle
pickle.dump(urls, open("istanbul_urls_flickr.pckl", 'wb'))
len(urls)
!mkdir istanbul
#pickle.dump(urls, open("taipei_urls_flickr.pckl", 'wb'))
#!mkdir taipei
for i, image_url in enumerate(urls):
    try:
        image = urllib.request.urlopen(image_url)
        # Resize the image and overwrite it
        image = Image.open(image)
        image = image.resize((256, 256), Image.ANTIALIAS)
        image.save(f'istanbul/{i}.jpg')
    except AttributeError:
        continue
```


Download Images
Convert Images per Model Input Spec
Image Input Size Requirement
Load TS_Hub Module
+
New classification layer
Compile Model
Train Model
Test Model



https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

Acquire training model and identify image size requirement

Using MobileNetV2 model without classification layer

```
feature_extractor_url = "https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/2"

def feature_extractor(x):
    feature_extractor_module = hub.Module(feature_extractor_url)
    return feature_extractor_module(x)

IMAGE_SIZE = hub.get_expected_image_size(hub.Module(feature_extractor_url))
```

Prepare hub layer and create module

```
features_extractor_layer = layers.Lambda(feature_extractor, input_shape=IMAGE_SIZE+[3])
```

- Freeze the variables in the feature extractor layer, so that the training only modifies the new classifier layer.

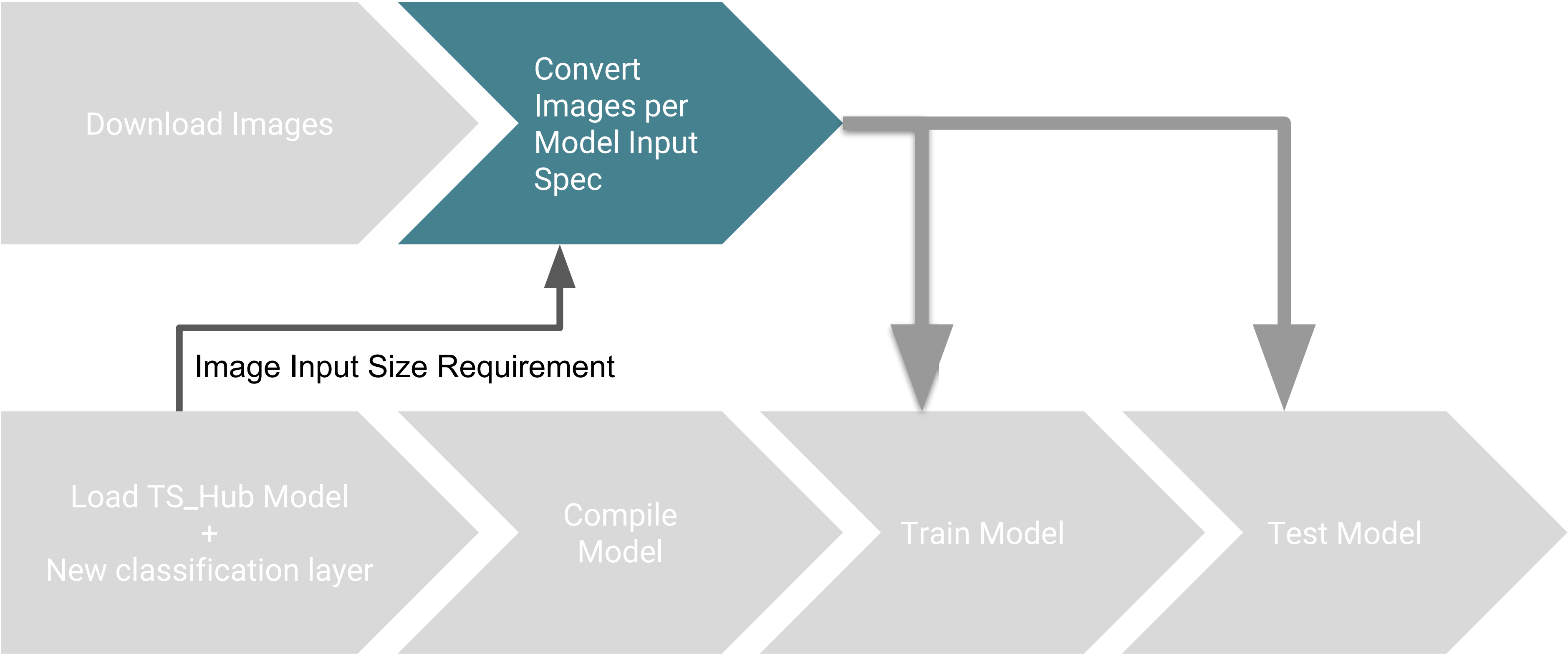
```
features_extractor_layer.trainable = False
```

- Wrap the hub layer in a `tf.keras.Sequential` model, and add a new classification layer.

```
model = tf.keras.Sequential([
  features_extractor_layer,
  layers.Dense(training_data.num_classes, activation='softmax')
])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2562
=================================================================
Total params: 2,562
Trainable params: 2,562
Non-trainable params: 0
_________________________________________________________________
```

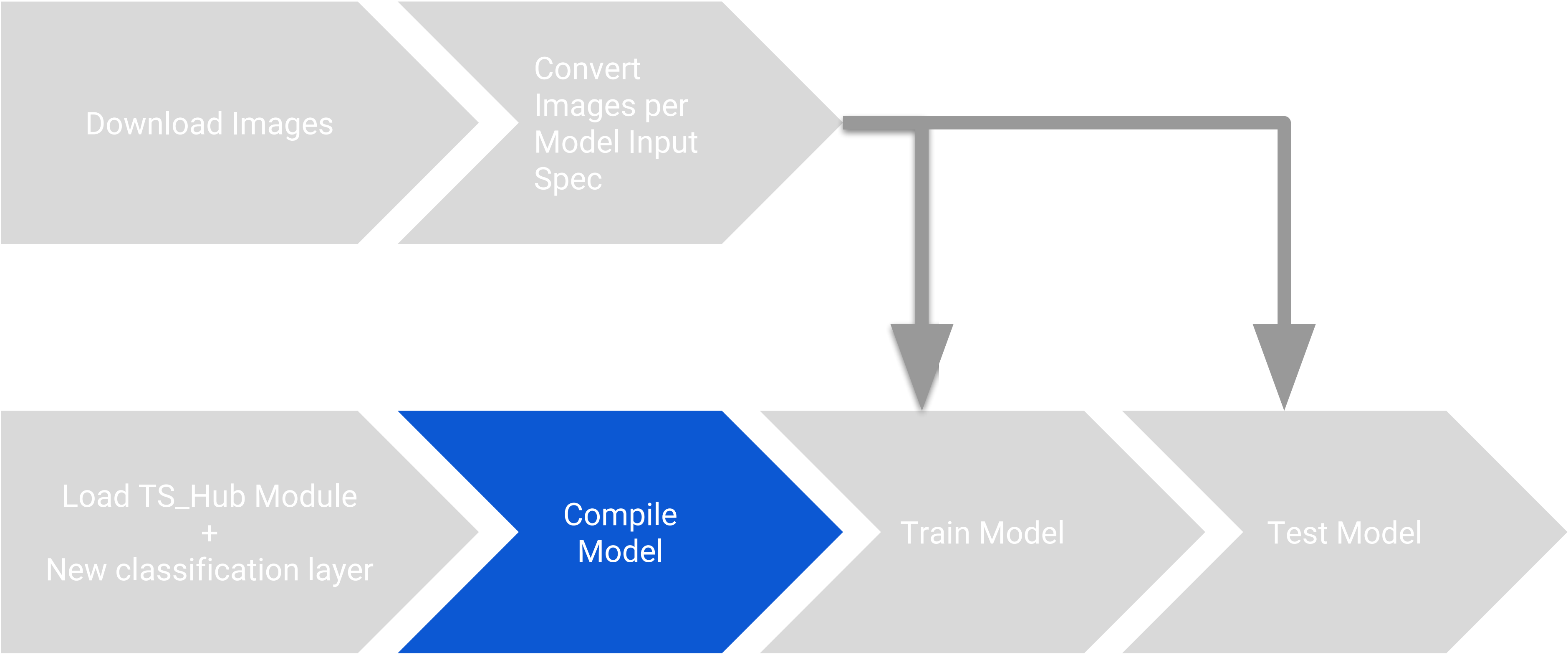
(~ 3M parameters)


Download Images
Convert Images per Model Input Spec
Image Input Size Requirement
Load TS_Hub Model
+
New classification layer
Compile Model
Train Model
Test Model

Setting up the data loader and splitting the data set into: **Training** & **Validation**

