```
import os
import numpy as np
import pandas as pd
from PIL import ImageOps
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
from sklearn.metrics import confusion_matrix, accuracy_score
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import DirectoryIterator, ImageDataGenerator
from tensorflow.keras.utils import load_img , img_to_array
from tensorflow.keras.layers import Input
import tensorflow as tf
from tensorflow.python.keras import Model
from tensorflow.python.keras.layers import Dense, Activation
from tensorflow.python.keras.utils.vis_utils import plot_model
from tensorflow import keras
from keras.layers import Reshape, Dropout , AveragePooling2D , Flatten
from tensorflow.keras.layers import Dense
!pip install wget
!pip install imutils
import wget
import cv2
from imutils import paths
import tarfile
import glob
import matplotlib.image as mpimg
%matplotlib inline
import shutil
from tensorflow.keras import datasets, layers, models
import tensorflow_hub as hub
import datetime
from tensorflow.keras.utils import Sequence
from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import AveragePooling2D
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: wget in /usr/local/lib/python3.7/dist-packages (3.2)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: imutils in /usr/local/lib/python3.7/dist-packages (0.5.4)
```

```
%load_ext tensorboard
```

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

To start, we firstly have to download the corresponding files for this assignment. We will download the data using a url and then unzip it, because it is compressed.

```
_URL = 'http://image.ntua.gr/iva/datasets/flickr_logos/flickr_logos_27_dataset.tar.gz'
wget.download(_URL)
```

```
'flickr_logos_27_dataset.tar (8).gz'
```

```
zip_dir = tf.keras.utils.get_file('./logo', origin=_URL, untar=True,extract=True)
```

After downloading the data and decompressing, we will open the files of text and images using an 'openfile' function :

```
def openfile(fname):
  if fname.endswith("tar.gz"):
     tar = tarfile.open(fname, "r:gz")
     tar.extractall()
     tar.close()
```

```
fname1 = './flickr_logos_27_dataset.tar.gz'
fname2 = './flickr_logos_27_dataset/flickr_logos_27_dataset_images.tar.gz'

openfile(fname1)
openfile(fname2)
```

Data Preparation

```
train_images = pd.read_csv("./flickr_logos_27_dataset/flickr_logos_27_dataset_training_set_annotation.txt",
                           sep='\s+',
                           usecols = [i for i in range(7)],
                           header=None,
                           names=["file_name",'label','subset','x1','y1','x2','y2'])
train_images.head()
```

	file_name	label	subset	x1	y1	x2	y2
0	144503924.jpg	Adidas	1	38	12	234	142
1	2451569770.jpg	Adidas	1	242	208	413	331

```
train_images.shape
```

```
(4536, 7)
```

```
train_images.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4536 entries, 0 to 4535
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   file_name  4536 non-null   object
 1   label      4536 non-null   object
 2   subset     4536 non-null   int64
 3   x1         4536 non-null   int64
 4   y1         4536 non-null   int64
 5   x2         4536 non-null   int64
 6   y2         4536 non-null   int64
dtypes: int64(5), object(2)
memory usage: 248.2+ KB
```

```
test_images = pd.read_csv("./flickr_logos_27_dataset/flickr_logos_27_dataset_query_set_annotation.txt",
                          sep='\t',
                           usecols = [i for i in range(2)],
                          header=None,
                          names=["file_name",'label'])
test_images
```

	file_name	label
0	2403695909.jpg	Adidas
1	2912587920.jpg	Adidas
2	3441398196.jpg	Adidas
3	4605630935.jpg	Adidas
4	4606245138.jpg	Adidas
...	...	...
265	3480640208.jpg	none
266	3486224308.jpg	none
267	3486430785.jpg	none
268	3490185235.jpg	none
269	3490913574.jpg	none

270 rows × 2 columns

```
test_images.shape
```

```
(270, 2)
```

As we can see, some photos have no label, so lets procede to count how many labeled images the test dataframe contains.

```
test_images.loc[test_images['label'] != 'none'].shape
```

```
(135, 2)
```

As we can observe, only 135 images are labeled.

```
test_images.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   file_name  270 non-null    object
 1   label      270 non-null    object
dtypes: object(2)
memory usage: 4.3+ KB
```

As we can see, there are no null values on neither the train nor the test dataset.

Furthermore, we will display the labels :

```
Labels= list(set(list(train_images.label)))
Labels.sort()
print('The number of classes are : ' , len(Labels) , '\n\nLabels : \n')
Labels
```

```
The number of classes are :  27

Labels :

['Adidas',
 'Apple',
 'BMW',
 'Citroen',
 'Cocacola',
 'DHL',
 'Fedex',
 'Ferrari',
 'Ford',
 'Google',
 'HP',
 'Heineken',
 'Intel',
 'McDonalds',
 'Mini',
 'Nbc',
 'Nike',
 'Pepsi',
 'Porsche',
 'Puma',
 'RedBull',
 'Sprite',
 'Starbucks',
 'Texaco',
 'Unicef',
 'Vodafone',
 'Yahoo']
```

Moreover, we will check for duplicated data entries, specifically in the 'file_name' column :

```
duplicate = test_images[test_images.duplicated('file_name')]

print("Duplicate Rows in Test df:\n" )
duplicate
```

```
Duplicate Rows in Test df:
```

file_name	label

```
duplicate = train_images[train_images.duplicated('file_name')]

print("Duplicate Rows in Train df:\n" )
duplicate
```

```
Duplicate Rows in Train df:
```

	file_name	label	subset	x1	y1	x2	y2
5	4763210295.jpg	Adidas	1	91	288	125	306
6	4763210295.jpg	Adidas	1	182	63	229	94
7	4763210295.jpg	Adidas	1	192	291	225	306
8	4763210295.jpg	Adidas	1	285	61	317	79
9	4763210295.jpg	Adidas	1	285	298	324	329
...	...	...	...	...	...	...	...
4531	2126991906.jpg	Yahoo	6	15	6	253	54
4532	217288720.jpg	Yahoo	6	136	161	304	222
4533	2472817996.jpg	Yahoo	6	2	4	499	106
4534	2514220918.jpg	Yahoo	6	1	69	342	157
4535	386891249.jpg	Yahoo	6	156	10	310	49

