TensorFlow Basics

Core basics

Check this link : https://www.javatpoint.com/tensorflow-interview-questions

How many kinds of Tensors are there?

▼ Constant tensor

Constant Tensors are used as constants

```
T = [
      [1,2,3],
      [4,5,6]
    ]
```

```
tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)
```

▼ Variable Tensor

- PlaceHolder Tensor

This section Core basics is yet to be completed .Please visit the link and finish this section.

▼ What are **callbacks** and how do we use them?

Callbacks are called in between epochs .They are used primarily to save models

```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

▼ What is **early stopping** and why would we want to use it ?

Early stopping is used to stop the model training if we are monitoring the data and a certain metric isnt changed

Example :

```
>>> callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
>>> # This callback will stop the training when there is no improvement in
>>> # the loss for three consecutive epochs.
>>> model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
>>> model.compile(tf.keras.optimizers.SGD(), loss='mse')
>>> history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                     epochs=10, batch_size=1, callbacks=[callback],
...                     verbose=0)
>>> len(history.history['loss'])  # Only 4 epochs are run.
4
```

reference link : https://keras.io/api/callbacks/early_stopping/

▼ Why do we use model.fit()

We use it to train the model

▼ Why do we use model.predict()

We use it to predict the outcome

▼

Reshape

What does -1 represent in reshapte?

It calculates the rest dimension by default such that the size remains sae

Fro example :

```
>>> t = [[1, 2, 3],
...      [4, 5, 6]]
>>> tf.reshape(t, [-1])
<tf.Tensor: shape=(6,), dtype=int32,
  numpy=array([1, 2, 3, 4, 5, 6], dtype=int32)>
>>> tf.reshape(t, [3, -1])
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
  array([[1, 2],
         [3, 4],
         [5, 6]], dtype=int32)>
>>> tf.reshape(t, [-1, 2])
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
  array([[1, 2],
         [3, 4],
         [5, 6]], dtype=int32)>
```

https://stackoverflow.com/questions/41848660/why-the-negative-reshape-1-in-mnist-tutorial/41848962

Simple Image Classifier using VGG or Densenet

Main file :

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/06bb5b5a-2ec5-4e89-8e2a-e712b606c103/Copy_of_image_classification_part1.ipynb

Model Creation

▼ Expand

Binary Classifier vs normal Softmax classifiers:

```
# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x=model.output
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units
x = layers.Dense(512, activation='relu')(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(2, activation='softmax')(x)

# Create model:
# input = input feature map
# output = input feature map + stacked convolution/maxpooling layers + fully
# connected layer + sigmoid output layer
model = Model(model.input, output)
```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

What is wrong with the above snipperts of code?

Answer:

> I had a similar issue when using model.fit(..). Turns out my output_size was defined as 2 while using "binary_crossentropy" as the loss function, when it should have been defined as 1.

We CANT use binary cross entropy for multi class predictions . We need to use "categorical croos_entropy"

```
model_final.compile(loss = "categorical_crossentropy",
optimizer = tf.keras.optimizers.SGD(lr=0.00001, momentum=0.9), metrics=["accuracy"])
```

We SHOULD ALSO CHANGE generator to categorical

```
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

to

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
```

```
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='categorical')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')
```

Downloading weights of imagenet

▼ Code to download **VGG net** (img_height,img_width,color_channels)

```
from keras.applications.vgg16 import VGG16
# load the model
model = VGG16(weights = "imagenet", include_top=False, input_shape = (256, 256, 3))
```

▼ Code to download **Densenet**

```
from tensorflow.keras import layers
from tensorflow.keras import Model
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers import Dense, Activation, Dropout, Flatten
from keras import optimizers
from keras.models import Sequential,load_model
from keras.preprocessing.image import ImageDataGenerator,load_img,img_to_array
import numpy as np
from keras.callbacks import ModelCheckpoint, History
import os
import urllib
import matplotlib.pyplot as plt
%matplotlib inline
import json
import seaborn as sns
from keras import applications
from keras.models import Sequential, Model
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
import tensorflow as tf

img_width = 256
img_height = 256
model = tf.keras.applications.DenseNet201(weights = "imagenet", include_top=False, input_shape = (img_width, img_height, 3))
```

Attaching net (VGG or densenet) with classification layer

Once we have downloaded the VGG net can we start classification right off the bat?

No , In order to convert our VGG or densenet to a classifier , we need to attach a fully connected layer.

▼ Code