```
data_root = 'istanbul_taipei'
VALIDATION_SPLIT = 0.15
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255, validation_split = VALIDATION_SPLIT)
training_data = image_generator.flow_from_directory(str(data_root), target_size=IMAGE_SIZE, subset = 'training')

validation_data = image_generator.flow_from_directory(str(data_root), target_size=IMAGE_SIZE
                                                       , subset = 'validation', shuffle = False)

for image_batch,label_batch in training_data:
    print("Image batch shape: ", image_batch.shape)
    print("Label batch shape: ", label_batch.shape)
    break
```

```
Found 5766 images belonging to 2 classes.
Found 1016 images belonging to 2 classes.
Image batch shape:  (32, 224, 224, 3)
Label batch shape:  (32, 2)
```


Download Images
Convert Images per Model Input Spec
Load TS_Hub Module
+
New classification layer
Compile Model
Train Model
Test Model

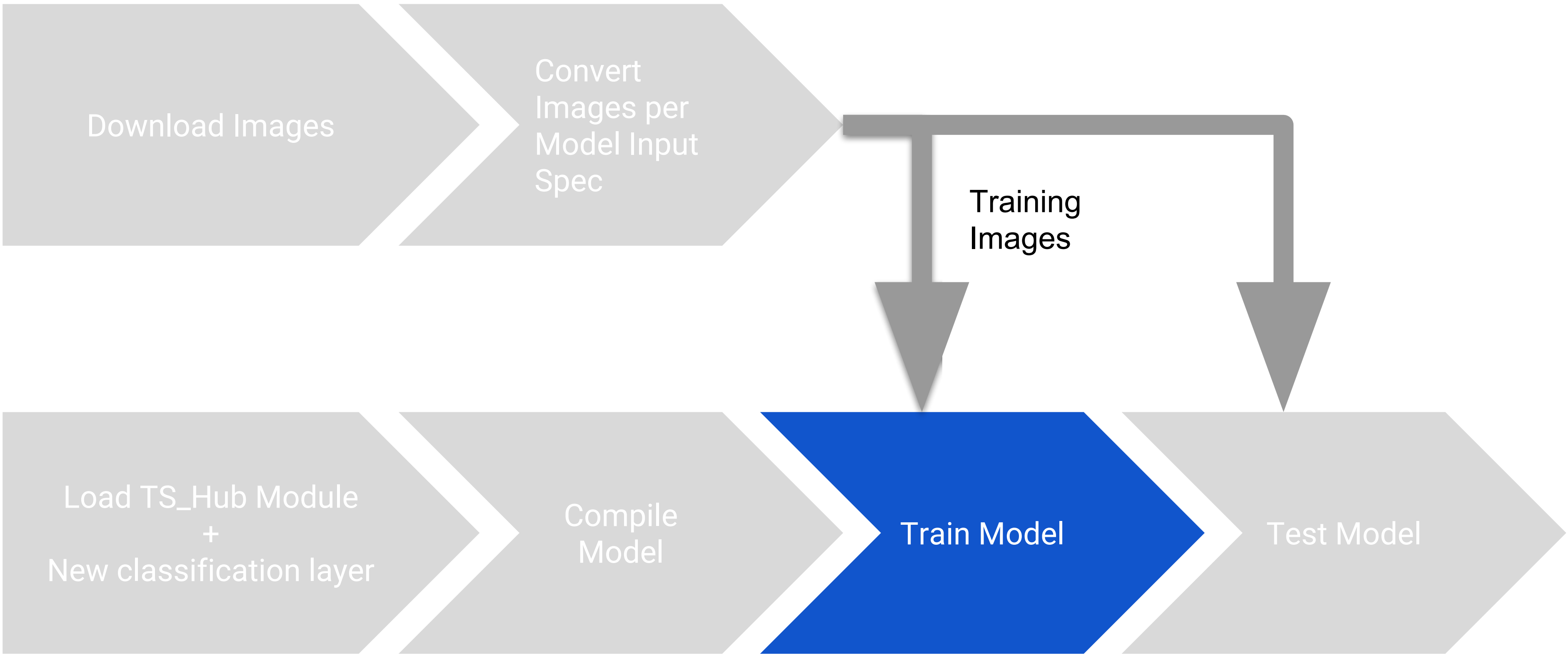
Train the model

Use compile to configure the training process:

```
model.compile(
  optimizer=tf.train.AdamOptimizer(),
  loss='categorical_crossentropy',
  metrics=['accuracy'])
```

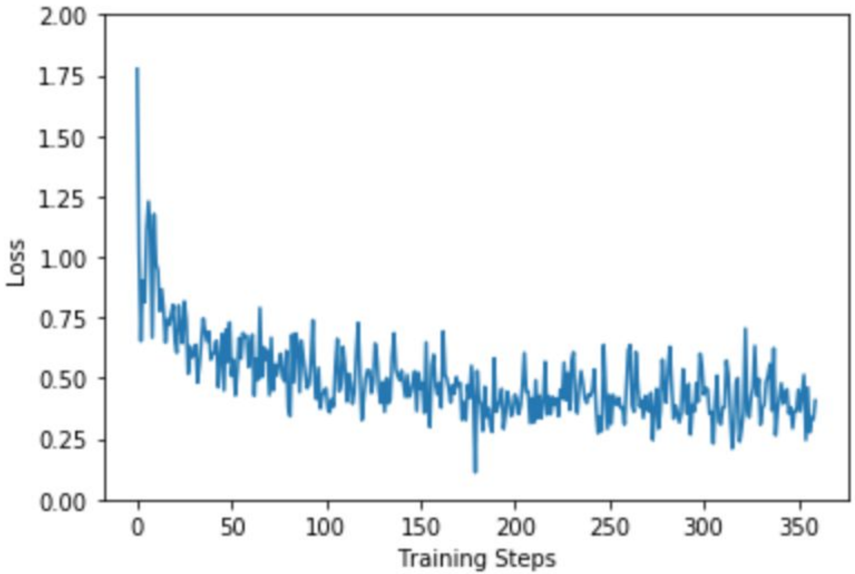
We use the fit method of Keras model to use custom Callback functions following:
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/hub_with_keras.ipynb#scrollTo=wC_AYRJU9NQe

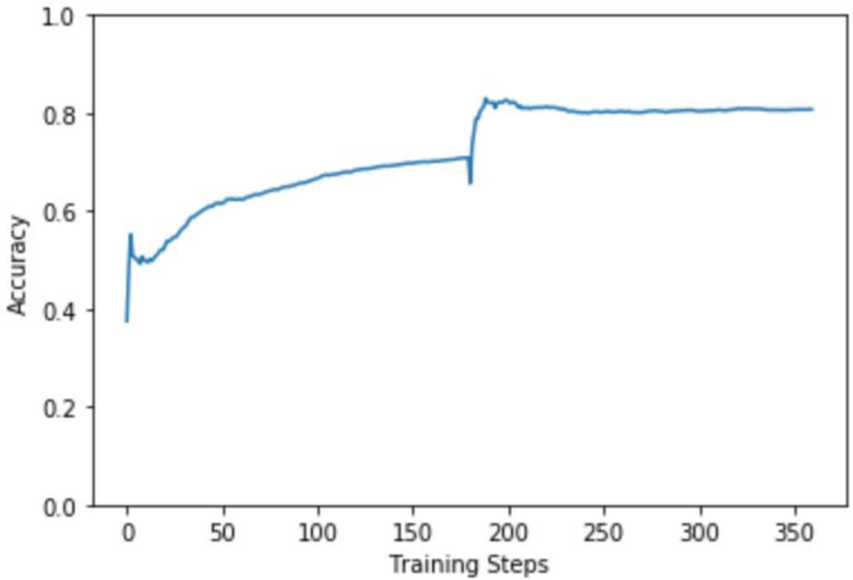
```
class CollectBatchStats(tf.keras.callbacks.Callback):
    def __init__(self):
        self.batch_losses = []
        self.batch_acc = []

    def on_batch_end(self, batch, logs=None):
        self.batch_losses.append(logs['loss'])
        self.batch_acc.append(logs['acc'])
```


Download Images
Convert Images per Model Input Spec
Training Images
Load TS_Hub Module + New classification layer
Compile Model
Train Model
Test Model

```python
steps_per_epoch = training_data.samples//training_data.batch_size
batch_stats = CollectBatchStats()
model.fit((item for item in training_data), epochs=2, # Our experiments show that the accuracy does not improve after 2
                    steps_per_epoch=steps_per_epoch,
                    callbacks = [batch_stats])
```

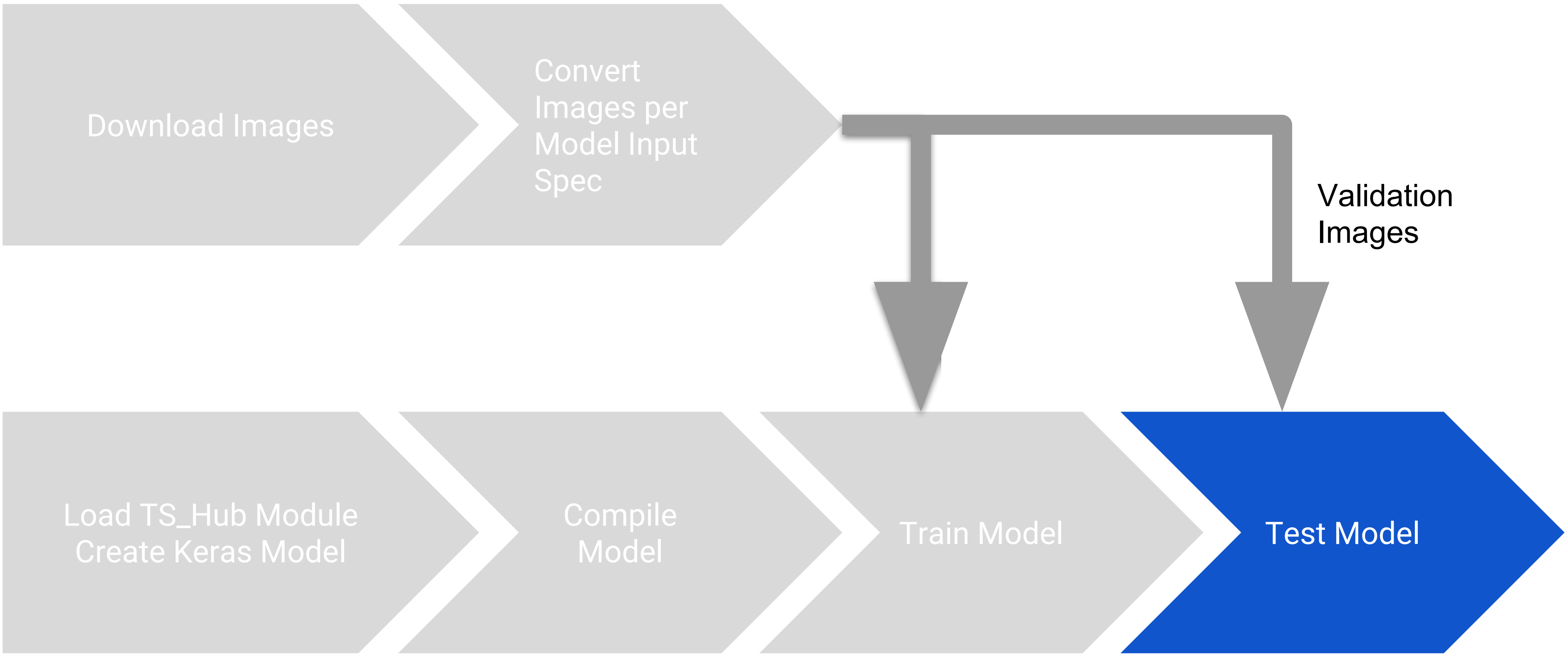
```
Epoch 1/2
 36/180 [=====>........................] - ETA: 1:34 - loss: 0.8016 - acc: 0.5877

