PAIN RECOGNITION THROUGH FACIAL EXPRESSION

Introduction:

A facial expression of pain in humans is characterized by lowering of the eyebrows, squeezing of the eyes, wrinkling of the nose, raising of the upper lip and opening of the mouth. Pain research has mainly used methods developed in emotion research such as the Facial Action Coding System (FACS), enabling trained researchers to code over 40 distinct muscle movements. Autonomous pain assessment has a huge number of applications, specially in the field of medical science.

Objectives:

- To build a robust deep learning model that could classify facial expression into two major classes that are is_in_pain, or not_in_pain.

Prerequisites:

- Python 3.7+
- Tensorflow, keras
- Opencv
- Matplotlib
- Dlib or haar cascade (for face detection)

Project outline:

- Go through different research paper to analyze the best approach for this problem statement
- Find appropriate dataset
- Build different relevant model and train the model
- Analyze the result of various model and select the best model

Key findings from research papers:

Below points are the key findings for various relevant research papers which need to be considered before the project kick-off.

- Use transfer learning for better results, where pre-trained DCNN(Deep Convolutional Neural Network) like VGG-16, AlexNet, InceptionNet etc) could

be used by replacing dense upper layers(s) as per the requirement. (https://www.mdpi.com/2079-9292/10/9/1036)

- It seems video based (BiLSTM) pain recognition has a little bit better performance than image based, however we can go with image based (since it will be more effective for real time pain recognition, also due to lack in dataset and implementation complexity) and if its accuracy is not that acceptable we can later try video based classifier.
  (https://www.frontiersin.org/articles/10.3389/fmed.2022.851690/full)
- Experimental results show that for the facial expression recognition task it does not matter, the color or grayscale image is fed to the input of the algorithm. Since, pretrained model like VGG16 expects 3 channel input, we should go with a color image.
  (https://www.researchgate.net/publication/335610128_The_Usage_of_Grayscale_or_Color_Images_for_Facial_Expression_Recognition_with_Deep_Neural_Networks)
- If an image and video based CNN model does not perform well then we can try working on facial landmark based pain recognition, since it seems it has slightly better accuracy.
  (
  https://www.researchgate.net/publication/341211036_Facial_Landmark-Based_Emotion_Recognition_via_Directed_Graph_Neural_Network
  ,
  https://www.mdpi.com/1424-8220/22/22/8704/pdf)

Dataset preparation:

The most challenging part of this project is to find the appropriate dataset. After searching for the dataset, it was found that dataset that are used in various pain assessment research are not available publicly, we need to contact the author for it. Fortunately, UNBC-McMaster Shoulder pain expression archive dataset is available on kaggle, however, after training on this dataset we have achieved validation accuracy up to 83% but this model failed to do well on the test dataset. Let's analyze the model.

Fig. screenshot of UNBC-McMaster dataset

We can see that subjects are making laughing faces while expressing pain. So the model was not able to clearly distinguish between a laughing face and a pain expressing face. And also this dataset does not contain other expressions like surprised, true laughing expressions etc. which are quite similar to the pain faces based on muscle contraction. So the model trained on this dataset did not do well while testing.

Now we need to prepare a dataset by ourselves, for which we need to create some python script to semi-automate the data generation process. For data collection, we can extract faces from video clips, by scraping from google, and integrate data from another dataset (such as a dataset containing sad faces, neutral faces, happy faces).

Python script to automate data collection process:

- **Video_to_dataset extractor:**
    - Video to dataset extractor allows us to help in extraction of faces in video frames and label them accordingly through human involvement. This script basically read video files frame by frame and extracts faces from each frame and display that in a window, then we can press keys “a” if it belongs no_pain class, “b” if is_pain class with pain score 1, “c” if it belongs to is_pain class with pain score 2, and any key other than “a”, “b”, “c”, “d” to ignore the displayed image. Those extracted faces will be saved in the related directory. In this way we can quickly extract and label faces from video clips.

Source code:

```python
import cv2
import numpy as np
import dlib
import os
import random

RANDOM_TOKEN = random.getrandbits(128)

def video_to_dataset(video_path):
    prefix = str(RANDOM_TOKEN)+video_path.split(".")[0].split("/")[-1]
    print(prefix)

    capture = cv2.VideoCapture(video_path)
    cv2.namedWindow("Display", cv2.WINDOW_NORMAL)
    # Using resizeWindow()
    cv2.resizeWindow("Display", 500, 500)

    face_detector = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
    DESTINATION_ROOT = "dataset/"
    c = 0
    IMG_SHAPE = 128

    while True:
        is_true, frame = capture.read()

        if not is_true:
            break
        # grayscaling
        gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        #face detection
        faces = face_detector.detectMultiScale(gray_image, 1.1, 4)
        second_value = 0

        if len(faces):
            for face in faces:
                x, y, w, h = face
                face_cropped = frame[y: y+h, x: x+w]
                try:
                    img_data = cv2.resize(face_cropped, (IMG_SHAPE, IMG_SHAPE))

                    cv2.imshow('Display', img_data)
                    key = cv2.waitKey(0) & 0xFF

                    if key == ord('a'):
                        cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "0"),
                                                 prefix+"_"+str(c)+".jpg"),
                                                 img_data)
                        c += 1
                    elif key == ord('b'):
                        cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "1"),
                                                 prefix+"_"+str(c)+".jpg"),
                                                 img_data)
                        c += 1
                    elif key == ord('c'):
                        cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "2"),
                                                 prefix+"_"+str(c)+".jpg"),
                                                 img_data)
                        c += 1
                    elif  key == ord('x'):
                        break

                except Exception as e:
                    print(e)
                    continue
        else:
            continue

        # cv2.imshow('Display', img_data)
        # key = cv2.waitKey(0) & 0xFF

        # if key == ord('a'):
        #     cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "0"),
        #                              prefix+"_"+str(c)+".jpg"),
        #                              img_data)
        #     c += 1
        # elif key == ord('b'):
        #     cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "1"),
        #                              prefix+"_"+str(c)+".jpg"),
        #                              img_data)
        #     c += 1

        # elif key == ord('c'):
        #     cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "2"),
        #                              prefix+"_"+str(c)+".jpg"),
        #                              img_data)
        #     c += 1
        # elif  key == ord('x'):
        #     break

        #skipping next three frame
        for i in range(3):
            capture.read()

    cv2.destroyWindow('Display')
    capture.release()
    return -1

if __name__ == "__main__":
    video_to_dataset("videos/video_sample_34.mp4")
```