3727 rows × 7 columns

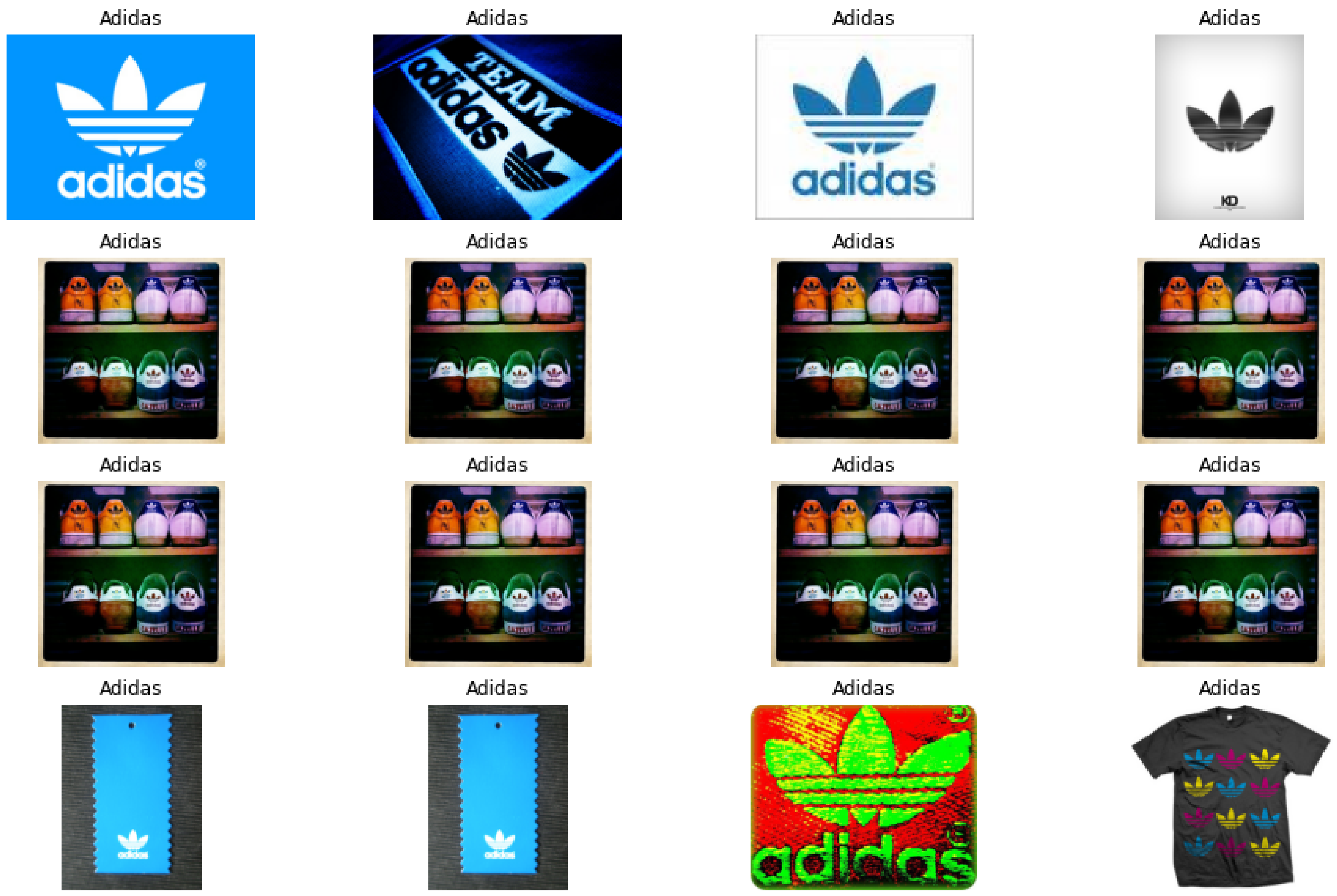
It is clear that eventhough no duplicates have been found in the test dataframe, a lot exist in the train dataframe, (and that can also be seen from the visualization below). To specify, in the training dataset there are only 809 unique images!

(the total number of rows (4536) minus the number of duplicated rows (3727))

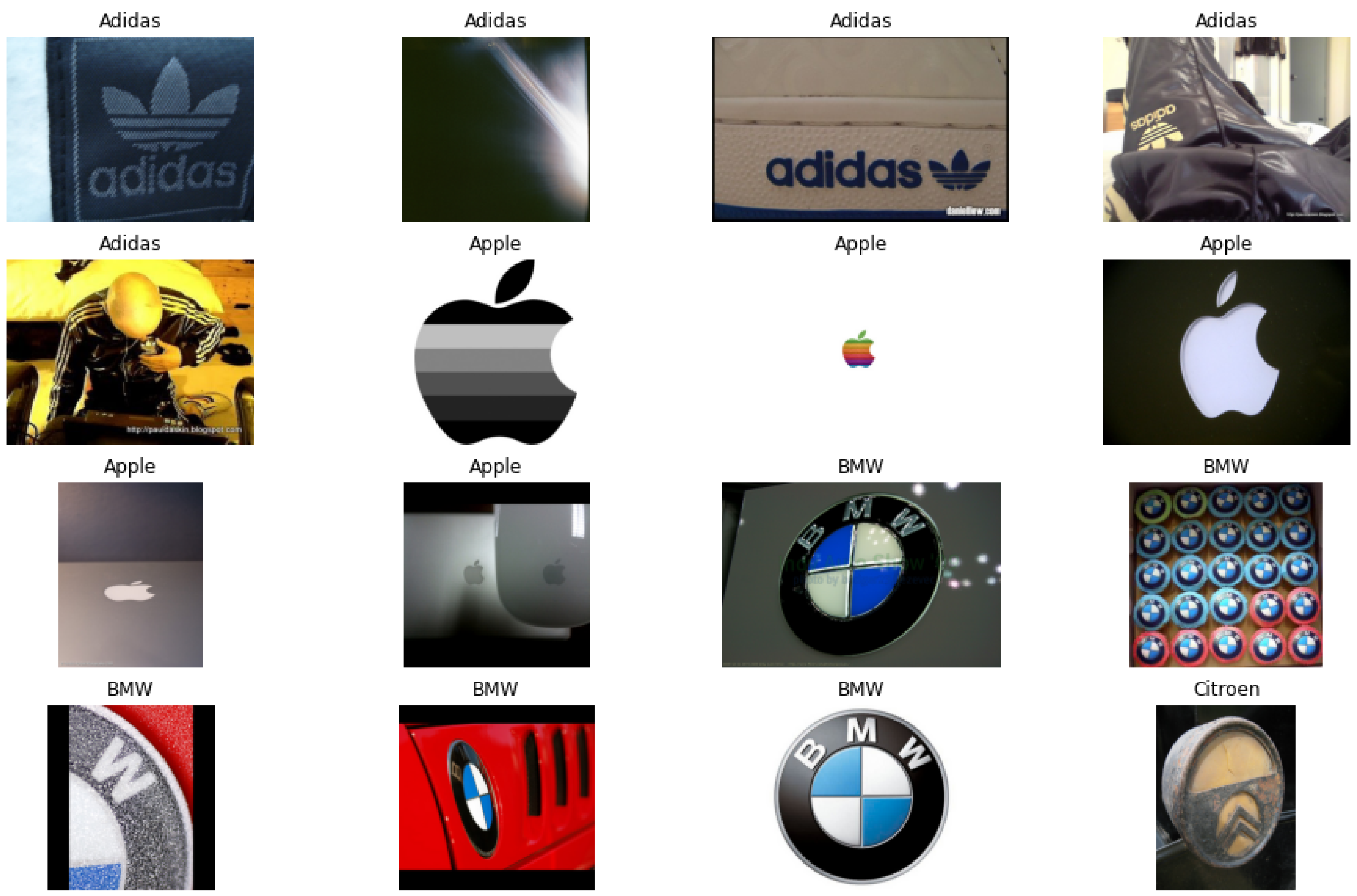
Perform some visualizations :

```
# Training images

plt.figure(figsize=(16, 10))
```

```
for idx in range(16):
    ax = plt.subplot(4, 4, idx + 1)
    plt.imshow(Image.open('./flickr_logos_27_dataset_images/'+ train_images.loc[idx,'file_name']))
    plt.title(train_images.loc[idx,'label'])
    plt.axis("off")
```




```
# Test images

plt.figure(figsize=(16, 10))
for idx in range(16):
    ax = plt.subplot(4, 4, idx + 1)
    plt.imshow(Image.open('./flickr_logos_27_dataset_images/'+ test_images.loc[idx,'file_name']))
    plt.title(test_images.loc[idx,'label'])
    plt.axis("off")
```