```
# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x=model.output
x = Flatten()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(256, activation="relu")(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(2, activation='softmax')(x)

# Create model:
# input = input feature map
# output = input feature map + stacked convolution/maxpooling layers + fully
```

```
# connected layer + sigmoid output layer
model = Model(model.input, output)
```

Model Training

▼ Code

```
x=model.output
x = Flatten()(x)
x = Dense(128, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(128, activation="relu")(x)
predictions = Dense(2, activation="softmax")(x)

model_final = Model(inputs = model.input, outputs = predictions)

model_final.compile(loss = "categorical_crossentropy", optimizer = tf.keras.optimizers.SGD(lr=0.00001, momentum=0.9), metrics=["accurac

train_datagen = ImageDataGenerator(
rescale = 1./128,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
shear_range=0.2,
rotation_range=40)

test_datagen = ImageDataGenerator(
rescale = 1./128,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
shear_range=0.2,
rotation_range=40)

train_generator = train_datagen.flow_from_directory(
train_dir,
target_size = (img_height, img_width),
batch_size = 8,
class_mode = "categorical")

validation_generator = test_datagen.flow_from_directory(
validation_dir,
target_size = (img_height, img_width),
class_mode = "categorical")

checkpoint = ModelCheckpoint("Location_densenet_final.h5", monitor='accuracy', verbose=1, save_best_only=True, save_weights_only=False,

#early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')

model_final.fit_generator(
train_generator,
steps_per_epoch  = train_generator.samples // 8,
epochs = 10,
validation_data = validation_generator,
validation_steps  = validation_generator.samples // 8,
callbacks = [checkpoint]) #,early
```

Model Output

How to check the model for random image in a dataset?

▼ Code

```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img
```

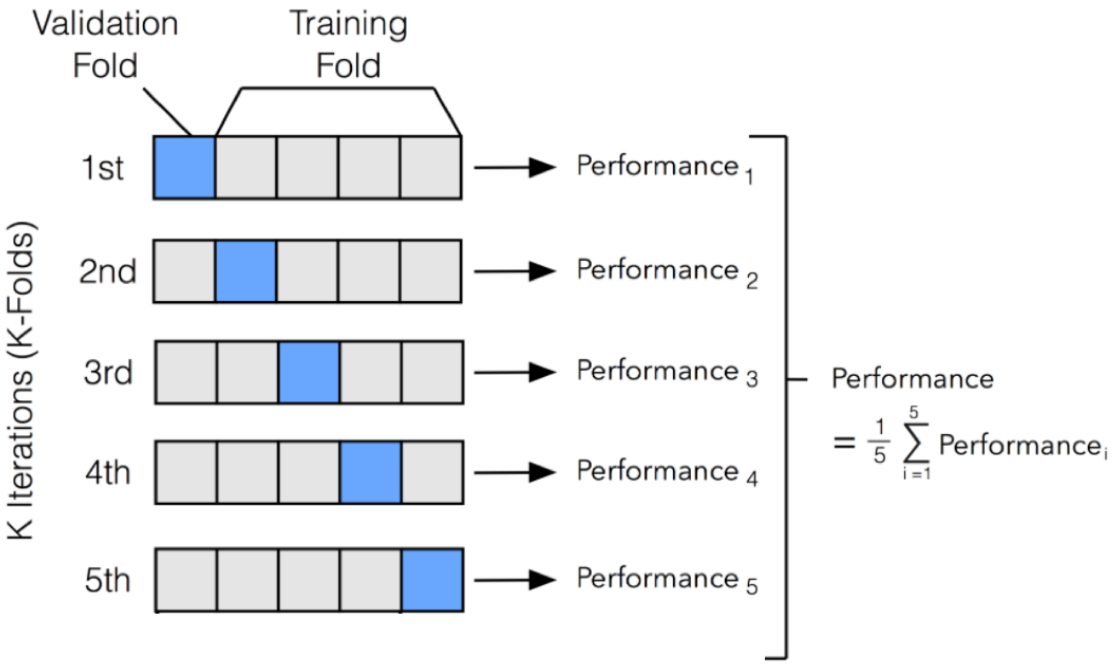
```
# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
# successive_outputs = [layer.output for layer in model.layers[1:]]
# visualization_model = Model(img_input, successive_outputs)

