The preprocessing steps are taken from kaggle notebook

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import shutil
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd


import os
import tensorflow as tf
import cv2
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Add, Dense, Dropout, Activation, ZeroPadding2D
, BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2
D,GlobalAveragePooling2D,Concatenate, ReLU, LeakyReLU,Reshape, Lambda
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopp
ing, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam,SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model, Model
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import metrics
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.imagenet_utils import preprocess_input
from tensorflow.keras.initializers import glorot_uniform
from tqdm import tqdm
import imgaug as ia
from imgaug import augmenters as iaa
from PIL import Image
import keras.backend as K
K.set_image_data_format('channels_last')
K.set_learning_phase(1)
```

```
/usr/local/lib/python3.7/dist-packages/keras/backend.py:450: UserWarning: `tf.keras.backe
nd.set_learning_phase` is deprecated and will be removed after 2020-10-11. To update it,
simply pass a True/False value to the `training` argument of the `__call__` method of you
r layer or model.
  warnings.warn('`tf.keras.backend.set_learning_phase` is deprecated and '
```

In [3]:

```
# getting the labels corresponding to the image
label_df = pd.read_csv('/content/drive/My Drive/crowd-counting/labels.csv')
label_df.columns = ['id' , 'people']
label_df.head()
```

Out[3]:

	id	people
0	1	35
1	2	41
2	3	41
3	4	44
4	5	41

In [4]:

```
# loading the images in vector format
img = np.load('/content/drive/My Drive/crowd-counting/images.npy')
#img = img.reshape(img.shape[0], img.shape[1], img.shape[2], img.shape[3],1)
img.shape
```

Out[4]:

```
(2000, 480, 640, 3)
```

In [5]:

```
labels = np.array(label_df['people'])
labels
```

Out[5]:

```
array([35, 41, 41, ..., 25, 26, 26])
```

In [6]:

```
# setting features and target value

x_train, x_test, y_train, y_test = train_test_split(img, labels, test_size=0.1)
print(x_train.shape[0])
print(x_test.shape[0])
```

```
1800
200
```

In [ ]:

```
!pip install split-folders
```

I will be using resnet upgraded for crowd counting

By using resnet

Customize way # 01

In [7]:

```
from tensorflow.keras.applications.resnet50 import ResNet50
```

In [8]:

```
resnet_model = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(480, 640, 3),
    pooling='avg',
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet
/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
94781440/94765736 [==============================] - 1s 0us/step
```

In [9]:

```
x = resnet_model.output
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='linear')(x)
```

In [10]:

```
model = Model(inputs=resnet_model.input, outputs=predictions)
```

In [11]:

```
k = -7
for layer in model.layers[:k]:
    layer.trainable = False
print('Trainable:')
for layer in model.layers[k:]:
    print(layer.name)
    layer.trainable = True
```

```
Trainable:
conv5_block3_3_conv
conv5_block3_3_bn
conv5_block3_add
conv5_block3_out
avg_pool
dense
dense_1
```

In [12]:

```
import tensorflow as tf
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['mae'])
```

Applying Callbacks

In [13]:

```
def scheduler(epoch, lr):
  if epoch < 10:
    return lr
  else:
    return lr * tf.math.exp(-0.1)
```

In [14]:

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patienc
e=5, min_lr=0.001)
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
learning_rate_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)
```

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
history = model.fit(x_train,
          y_train,
          validation_data=(x_test, y_test),
          epochs=20,
          batch_size=8,
        )
```

In [ ]:

```
# Check libcudnn8 version
!apt-cache policy libcudnn8

# Install latest version
!apt install --allow-change-held-packages libcudnn8=8.4.1.50-1+cuda11.6
```

```
# Export env variables
!export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}
!export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH
!export LD_LIBRARY_PATH=/usr/local/cuda-11.4/include:$LD_LIBRARY_PATH
!export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/lib64