Before we move on to Data Preprocessing ,lets take a look of the sizes of the first 5 images saved in the train dataset:

```
size = train_images.iloc[:,3:]
size.head()
```

	x1	y1	x2	y2
0	38	12	234	142
1	242	208	413	331

As we can see, the images vary in size, so we have to reshape them.

Data Preprocessing

Creation of paths for the augmented images of the train and test dataframes to be saved

```
try:
    shutil.rmtree("Train")
    if not os.path.exists('Train'):
           os.makedirs('Train')
    for i in Labels:
            os.makedirs(os.path.join('Train',i))

except FileNotFoundError:
      print("Oops!  Something went wrong.  Try again...")
      if not os.path.exists('Train'):
           os.makedirs('Train')
      for i in Labels:
            os.makedirs(os.path.join('Train',i))
```

```
try:
    shutil.rmtree("Test")
    if not os.path.exists('Test'):
           os.makedirs('Test')
    for i in Labels:
            os.makedirs(os.path.join('Test',i))

except FileNotFoundError:
      print("Oops!  Something went wrong.  Try again...")
      if not os.path.exists('Test'):
           os.makedirs('Test')
      for i in Labels:
            os.makedirs(os.path.join('Test',i))
```

Reshape and remove corrupted images :

```
size = size.values.tolist()
```

```
for i in range(len(train_images.iloc[:,0])):
    try:
        destrain = os.path.join('Train',train_images.iloc[:,1][i])
        savepath = os.path.join(destrain,train_images.iloc[:,0][i])
        img  = os.path.join('./flickr_logos_27_dataset_images/',train_images.iloc[:,0][i])
        image = cv2.imread(img)
        image = image[size[i][1]:size[i][3],size[i][0]:size[i][2]]
        image = cv2.resize(image,(224,224))
        cv2.imwrite(savepath,image)
    except:
        print('error')
        pass
```

```
error
error
error
error
error
```

```
for i in range(len(test_images.iloc[:,0])):
    try:
        destrain = os.path.join('Test',test_images.iloc[:,1][i])
        savepath = os.path.join(destrain,test_images.iloc[:,0][i])
        img  = os.path.join('./flickr_logos_27_dataset_images/',test_images.iloc[:,0][i])
        image = cv2.imread(img)
        image = cv2.resize(image,(224,224))
        cv2.imwrite(savepath,image)
    except:
        print('error')
        pass
```

Now, lets check the result of our changes :

```
imagePaths = list(paths.list_images('Train'))
```

```
rand_image = Image.open(imagePaths[10])
rand_image
```

```
rand_image = np.array(rand_image)
rand_image.shape
```

```
(224, 224, 3)
```

Image augmentation

Before we build and train a CNN model, we have to prevent potential overfitting and increase the model's performance. This could be achieved via Image Augmentation, where you apply various changes to the initial data to create more data for training. Following this strategy, we will construct a data augmentation function using the Keras 'ImageDataGenerator' class.

```
# We create a path to the directory 'keras_augmentations',
# where augmented images will be stored for us to preview the results of the image augmentation procedure.

try:
      shutil.rmtree("keras_augmentations")
      if not os.path.exists('keras_augmentations'):
         os.makedirs('keras_augmentations')

except FileNotFoundError:

      print("Oops!  Something went wrong.  Try again...")

      if not os.path.exists('keras_augmentations'):
         os.makedirs('keras_augmentations')
```

For the Data augmentation, we will do:

- Rescale
- Rotation
- Zooming
- Horizontal / Vertical flip
- Adjust brightness range
- Shift the width / height

```
Image_generator = ImageDataGenerator(rescale = 1/255, #normalization
                                     rotation_range = 40,
                                     shear_range = 0.2,
                                     zoom_range = 0.2,
                                     horizontal_flip = True,
                                     vertical_flip=True,
                                     brightness_range = (0.5, 1.5),
                                     width_shift_range=0.2,
                                     height_shift_range=0.2,
                                     validation_split = 0.2,
                                     featurewise_center=True,
                                     featurewise_std_normalization=True) # Divide inputs by std of the dataset, feature-wise
```

Lets see the outcome by selecting a random image and then plotting the image augmentation results:

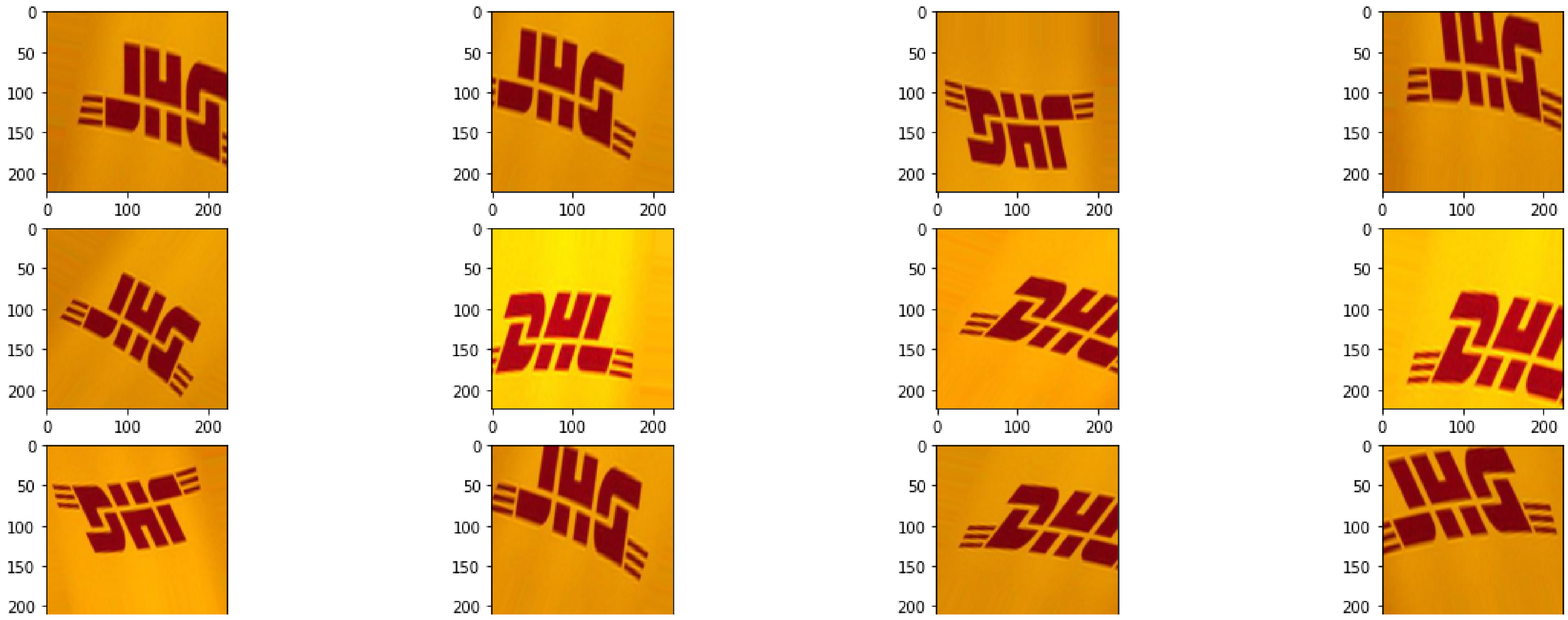
```
rand_image = Image.open(imagePaths[60])
rand_image
# This is a random image of the Train dataset
```

```
x = img_to_array(rand_image)  # shape (3, 224, 224)
x = x.reshape((1,) + x.shape)  # shape (1, 3, 224, 224)

#the code below generates batches of randomly transformed images and saves the results to the `keras_augmentations` directory
i = 0
for batch in Image_generator.flow(x, batch_size=1,
                          save_to_dir='keras_augmentations', save_prefix='yh', save_format='jpeg'):
    i += 1
    if i > 11:   #we want 12 augmented photos
        break
```

```
images = os.listdir('keras_augmentations')
images
```

```
['yh_0_8950.jpeg',
 'yh_0_7619.jpeg',
 'yh_0_4778.jpeg',
 'yh_0_9760.jpeg',
 'yh_0_4325.jpeg',
 'yh_0_27.jpeg',
 'yh_0_7378.jpeg',
 'yh_0_9033.jpeg',
 'yh_0_8690.jpeg',
 'yh_0_3335.jpeg',
 'yh_0_2328.jpeg',
 'yh_0_278.jpeg']
```

```
# Let's look at the augmented images
aug_images = []
for img_path in glob.glob('keras_augmentations/*.jpeg'):
    aug_images.append(mpimg.imread(img_path))

plt.figure(figsize=(20,10))
columns = 4
for i, image in enumerate(aug_images):
    plt.subplot(len(aug_images) / columns + 1, columns, i + 1)
    plt.imshow(image)
```



Creating dataset and splitting into training and validation sets

We will achieve data augmentation using Keras functions. Firstly, the images will be read from folders containing images with the 'flow_from_directory()' function and then the images will be included in the data pipeline with a function like ImageDataGenerator.

```
#The 'flow_from_directory()' is a method to read the images from folders containing images.

Train = Image_generator.flow_from_directory('Train',
target_size = (224,224),
batch_size = 32,   # We set batch size equal to 32
shuffle=False,
seed=42,
color_mode='rgb', # color_mode='rgb' because the image has three color channels
subset = 'training',
class_mode='categorical')
```

```
Found 648 images belonging to 27 classes.
```

```
Validation = Image_generator.flow_from_directory('Train',
target_size = (224,224),
batch_size = 32,
shuffle=False,
seed=42,
color_mode='rgb',
subset = 'validation',
class_mode='categorical')
```

```
Found 161 images belonging to 27 classes.
```

809 unique photos distributed into Train and Validation set, 648 for Train and 161 for Validation

Now, we will calculate the steps per epoch --> the total number of images per directory divided by the batch size:

```
#Training steps

STEP_SIZE_TRAIN=Train.n//Train.batch_size
STEP_SIZE_TRAIN
```

```
20
```

```
#Validation steps

STEP_SIZE_VALID=Validation.n//Validation.batch_size
STEP_SIZE_VALID
```

```
5
```

Building a Convolutional Neural Network (CNN) for image classification

Firstly, we are going to try a CNN model and plot the accuracy. Our CNN model is going to include 11 layers, 4 Conv2D, 2 MaxPooling2D, an AveragePooling layer, a flatten layer, a dropout layer and lastly 2 dence layers.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
#Batch size = 32
model.add(layers.MaxPooling2D((5, 5)))
# Pooling layers are used to reduce the dimensions of the feature maps.
# Max pooling, in particular, is a pooling operation that selects the maximum element from the region of the feature map covered by the filter.
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((5, 5)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.AveragePooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
 # The flatten layer converts 3D arrays into 1D arrays to then be iserted into the dence layer
model.add(layers.Dense(64, activation='softmax'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(27))
 # Because the number of classes is 27
```

