SUMMARY

ON

FACE EMOTION RECOGNITION

ALMABETTER CAPSTONE PROJECT

-By Yamini Peddireddi

1. Introduction:

1.1. Deep Learning:

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called **artificial neural networks**.

Andrew Ng from Coursera and Chief Scientist at Baidu Research formally founded Google Brain that eventually resulted in the productization of deep learning technologies across a large number of Google services.

The growth of and interest in AI is due, in no small part, to the recent advances in deep learning for which Bengio, Hinton and LeCun laid the foundation. These technologies are used by billions of people. Anyone who has a smartphone in their pocket can tangibly experience advances in natural language processing and computer vision that were not possible just 10 years ago. In addition to the products, we use every day, new advances in deep learning have given scientists powerful new tools--in areas ranging from medicine, to astronomy, to materials science.
“At the heart of this progress are fundamental techniques developed starting more than 30 years ago by this year's Turing Award winners, Yoshua Bengio, Geoff Hinton, and Yann LeCun. By dramatically improving the ability of computers to make sense of the world, deep neural networks are changing not just the field of computing, but nearly every field of science and human endeavor.”

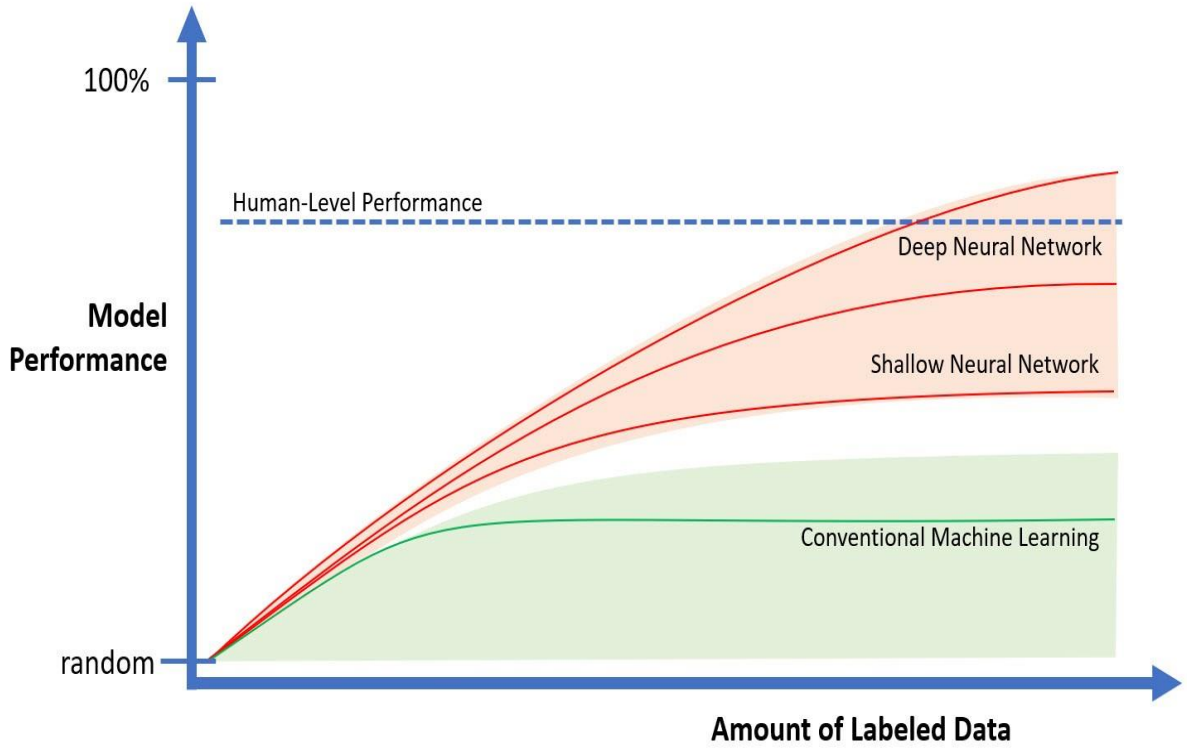
Machine Learning, Neural Networks and Deep Learning

In traditional computing, a computer program directs the computer with explicit step-by-step instructions. In deep learning, a subfield of AI research, the computer is not explicitly told how to solve a particular task such as object classification. Instead, it uses a learning algorithm to extract patterns in the data that relate the input data, such as the pixels of an image, to the desired output such as the label "cat." The challenge for researchers has been to develop effective learning algorithms that can modify the weights on the connections in an artificial neural network so that these weights capture the relevant patterns in the data.

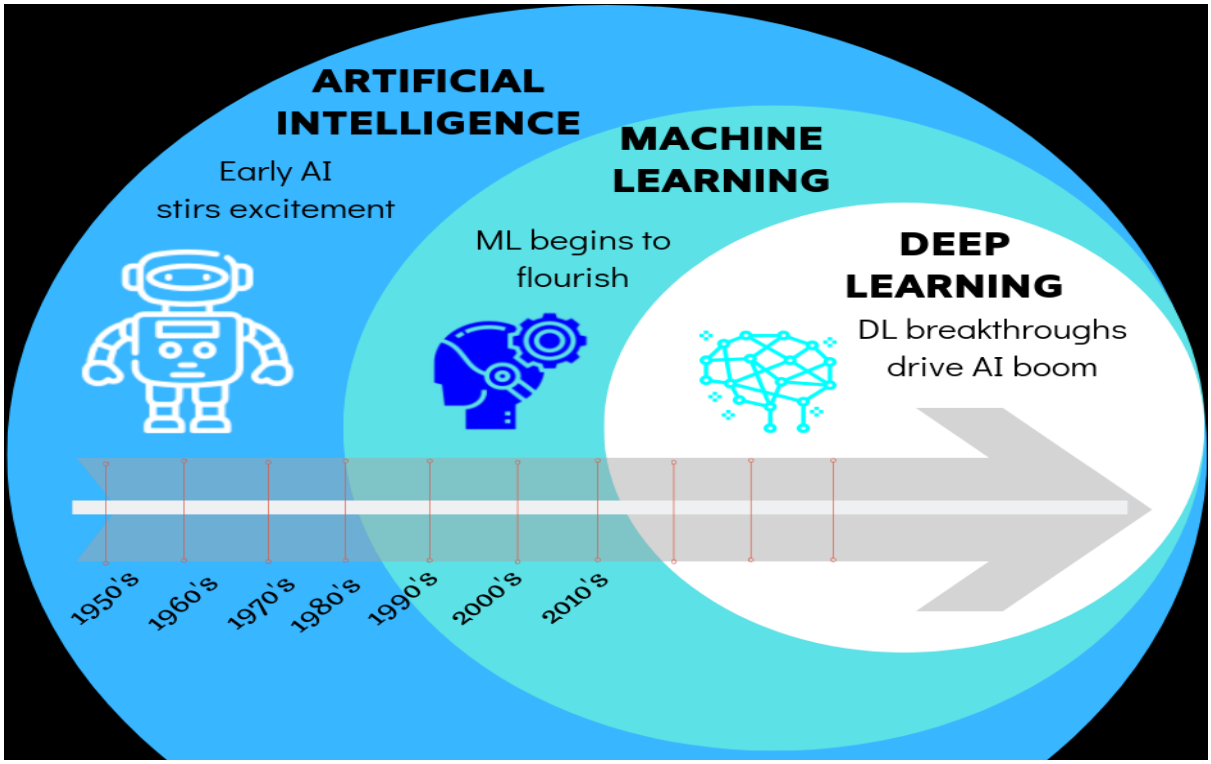
Geoffrey Hinton, who has been advocating for a machine learning approach to artificial intelligence since the early 1980s, looked to how the human brain functions to suggest ways in which machine learning systems might be developed. Inspired by the brain, he and others proposed "artificial neural networks" as a cornerstone of their machine learning investigations.