Fig: screenshot of the source code of video_to_dataset converter

Output:

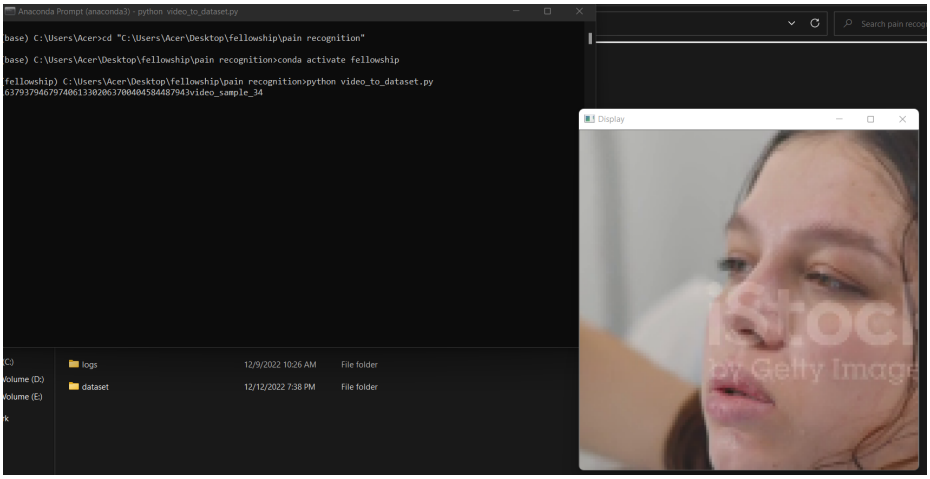


Fig: Screenshot of the execution of video_to_dataset script

- **Scrap image from google:**
    - Google image search is one of the best platforms for searching required images. So are going to use a python tool – google image scraper (available here: https://github.com/ohyicong/Google-Image-Scraper) to scrape 500 most relevant images. However, not all the images will be relevant so we will create a python script similar to video_to_dataset converter to semi-automate the image filtering process.

Source code:

```
from tensorflow import keras
import cv2
import numpy as np
import os
import random


IMG_DIR = "Google-Image-Scraper-master/photos/sad faces"
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
DESTINATION_ROOT = "dataset/"
RANDOM_TOKEN = random.getrandbits(128)

if not os.path.exists("dataset"):
    os.makedirs("dataset/")
    os.makedirs("dataset/0")
    os.makedirs("dataset/1")
    os.makedirs("dataset/2")


# cv2.namedWindow("Display", cv2.WINDOW_NORMAL)
# # Using resizeWindow()
# cv2.resizeWindow("Display", 500, 500)

IMG_SHAPE = 128
def process():
    c = 0
    prefix = str(RANDOM_TOKEN)+"_google_"

    for img_name in sorted(os.listdir(IMG_DIR)):
        img_path = os.path.join(IMG_DIR, img_name)
        img = cv2.imread(img_path,cv2.IMREAD_COLOR)
        extension = img_name.split('.')[1].lower()

        if extension in ['gif', 'png', 'web']:
            continue

        gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        #face detection
        faces = face_cascade.detectMultiScale(gray_image, 1.1, 4)
        if len(faces):
            x, y, w, h = faces[0]
            face_cropped = img[y: y+h, x: x+w]
            try:
                img_data = cv2.resize(face_cropped, (IMG_SHAPE, IMG_SHAPE))
            except Exception as e:
                print("exception: ", e)
                continue
        else:
            print("no face detected")
            continue

        print("showing image")
        cv2.imshow('Display', img_data)
        key = cv2.waitKey(0) & 0xFF

        if key == ord('a'):
            cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "0"),
```

```
        print("showing image")
        cv2.imshow('Display', img_data)
        key = cv2.waitKey(0) & 0xFF

        if key == ord('a'):
            cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "0"),
                                     prefix+"_"+str(c)+".jpg"),
                                     img_data)
            c += 1
        elif key == ord('b'):
            cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "1"),
                                     prefix+"_"+str(c)+".jpg"),
                                     img_data)
            c += 1

        elif key == ord('c'):
            cv2.imwrite(os.path.join(os.path.join(DESTINATION_ROOT, "2"),
                                     prefix+"_"+str(c)+".jpg"),
                                     img_data)
            c += 1
        elif  key == ord('x'):
            break

        cv2.destroyWindow('Display')

# start
process()
```

- **By integrating subset of other datasets:**
    - Collect happy, neutral and surprised faces from here (https://www.kaggle.com/datasets/aadityasinghal/facial-expression-dataset) and add them in no_pain class.

Screenshot of collected data:

Fig: Screenshot of facial expression that belongs to class is_pain

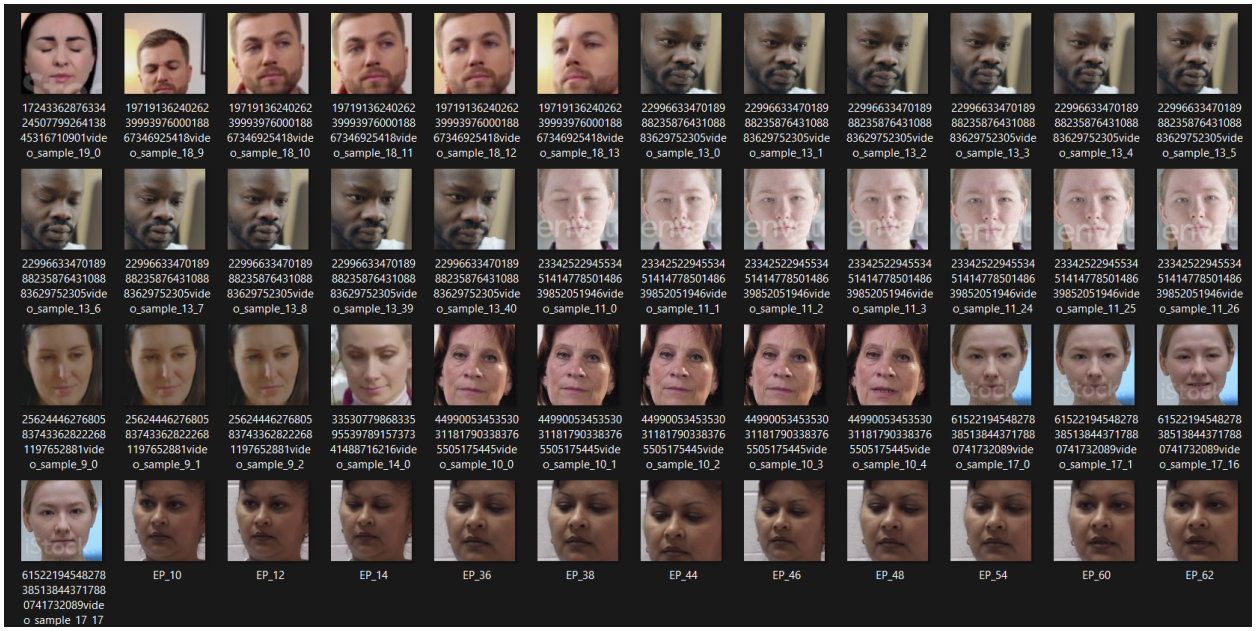


Fig: Screenshot of collected faces that are in not_in_pain class

Dataset summary:

After extracting and integrating dataset from various source we have:

- About 16k faces that belongs to not_in_pain class
- About 400 faces that belongs to in_pain with pain score 1 class
- About 200 faces that belongs to in_pain with pain score 2 class

We can clearly see that we have highly unbalanced data, so it is better to merge pain score 1 and pain score 2 classes into a single in_pain class for now.

Model Building:

- **Model Architecture**

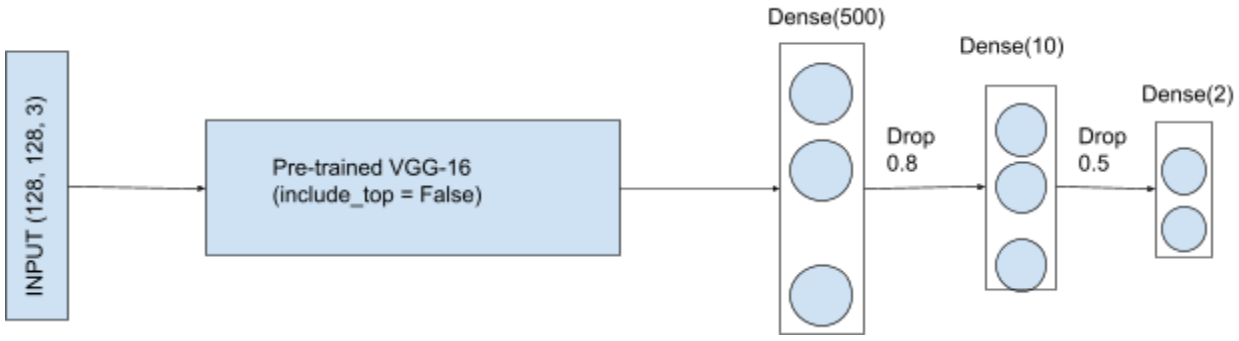


Fig: Architecture of pain recognition model

- **VGG16**

  VGG16 is a convolutional neural network (CNN) architecture that was developed by the Visual Geometry Group (VGG) at the University of Oxford. It was introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014.

  The VGG16 architecture is characterized by its simplicity, using only 3x3 convolutional filters and a stack of convolutional and max pooling layers. It consists of 16 weight layers, which is why it is called "VGG16." **The network is trained on the ImageNet dataset, which contains over 1 million images with 1000 categories.**

  VGG16 has been widely used as a base model for many image classification tasks and has achieved state-of-the-art performance on several benchmarks. It is known for its good performance and efficiency, making it a popular choice for many applications. However,

it is also relatively large and may be slower to train and test compared to more recent CNN architectures.

- **Xception**

  Xception is a convolutional neural network (CNN) architecture that was introduced by François Chollet in his paper "Xception: Deep Learning with Depthwise Separable Convolutions" in 2017. It was developed as an improvement over the Inception architecture, which was introduced in 2014 by Google.

  The Xception architecture is characterized by its use of depthwise separable convolutions, which are a type of convolution that is computationally more efficient than regular convolutions. Depthwise separable convolutions decompose a standard convolution into a depthwise convolution (which applies a single filter to each input channel) and a pointwise convolution (which combines the output of the depthwise convolution using a 1x1 convolution). This allows the network to reduce the number of parameters and computational cost while maintaining good performance.

  Xception has been widely used for image classification tasks and has achieved state-of-the-art performance on several benchmarks. It is known for its good performance and efficiency, making it a popular choice for many applications. However, it is also relatively large and may be slower to train and test compared to some other CNN architectures.

- **VGGFace2**

  VGGFace2 is a convolutional neural network (CNN) architecture for face recognition developed by the Visual Geometry Group (VGG) at the University of Oxford. It is an extension of the original VGGFace architecture, which was introduced in 2014.

  VGGFace2 is trained on a large dataset of celebrity faces, which includes over 3 million images of more than 9,000 celebrities. It is designed to perform face recognition by identifying unique features in the face and comparing them to a database of known faces.

  VGGFace2 has been widely used for face recognition tasks and has achieved good performance on several benchmarks. It is known for its good performance and efficiency, making it a popular choice for many applications. However, it is also relatively large and may be slower to train and test compared to some other CNN architectures.

Source code:

```
def Model_vgg(img_shape, fine_tune=0):
    pretrained_model = applications.vgg16.VGG16(weights='imagenet', include_top=False,input_shape=img_shape)
    model = Sequential()

    for layer in pretrained_model.layers:
        model.add(layer)

    # freezing vgg16 layers
    if fine_tune > 0:
        for layer in model.layers[:-fine_tune]:
            layer.trainable = False
    elif fine_tune == 'all':
        pass
    else:
        for layer in model.layers:
            layer.trainable = False
    # building dense layer
    model.add(Flatten())
    model.add(BatchNormalization())
    model.add(Dense(500,activation='relu'))
    model.add(Dropout(0.8))
#     model.add(Dense(500, activation='relu'))
#     model.add(Dropout(0.6))
#     model.add(Dense(50, activation='relu'))
#     model.add(Dropout(0.3))
    model.add(Dense(10, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2,activation='softmax'))

    opt = keras.optimizers.Adam(learning_rate=0.0001)

    model.compile(optimizer=opt,
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])

    return model
```

Training model with unbalanced data:

We have achieved validation accuracy of approximate 63% while training the model with the unbalanced data (ie. about 16k not_in_pain class and about 900 in_pain class). Now we are going to upsample the in_pain class by data augmentation and down sample the not_in_pain class by removing faces that are similar.