```
model.summary()
```

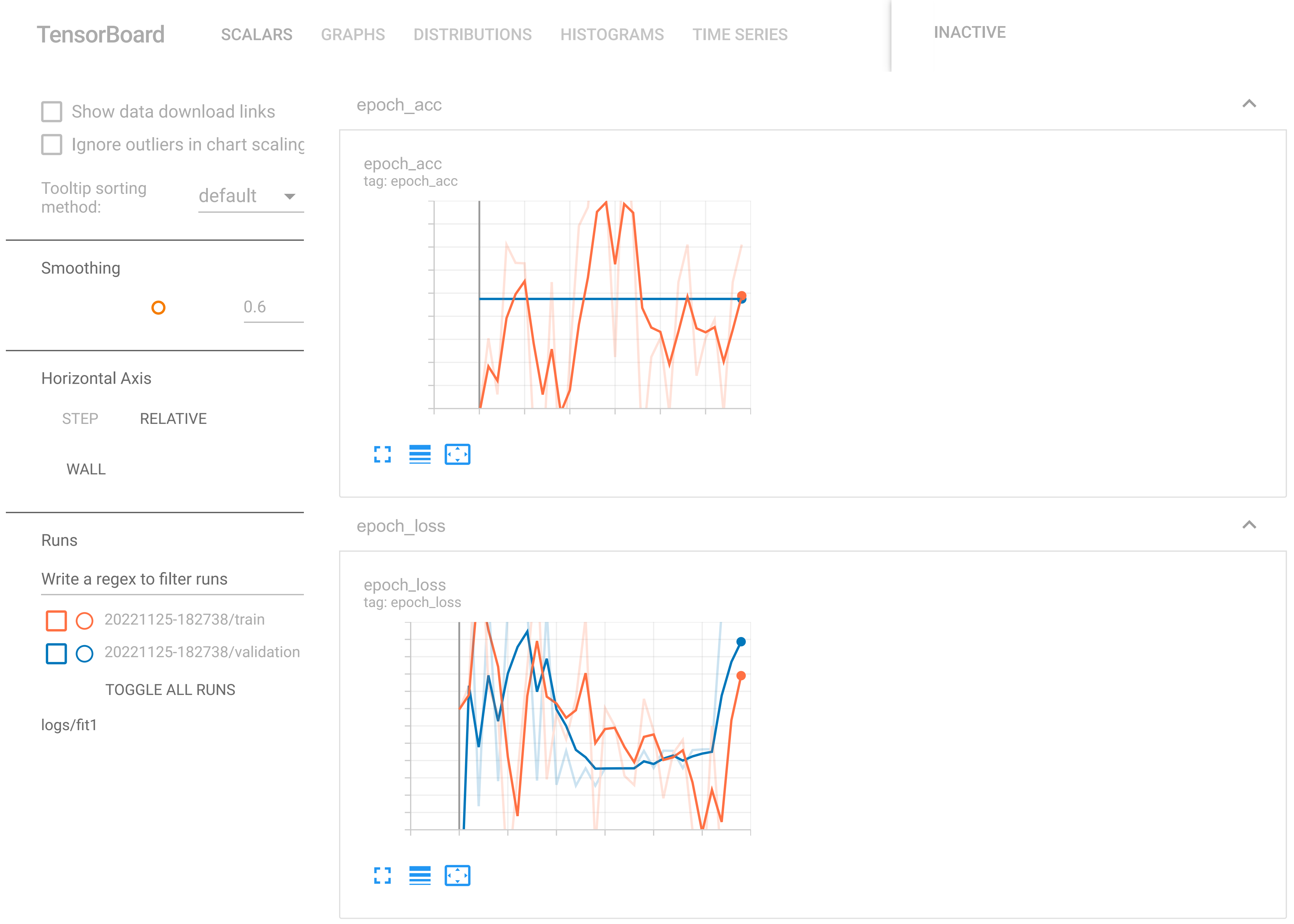
```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_76 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_30 (MaxPoolin  (None, 44, 44, 32)       0
 g2D)

 conv2d_77 (Conv2D)          (None, 42, 42, 64)        18496

 max_pooling2d_31 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_78 (Conv2D)          (None, 6, 6, 64)          36928

 average_pooling2d_19 (Avera  (None, 3, 3, 64)         0
 gePooling2D)

 conv2d_79 (Conv2D)          (None, 1, 1, 64)          36928

 flatten_9 (Flatten)         (None, 64)                0

 dense_50 (Dense)            (None, 64)                4160

 dropout_24 (Dropout)        (None, 64)                0

 dense_51 (Dense)            (None, 27)                1755

=================================================================
Total params: 99,163
Trainable params: 99,163
Non-trainable params: 0
_________________________________________________________________
```

Furthermore, We will use the Adam optimizer with a learning rate equal to 0.001. In addition, for the loss function we will use Categorical Crossentropy.

```
model.compile(
  optimizer=tf.keras.optimizers.Adam(0.001),  #Setting learning time equal to 0.001
  loss= 'categorical_crossentropy',
  metrics=['acc'])


log_dir = "logs/fit1/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.



model.fit(Train,
          validation_data = Validation,
          steps_per_epoch=STEP_SIZE_TRAIN,
          validation_steps=STEP_SIZE_VALID,
          epochs=30,
          callbacks = tensorboard_callback)
```

```
%tensorboard --logdir logs/fit1
```

```
Reusing TensorBoard on port 6006 (pid 8600), started 1:29:23 ago. (Use '!kill 8600' to kill it.)
```



As we can observe, the accuracy is near zero, thus we have to try out new methods. One method is Transfer Learning, i.e. the procedure in which an already trained model will be used as a layer of a neural network model for the classification of new unknown data (unknown classes).

Transfer Learning :

Try No1 :

```
# Finding the shapes
for image_batch, labels_batch in Train:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 224, 224, 3)
(32, 27)
```

32 batches x shape (224,224,3) --> 224x224 images with 3 color channels (red, green and blue)

32 batches x 27 classes

We will utilise the already trained model 'MobileNetV2', a model trained on the imagenet dataset. Then, We will additionally add a MaxPooling layer, a Flatten layer, 2 Dense Layers and a Dropout layer.

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

base_Model = tf.keras.applications.MobileNetV2(input_shape=(224,224,3),
                                               include_top=False,
                                               weights='imagenet')
head_Model = base_Model.output

Max_pooling_layer = tf.keras.layers.MaxPooling2D((5,5))
head_Model = Max_pooling_layer(head_Model)

Flatten_layer = tf.keras.layers.Flatten(name="flatten")
head_Model = Flatten_layer(head_Model)

Dense_layer1 = tf.keras.layers.Dense(128, activation="relu")
head_Model = Dense_layer1(head_Model)

Dropout_layer = tf.keras.layers.Dropout(0.5)
head_Model = Dropout_layer(head_Model)

Dense_layer2 = tf.keras.layers.Dense(27, activation="softmax")
head_Model = Dense_layer2(head_Model)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
Model = tf.keras.Model(base_Model.input , head_Model)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in base_Model.layers:
    layer.trainable = False
```

```
feature_batch = base_Model(image_batch)
print(feature_batch.shape)
```

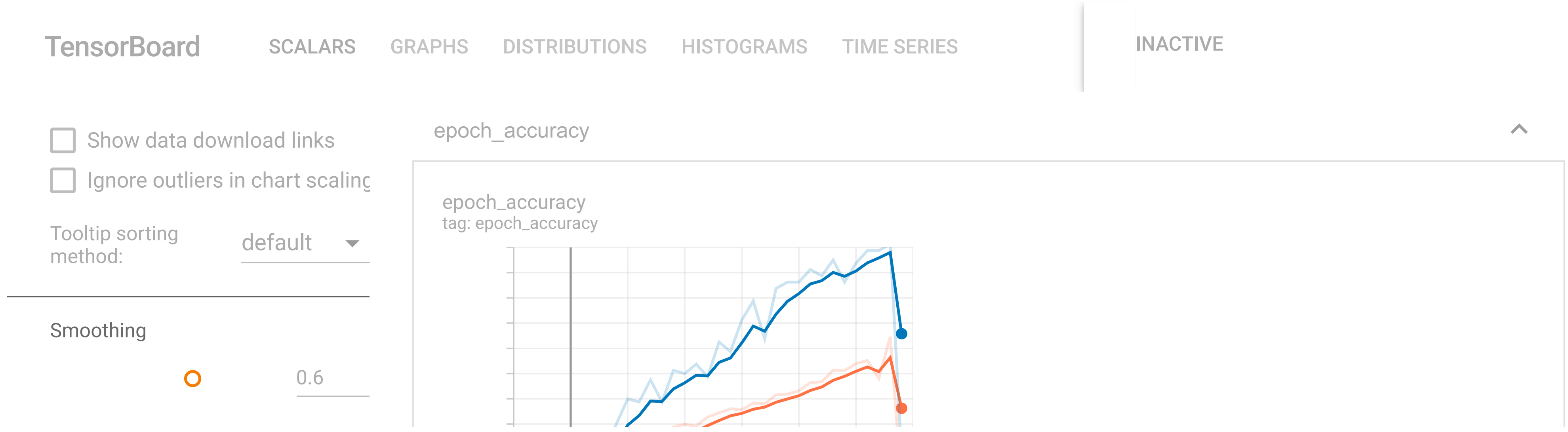
```
(32, 7, 7, 1280)
```

```
Model.compile(loss="categorical_crossentropy",
              optimizer='Adam',
              metrics=['accuracy'])


log_dir = "logs/fit9/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.
```

```
history = Model.fit(Train,
                    validation_data = Validation,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=30,
                    callbacks = tensorboard_callback)
```

```
%tensorboard --logdir logs/fit9
```

```
Reusing TensorBoard on port 6007 (pid 10402), started 1:15:25 ago. (Use '!kill 10402' to kill it.)
```