In computer science, the term "neural networks" refers to systems composed of layers of relatively simple computing elements called "neurons" that are simulated in a computer. These "neurons," which only loosely resemble the neurons in the human brain, influence one another via weighted connections. By changing the weights on the connections, it is possible to change the computation performed by the neural network. Hinton, LeCun and

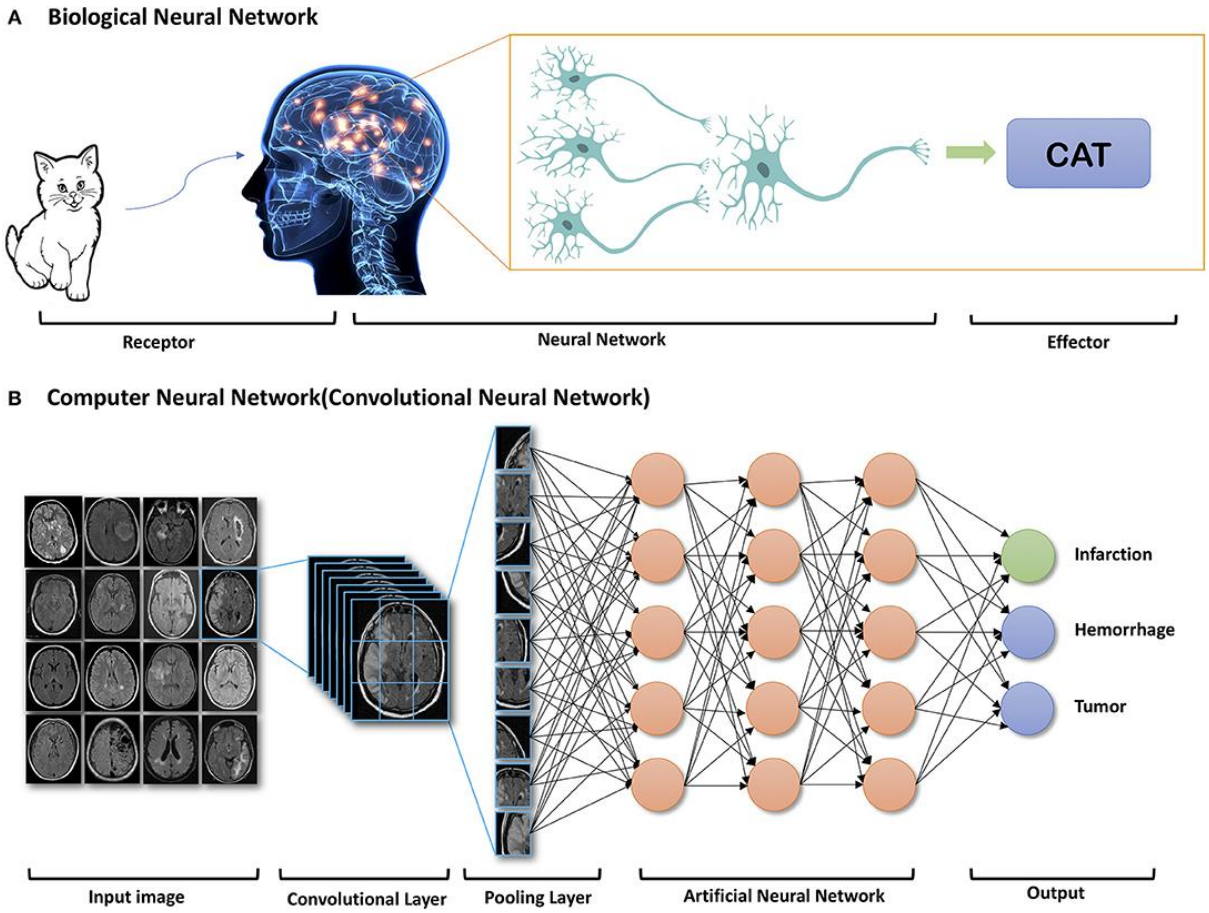
Bengio recognized the importance of building deep networks using many layers--hence the term "deep learning."



Transformation from AI, ML to advanced networks



1.2. Neural networks diagram or representation



2. Methodology/Approach in solving:

2.1. Problem statement

The Indian education landscape has been undergoing rapid changes for the past 10 years owing to the advancement of web-based learning services, specifically, eLearning platforms. Global E-learning is estimated to witness an 8X over the next 5 years to reach USD 2B in 2021. India is expected to grow with a CAGR of 44% crossing the 10M users mark in 2021. Although the market is growing on a rapid scale, there are major challenges associated with digital learning when compared with brick-and-mortar classrooms. One of many challenges is how to ensure quality learning for students. Digital platforms might overpower physical classrooms in terms of content quality but when it comes to understanding whether students are able to grasp the content in a live class scenario is yet an open-end challenge. In a physical classroom during a lecturing teacher can see the faces and assess the emotion

of the class and tune their lecture accordingly, whether he is going fast or slow. He can identify students who need special attention. Digital classrooms are conducted via video telephony software program (ex: Zoom) where it's not possible for medium scale class (25-50) to see all students and access the mood. Because of this drawback, students are not focusing on content due to lack of surveillance. While digital platforms have limitations in terms of physical surveillance but it comes with the power of data and machines which can work for you. It provides data in the form of video, audio, and texts which can be analysed using deep learning algorithms. Deep learning backed system not only solves the surveillance issue, but it also removes the human bias from the system, and all information is no longer in the teacher's brain rather translated in numbers that can be analysed and tracked.

We will solve the above-mentioned challenge by applying deep learning algorithms to live video data. The solution to this problem is by recognizing facial emotions.

This project will be able to detect the emotion of a person or the people.

2.2. Steps in problem solving

According to the problem statement, it is clear that here we have to create a model which would be able to detect the emotion of students or people in the classroom or any other place who would like to know the state of emotions of a person or people in a group so that it becomes easy for the teacher or supervisor or any authority person to bring new things which would make the people happier and help in completing their mission successfully.

The steps I followed to solve this problem are:

1) Business understanding
2) Data Collection
3) Data Cleaning
4) Data Visualization
5) Data pre-processing
6) Data Analysis
7) Models that can be used
8) Model or technique used for solving
9) Transfer learning
10) Creating and training the model
11) Evaluating the model
12) Retraining the model and tuning the model
13) Evaluating and interpreting the results
14) Making some sample predictions
15) Model ready for deployment
16) Creating necessary files for deployment
17) Making the necessary installations
18) Streamlit app is created and deployed in GCP

2.3. Challenges

The challenges faced in completing this project are:

a) Mislabelled data or data was not correctly labelled in the raw data
b) Data limitation
c) Computational cost and time. There requires high computational power and resources to train highly accurate models and takes lot of time which is practically not possible with low memory and computational systems
d) There is some mismatch problem with the libraries and compatibility with one another.
e) OpenCV library has some compatibility issues with streamlit and live camera access was not possible with it. When the model is deployed in GCP/AWS/Azure platforms OpenCV has some errors and bugs.
f) Heroku cannot deploy the models with project size greater than 500 MB. So large projects cannot be deployed here.
g) Kernel dies sometimes due to the huge work assigned to it.
h) Memory crash problems can occur in some devices which have less resources to handle the data and models.
i) GCP has some charge if trial period ends and it is highly charged if you require to keep the projects running in GCP. So, for some period of time, it would be better not for longer time who wants to just try whereas it is very much useful for companies.

