```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kagg1e/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROH KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.ennon import HTTPEnnor
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_S0URCE_MAPPING = 'facial-expression:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F31050%2F39603%2Fbundle%2Farchive.zip%3FX-G‹

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)


  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_W0RKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FiIeEx1stsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading facial-expression, 202559978 bytes compressed
[==================================================] 202559978 bytes downloaded
```

```
Downloaded and uncompressed: facial-expression
Data source import complete.
```

```
# This Python 3 euviroument comes with mauy helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the “../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdin(“../input/facial-expression/fer2013/“))

# Any results you write to the current directory are saved as output.
```

```
['fer2013. csv']
```

```
import tensorflow as tf

import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layens import Dense, Activation, Dropout, Flatten

from keras.preprocessing import image
from keras.prepnocessing.image import ImageDataGenerator
import pandas as pd
import uumpy as np
import matplotlib.pyplot as pit
```

```
h
fil ame ' t input/facial-expression/fen2013/fer2013.csv'
label_map = ['Anger', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
names=['emotion','pixels','usage']
df=pd.read_csv('../input/facial-expression/fer2013/fer2013.csv',names=names, na_filter=False)
im=df['pixels']
df.head(10)
```

	emot1on	p1xels	usage
0	emotion	pixels	Usage
1	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
2	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
3	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
4	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
5	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training
6	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 ...	Training
7	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 1...	Training
8	3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 5...	Training
9	3	85 84 90 121 101 102 133 153 153 169 177 189 1...	Training

Next steps: Generate code with df GZi View recommended plots

```
def getData(filname):
    R images are 48x48
    O N = 35887


    first = True
    for line in open(filname):
        if first:
            first = False
        else:
            row = line.split(',')
            Y.append(int(row[0]))
            X.append([int(p) for p in now[1].split()])

    X, Y = np.array(X) / 255.0, np.array(Y)
    return  X,  Y
```

```
X, Y = getData(filuame)
num_class = len(set(Y))
print(num_class)
```

```
7
```

```
# keras with tensorflow backend
N, D = X.shape
X = X.reshape(N, 48, 48, 1)
```

Start coding or generate with AI.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=10)
y_train = (np.arange(num_class) == y_train[:, None]).astype(np.float32)
y_test = (np.arange(num_class) == y_test[:, None]).astype(np.float32)
```

```
from keras.models import Sequential
from keras.layers import Dense , Activation , Dropout ,Flatten
from keras.metrics import categorical_accuracy
from keras.models import model_from_json
from keras.callbacks impont ModelCheckpoint
from keras.optimizers import *
```

- Activation layer 'relu'

```
import tensorflow as tf
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, load_model
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.applications.inception_v3 import InceptionV3
from keras.callbacks impont ModelCheckpoint
from keras import metrics
from keras.optimizers import Adam
from keras import backend as K
import keras
```

```
def my_model():
    model = Sequential()
    input_shape = (48,48,1)
    model.add(Conv2D(64, (5, 5), input_shape=input_shape,activation='relu', padding='same'))
    model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Couv2D(128, (5, 5),activatiou='relu',paddiug='same'))
    model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
    model.add(MaxPooliug2D(pool_size=(2, 2)))

    model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
    model.add(Conv2D(256, (3, 3),activation='nelu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dnopout(0.2))
    model.add(Dense(7))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer='adam')
    # UNCOMMENT THIS TO VIEW THE ARCHITECTURE
    #model.summary()

    returu model
model=my_model()
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Paraa #
=================================================================
 conv2d_20 (Conv2D)          (None, 48, 48, 64)        1664

 conv2d_21 (Conv2D)          (None, 48, 48, 64)        102464

 max_pooling2d_6 (MaxPoolin  (None, 24, 24, 64)        0
 g2D)

 conv2d_22 (Conv2D)          (None, 24, 24, 128)       204928

 conv2d_23 (Conv2D)          (None, 24, 24, 128)       409728

  max_poo11ng2d_7 (NaxPoo1In (None, 12, 12, 128)       0
 g2D)

 couv2d_24 (Conv2D)          (None, 12, 12, 256)       295168

 couv2d_25 (Conv2D)          (None, 12, 12, 256)       590086

 max_pooling2d_8 (MaxPoolin  (Noue, 6, 6, 256)         0
 g2D)

 flatten_l (Flatten)         (None, 9216)              0

 dense_1 (Dense)             (None, 128)               1179776

 act 1v at ton (Actlvatton)  (None, 128)               0

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 7)                 903

 activation_1 (Activation)   (None, 7)                 0

=================================================================
Total params: 2784711 (10.62 MB)
Trainable params: 2784711 (10.62 MB)
Non-trainable panams: 0 (0.00 Byte)
_________________________________________________________________
```

In the following model the accuracy is increasing but model is performing very bad on validation set. This is the case of overfitting.