Fine tuning:

So far, We were only training a few layers on top of an MobileNetV2 base model, thus the weights of the pre-trained network were not updated during training. One way to increase performance even further is to train the weights of the top layers of the pre-trained model alongside the training of the classifier we added. This is called ***Fine Tuning***

```
# Un-freeze the top layers of the model:
base_Model.trainable = True
```

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

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_Model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  154
```

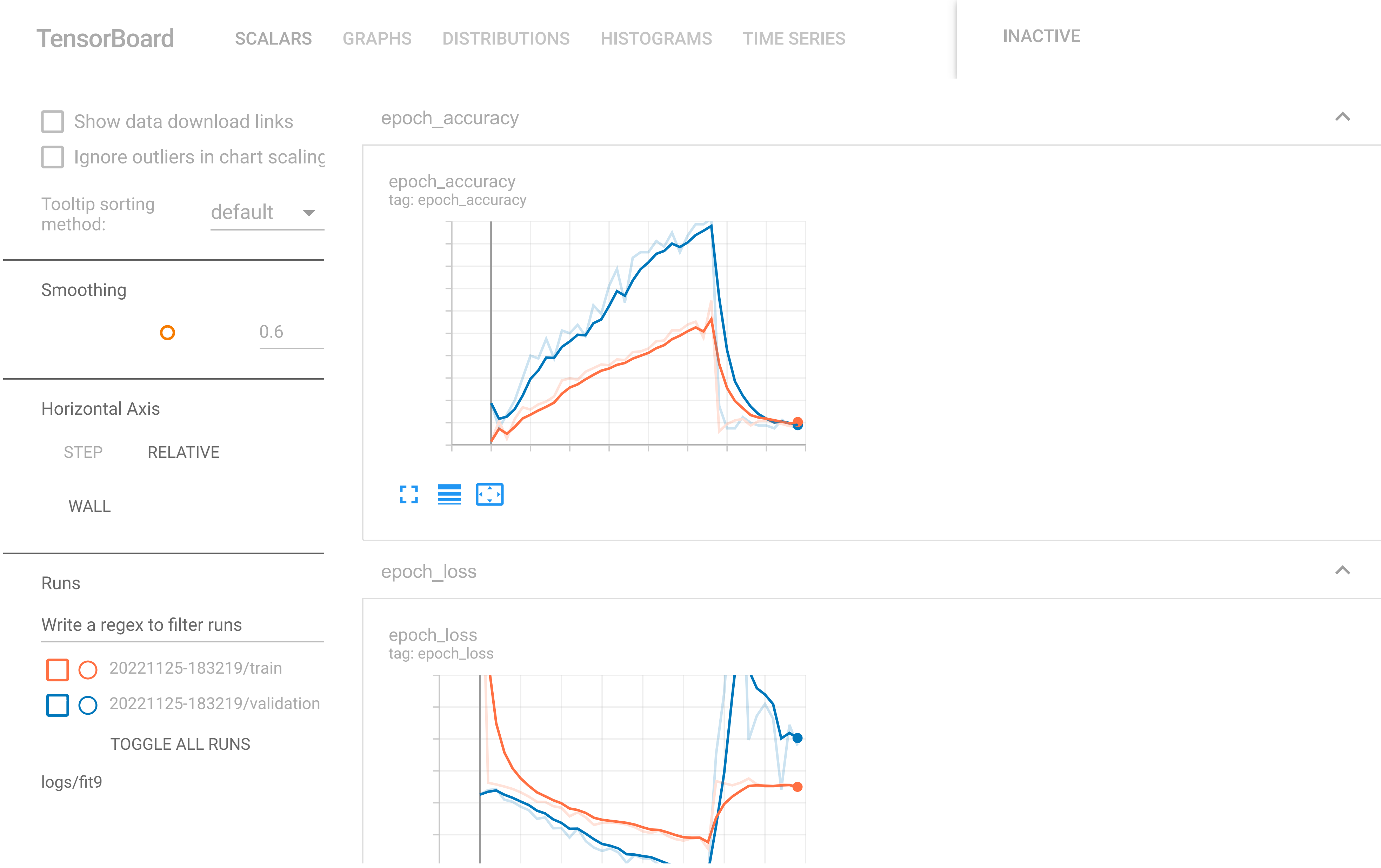
```
Model.compile(loss="categorical_crossentropy",
              optimizer='Adam',
              metrics=['accuracy'])
```

```
fine_tune_epochs = 10
total_epochs =  30 + fine_tune_epochs

Model.fit(Train,
          epochs=total_epochs,
          steps_per_epoch=STEP_SIZE_TRAIN,
          validation_steps=STEP_SIZE_VALID,
          initial_epoch=history.epoch[-1],
          validation_data=Validation,
          callbacks = tensorboard_callback)
```