3. Data collection and understanding

3.1. Data collection

I have collected the dataset link from some blog and could not find it directly and it was found in Kaggle. There are many other related datasets and I have taken the first found dataset and there are even other datasets in which there are some animated images and also some disguised images. I only choose the images from first set which is direct person images and no other things involved in it as it matches my requirement.

The link to the dataset can be found in my coding notebook.

3.2. Data Cleaning and pre-processing

From the images collected there are some redundant images that have been removed from my dataset as it is some other images which are not related to my project. There are some misplaced images which are of different target class so I moved them to the respective folder or to the respective emotion value.

3.3. Data processing, visualization and analysis

I have tried visualize the images from different ways like using Pillow, matplotlib and opencv. Next, I have prepared my directory of images for training them. I have visualized a bunch of images from the prepared data directory from which images are going to be trained. Found the target classes and number of train and test images present in it.

Target classes in the dataset

```
ic(data.classes);
ic(len(data.classes));
ic(data.c);
```

```
ic| data.classes: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
ic| len(data.classes): 7
ic| data.c: 7
```

4. Models

4.1. Models that can be used

There are many models that can be used for completing this task like even ML models like SVM, KNN, Boosting can be used then using more advanced algorithms in CNN would make the task easier as it involves more deep layers which would result in increasing the accuracy. Yes, there is no one algorithm which fits all the tasks or projects. Hence, there are numerous algorithms for completing the tasks.

4.2. The models available and chosen for the project

I have chosen fastai one of the advanced transfer learning technique

Fastai is the model based on pytorch. It is something like keras then keras is dependent on tensorflow.

PyTorch is an optimized tensor library primarily **used** for Deep Learning applications using GPUs and CPUs. It is an open-source machine learning library for Python, mainly developed by the Facebook AI Research team. It is one of the widely **used** Machine learning libraries, others being TensorFlow and Keras.

Refer this for more understanding on PyTorch: https://ai.facebook.com/tools/pytorch/

fastai is organized around two main design goals: to be approachable and rapidly productive, while also being deeply hackable and configurable. It is built on top of a hierarchy of lower-level APIs which provide composable building blocks. This way, a user wanting to rewrite part of the high-level API or add particular behavior to suit their needs does not have to learn how to use the lowest level.

It's very easy to migrate from plain PyTorch, Ignite, or any other PyTorch-based library, or even to use fastai in conjunction with other libraries. Generally, you'll be able to use all your

existing data processing code, but will be able to reduce the amount of code you require for training, and more easily take advantage of modern best practices.

fastai includes many modules that add functionality, generally through callbacks. Thanks to the flexible infrastructure, these all work together, so you can pick and choose what you need (and add your own), including: mixup and cutout augmentation, a uniquely flexible GAN training framework, a range of schedulers (many of which aren't available in any other framework) including support for fine tuning following the approach described in ULMFiT, mixed precision, gradient accumulation, support for a range of logging frameworks like Tensorboard (with particularly strong support for Weights and Biases, as demonstrated here), medical imaging, and much more.

```
#Models available in fastai with transfer learning models
dir(fastai.vision.models)
```

```
]: ['BasicBlock',
    'Darknet',
    'DynamicUnet',
    'ResLayer',
    'ResNet',
    'SqueezeNet',
    'UnetBlock',
    'WideResNet',
    'XResNet',
    '__builtins__',
    '__cached__',
    '__doc__',
    '__file__',
    '__loader__',
    '__name__',
    '__package__',
    '__path__',
    '__spec__',
    'alexnet',
    'darknet',
    'densenet121',
    'densenet161',
    'densenet169',
    'densenet201',
    'mobilenet_v2',
    'resnet101',
    'resnet152',
```

4.3. Transfer Learning

Transfer learning is a mechanism in which the knowledge gained during training one type of process or project is used for another project which are somehow related to one another.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

There are two ways to customize a pretrained model:

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful features from new samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained model so that you can repurpose the feature maps learned previously for the dataset.

   You do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.
2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

You will follow the general machine learning workflow.

1. Examine and understand the data
2. Build an input pipeline, in this case using Keras ImageDataGenerator or Pytorch ImageDataBunch
3. Compose the model
   - Load in the pretrained base model (and pretrained weights)
   - Stack the classification layers on top
4. Train the model
5. Evaluate model

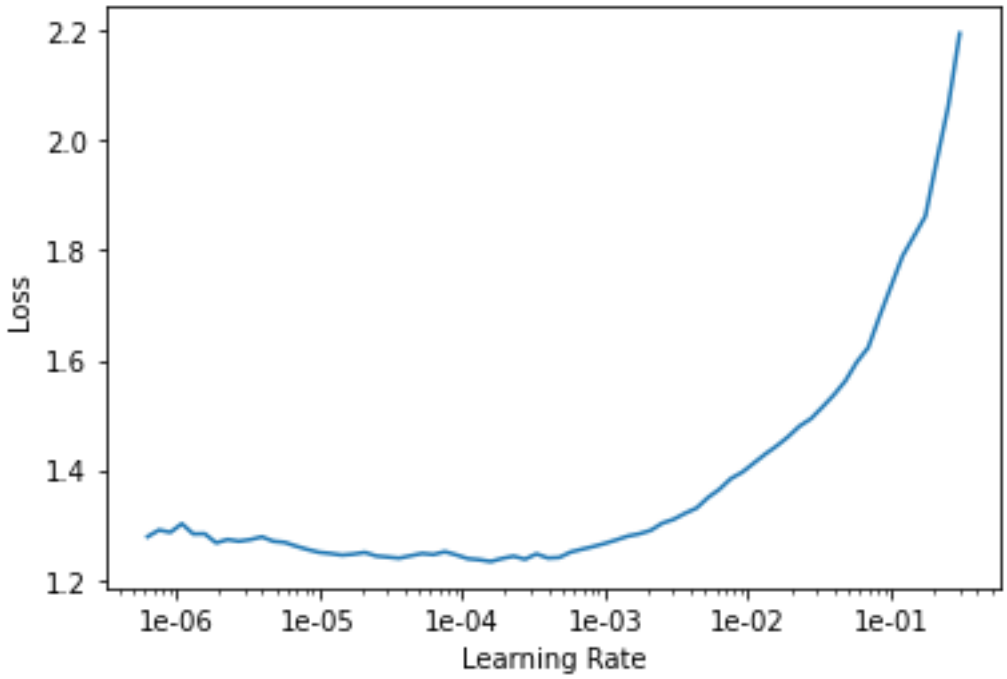
5. Model building and evaluation

5.1. Model building

After choosing the model(resnet34) I have build my model using cnn learner where I give my dataset directory which is divided into train and test and choose the evaluation metric like

error rate, accuracy, r2 score, roc auc score, mse, rmse, explained variance, kappa score, precision, jaccard index etc. Giving all these details and due to the transformations applied on images there is a lot of training taking place than we can do it with the single model and few images.

I have used cyclical learning rate method of finding the optimal learning rate where it runs for certain number of epochs (as given). The optimal learning rate is the best rate that can be used for any dataset which gives good predictions.



5.2. Retraining the model and tuning

Now after finding the optimal rate the range can be chosen where the model again oscillates between the min and max learning rates to give the error rates for certain number of epochs and performance can be tracked at every stage.