# Let's prepare a random input image of a cat or dog from the training set.
cat_img_files = [os.path.join(train_cats_dir, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(train_dogs_dir, f) for f in train_dog_fnames]
img_path = random.choice(cat_img_files + dog_img_files)

img = load_img(img_path, target_size=(256, 256))  # this is a PIL image
x = img_to_array(img)
x = x.reshape((1,) + x.shape)/128

img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(256, 256))
x = img_to_array(img)
x = x.reshape((1,) + x.shape)/128
pred = model_final.predict(x)
plt.imshow(img)
# print(pred)
pred_label = np.argmax(pred, axis=1)
key_dict = {0:"Cats",1:"Dogs"}
# print(pred_label)
print(key_dict[pred_label[0]])
# print(key_dict[pred_label])
```

▼ **What is cross Validation?**



It is a statistical method that is used to find the performance of machine learning models. It is used to protect our model against **overfitting** in a predictive model, particularly in those cases where the amount of data may be limited.

In cross-validation, we partitioned our dataset into a fixed number of folds (or partitions), run the analysis on each fold, and then averaged the overall error estimate.

Getting started with tensorboard Visualization

Marking accuracy of training and testing data

Basic File

https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb

Instructions to get tensorboard visul

https://neptune.ai/blog/tensorboard-tutorial

Logging custom Loss using Tensorflow

Link : https://www.tensorflow.org/tensorboard/scalars_and_keras

Tesnorflow : Udemy

Transfer learning : Theory

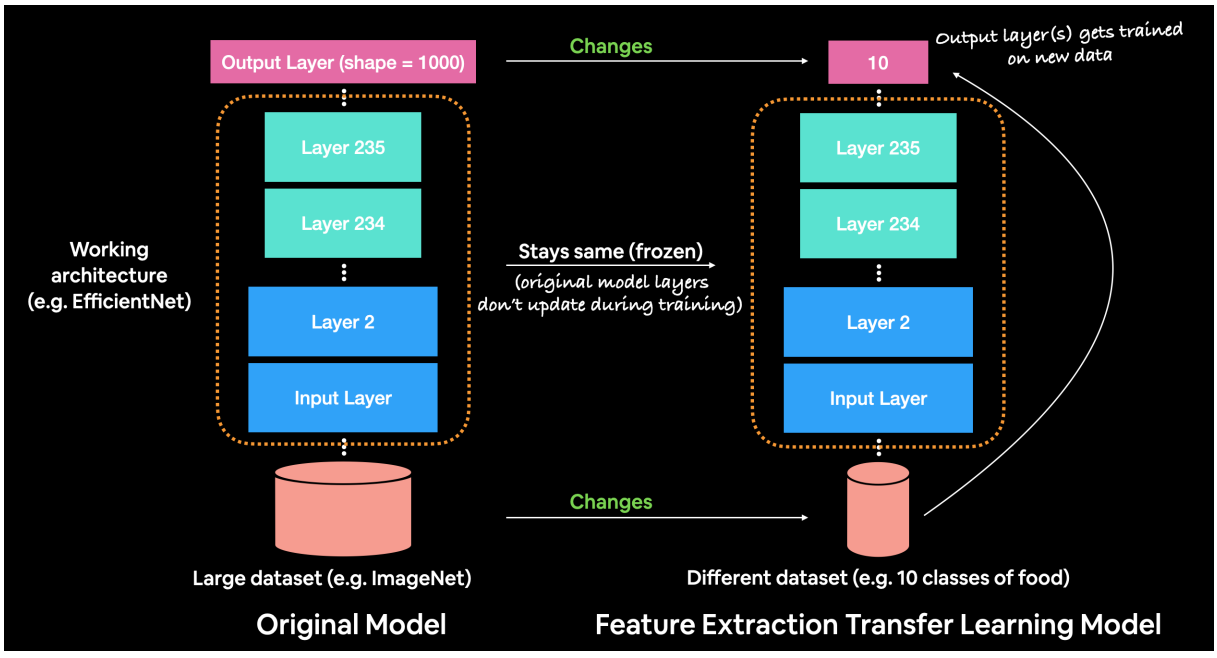
▼ What is transfer learning ?

Transfer learning is creating a model from an already pre-existing model architecture

One famous example is **imagenet** (Computer vision)

▼ What would be the difference between the ofifical imagenet model and our model?

The difference would be the **output** category . In the official one there would probably be 1000 classes , but in our cases , we might not require those many classes .So we only **change the output layer**



Transfer learning : Practical

To check the main code for practice visit the following link : https://colab.research.google.com/github/mrdbourke/tensorflow-deep-learning/blob/main/04_transfer_learning_in_tensorflow_part_1_feature_extraction.ipynb#scrollTo=Nq4kxIpQMpZT

▼ How can we check if we are running code on GPU?