```
Epoch 30/40
20/20 [==============================] - 15s 588ms/step - loss: 3.3393 - accuracy: 0.0308 - val_loss: 3.7649 - val_accuracy: 0.0875
Epoch 31/40
20/20 [==============================] - 10s 526ms/step - loss: 3.3051 - accuracy: 0.0471 - val_loss: 4.7289 - val_accuracy: 0.0375
Epoch 32/40
20/20 [==============================] - 11s 548ms/step - loss: 3.2744 - accuracy: 0.0552 - val_loss: 6.5089 - val_accuracy: 0.0375
Epoch 33/40
20/20 [==============================] - 11s 558ms/step - loss: 3.3150 - accuracy: 0.0584 - val_loss: 7.5014 - val_accuracy: 0.0625
Epoch 34/40
20/20 [==============================] - 11s 535ms/step - loss: 3.3801 - accuracy: 0.0438 - val_loss: 3.9829 - val_accuracy: 0.0500
Epoch 35/40
20/20 [==============================] - 11s 532ms/step - loss: 3.2910 - accuracy: 0.0536 - val_loss: 4.3634 - val_accuracy: 0.0437
Epoch 36/40
20/20 [==============================] - 10s 525ms/step - loss: 3.2574 - accuracy: 0.0552 - val_loss: 4.5473 - val_accuracy: 0.0437
Epoch 37/40
20/20 [==============================] - 11s 531ms/step - loss: 3.2555 - accuracy: 0.0503 - val_loss: 4.3191 - val_accuracy: 0.0375
Epoch 38/40
20/20 [==============================] - 10s 545ms/step - loss: 3.2902 - accuracy: 0.0471 - val_loss: 3.2021 - val_accuracy: 0.0562
Epoch 39/40
20/20 [==============================] - 10s 525ms/step - loss: 3.2847 - accuracy: 0.0422 - val_loss: 4.2214 - val_accuracy: 0.0437
Epoch 40/40
20/20 [==============================] - 11s 531ms/step - loss: 3.2096 - accuracy: 0.0568 - val_loss: 3.8965 - val_accuracy: 0.0375
<keras.callbacks.History at 0x7f2be96e4210>
```

```
%tensorboard --logdir logs/fit9
```

```
Reusing TensorBoard on port 6007 (pid 10402), started 1:17:27 ago. (Use '!kill 10402' to kill it.)
```



Unfortunately, fine tuning resulted in a significant drop in the model's accuracy, thus we will not use it.

In addition, we will try some more models:

Try No2 : - a more simple model

```
mobilenet_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
inception_v3 = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4"

feature_extractor_model = mobilenet_v2
```

```
feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(224, 224, 3),
    trainable=False)
```

```
Model_1 = tf.keras.Sequential([
  feature_extractor_layer,
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(27)
])

Model_1.summary()
```

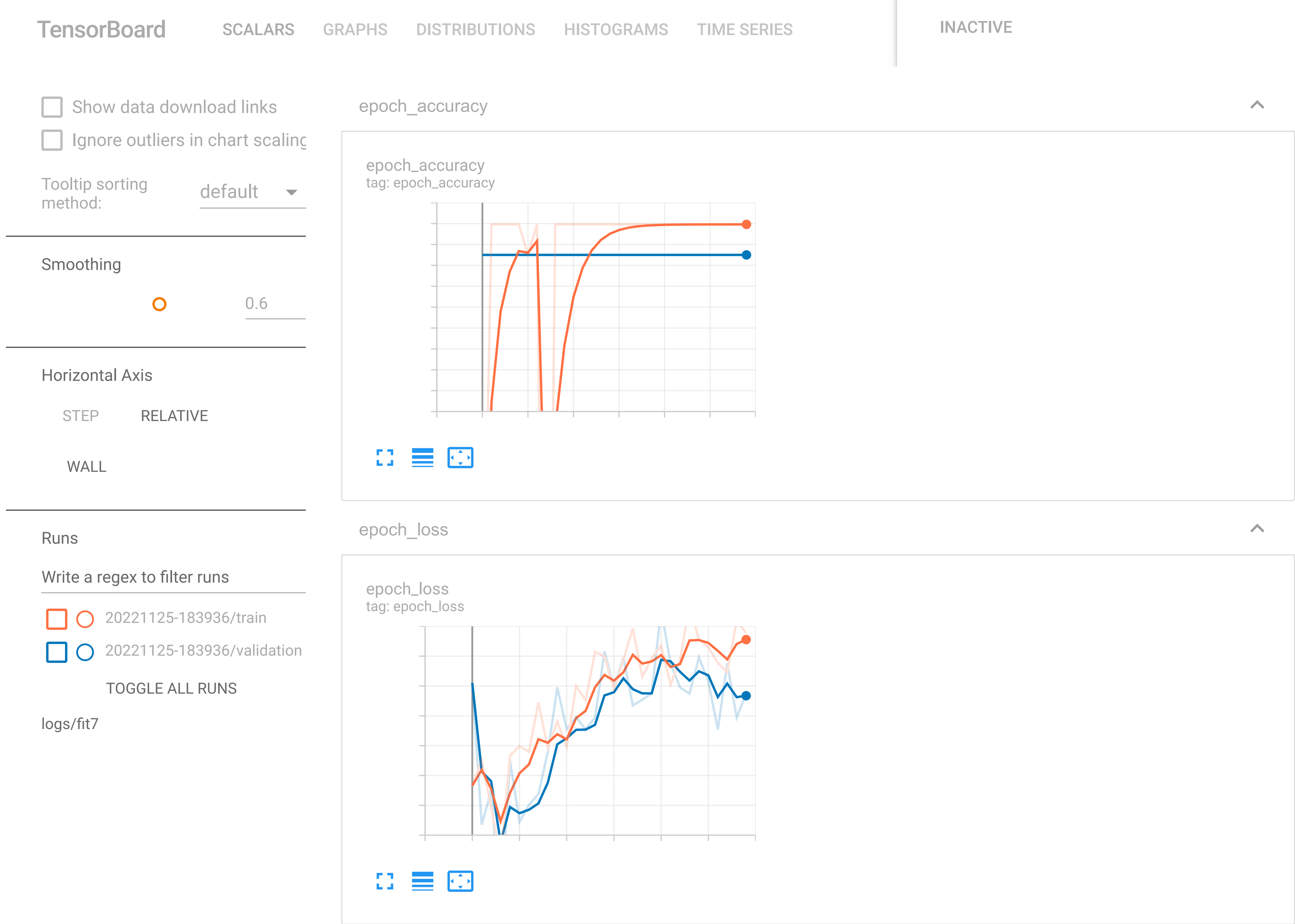
```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_1 (KerasLayer)  (None, 1280)              2257984

 dense_54 (Dense)            (None, 64)                81984

 dense_55 (Dense)            (None, 27)                1755

=================================================================
Total params: 2,341,723
Trainable params: 83,739
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
Model_1.compile(loss="categorical_crossentropy",
              optimizer='Adam',
              metrics=['accuracy'])

log_dir = "logs/fit7/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.
```

```
Model_1.fit(
        Train,
        validation_data = Validation,
        steps_per_epoch=STEP_SIZE_TRAIN,
        validation_steps=STEP_SIZE_VALID,
        epochs=30,
        callbacks = tensorboard_callback)
```