Again, that model parameters can be called back when call-backs are used while starting the training which is one of the efficient of processing saving the work at every stage compared to other ways of training.

At every stage even the results can be tracked. Splendid! How useful is this method?

You have to try it when you have to know it.

```
learn.show_results()
```

5.3. Evaluating and interpreting the results

I found model performance was good at stage-4 and so I have used that stage of learner as my final model.

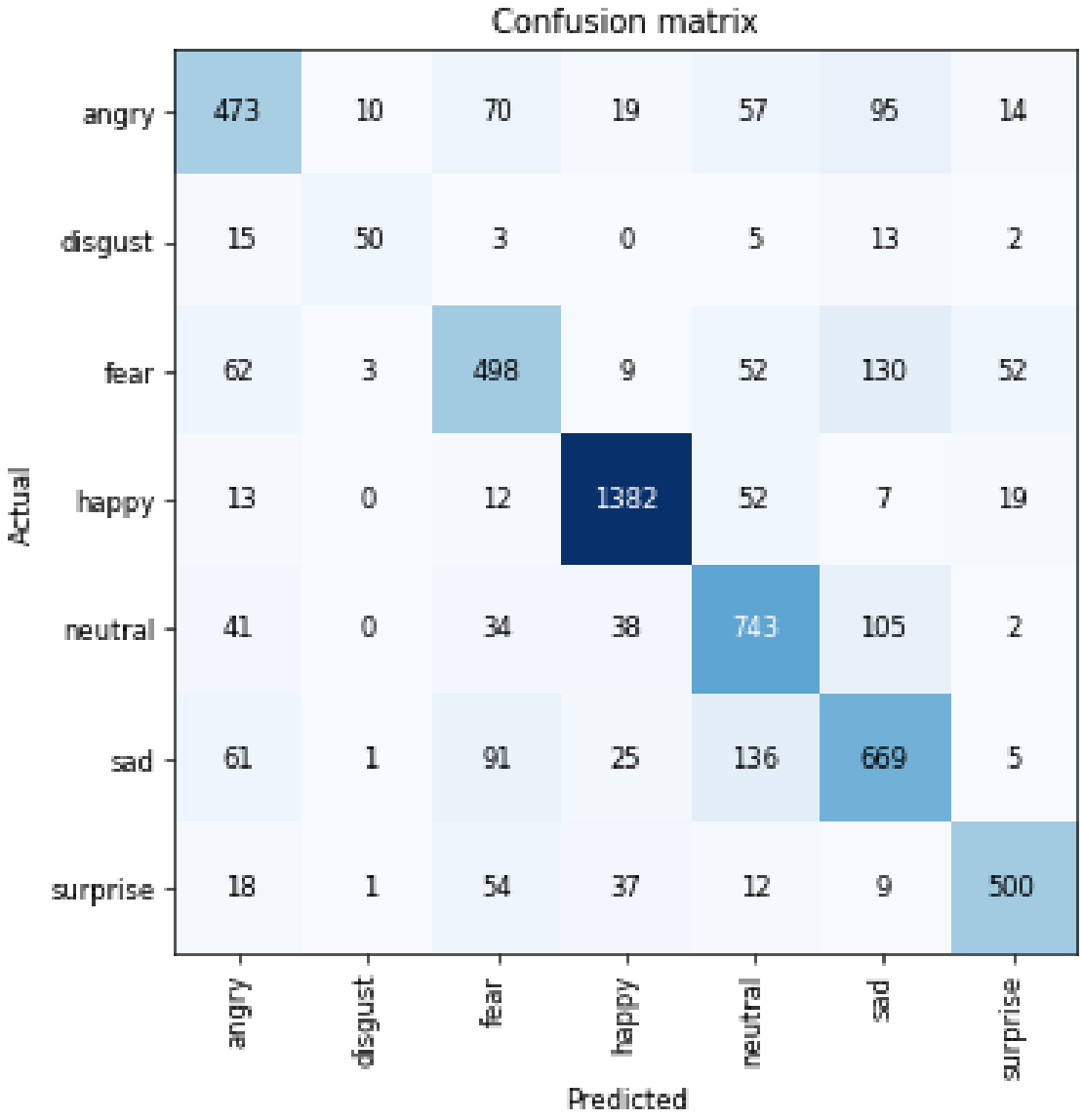
```
learn.fit_one_cycle(3, max_lr=slice(1e-4,40e-5), callbacks=[callbacks.SaveModelCallback(learn,every='epoch',\
                                                                                        monitor='error_rate',name='callba
```

epoch	train_loss	valid_loss	error_rate	time
0	0.742474	0.793938	0.284611	4:16:12
1	0.677608	0.735303	0.271802	3:24:10
2	0.473517	0.702031	0.242850	3:14:00

- Tracking and saving the performance in every epoch
- It can be seen that the error rate has drastically decreased from 57% to 24%
- Also learn-5 stage gives less residual state then there is some overfitting happened with train and validation set so I have chosen learn-4 which is having almost the same accuracy and error rate.

The accuracy of this model is 75.715% ¶

You can see the time it took for running 3 epochs continuing from other batch of learning. So, it is time consuming process then yields good results when trained for good number of epochs and time period.



This is the resultant confusion matrix from the stage-4 of learning made by the model.

- It can be seen that there are good number of correct predictions
- There are some predictions when sad and fear images are taken.
- There is even some gap in understanding the fear and angry images, sad and angry images.
- The model is able is predict with 76% of accuracy and with 24% error rate even though real time prediction with unknown images are still correct.

6.Sample predictions and deployment ready model

6.1. Making some sample predictions

One more advantage of dealing with DL problems or with images are it easy to find out whether the performance is good or not because we can test them with real images and real persons around us whereas ML or tabular data cannot be interpreted or seen or cannot know the actual results until used by client.

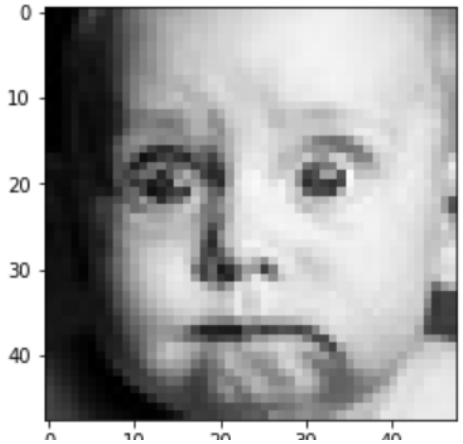
So, let's see some sample prediction made by the model

```
test1=cv2.imread('./3756.jpg')
t = pil2tensor(test1, dtype=np.float32) # converts to numpy tensor
#t = t.permute(2,0,1) # Move num_channels as first dimension
im = Image(t) # Convert to fastAi Image - this class has "apply_tfms"
model3_test.predict(im)
```

```
(Category tensor(0),
 tensor(0),
 tensor([9.9617e-01, 0.0000e+00, 5.2409e-43, 0.0000e+00, 1.6755e-03, 2.1535e-03,
         0.0000e+00]))
```

```
plt.imshow(test1)
```

```
<matplotlib.image.AxesImage at 0x2573261f550>
```