```
path_mode1='model_filter.h5' # save model at this location after each epoch
```

```
model=my_model() # create the model
K.set_value(model.optimizer.lr,0.0005) # set the learning rate
# fit the model
lh=model.fit(x=X_train,
             y=y_traiu,
             batch_size=64,
             epochs=20,
             verbose=1,
             validation_data=(X_test,y_test),
             shuffle=True,
             callbacks=[
                 ModelCheckpoint(filepath=path_model),
```

Epoch l/2B

```
Train on 32298 samples, validate on 3589 samples
Epoch 1/20
32298/32298 [==============================] - 23s 706us/step - loss: 1.6426 - accuracy: 0.3741 - val_loss: 1.6015 - val_accura
cy: 0.3759
Epoch 2/20
32298/32298 [==============================] - 18s 547us/step - loss: 1.2735 - accuracy: 0.5181 - val_loss: 1.4109 - val_accura
cy: 0.4595
Epoch 3/20
32298/32298 [==============================] - 17s 538us/step - loss: 1.0894 - accuracy: 0.5906 - val_loss: 1.1277 - val_accura
cy: 0.5665
Epoch 4/20
32298/32298 [==============================] - 17s 535us/step - loss: 0.9575 - accuracy: 0.6464 - val_loss: 1.1122 - val_accura
cy: 0.5879
Epoch 5/20
32298/32298 [==============================] - 18s 549us/step - loss: 0.8205 - accuracy: 0.6977 - val_loss: 1.2784 - val_accura
cy: 0.5394
Epoch 6/20
32298/32298 [==============================] - 17s 538us/step - loss: 0.6528 - accuracy: 0.7661 - val_loss: 1.2114 - val_accura
cy: 0.5784
Epoch 7/20
32298/32298 [==============================] - 17s 537us/step - loss: 0.4788 - accuracy: 0.8353 - val_loss: 1.4268 - val_accura
cy: 0.5659
Epoch 8/20
32298/32298 [==============================] - 17s 536us/step - loss: 0.3296 - accuracy: 0.8895 - val_loss: 1.3633 - val_accura
cy: 0.5960
Epoch 9/20
32298/32298 [==============================] - 18s 544us/step - loss: 0.2132 - accuracy: 0.9319 - val_loss: 1.5412 - val_accura
cy: 0.5971
Epoch 10/20
32298/32298 [==============================] - 17s 538us/step - loss: 0.1523 - accuracy: 0.9548 - val_loss: 1.6044 - val_accura
cy: 0.6169
Epoch 11/20
32298/32298 [==============================] - 17s 534us/step - loss: 0.1291 - accuracy: 0.9608 - val_loss: 1.6944 - val_accura
cy: 0.5968
Epoch 12/20
32298/32298 [==============================] - 17s 541us/step - loss: 0.1247 - accuracy: 0.9604 - val_loss: 1.8333 - val_accura
cy: 0.6030
Epoch 13/20
32298/32298 [==============================] - 17s 534us/step - loss: 0.1105 - accuracy: 0.9637 - val_loss: 1.8680 - val_accura
cy: 0.6105
Epoch 14/20
32298/32298 [==============================] - 17s 534us/step - loss: 0.0994 - accuracy: 0.9685 - val_loss: 1.9827 - val_accura
cy: 0.6032
Epoch 15/20
32298/32298 [==============================] - 17s 532us/step - loss: 0.0926 - accuracy: 0.9699 - val_loss: 1.8939 - val_accura
cy: 0.6049
Epoch 16/20
32298/32298 [==============================] - 18s 542us/step - loss: 0.0838 - accuracy: 0.9730 - val_loss: 1.9510 - val_accura
cy: 0.6043
Epoch 17/20
32298/32298 [==============================] - 17s 534us/step - loss: 0.0871 - accuracy: 0.9709 - val_loss: 2.0470 - val_accura
cy: 0.5943
Epoch 18/20
32298/32298 [==============================] - 17s 538us/step - loss: 0.0709 - accuracy: 0.9772 - val_loss: 2.2031 - val_accura
cy: 0.5890
Epoch 19/20
32298/32298 [==============================] - 17s 540us/step - loss: 0.0634 - accuracy: 0.9798 - val_loss: 2.2859 - val_accura
cy: 0.6004
Epoch 20/20
12480/32298 [==========>..................] - ETA: 10s - loss: 0.0585 - accuracy: 0.9817
```