```
%tensorboard --logdir logs/fit7
```

```
Reusing TensorBoard on port 6008 (pid 10839), started 1:18:29 ago. (Use '!kill 10839' to kill it.)
```



As we can observe, the model did not performed well.

Final model:

Considering the above performances, our final model will be the first model trained with transfer learning, however we will not use fine tuning as it didint work as expected.

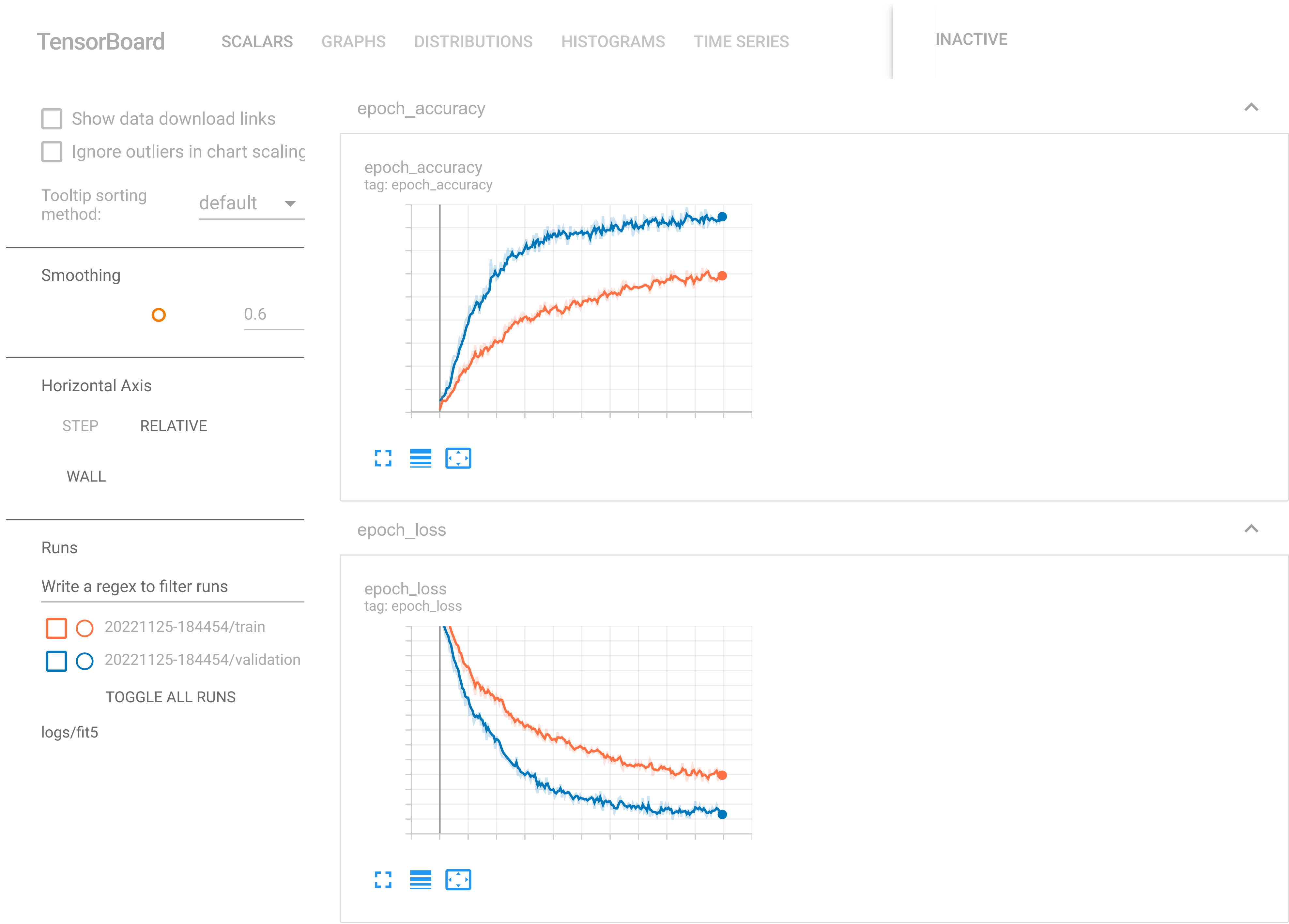
```
# Create the base model from the pre-trained model MobileNet V2

base_Model = tf.keras.applications.MobileNetV2(input_shape=(224,224,3),
                                               include_top=False,
                                               weights='imagenet')
head_Model = base_Model.output

Max_pooling_layer = tf.keras.layers.MaxPooling2D((5,5))
head_Model = Max_pooling_layer(head_Model)

Flatten_layer = tf.keras.layers.Flatten(name="flatten")
head_Model = Flatten_layer(head_Model)

Dense_layer1 = tf.keras.layers.Dense(128, activation="relu")
head_Model = Dense_layer1(head_Model)

Dropout_layer = tf.keras.layers.Dropout(0.5)
head_Model = Dropout_layer(head_Model)

Dense_layer2 = tf.keras.layers.Dense(27, activation="softmax")
head_Model = Dense_layer2(head_Model)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
Model = tf.keras.Model(base_Model.input , head_Model)
```

```
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in base_Model.layers:
    layer.trainable = False
```

```
Model.compile(loss="categorical_crossentropy",
              optimizer='Adam',
              metrics=['accuracy'])


log_dir = "logs/fit5/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.
```

```
history = Model.fit(Train,
                    validation_data = Validation,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=200,
                    callbacks = tensorboard_callback)
```

```
%tensorboard --logdir logs/fit5
```



Training loss: 1.1834 , Training accuracy: 0.5958

val_loss: 0.6440 - val_accuracy: 0.8625

Predict images

```
Images_for_prediction = list(paths.list_images('./flickr_logos_27_dataset_images'))
```

```
Model.get_config
```

```
<bound method Functional.get_config of <keras.engine.functional.Functional object at 0x7f2a5f6c7210>>
```

```
def prediction(path):
    image = Image.open(path)
    #plt.imshow(image)
```

```
    test = load_img(path,target_size=(224,224))
    test = img_to_array(test)
    test = np.expand_dims(test,axis=0)
    test /= 255
    result = Model.predict(test,batch_size = 32)
    y_class = result.argmax(axis=-1)
    result = (result*100)
    result = list(np.around(np.array(result),1))
    return [ image , Labels[y_class[0]] ]
```

```
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)

for n in range(9):
  Im = Images_for_prediction[n]
  image , Label = prediction(Im)


  plt.subplot(3,3,n+1)
  plt.imshow(image)
  plt.title(Label)
  plt.axis('off')
_ = plt.suptitle("Model predictions")
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
```

Bibliography

https://www.kaggle.com/code/sushovansaha9/xception-flickr27-brand-logo-detection

https://www.geeksforgeeks.org/find-duplicate-rows-in-a-dataframe-based-on-all-or-selected-columns/

https://www.kaggle.com/code/parulpandey/overview-of-popular-image-augmentation-packages/notebook#6.-Keras-Image-Data-Generator

https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720

https://www.tensorflow.org/tutorials/images/cnn

https://keras.io/api/applications/xception/

https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

https://www.tensorflow.org/tutorials/images/transfer_learning

https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub

https://www.kaggle.com/code/ryanholbrook/data-augmentation