6.1. Deployment ready model

The layers in the resnet model constructed using fastai transfer learning can be seen by loading and calling that model or just printing that model as below

```
model4=learn.load('learn-4')
model4
```

```
y: CategoryList
angry,angry,angry,angry,angry
Path: D:\Data science\Alma better\DL Facial emotion recognition\Images\images\train;

Valid: LabelList (5699 items)
x: ImageList
Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224)
y: CategoryList
happy,sad,angry,neutral,surprise
Path: D:\Data science\Alma better\DL Facial emotion recognition\Images\images\train;

Test: LabelList (7066 items)
x: ImageList
Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224)
y: EmptyLabelList
,,,,
Path: D:\Data science\Alma better\DL Facial emotion recognition\Images\images\train, model=Sequential(
  (0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
```

```
(0): Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

More detailed view can be found in the code notebook.

7.Deployment

7.1. Creating files for deployment

For creating and executing the deployment files a new IDE has to be chosen instead of python notebook as python notebook are not sufficient and capable of handling web development tools.

I have created my files for deployment using Pycharm and even VS code can be used.

Some of the files which are necessary for deployment are:

a. Requirements file
b. App.py file
c. Model
d. Code file
e. Input and output folder
f. .gitignore file

The above are files required for deployment of any project in Heroku, GCP, AWS, Azure or any other. There may be requirement of many more files also after including all the above base files.

The files I have created for deploying this project and running it smoothly in cloud are:

a) Requirement file
b) Streamlit.app.py file
c) Classify.py file
d) Harcasscade file
e) Code files
f) Image's folder
g) .gitignore
h) .slugignore
i) Packages.txt
j) Docker file
k) Setup.sh shell file
l) Runtime.txt file

7.2. Making necessary installations

