I’M BESIDE YOU PROJECT 2K22

DATA SCIENTIST ASSIGNMENT

Submitted By

IDRIS KAGDI

(Institute Application No.: 21MT0169)

Master of Technology (M.Tech.), Petroleum Engineering

Indian Institute of Technology (Indian School of Mines), Dhanbad

Project 1 : Smile Classification

• AIM : To classify faces into :

a. NOT smile : The face doesn’t have a smile.

b. POSITIVE smile : The face has a real smile.

c. NEGATIVE smile : The face has a fake smile.

• DATASET :

https://drive.google.com/drive/folders/1YZj1F3MhD7kdyc2LBm4YZYPZK1giAIk2?usp=sharing

• Dataset Information : train.csv and test.csv consists of annotations for training and testing respectively. happy_images.zip consists of the face images.

1. **Code and Dataset Links:**

Core code on Google Colab-

https://colab.research.google.com/drive/1YCmBRVkQJsI-rgkLU2mqKRJlvWCHc0qW?usp=sharing

Given Dataset on my Google Drive-

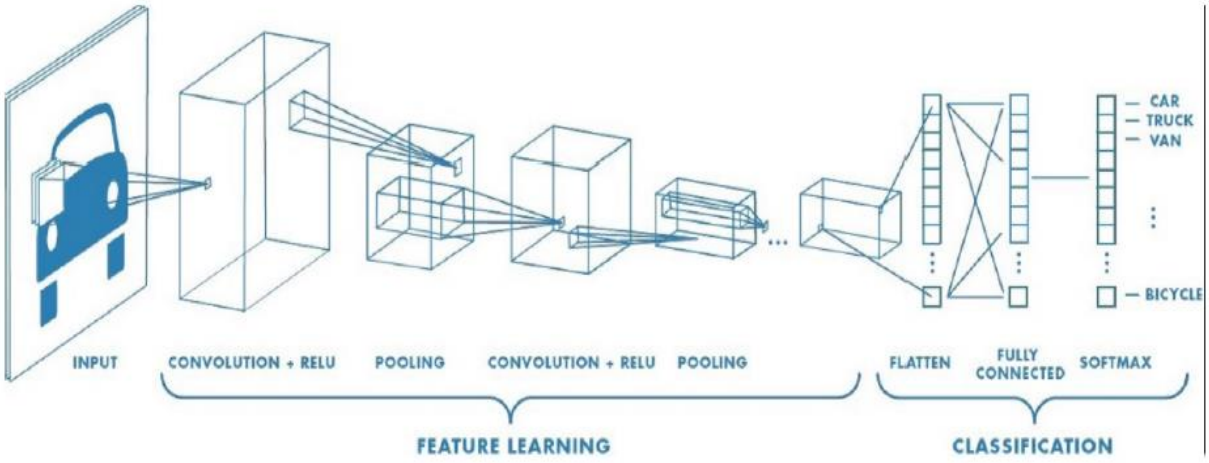
https://drive.google.com/drive/folders/1C2TxbwbOdBZWsiJvd73gwE9Oqb8ssHua?usp=sharing

Modified Dataset sorted into 'train' and 'test' and labelled folders as the output classes-

https://drive.google.com/drive/folders/1-56OkBwvn24ufJTPimDiayaaSwjCiNuV?usp=sharing