# Install tensorflow
!pip install tflite-model-maker==0.4.0
!pip uninstall -y tensorflow && pip install -q tensorflow==2.9.1
!pip install pycocotools==2.0.4
!pip install opencv-python-headless==4.6.0.66
```

Scores for resnet50

In [ ]:

```
# model error on training dataset
score = model.evaluate(x_train,
                               y_train,
                               verbose = 0)
print("\nTrain error: %.1f%%" % (100.0 * score[1]))
```

```
Train error: 3122.7%
```

In [ ]:

```
# model error on test dataset
score = model.evaluate(x_test,
                               y_test,
                               verbose = 0)
print("\nTest error: %.1f%%" % (100.0 * score[1]))
```

```
Test error: 3087.7%
```

In [ ]:

```
eval_score = model.evaluate(x_test, y_test)
print("Test loss:", eval_score[0])
print("Test error:", eval_score[1])
```

```
13/13 [==============================] - 7s 555ms/step - loss: 1005.2656 - mae: 30.8765
Test loss: 1005.265625
Test error: 30.876514434814453
```

Graphs

Loss vs error graphs

In [ ]:

```
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model error per epoch')
plt.ylabel('mae')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss per epoch')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

ResNet50 model prediction

In [ ]:

```
re = model.predict(x_train)
print('Training set --')
print('    ground truth: ', np.sum(y_train, axis=0))
print('  evaluate count: ', np.sum(re*(re>0.3), axis=0).astype('int'))

re = model.predict(x_test)
print('Testing set --')
print('    ground truth: ', np.sum(y_test, axis=0))
print('   predict count: ', np.sum(re*(re>0.3), axis=0).astype('int'))
```

By using efficientnetb0

In [ ]:

```
from tensorflow.keras.applications.efficientnet import EfficientNetB0
```

In [ ]:

```
efficientnetb0_model = EfficientNetB0(
    weights='imagenet',
    include_top=False,
    input_shape=(480, 640, 3),
    pooling='avg',
)
```

In [ ]:

```
x = efficientnetb0_model.output
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='softmax')(x)
```

In [ ]:

```
model = Model(inputs=efficientnetb0_model.input, outputs=predictions)
```

In [ ]:

```
k = -7
for layer in model.layers[:k]:
    layer.trainable = False
print('Trainable:')
for layer in model.layers[k:]:
    print(layer.name)
    layer.trainable = True
```

In [ ]:

```
import tensorflow as tf
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['mae'])
```

In [ ]:

```
history = model.fit(x_train,
          y_train,
          validation_data=(x_test, y_test),
          epochs=20,
          batch_size=8,
          callbacks = [reduce_lr, early_stopping, learning_rate_scheduler]
        )
```

Scores for efficientnetb0

In [ ]:

```
# model error on training dataset
score = model.evaluate(x_train,
                               y_train,
                               verbose = 0)
print("\nTrain error: %.1f%%" % (100.0 * score[1]))
```

In [ ]:

```
# model error on test dataset
score = model.evaluate(x_test,
                               y_test,
                               verbose = 0)
print("\nTest error: %.1f%%" % (100.0 * score[1]))
```

In [ ]:

```
eval_score = model.evaluate(x_test, y_test)
print("Test loss:", eval_score[0])
print("Test error:", eval_score[1])
```

Graphs

Error vs accuracy grapghs

In [ ]:

```
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model error per epoch')
plt.ylabel('mae')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [ ]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss per epoch')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

EfficientNetB0 model prediction

In [ ]:

```
result = model.predict(x_train)
print('Training set --')
print('    ground truth: ', np.sum(y_train, axis=0))
print('  evaluate count: ', np.sum(result*(result>0.3), axis=0).astype('int'))

result = model.predict(x_test)
print('Testing set --')
print('    ground truth: ', np.sum(y_test, axis=0))
print('   predict count: ', np.sum(result*(result>0.3), axis=0).astype('int'))
```