```
!nvidia-smi
```

```
# Are we using a GPU?
!nvidia-smi
```

```
Fri Feb 12 03:39:41 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

▼ What is code to unzip files programatically via python?

```
# Unzip the downloaded file
zip_ref = zipfile.ZipFile("10_food_classes_10_percent.zip", "r")
zip_ref.extractall()
zip_ref.close()
```

▼ Code to check number of **images that come under different classes**, based on file structure (Using ImageDataGenerator)

```
# We need to replace the file location here to the one where we want.
train_dir = '/content/drive/MyDrive/CarLocationData/training'
validation_dir = '/content/drive/MyDrive/CarLocationData/validation'
img_height = 256
img_width = 256
train_datagen = ImageDataGenerator(
rescale = 1./255,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
shear_range=0.2,
rotation_range=40)

test_datagen = ImageDataGenerator(
rescale = 1./255,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
shear_range=0.2,
rotation_range=40)
train_generator = train_datagen.flow_from_directory(
train_dir,
target_size = (img_height, img_width),
batch_size = 10,
class_mode = "categorical")

validation_generator = test_datagen.flow_from_directory(
validation_dir,
target_size = (img_height, img_width),
class_mode = "categorical")
```

```
Found 979 images belonging to 3 classes.
Found 171 images belonging to 3 classes.
```

▼ What are callbacks and how many kinds of callbacks are there?

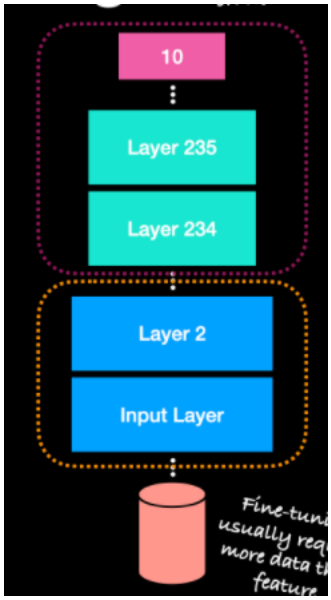
Callbacks are extra functionality you can add to your models to be performed during or after training. Some of the most popular callbacks include:

- **Experiment tracking with TensorBoard** - log the performance of multiple models and then view and compare these models in a visual way on TensorBoard (a dashboard for inspecting neural network parameters). Helpful to compare the results of different models on your data.
- **Model checkpointing** - save your model as it trains so you can stop training if needed and come back to continue off where you left. Helpful if training takes a long time and can't be done in one sitting.
- **Early stopping** - leave your model training for an arbitrary amount of time and have it stop training automatically when it ceases to improve. Helpful when you've got a large dataset and don't know how long training will take.

▼ When given a pretrained model which part of the model do we focus our training on?

Short answer : Only the top 2 or 3 layers

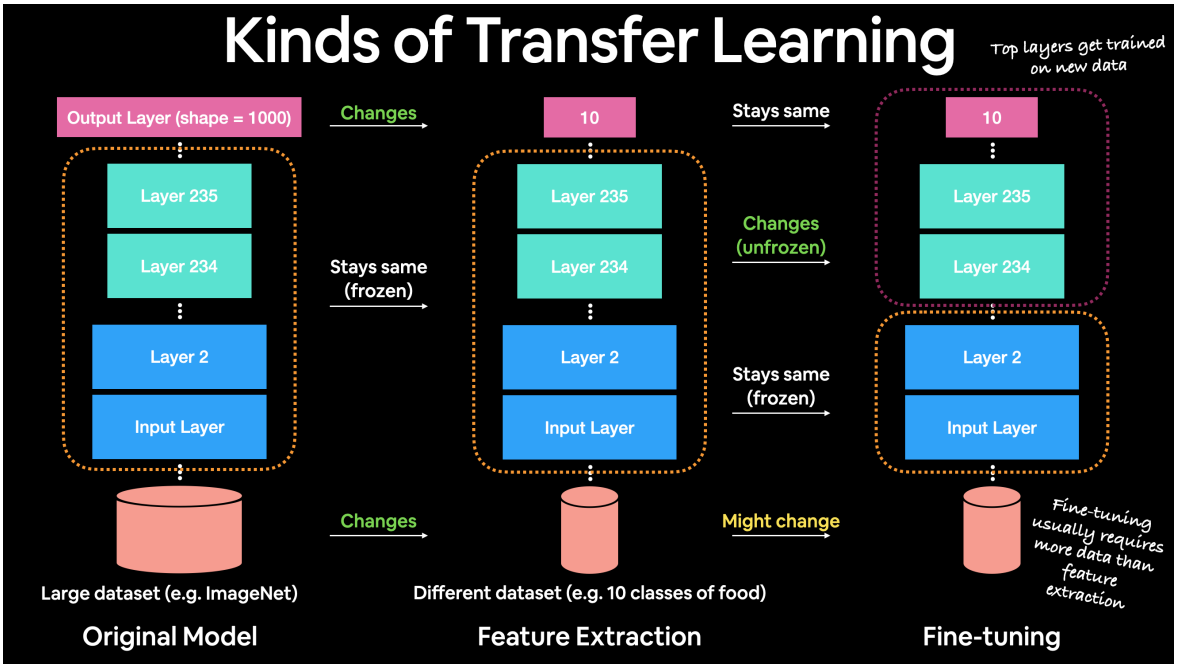
Short answer : the top layer , or the layer closer to output layer