To make the project work locally first in pycharm with a web page opening you have to make sure all the app.py file requirements are satisfied and also it is important that they are placed in requirements file if not there is a high chance that in cloud while you deploy it in herouku/GCP/Azure/AWS it may not show the expected results as it does in local system.

This is the main reason behind the failure of the application not working in the cloud and working in local. Sometimes there might be some other reasons like memory, the resources available and membership or some other issues also for not application not getting deployed in cloud.

So, this step is very important for deployment to run successfully.

7.3. Deployment in GCP, Heroku and Streamlit sharing

Heroku is free to use and doesn't require any payment then it does not work for project size > 500 MB. When GCP/Azure/AWS requires payment for deployment of your projects and support any type of project even though there are some trial versions available to use for few days and cancel it anytime before the period ends.

Heroku might work very well for small scale projects and may not work for large scale projects. Coming to streamlit sharing if your app is created using streamlit you have to get a streamlit invite to be able to share your created project to the world using streamlit sharing. Mostly streamlit sharing team might approve it in few days after which you can deploy any project created through streamlit and share your work with anyone.

I deployed the project through Heroku first and ended up with slug size error as my project size is greater than 500 MB and my project size at last was 744 KB. So, I couldn't deploy through Heroku and deployed it through GCP in trial period having some free credit and after which it can be cancelled or will be charged.

The Steps I followed to deploy the project in GCP are:

a) Create an account which runs on free credit for 90 days
b) Go to Google Cloud console
c) Activate the cloud shell