Epoch 1/2
104/180 [================>.............] - ETA: 52s - loss: 0.6407 - acc: 0.6722

Epoch 1/2
163/180 [==========================>...] - ETA: 11s - loss: 0.5866 - acc: 0.7013

Epoch 1/2
180/180 [==============================] - 125s 692ms/step - loss: 0.5716 - acc: 0.7089
Epoch 2/2
 21/180 [==>...........................] - ETA: 1:46 - loss: 0.3903 - acc: 0.8229

Epoch 2/2
 75/180 [===========>..................] - ETA: 1:12 - loss: 0.4139 - acc: 0.8025

Epoch 2/2
123/180 [===================>..........] - ETA: 39s - loss: 0.4192 - acc: 0.8039

Epoch 1/2
180/180 [==============================] - 125s 692ms/step - loss: 0.5716 - acc: 0.7089
Epoch 2/2
180/180 [==============================] - 124s 689ms/step - loss: 0.4139 - acc: 0.8075
```






```
plt.figure()
plt.ylabel("Loss")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(batch_stats.batch_losses)
```

```
plt.figure()
plt.ylabel("Accuracy")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(batch_stats.batch_acc)
```

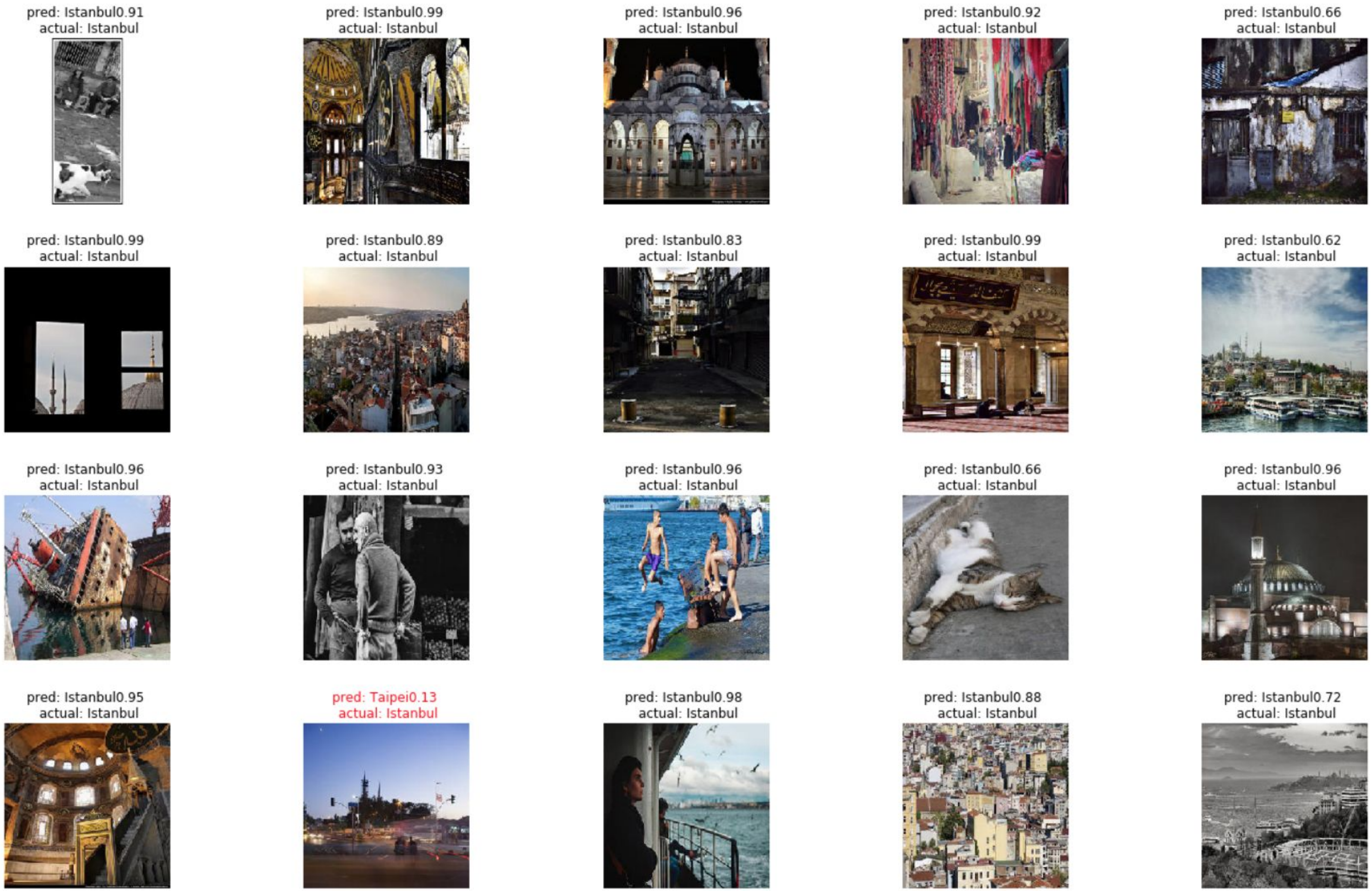

Download Images
Convert Images per Model Input Spec
Validation Images
Load TS_Hub Module
Create Keras Model
Compile Model
Train Model
Test Model

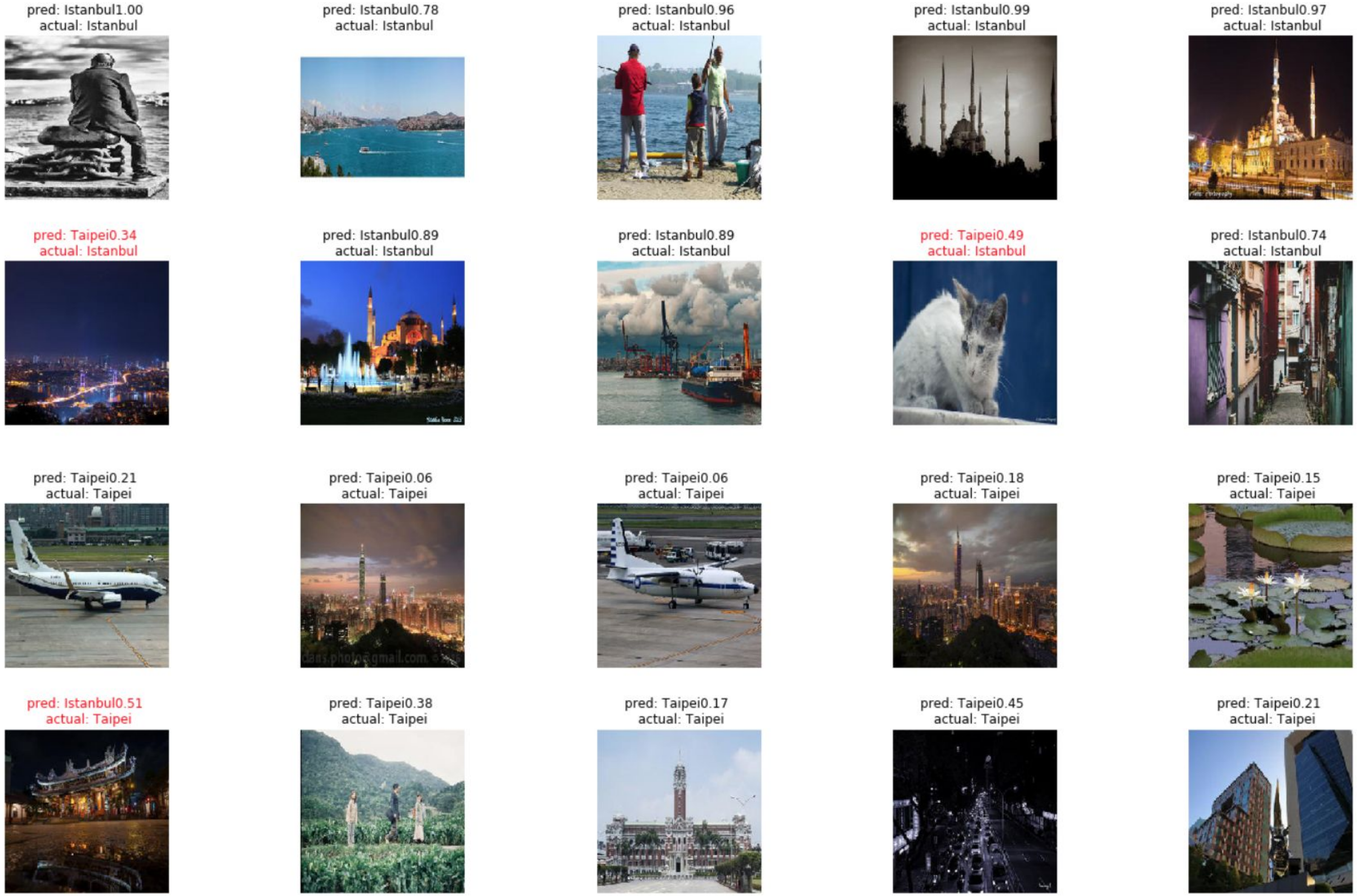
Evaluating the model on the Validation Set

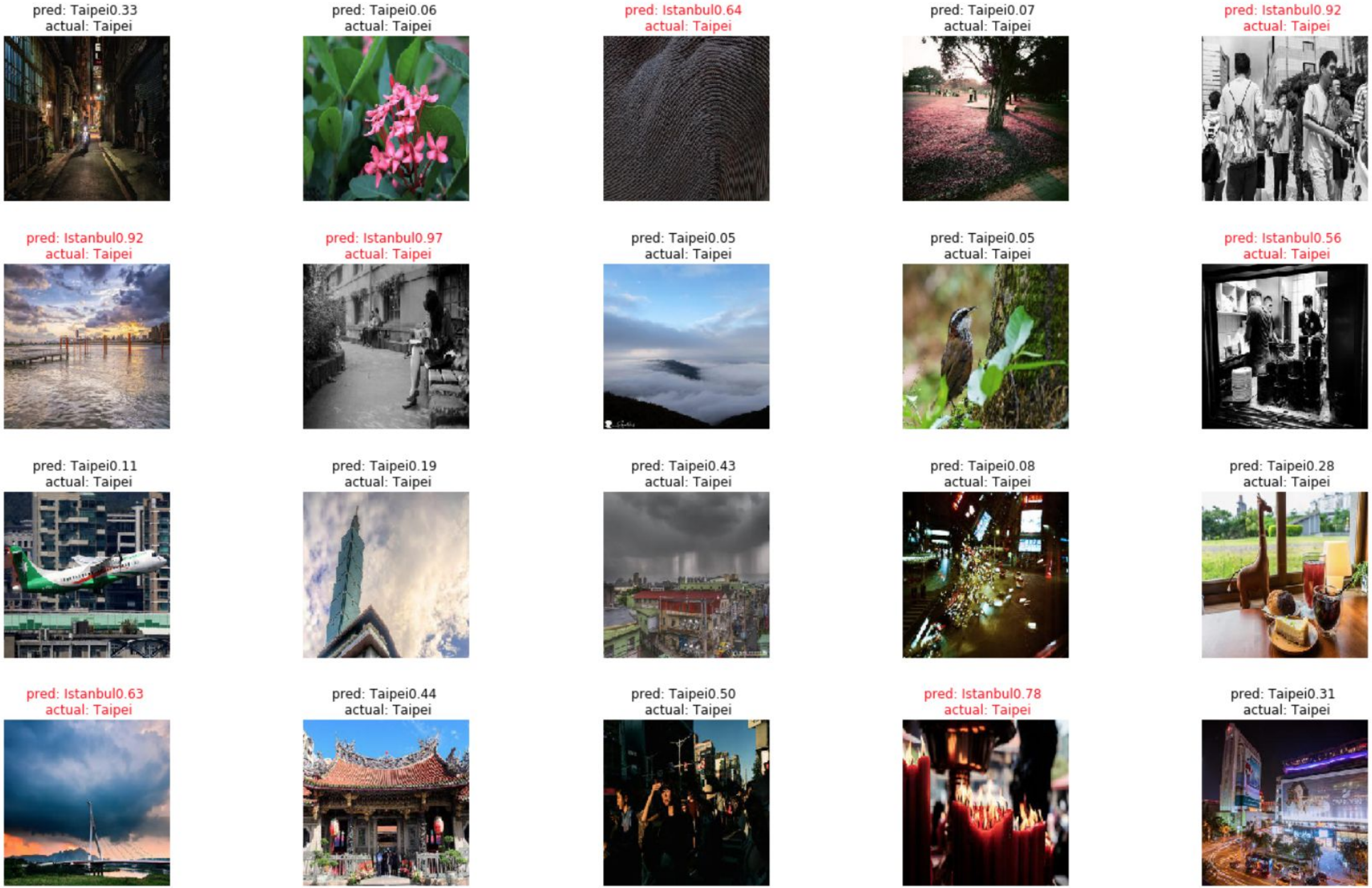
```
valid_scores = []
count_ = 0
num_batch = len(validation_data.filenames) // 32
validation_data.reset() # resetting the generator so that we always get the same set of images for validation
for x_test, y_test in validation_data:
    scores = model.evaluate(x_test, y_test, verbose=1)
    valid_scores.append(scores)
    count_ += 1
    if count_ >= num_batch:
        break
print("validation accuracy", np.mean(list((uu[1] for uu in valid_scores))))
```