Additional note : A common workflow is to "freeze" all of the learned patterns in the bottom layers of a pretrained model so they're untrainable. And then train **the top 2-3 layers of so the pretrained model can adjust its outputs to your custom data** (feature extraction).

▼ What is the **lifecycle of training a model** ? How can we use a pretrained model and ine tune it further and further?

first we train only near the ouput layer , then we move on the fine tuning where we train a bunch of layers that are assoscaited with the ouput layer.(Example: 234, 235 etc)



▼

Back to coding

▼ Code to create a **classifier model** from pre-existing models available **online**.

Model creation function with variable classes

```
def create_model(model_url, num_classes=10):
  """Takes a TensorFlow Hub URL and creates a Keras Sequential model with it.

  Args:
    model_url (str): A TensorFlow Hub feature extraction URL.
    num_classes (int): Number of output neurons in output layer,
      should be equal to number of target classes, default 10.

  Returns:
    An uncompiled Keras Sequential model with model_url as feature
    extractor layer and Dense output layer with num_classes outputs.
  """
  # Download the pretrained model and save it as a Keras layer
  feature_extractor_layer = hub.KerasLayer(model_url,
                                           trainable=False, # freeze the underlying patterns
                                           name='feature_extraction_layer',
                                           input_shape=IMAGE_SHAPE+(3,)) # define the input image shape

  # Create our own model
  model = tf.keras.Sequential([
    feature_extractor_layer, # use the feature extraction layer as the base
    layers.Dense(num_classes, activation='softmax', name='output_layer') # create our own output layer
  ])

  return model
```

Model Link→ Create model → compile loss

```
# Resnet 50 V2 feature vector
resnet_url = "https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/4"

# Original: EfficientNetB0 feature vector (version 1)
```

```
efficientnet_url = "https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1"


# Create model
resnet_model = create_model(resnet_url, num_classes=train_data_10_percent.num_classes)

# Compile
resnet_model.compile(loss='categorical_crossentropy',
                     optimizer=tf.keras.optimizers.Adam(),
                     metrics=['accuracy'])
```

Note that we can also tweak the number of output classes in this case which is awesome.

```
# Download the pretrained model and save it as a Keras layer
feature_extractor_layer = hub.KerasLayer(model_url,
                                         trainable=False, # freeze the underlying patterns
                                         name='feature_extraction_layer',
                                         input_shape=IMAGE_SHAPE+(3,)) # define the input image shape
```

▼ In the above code why are we setting trainable to false?

We are doing this to freeze the **level of abstraction for objects is retained .**

Lower level features have generalised traits , something common to majority of objects

Whereas higher level features are more **specific** to the kind of objects that we want to train.Hence we are setting higher level trainable to false.

> For example, a bottom layer in a computer vision model to identify images of cats or **dogs might learn the outline of legs, where as, layers closer to the output might learn the shape of teeth**. Often, you'll want the larger features (learned patterns are also called features) to remain, since these are similar for both animals, where as, the differences remain in the more fine-grained features.