2. **Ideology used:**

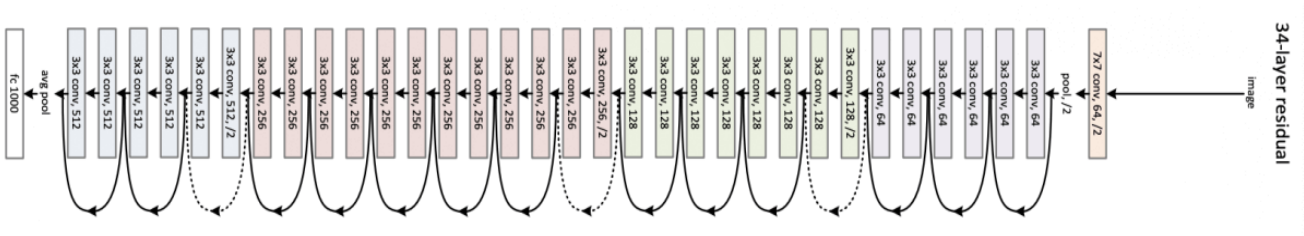
A special-type of Machine Learning model, called **Convolutional Neural Network (CNN)**, is used in this multi-class classification problem, along with the Pre-trained **Transfer Learning Model ResNet-50** (short for Residual Network with 50 dense layers) on **Python**. CNNs are a powerful type of neural network that is used primarily for image classification. Convnets consist primarily of three different types of layers: convolutions, pooling layers, and one fully connected layer. In the convolutional layers, a matrix known as a kernel is passed over the input matrix to create a feature map for the next layer. The dimensions of the kernel can also be adjusted to produce a different feature map, or to expand the data along one dimension while reducing its size along the other axes. Similarly, Stride length, Padding, Pooling and Activation function are designed for a network. Below figure shows a general architecture of a CNN.



Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. It is very effective, these days, to perform transfer learning with predictive modelling problems that use image data as input. Many popular organizations make innovative CNN models and compete in ImageNet Classification Challenge, some of which are:

- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model

Many of these Models, like AlexNet, VGG16, GoogLeNet/Inception, ResNet, etc. are trained to classify into more than 1000 categories. So, in this project, I had chosen pre-trained ResNet50 model, which is 50 layers deep, attained top-1 accuracy of 75.2% and top-5 accuracy of 93% in 2015. The ResNet-50 has over 23 million trainable parameters. Its architecture is shown below:



Also, Data Augmentation is used to increase the training data size to increase performance as given data is less, by random Rotation, horizontal flip, etc. For coloured images, each pixel has 3 channels: **R**ed, **G**reen and **B**lue (**RGB**), which is used in creating tensor. The data contains 4831 training images and 1611 testing images.

3. Methodology and Work-Plan used:

Following steps entrail the methodology:

1. Mounting the given images and train and test data on google drive.
2. Reading .csv train and test files by pandas and allocating classes as **0 for negative smile, 1 for positive smile and 2 for NOT smile**, for converting string into integer.
3. Creating separate 'train' and 'test' folders in my google drive and inside it, separate classes 0, 1 and 2 for both 'train' and 'test' folders. This operation is required for loading data for our model.
4. Creating train and test Data Loader and normalizing the image channels by dividing it by 255. Also, setting random image rotation, horizontal/vertical flip, shear, etc. for Data augmentation.

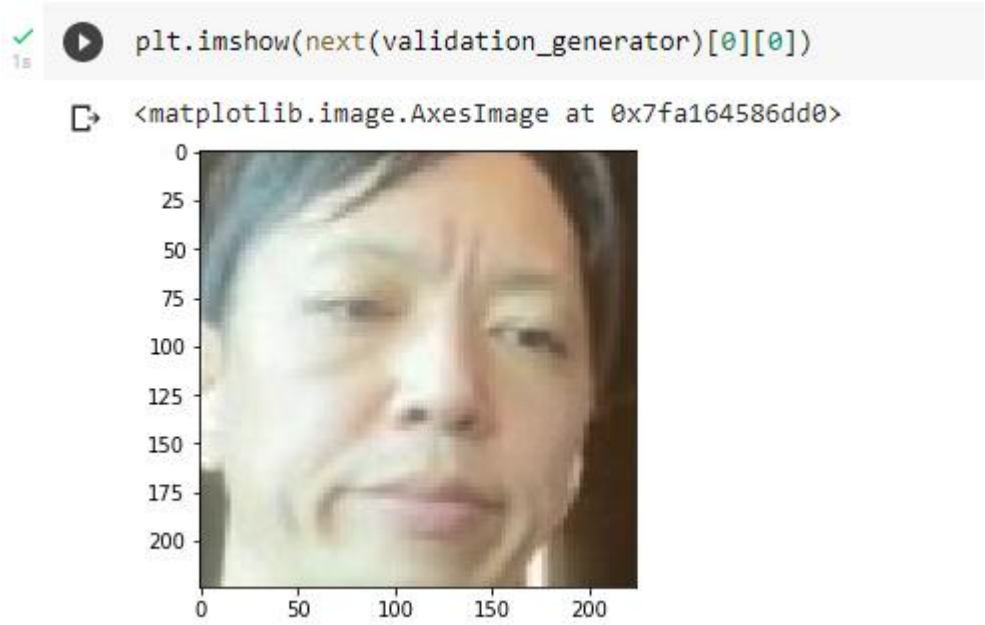
5. Now, Creating train and validation (test, in this case) Data generator by fitting our images to target size of 224×224 pixels, as it is the input size required for the ResNet50 Model, and splitting our images into mini-batches for averaging the weight gain step for whole batch by Batch Size. Batch Size, here, is the hyperparameter to be tuned.
6. Creating a plotting function to analyze our train and test accuracy and losses for the successive epochs.
7. Making a '.h5' Checkpoint file generator for saving our best model parameters.
8. **Importing and Training the ResNet-50 Model**- Pre-trained ResNet50 model is concatenated with 2 fully-connected dense layers to connect to our 3-class output layer. Compiling the model with Activation functions, Learning rate, loss criterion, optimizer, etc. which are the hyperparameters to be tuned.