Data augmentation:

We are going to augment the is_in_pain class data with the following parameters and conditions:

- horizontal_flip=True
- rescale=1.1
- rotation_range= 15 #-15 degree - 15 degree
- width_shift_range = 0.2
- height_shift_range = 0.2

- Upsample up to 50% of not_in_plain class size.

Screenshot:

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import random
import cv2



RANDOM_TOKEN = random.getrandbits(128)

DATA_DIR = "1/"
BATCH_SIZE = 64

data_generator = ImageDataGenerator(
    horizontal_flip=True,
    rescale=1.1,
    rotation_range= 15, #-10 degree - 10 degree
    width_shift_range = 0.2,
    height_shift_range = 0.2
)

if not os.path.exists("output/"):
    os.makedir("output/")

def img_augmentor():
    n_class_0 = round(0.5 * len(os.listdir("0/")))
    n_output = n_class_0 // BATCH_SIZE
    c = 0
    train_generator = data_generator.flow_from_directory(
        directory= DATA_DIR,
        target_size=(128, 128),
        color_mode="rgb",
        batch_size=BATCH_SIZE,
        class_mode="sparse",
        subset='training',
        shuffle=True,
        seed=42
    )

    for i in range(n_output):
        batch  = next(train_generator)
        img_b, l_b = batch
        for img in img_b:
            print(f"saving image : {c}")
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            cv2.imwrite(os.path.join("output", str(RANDOM_TOKEN)+"_aug_"+str(c)+".jpg"), img_rgb)
            c+= 1