d) Git clone the respective repository or project
e) Build the docker image on cloud
f) Check the docker images whether it was created or not
g) Push that docker image to cloud repository in GCP
h) Go to Kubernetes engine, create the cluster on which docker image resides
i) Next go to workloads in Kubernetes engine and give the same port number on which you want to deploy this project on and this port number has to be stored which would be used to create secure port
j) By the above process you would be able to create and deploy your project through load balancer and unsecure link i.e.., http:/3.45.65.80 (80 is port number)
k) To create a secure link, you have to go to cloud run and deploy it there with the same docker image and giving the same port number otherwise it will not get deployed because the application is work only in the given port number not on any other.
l) If everything was given properly you will get the url(secure link) beside the application and cluster region.
m) That's it now your project is running and can be shared with anyone.

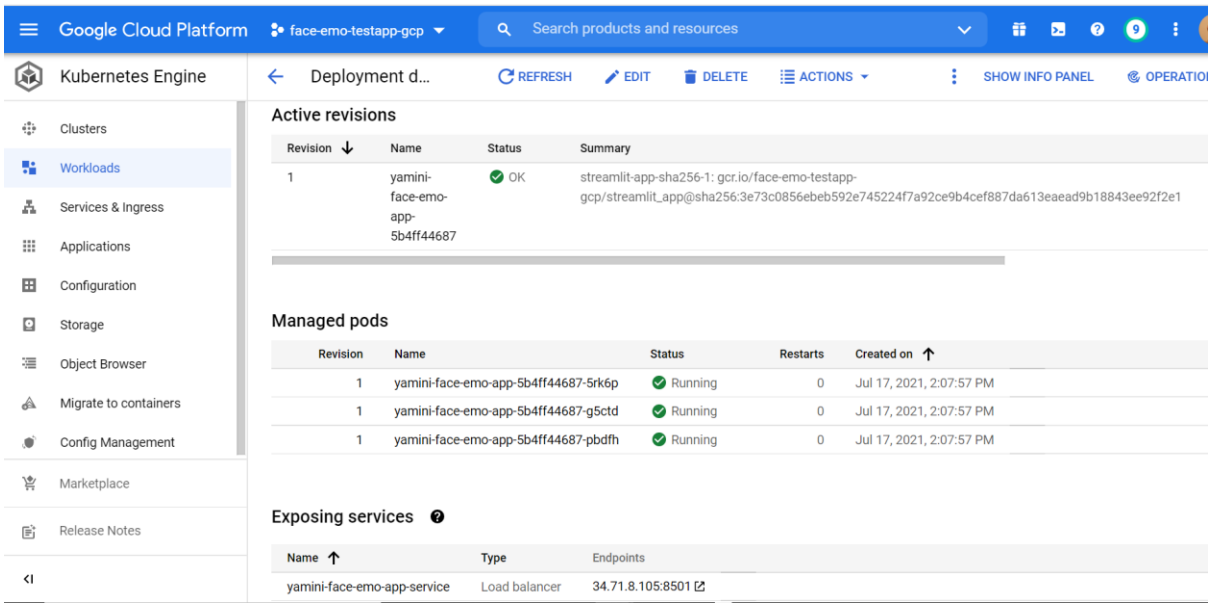


Figure 1.GCP Kubernates engine endpoint working

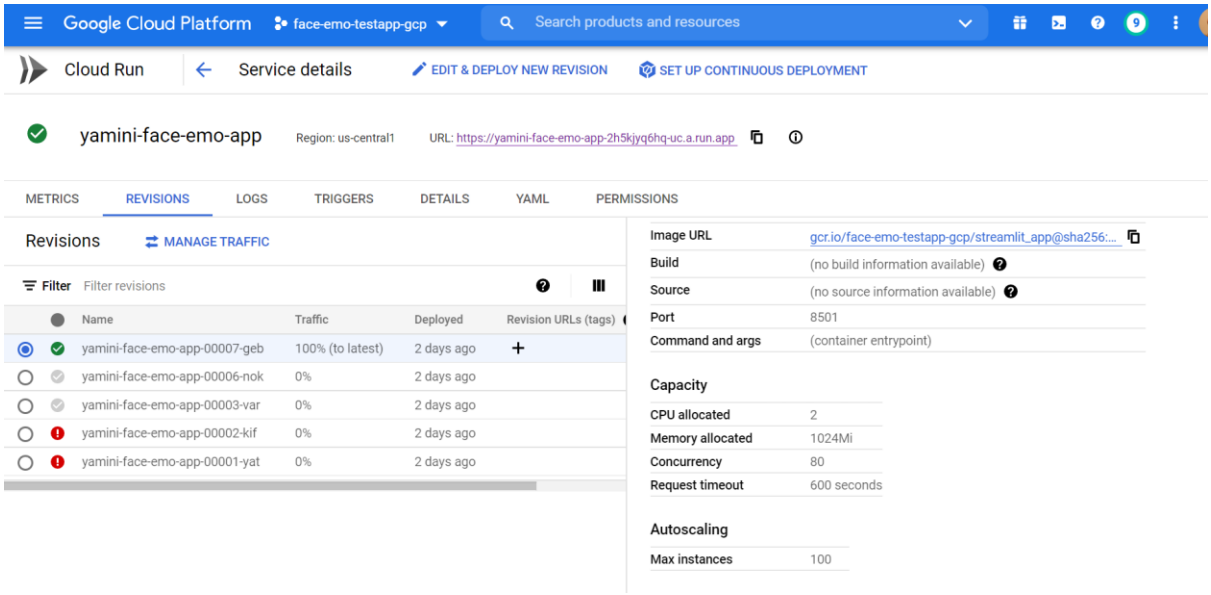
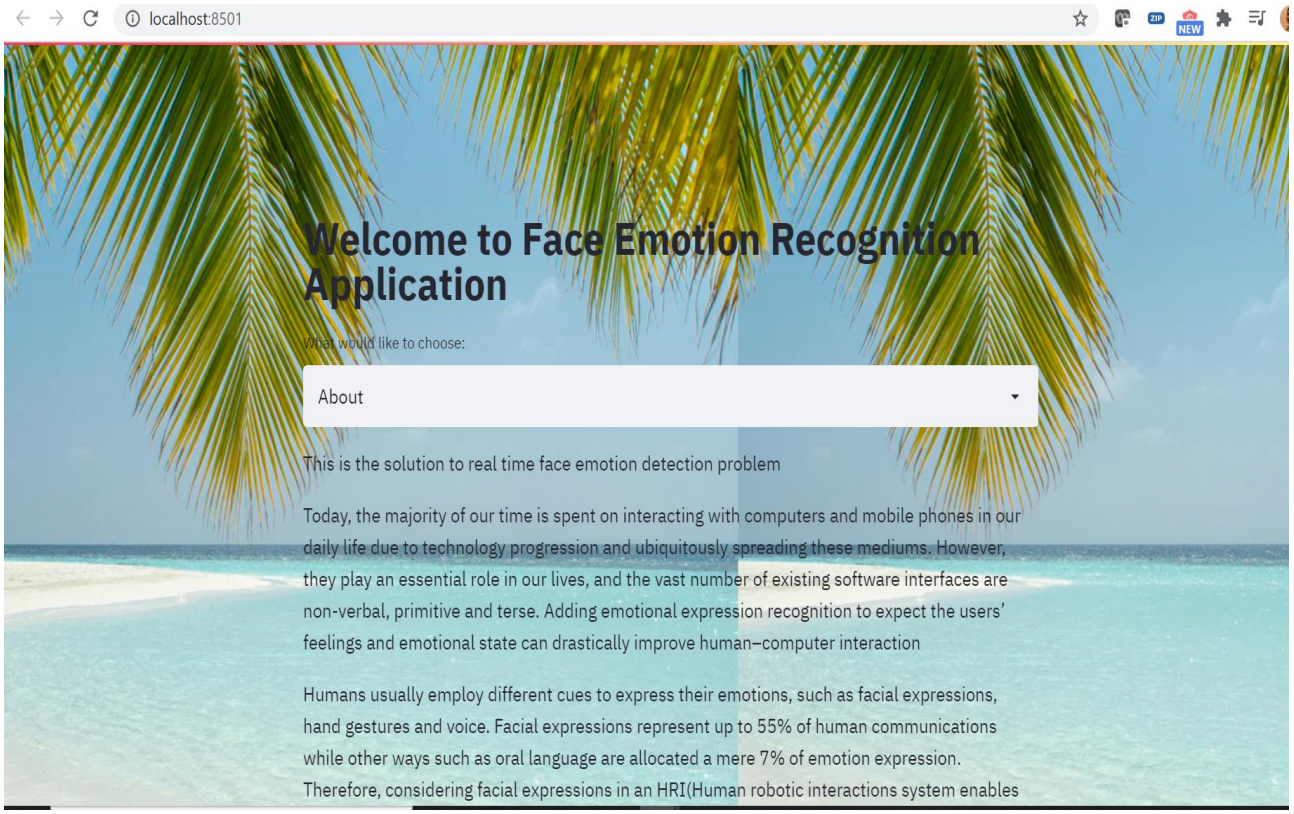
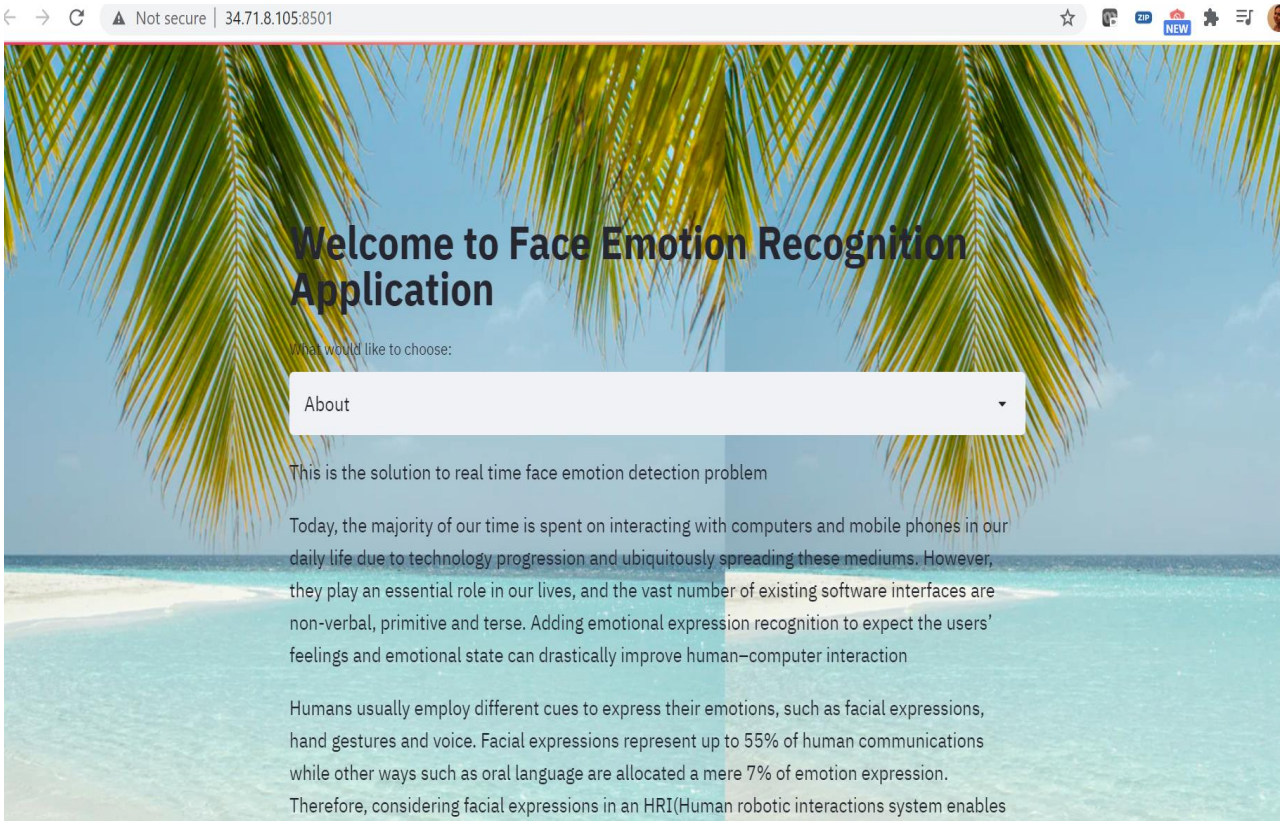


Figure 2. GCP Cloud run url working status

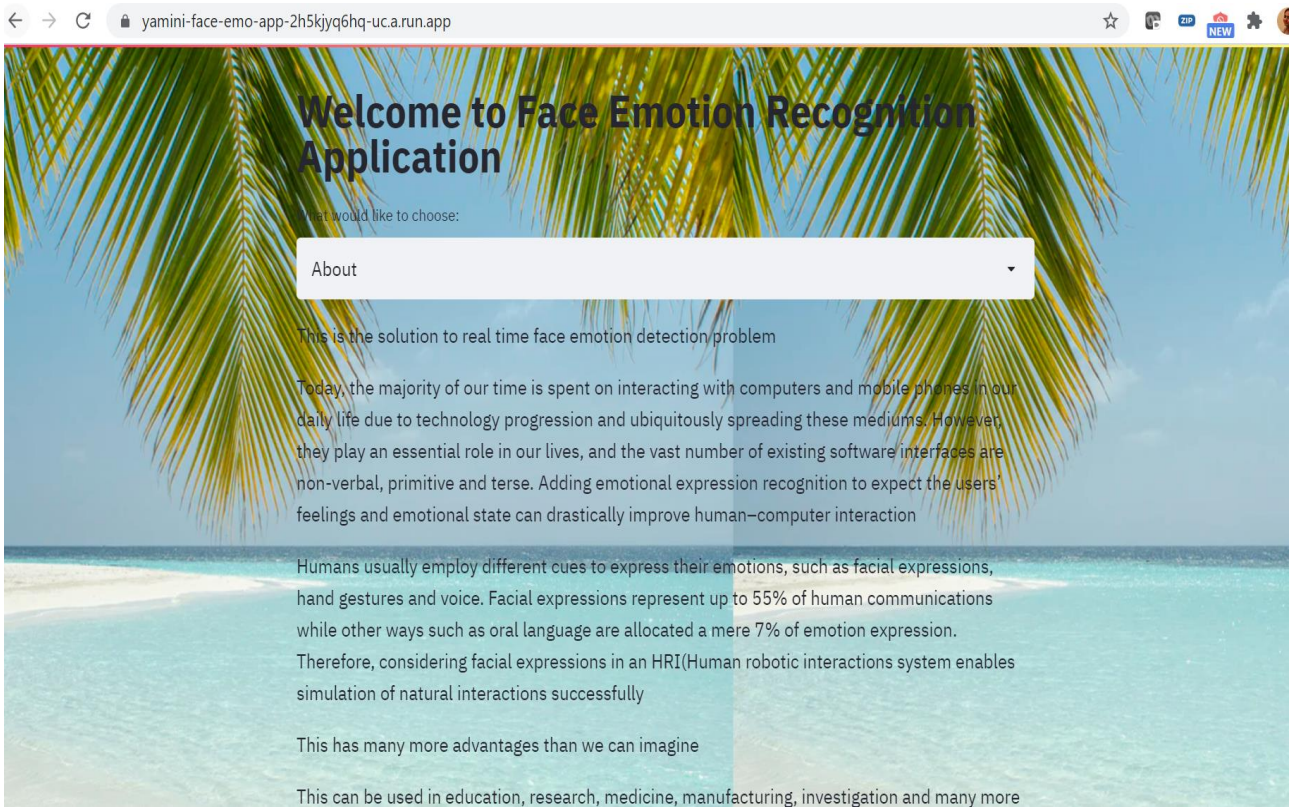
7.4. Deployment images



3.Running streamlit app locally

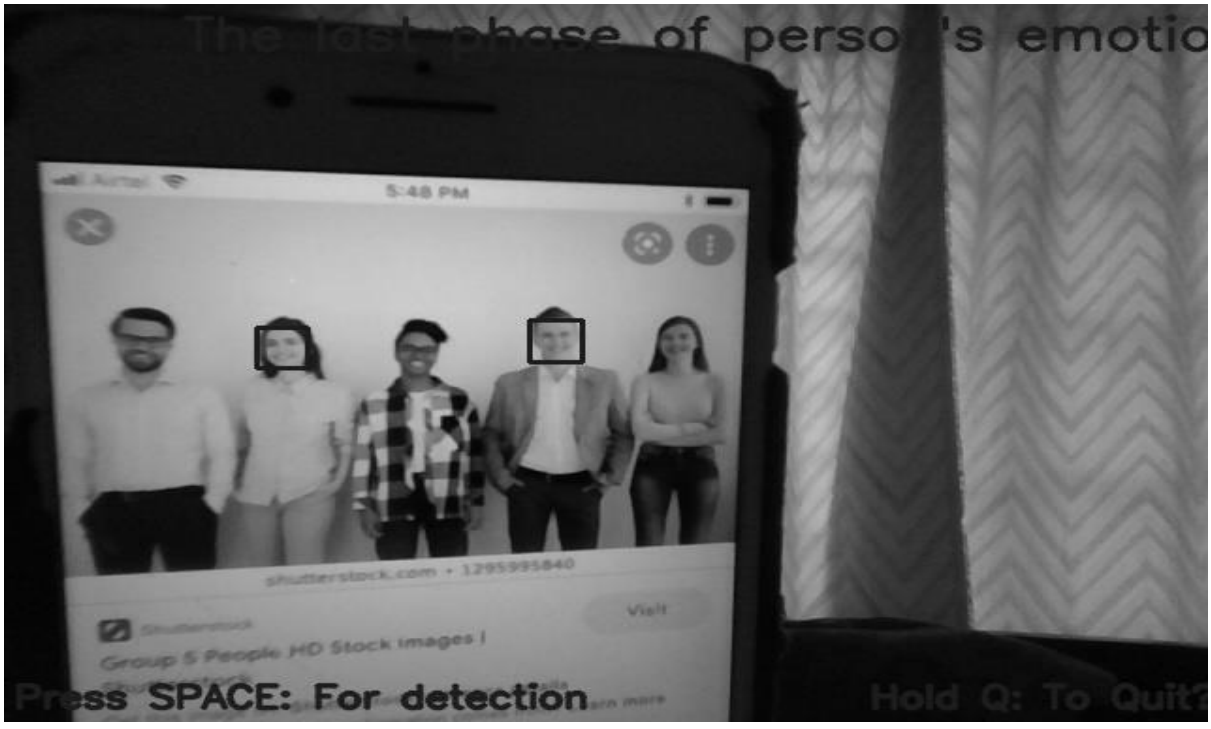


4.Running streamlit app after deploying in GCP

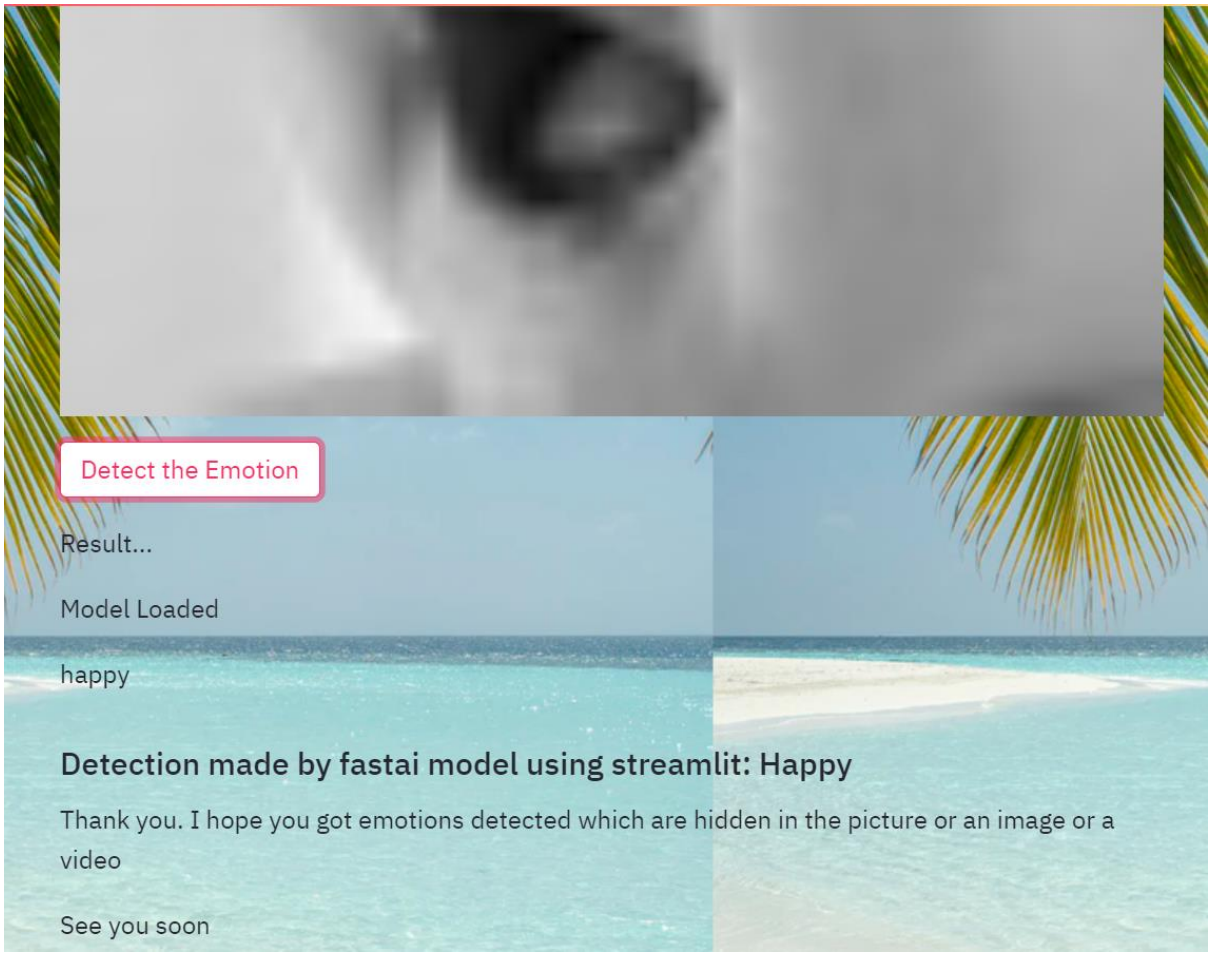


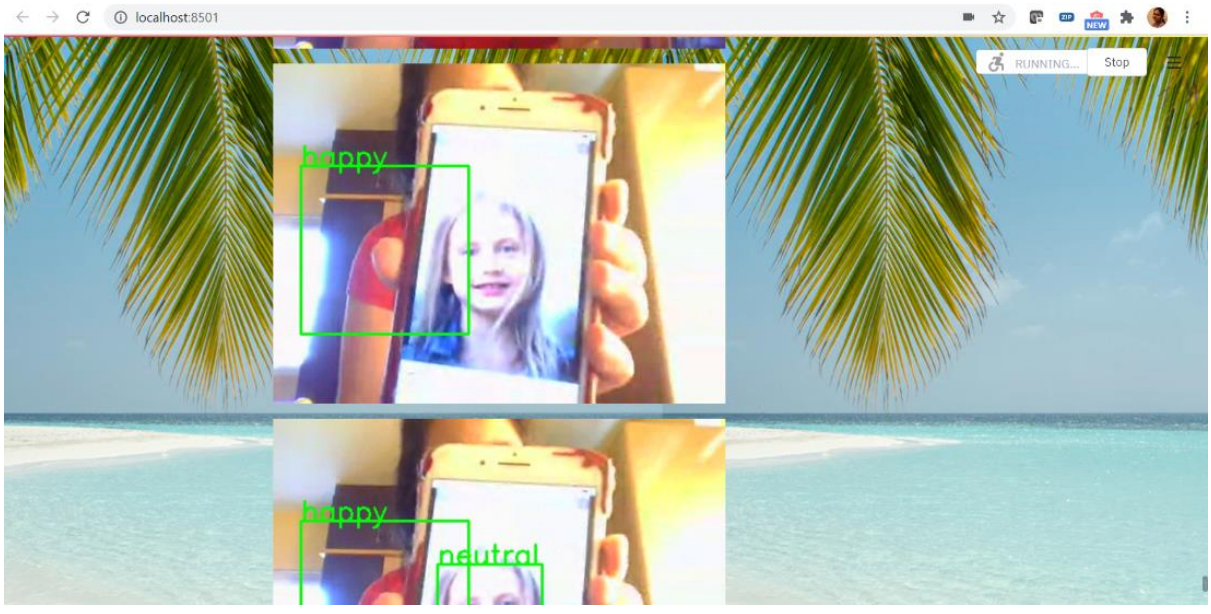
5.Secure link obtained after deployment in GCP from Cloud run in GCP

7.5. Detected images



6.Live Detection made through opencv using flask





7.Live video streamlit detection

8.Conclusion

- The accuracy of this model came up to 76% then there are some challenges because of which there was more error rate here. There are some of the images which are misclassified. Even though I have cleaned some images as there are more than 3000+ images in each category it was difficult to check every image.