4. Code-flow:

1. Importing required libraries in python, like
    - Keras form Tensorflow to import model
    - tqdm to get progress bar with metrics
    - Pandas, Numpy, Matplotlib for basic python commands like, reading .csv, creating arrays, plotting data,etc.
    - ImageDataGenerator from tensorflow.keras.preprocessing.image for data augmentation
    - Shutin for copying images in our sorted directory
    - Import Input, Lambda, Dense, Flatten from Keras
    - Image from keras.preprocessing
    - Sequential form keras.models
    - Time to save our best performance model file
2. Import train.csv and test.csv by the help of pd.read_csv and name the columns as ID and labels
3. Creation of encoding_map dictionary for converting categorical variables in the form of string to integers with 0 representing negative smile, 1 representing positive smile and 2 representing not smile
4. Check the normalized distribution of the image dataset to know more about the images by checking the value counts of the class dictionary created on the previous line of code.
5. Now, some Linux commands, like 'mkdir' (make directory) is used to create a folder 'home' in my google drive and inside it, 'train' and 'test' folders and then classes '0', '1' and '2' folders, in each of them. Then, 'Shutil' module is used to put the sorted face images into each folder by creating loop using tqdm to copy our images from source

and pasting to destination file, i.e. the sorted 0, 1 and 2 folders. [Note: should be run only once.]

6. Optional: Check whether the a particular file path exists in the happy_images folder or not using os.path.exists the output will be in a Boolean form
7. Create train_loader and test_loader as Data Loader by ImageDataGenerator by inputting parameters of your choice. Data Augmentation is done in this step, vertical flip, shear and other functions are disabled.
8. Create a Data generator function for loading from the directory defining target size as (224,224) and batch size as 16.
9. Below is an example of resized image:



10. Defining plotmodelhistory function (by history()) using Matplotlib for plotting for train and test data with Model Accuracy and Loss on Y-axis and epochs on X-axis. It will generate 2 separate Accuracy and Loss plots for train and validate (test).
11. Building of transfer learning model:
    a) Base_model is imported as ResNet50 model, pre-trained from ImageNet. Last layer is flattened by using Flatten().
    b) Last layer (1000 neurons) is concatenated with 2 extra Dense() layers, ‘class1’ (1024 neurons with activation ReLU) and ‘output’ layer (3 neurons with activation Softmax), as Softmax is used for multi-class classification.

c) Now, create the model using Model and compile the model using loss function as categorical_crossentropy, optimizer as adam and metrics as the accuracy, precision and recall.