img_augmentor()
```

Fig: screenshot of data augmentor

Dataset summary after data augmentation:

After down and upsampling dataset we have:

- About 5k faces that belongs to not_in_pain class
- About 4.7k faces that belongs to in_pain

We still don’t have a perfectly balanced dataset, to handle this we are going to use class_weights while model training.

Training Model with balanced dataset:

After training the same model with the balanced dataset we have got 98% validation accuracy which is pretty good accuracy. However, due to the limitation of the dataset the model might not be able to fit in every scenario.

Screenshots:

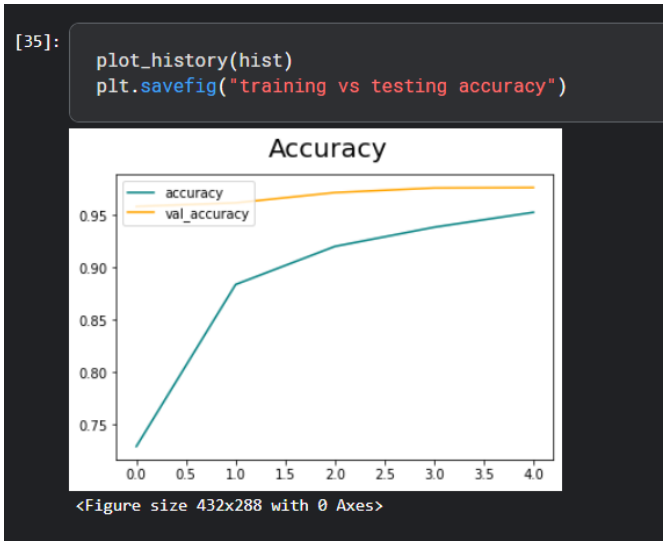


Fig: Training vs validation accuracy

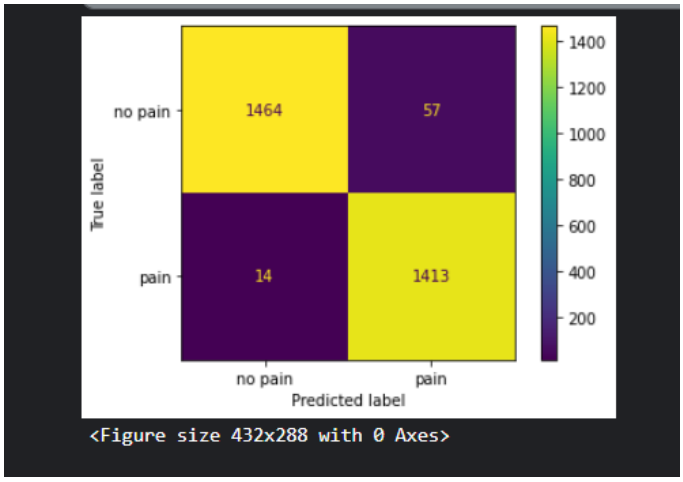


Fig: confusion matrix

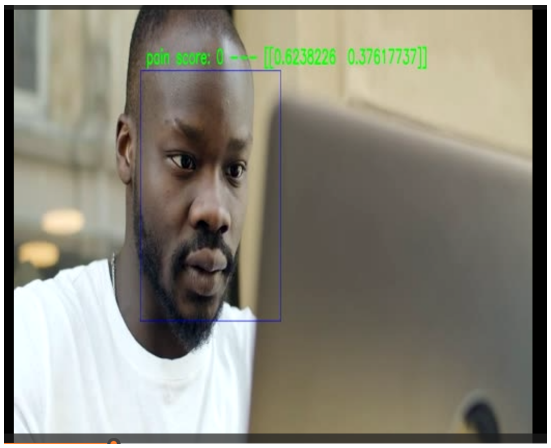
Classification Report

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1521
1	0.96	0.99	0.98	1427
accuracy			0.98	2948
macro avg	0.98	0.98	0.98	2948
weighted avg	0.98	0.98	0.98	2948

Fig: Classification Report

"Testing" results:

Screenshot of test result:



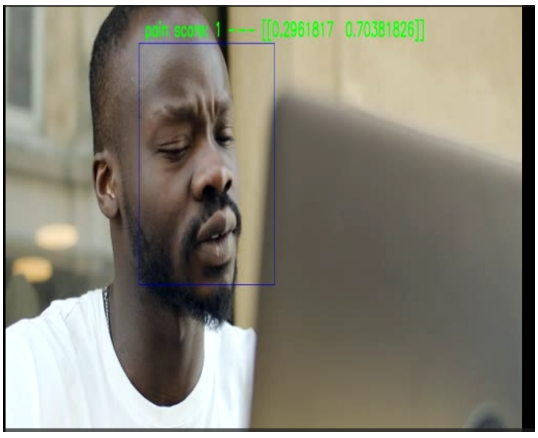


Fig: successfully recognize not_in_pain class (pain score: 0) and in_pain class (pain score: 1)

Fig: successfully distinguished pain face and laughing face

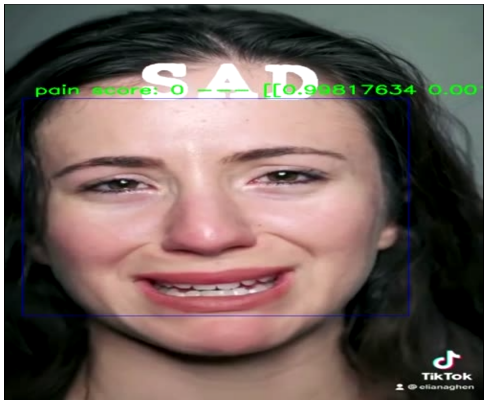


Fig: unable to detect pain face by vggface2