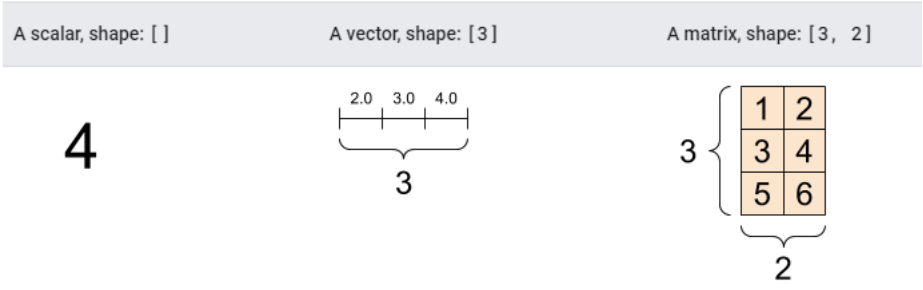
▼ What does IMAGE_SHAPE+(3,) mean?

Appending to tuple

```
(224,224)+(3,)=(224,224,3)
```

▼

What is the difference between a scalar , vector and a matrix?



Assertions using tensorflow

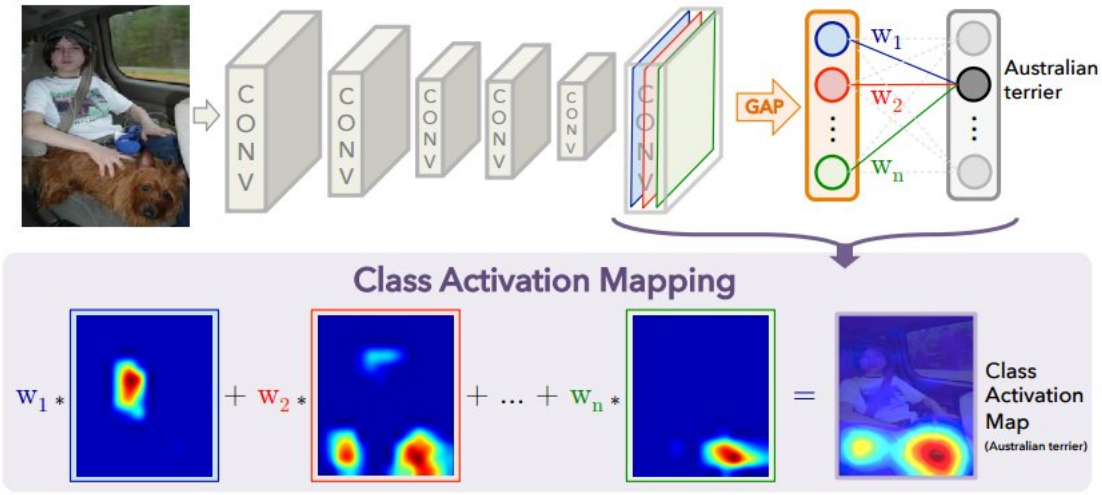
How do we perform assertions using tensorflow?

```
asserts = [
        tf.Assert(tf.greater(tf.shape(input=proposals)[0], 0), [proposals],
                  name="roi_assertion"),
    ]
    with tf.control_dependencies(asserts):
        proposals = tf.identity(proposals)
```

Visulization via heatmaps

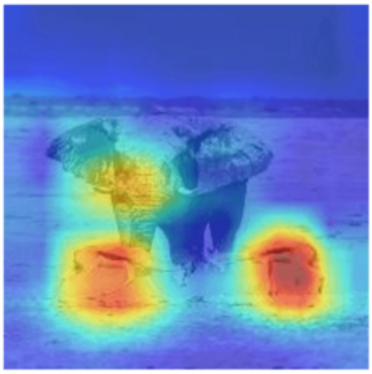
Link to follow : https://towardsdatascience.com/visualizing-neural-networks-decision-making-process-part-1-class-activation-maps-cams-bd1a218fa977?source=follow_footer-----5887a6aa135d----0----------------------------

Class activation maps And

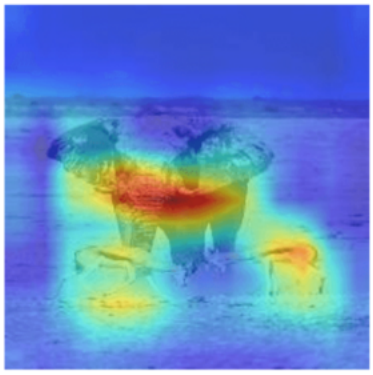


We can get class activation maps

Gradient Weighted Class Activation Mapping (Grad-CAM)



gazelle



elephant