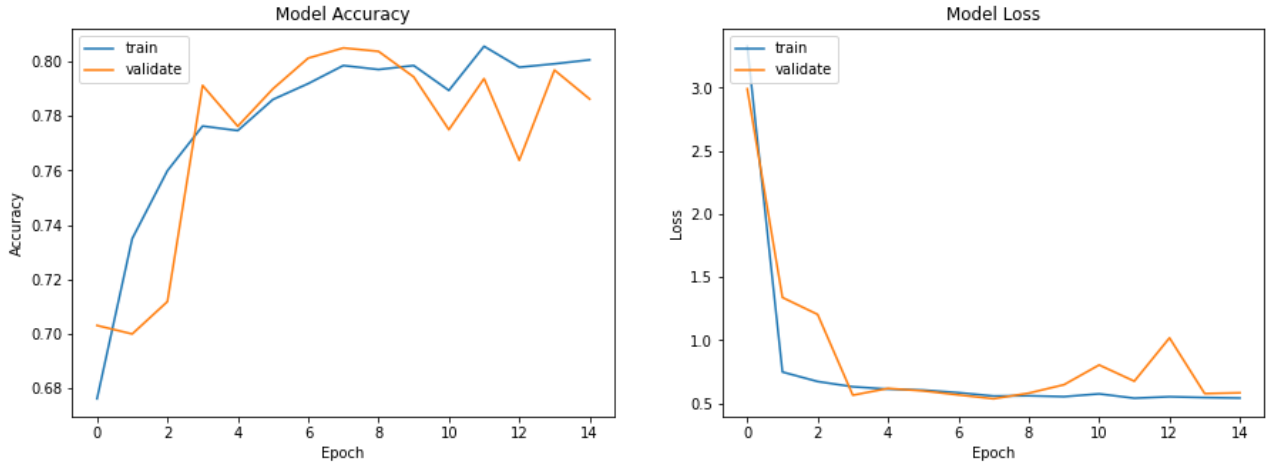
d) Fit the model (model.fit()) using generator function (history()) and epochs as 15 (tuned hyperparameter). Here, steps_per_epoch and validation_steps is taken as floor value of the no. of train and test data divided by batch size (which is 16), respectively.

e) Run the code to get the plots and results.

f) Model.summary() to get the layers used in our transfer learning model.

5. Performance and Results:

Following plot is obtained:



As it can be seen, during training Loss is decreasing and Accuracy is increasing our 15 epochs. It is very much compatible with the test Accuracy and Loss. Also, the above hyperparameters are chosen after some experimentation and are not random. Following is the result over 15 epochs-

Epoch 1/15

301/301 [==============================] - 163s 514ms/step - loss: 3.3276 - accuracy: 0.6762 - recall_1: 0.5562 - precision_1: 0.6999 - val_loss: 2.9903 - val_accuracy: 0.7031 - val_recall_1: 0.7031 - val_precision_1: 0.7031

Epoch 2/15

301/301 [==============================] - 150s 496ms/step - loss: 0.7482 - accuracy: 0.7350 - recall_1: 0.6687 - precision_1: 0.7632 - val_loss: 1.3382 - val_accuracy: 0.7000 - val_recall_1: 0.7000 - val_precision_1: 0.7000

Epoch 3/15

301/301 [==============================] - 153s 509ms/step - loss: 0.6728 - accuracy: 0.7599 - recall_1: 0.7105 - precision_1: 0.7978 - val_loss: 1.2041 - val_accuracy: 0.7119 - val_recall_1: 0.7081 - val_precision_1: 0.7135

Epoch 4/15

301/301 [==============================] - 154s 509ms/step - loss: 0.6315 - accuracy: 0.7763 - recall_1: 0.7298 - precision_1: 0.8074 - val_loss: 0.5646 - val_accuracy: 0.7912 - val_recall_1: 0.7556 - val_precision_1: 0.8247

Epoch 5/15

301/301 [==============================] - 149s 494ms/step - loss: 0.6130 - accuracy: 0.7747 - recall_1: 0.7275 - precision_1: 0.8118 - val_loss: 0.6183 - val_accuracy: 0.7763 - val_recall_1: 0.7169 - val_precision_1: 0.8378