```
32/32 [==============================] - 1s 21ms/sample - loss: 0.5485 - acc: 0.7500
32/32 [==============================] - 1s 21ms/sample - loss: 0.7346 - acc: 0.6250
32/32 [==============================] - 1s 21ms/sample - loss: 0.4502 - acc: 0.7812
32/32 [==============================] - 1s 22ms/sample - loss: 0.4661 - acc: 0.8438
32/32 [==============================] - 1s 22ms/sample - loss: 0.7179 - acc: 0.6562
32/32 [==============================] - 1s 22ms/sample - loss: 0.3870 - acc: 0.7812
validation accuracy 0.8054435
```

Results:


pred: Istanbul0.91
actual: Istanbul
pred: Istanbul0.99
actual: Istanbul
pred: Istanbul0.96
actual: Istanbul
pred: Istanbul0.92
actual: Istanbul
pred: Istanbul0.66
actual: Istanbul
pred: Istanbul0.99
actual: Istanbul
pred: Istanbul0.89
actual: Istanbul
pred: Istanbul0.83
actual: Istanbul
pred: Istanbul0.99
actual: Istanbul
pred: Istanbul0.62
actual: Istanbul
pred: Istanbul0.96
actual: Istanbul
pred: Istanbul0.93
actual: Istanbul
pred: Istanbul0.96
actual: Istanbul
pred: Istanbul0.66
actual: Istanbul
pred: Istanbul0.96
actual: Istanbul
pred: Istanbul0.95
actual: Istanbul
pred: Taipei0.13
actual: Istanbul
pred: Istanbul0.98
actual: Istanbul
pred: Istanbul0.88
actual: Istanbul
pred: Istanbul0.72
actual: Istanbul


pred: Istanbul1.00
actual: Istanbul
pred: Istanbul0.78
actual: Istanbul
pred: Istanbul0.96
actual: Istanbul
pred: Istanbul0.99
actual: Istanbul
pred: Istanbul0.97
actual: Istanbul
pred: Taipei0.34
actual: Istanbul
pred: Istanbul0.89
actual: Istanbul
pred: Istanbul0.89
actual: Istanbul
pred: Taipei0.49
actual: Istanbul
pred: Istanbul0.74
actual: Istanbul
pred: Taipei0.21
actual: Taipei
pred: Taipei0.06
actual: Taipei
pred: Taipei0.06
actual: Taipei
pred: Taipei0.18
actual: Taipei
pred: Taipei0.15
actual: Taipei
pred: Istanbul0.51
actual: Taipei
pred: Taipei0.38
actual: Taipei
pred: Taipei0.17
actual: Taipei
pred: Taipei0.45
actual: Taipei
pred: Taipei0.21
actual: Taipei


pred: Taipei0.33
actual: Taipei
pred: Taipei0.06
actual: Taipei
pred: Istanbul0.64
actual: Taipei
pred: Taipei0.07
actual: Taipei
pred: Istanbul0.92
actual: Taipei
pred: Istanbul0.92
actual: Taipei
pred: Istanbul0.97
actual: Taipei
pred: Taipei0.05
actual: Taipei
pred: Taipei0.05
actual: Taipei
pred: Istanbul0.56
actual: Taipei
pred: Taipei0.11
actual: Taipei
pred: Taipei0.19
actual: Taipei
pred: Taipei0.43
actual: Taipei
pred: Taipei0.08
actual: Taipei
pred: Taipei0.28
actual: Taipei
pred: Istanbul0.63
actual: Taipei
pred: Taipei0.44
actual: Taipei
pred: Taipei0.50
actual: Taipei
pred: Istanbul0.78
actual: Taipei
pred: Taipei0.31
actual: Taipei

Confusion Matrix

```
predicted_istanbul=[None]*len(prob_istanbul)
actual_istanbul=[None]*len(true_istanbul)
for i in np.arange(len(true_istanbul)):
    if (prob_istanbul[i] >0.5): #threshold value is taken as 0.5
        predicted_istanbul[i]="Istanbul"
    else:
        predicted_istanbul[i]="Taipei"
for i in np.arange(len(true_istanbul)):
    if (true_istanbul[i]==1):
        actual_istanbul[i]="Istanbul"
    else:
        actual_istanbul[i]="Taipei"
```

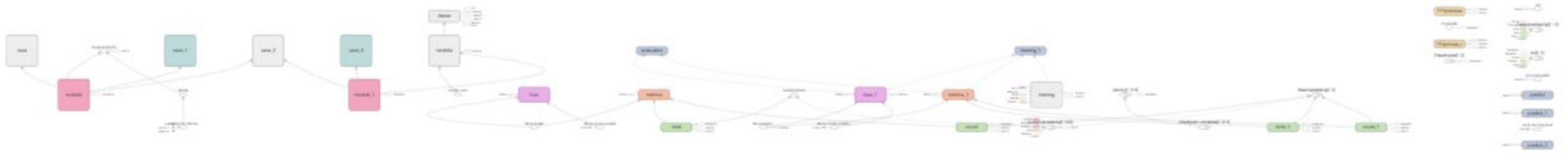
```
import pandas as pd
actual = actual_istanbul
predicted = predicted_istanbul
df = pd.DataFrame({'predicted':predicted,'actual':actual})