- So, there may be some images where it has been not correctly labelled and also training for some more epochs would increase the accuracy.
- May be using resenet50 or more advanced algorithm would yield greater accuracy then it requires greater computational power and time to use them.
- In spite of all these when tested with new images or videos the detection was very good and accurate.
- Some of the detected images and videos results are provided in the pipeline notebook.
- Automatic facial expression recognition system has many applications including, but not limited to, human behaviour understanding, detection of mental disorders, and synthetic human expressions.
- This is one most useful project or topic for future technologies which brings humans closer to the technology and make the tasks easier.
- Facial emotion recognition has many advantages than can be made or thought of. This helps in human-robot interactions, medical field, teaching field, Human resources field do their task much easily.

References:

https://www.eurekalert.org/pub_releases/2019-03/afcm-fod032619.php

https://machinelearningmastery.com/what-is-deep-learning/

https://henrikhain.io/slides/Recent-Advancements-in-Deep-Reinforcement-Learning/Recent-Advancements-in-Deep-Reinforcement-Learning.html#27

https://www.frontiersin.org/articles/10.3389/fneur.2019.00869/full

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

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

https://cloud.google.com/load-balancing/docs/https/setting-up-https-serverless

https://towardsdatascience.com/how-to-deploy-docker-containers-to-the-cloud-b4d89b2c6c31

https://www.caxy.com/blog/starting-scratch-creating-and-installing-python-app-heroku

https://towardsdatascience.com/deploying-panel-holoviz-dashboards-using-heroku-container-registry-5221eb0538ba

https://www.section.io/engineering-education/how-to-deploy-streamlit-app-with-docker/