Epoch 6/15

301/301 [==============================] - 149s 494ms/step - loss: 0.6049 - accuracy: 0.7861 - recall_1: 0.7369 - precision_1: 0.8253 - val_loss: 0.5986 - val_accuracy: 0.7900 - val_recall_1: 0.7506 - val_precision_1: 0.8249

Epoch 7/15

301/301 [==============================] - 152s 505ms/step - loss: 0.5844 - accuracy: 0.7919 - recall_1: 0.7414 - precision_1: 0.8239 - val_loss: 0.5668 - val_accuracy: 0.8012 - val_recall_1: 0.7656 - val_precision_1: 0.8305

Epoch 8/15

301/301 [==============================] - 153s 507ms/step - loss: 0.5579 - accuracy: 0.7985 - recall_1: 0.7553 - precision_1: 0.8332 - val_loss: 0.5363 - val_accuracy: 0.8050 - val_recall_1: 0.7675 - val_precision_1: 0.8428

Epoch 9/15

301/301 [==============================] - 150s 497ms/step - loss: 0.5600 - accuracy: 0.7971 - recall_1: 0.7479 - precision_1: 0.8416 - val_loss: 0.5807 - val_accuracy: 0.8037 - val_recall_1: 0.7331 - val_precision_1: 0.8379

Epoch 10/15

301/301 [==============================] - 150s 496ms/step - loss: 0.5531 - accuracy: 0.7985 - recall_1: 0.7520 - precision_1: 0.8397 - val_loss: 0.6473 - val_accuracy: 0.7944 - val_recall_1: 0.7600 - val_precision_1: 0.8266

Epoch 11/15

301/301 [==============================] - 149s 495ms/step - loss: 0.5751 - accuracy: 0.7894 - recall_1: 0.7394 - precision_1: 0.8308 - val_loss: 0.8039 - val_accuracy: 0.7750 - val_recall_1: 0.7494 - val_precision_1: 0.7940

Epoch 12/15

301/301 [==============================] - 149s 495ms/step - loss: 0.5408 - accuracy: 0.8056 - recall_1: 0.7639 - precision_1: 0.8407 - val_loss: 0.6754 - val_accuracy: 0.7937 - val_recall_1: 0.7750 - val_precision_1: 0.8052

Epoch 13/15

301/301 [==============================] - 150s 496ms/step - loss: 0.5524 - accuracy: 0.7979 - recall_1: 0.7562 - precision_1: 0.8368 - val_loss: 1.0183 - val_accuracy: 0.7638 - val_recall_1: 0.7462 - val_precision_1: 0.7850

Epoch 14/15

301/301 [==============================] - 150s 496ms/step - loss: 0.5459 - accuracy: 0.7992 - recall_1: 0.7562 - precision_1: 0.8415 - val_loss: 0.5773 - val_accuracy: 0.7969 - val_recall_1: 0.7387 - val_precision_1: 0.8401

Epoch 15/15

301/301 [==============================] - 149s 495ms/step - loss: 0.5418 - accuracy: 0.8006 - recall_1: 0.7599 - precision_1: 0.8408 - val_loss: 0.5834 - val_accuracy: 0.7862 - val_recall_1: 0.7394 - val_precision_1: 0.8279

dict_keys(['loss', 'accuracy', 'recall_1', 'precision_1', 'val_loss', 'val_accuracy', 'val_recall_1', 'val_precision_1'])

The best model (epoch 8) has following results-

Training loss: 0.5579

Validation loss: 0.5363

Training accuracy: 0.7971

Validation accuracy: 0.8037

Precision score: 0.8379

Recall score: 0.7331

6. **Future scope of the model**:

1) This model may be tried with different transfer learning models
2) The model may be created by pytorch library. (I tried it, but it showed some errors)
3) Also, the data given has almost 70% images labelled as NOT Smile, which is the problem called Dataset Imbalance, so this issue can be addressed by assigning weights to the losses of each class or duplicating images in same class with less images.

NOTE: Please ignore some hashtag (comments) in the code file.

-x-x-x-