pd.crosstab(df['predicted'],df['actual'])
```

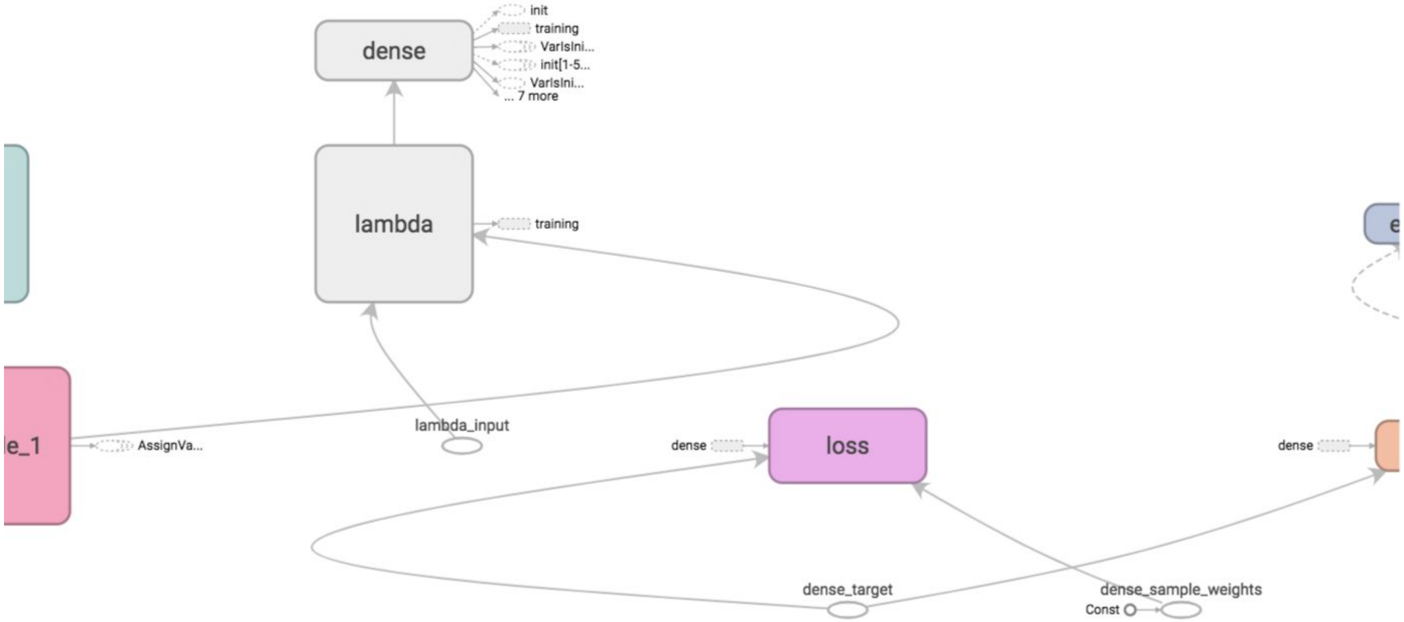
Confusion Matrix:

actual	Istanbul	Taipei
predicted		
Istanbul	439	133
Taipei	69	375

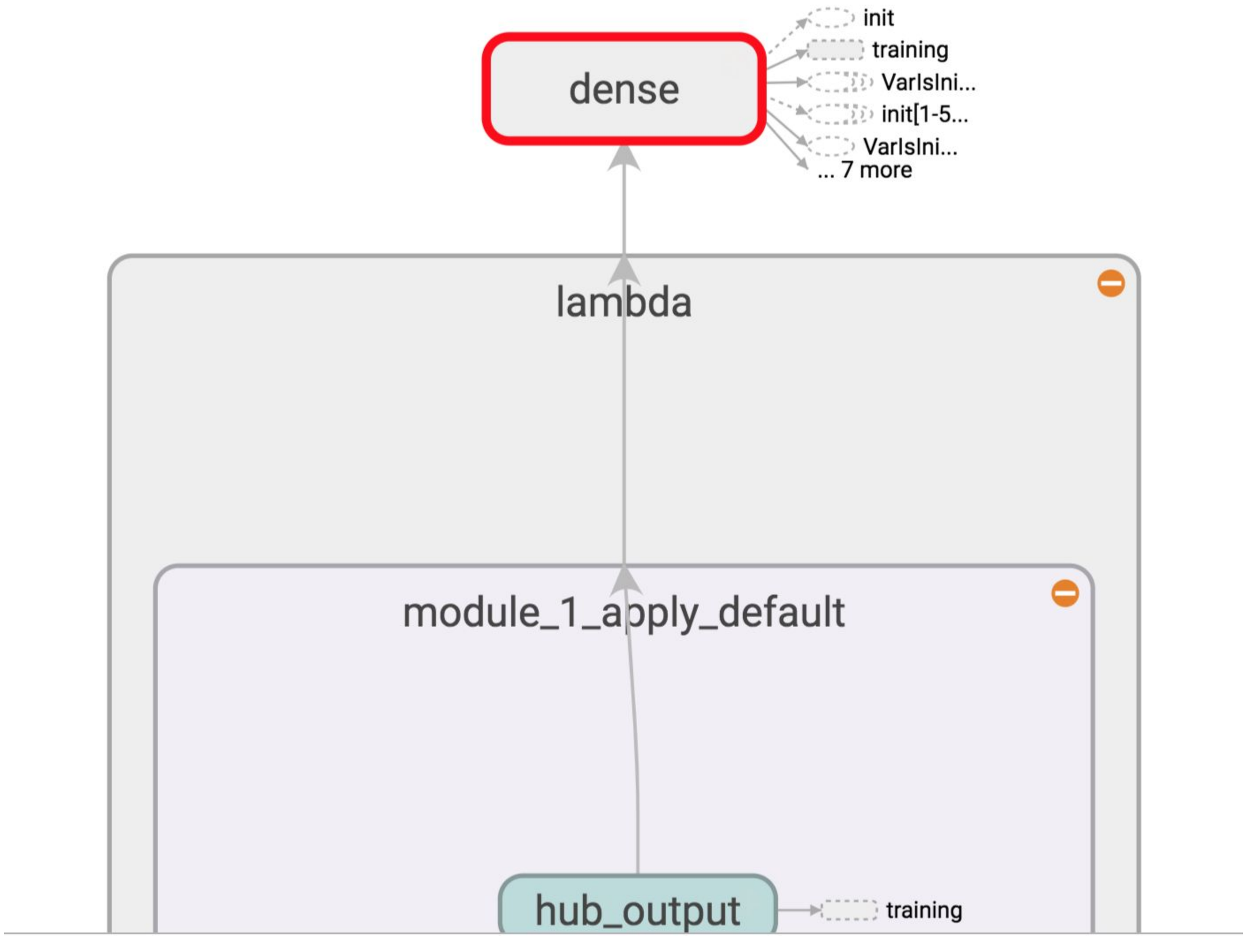
```
Accuracy: 0.801
```

TensorBoard Graph:

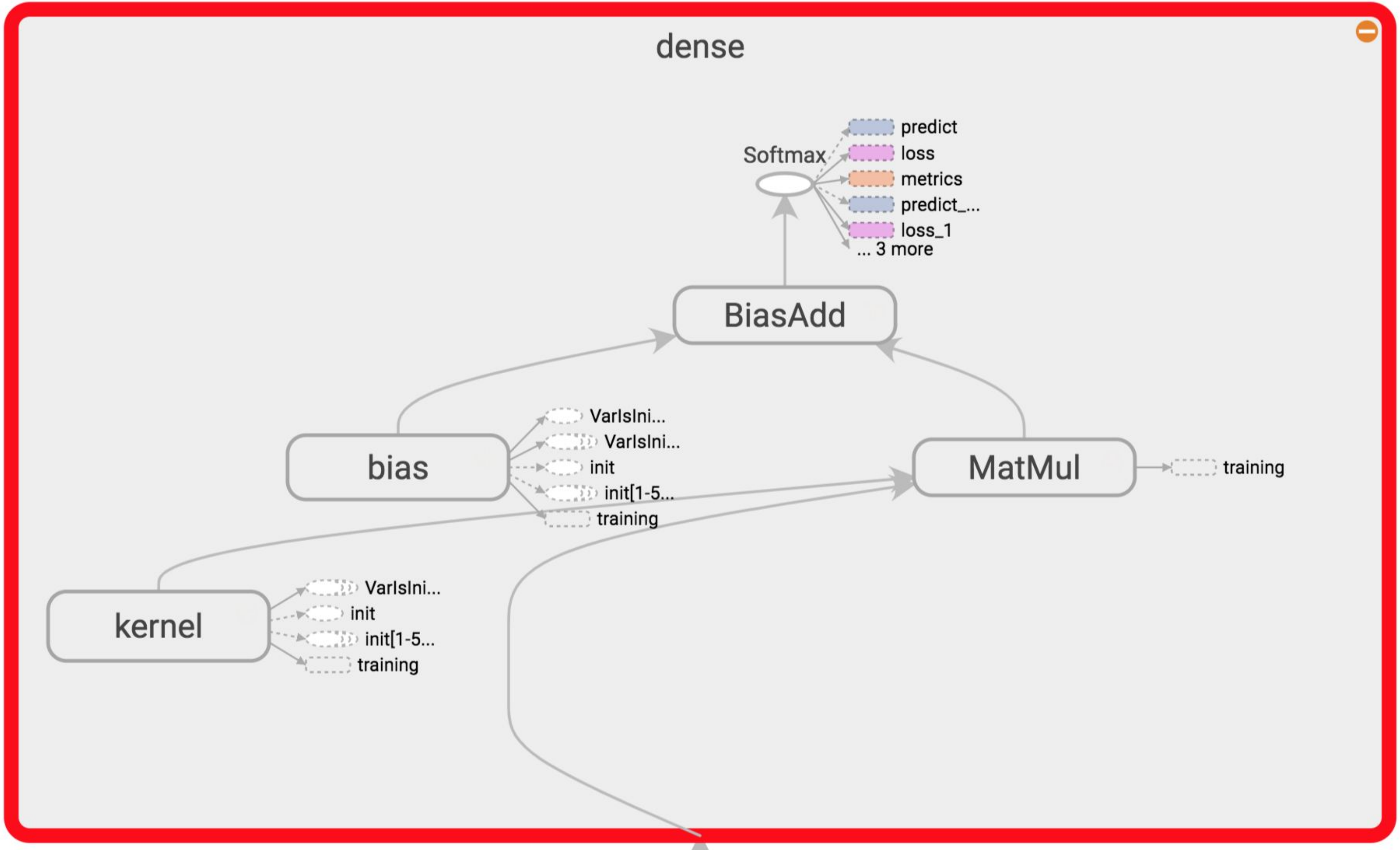



dense
init
training
VarIsIni...
init[1-5...
VarIsIni...
... 7 more
lambda
training
le_1
AssignVa...
lambda_input
dense
loss
dense
dense_target
dense_sample_weights
Const


dense
init
training
VarIsIni...
init[1-5...
VarIsIni...
... 7 more
lambda
module_1_apply_default
hub_output
training
MobilenetV2
Logits
embedding
module_1
Conv_1
module_1
expanded_conv_16
module_1
expanded_conv_16
module_1
expanded_conv_15
module_1
expanded_conv_14
module_1
expanded_conv_13
module_1
expanded_conv_12
module_1
expanded_conv_11
module_1
expanded_conv_10
module_1
expanded_conv_9
module_1
expanded_conv_8
module_1
expanded_conv_7
module_1
expanded_conv_6
module_1
expanded_conv_5
module_1
expanded_conv_4
module_1
expanded_conv_3
module_1
expanded_conv_2
module_1
expanded_conv_1
module_1
expanded_conv_2
module_1
expanded_conv_1
module_1
expanded_conv
module_1
Conv
input
hub_input



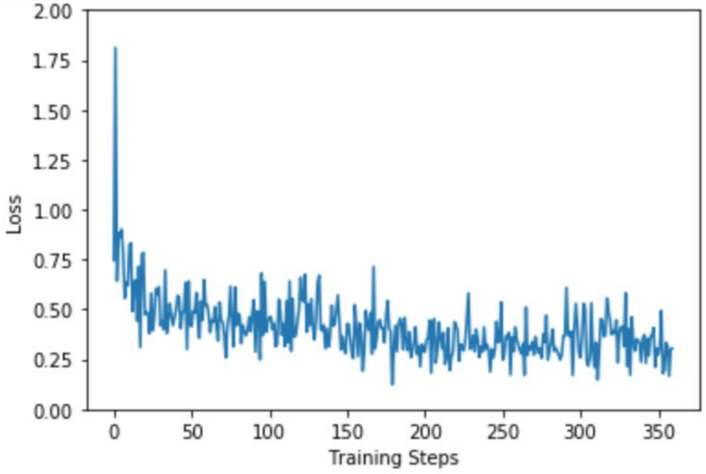
dense
init
training
VarIsIni...
init[1-5...
VarIsIni...
... 7 more
lambda
module_1_apply_default
hub_output
training

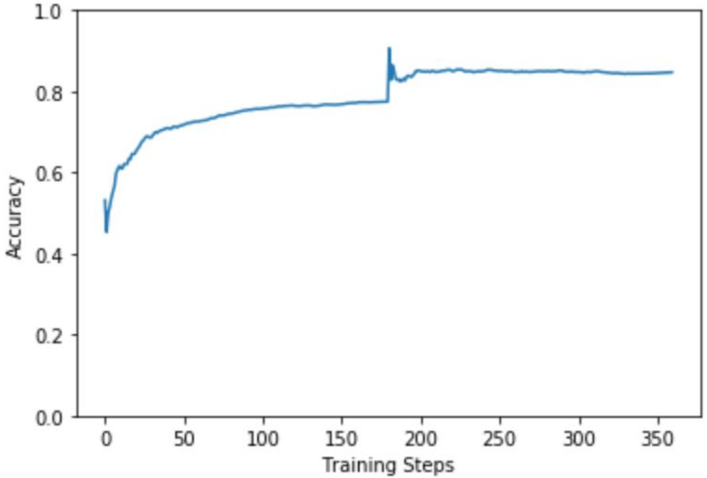

dense
Softmax
predict
loss
metrics
predict_...
loss_1
... 3 more
BiasAdd
bias
VarIsIni...
VarIsIni...
init
init[1-5...
training
MatMul
training
kernel
VarIsIni...
init
init[1-5...
training

Adding one more dense layer:

```
model = tf.keras.Sequential([
  features_extractor_layer,
  layers.Dense(100, activation='relu'),
  layers.Dense(training_data.num_classes, activation='softmax')
])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 100)               128100
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 202
=================================================================
Total params: 128,302
Trainable params: 128,302
Non-trainable params: 0
_________________________________________________________________
```





actual	Istanbul	Taipei
predicted		
Istanbul	436	73
Taipei	72	435

```
Accuracy: 0.857
```

Adding one more dense layer:

```
model = tf.keras.Sequential([
  features_extractor_layer,
  layers.Dense(150, activation='relu'),
  layers.Dense(100, activation='relu'),
  layers.Dense(training_data.num_classes, activation='softmax')
])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 150)               192150
_________________________________________________________________
dense_1 (Dense)              (None, 100)               15100
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202
=================================================================
Total params: 207,452
Trainable params: 207,452
Non-trainable params: 0
```

actual	Istanbul	Taipei
predicted		
Istanbul	436	73
Taipei	72	435

```
Accuracy: 0.859
```