We got pretty good training and validation accuracy, however upon testing on training dataset and live stream, the model was unable to recognize pain class properly. The possible reason for this result is a highly unbalanced dataset and the quality of the data. Now, instead of working with this dataset we can create our own custom dataset.

CUSTOM DATASET (Fellows' dataset)

To collect our custom dataset, we record various facial expressions (neutral, laughing, yawning, expression of having pain) of 5 fellows which are used only for training and facial expressions of other 2 fellows are used for validation only and the remaining 1 fellow's facial expressions are used for testing.

Screenshot:

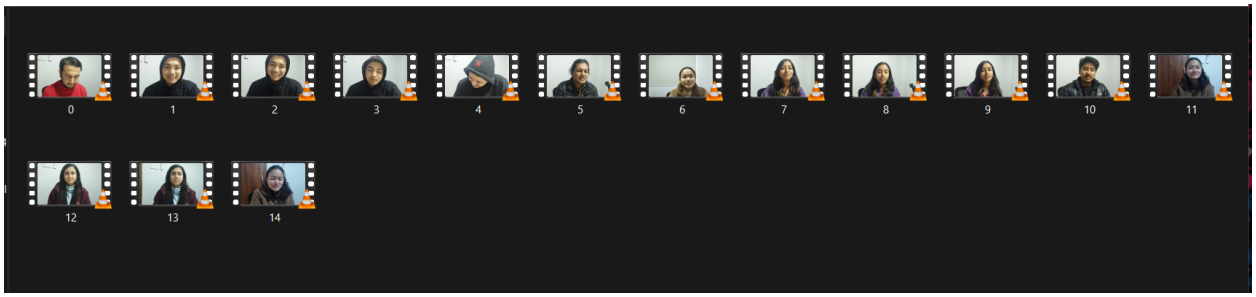


Fig: recording of facial expressions of 7 fellows

Extracting and labeling facial expression from recordings:

We are going to use video_to_dataset script to extract and label the facial expression into two classes i.e "not_in_pain" and "in_pain".

Screenshot:

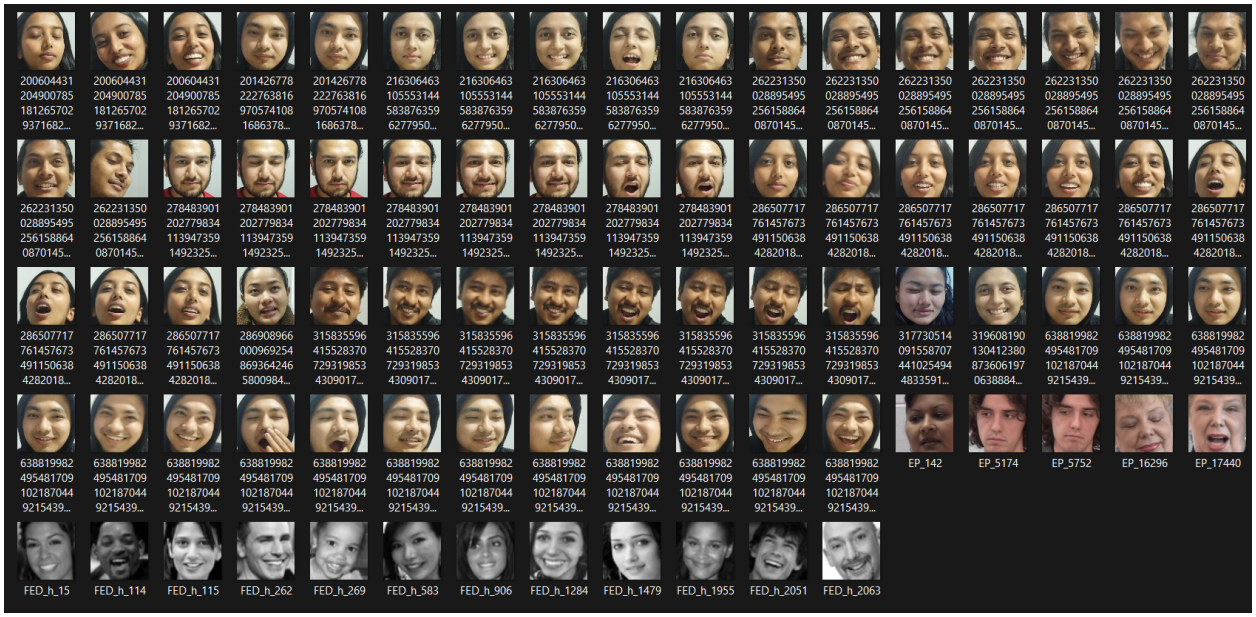


Fig: screenshot of “not_in_pain” class dataset

We have added some data from our previous dataset that could help to avoid overfitting.

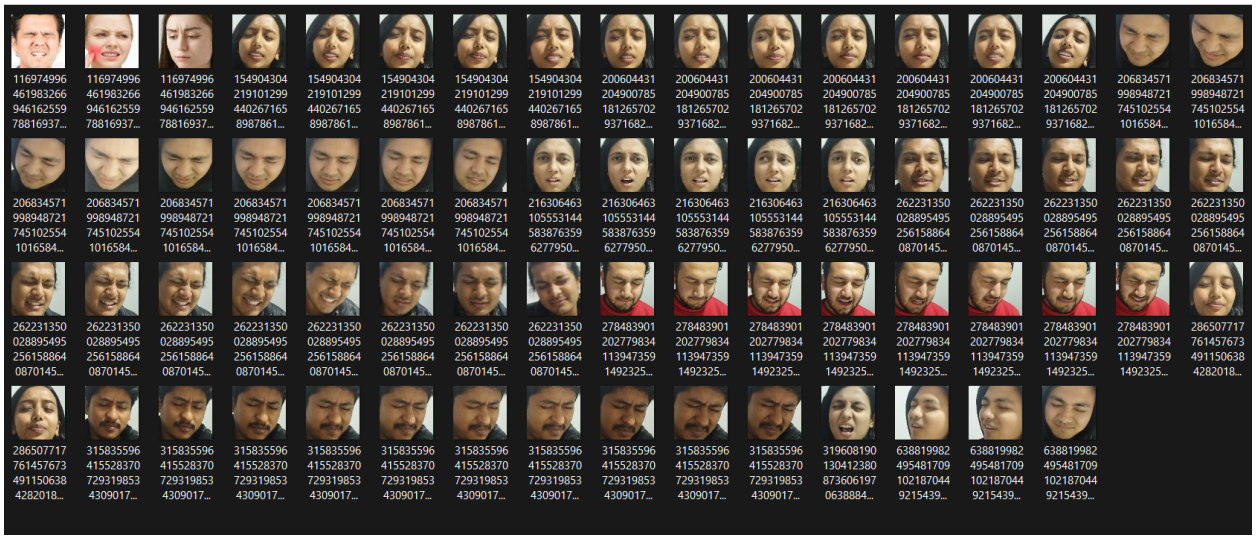


Fig: Screenshot of “is_in_pain” class dataset

Dataset Summary:

Dataset	class	size
Training	not_in_pain	80
	in_pain	66
Validation	not_in_pain	18
	in_pain	23
Test	not_in_pain	102
	in_pain	47

Table: summary of the custom dataset

Model Building:

We have used the concept of transferred learning where vgg16 and vggface2 pretrained models to create a pain recognition model. The performance of the vggface2 model was better than the vgg16.

Screenshot:

```python
from keras_vggface.vggface import VGGFace

def VGGFace_Custome(model_n="resnet50", fine_tune=0):
    vggface = VGGFace(model=model_n, include_top=False, input_shape=(224, 224, 3), pooling='avg')
    model = Sequential()

    model.add(vggface)

    # freezing vgg16 layers
    if fine_tune > 0:
        for layer in model.layers[:-fine_tune]:
            layer.trainable = False
    else:
        for layer in model.layers:
            layer.trainable = False

    # building dense layer
    model.add(Flatten())
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(0.6))

    model.add(Dense(50, activation='relu'))
    model.add(Dropout(0.4))

    model.add(Dense(2,activation='softmax'))

#     lr_schedule = keras.optimizers.schedules.ExponentialDecay(
#     initial_learning_rate=1e-2,
#     decay_steps=10,
#     decay_rate=0.1)

#     opt = keras.optimizers.Adam(learning_rate=lr_schedule)

    model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

    return model
```

Fig: the source code of pain recognition model with vggface2 pretrained model

Performance evaluation of vgg16 and vggface2:

Below figure shows how the performance of vggface2 is better than vgg16.

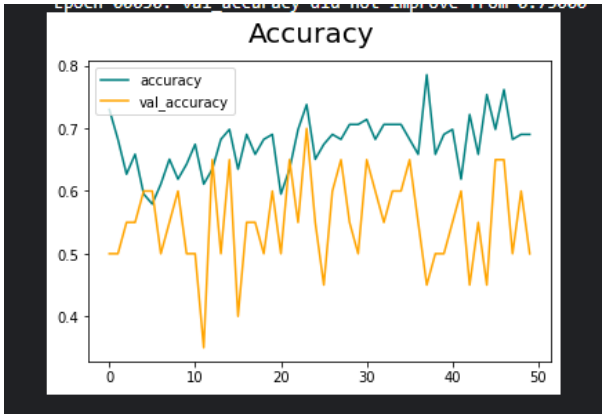


Fig: Train vs validation accuracy curve of model with vgg16

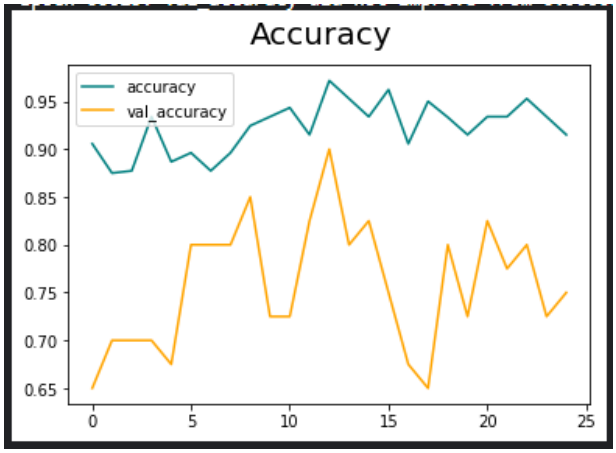


Fig: Train vs validation accuracy curve of model with vggface2

Fine Tuning:

We did fine tuning of the pain recognition model with vggface2 to improve its accuracy, and were able to achieve train accuracy of 95% and validation accuracy of 85%. The reason behind this high variance between train and validation accuracy is the number of images available in them.

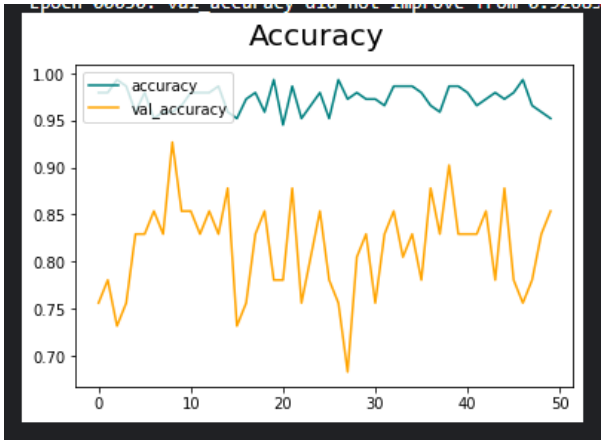


Fig: Train vs validation accuracy curve after fine tuning

Performance evaluation:

we can see the below confusion matrix, and performance report to analyze the performance of the model. And we can clearly conclude that, based on the size of the dataset, the performance of the model is quite acceptable.

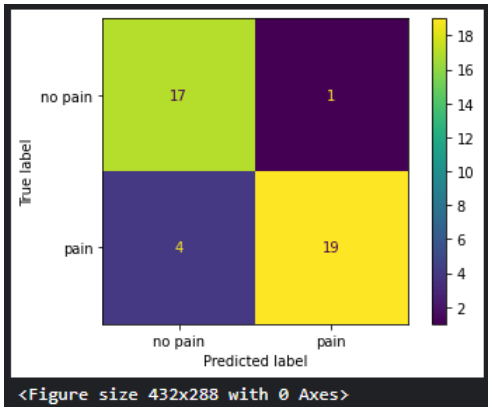


Fig: confusion matrix of validation result

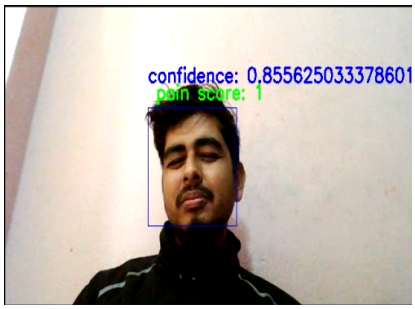
Classification Report

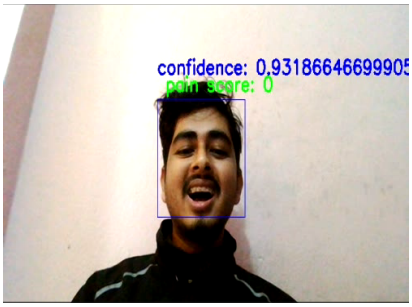
	precision	recall	f1-score	support
0	0.81	0.94	0.87	18
1	0.95	0.83	0.88	23
accuracy			0.88	41
macro avg	0.88	0.89	0.88	41
weighted avg	0.89	0.88	0.88	41

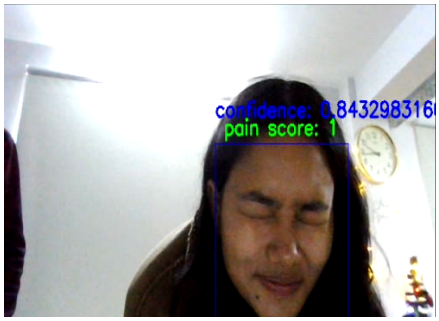
Fig: screenshot of the classification report

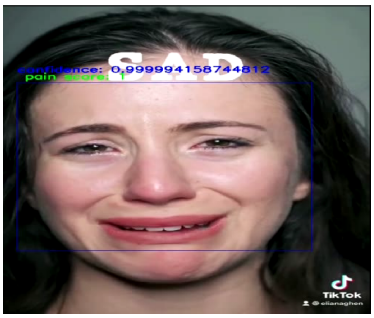
Test result:

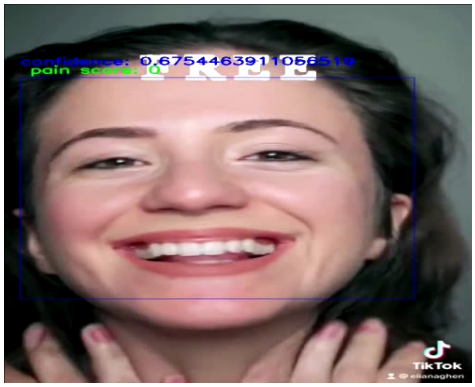
Figures below are the screenshot of the performance of the model while testing. Note that, pain score : 1 represents **in_pain** class, and pain score: 0 represents **not_in_pain** class.











TEST DATASET:

To evaluate the model performance, a separate test dataset is prepared which contains about 149 images out of which 102 images belong to class not_in_pain class and the rest 47 images belong to in_pain_class.

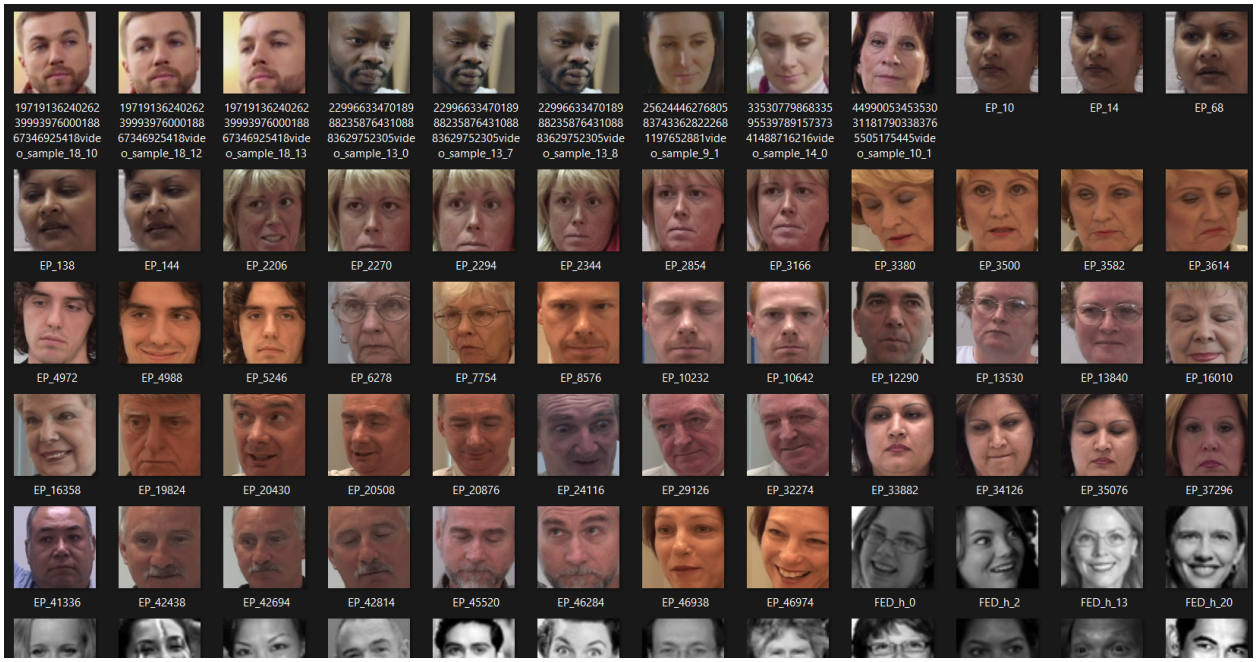
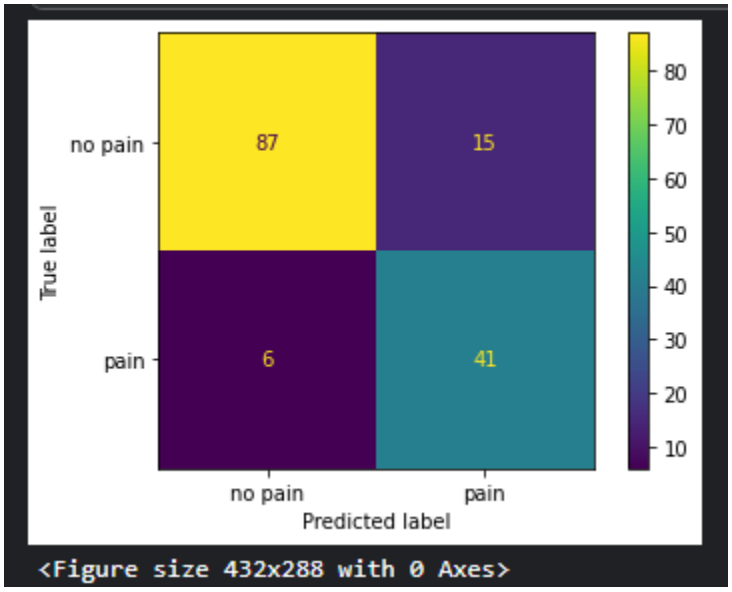


Fig: sample images of not_in_pain class

Fig: sample images of in_pain class

TEST RESULT:




```
/opt/conda/lib/python3.7/site-packages/keras/engine/train
sion. Please use `Model.predict`, which supports generato
  warnings.warn('`Model.predict_generator` is deprecated
Confusion Matrix:
[[87 15]
 [ 6 41]]
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.85      0.89       102
           1       0.73      0.87      0.80        47

    accuracy                           0.86       149
   macro avg       0.83      0.86      0.84       149
weighted avg       0.87      0.86      0.86       149
```

NEW DATASET SUMMARY:

Dataset	class	size
Training	not_in_pain	107
	in_pain	78
Validation	not_in_pain	18
	in_pain	23
Test	not_in_pain	75
	in_pain	35

HYPER-PARAMETER:

Learning rate = 0.001
Beta_1 = 0.88

NEW DENSE LAYER:

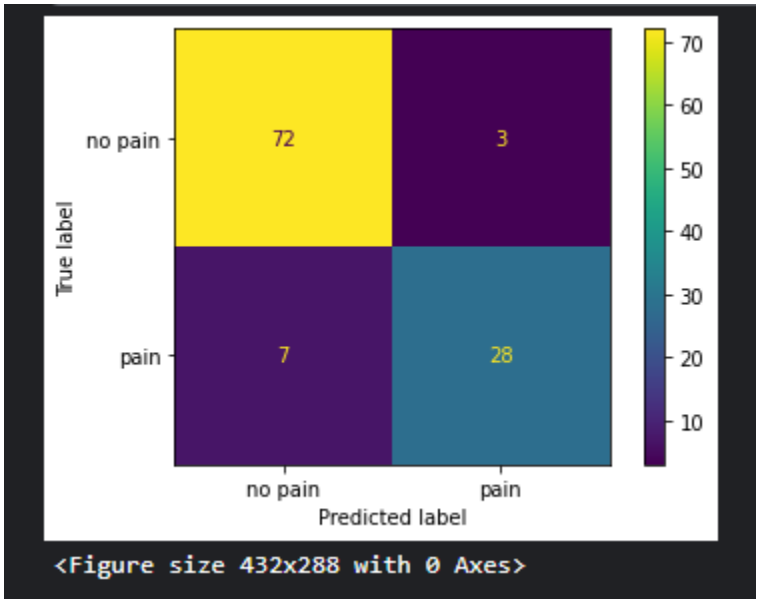
```
# building dense layer
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(500, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(10, activation='relu'))

model.add(Dense(2,activation='softmax'))
```

VALIDATION NEW RESULT:

Classification Report

	precision	recall	f1-score	support
0	0.82	1.00	0.90	18
1	1.00	0.83	0.90	23
accuracy			0.90	41
macro avg	0.91	0.91	0.90	41
weighted avg	0.92	0.90	0.90	41

TEST RESULT NEW:



Classification Report

	precision	recall	f1-score	support
0	0.91	0.96	0.94	75
1	0.90	0.80	0.85	35
accuracy			0.91	110
macro avg	0.91	0.88	0.89	110
weighted avg	0.91	0.91	0.91	110

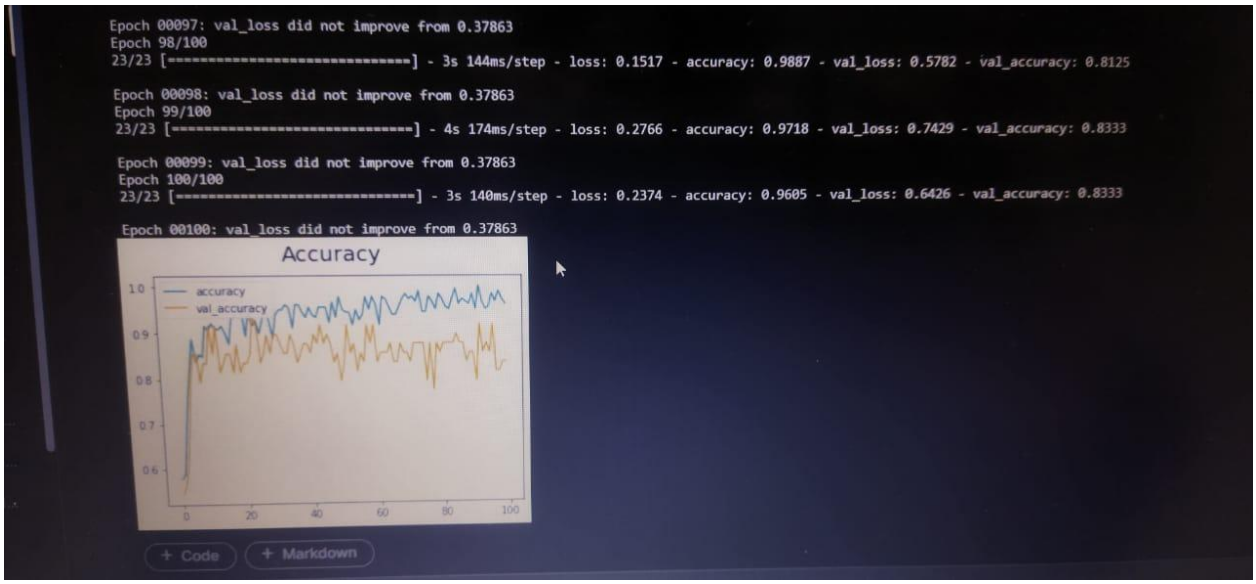
Analyzing Confusion Matrix

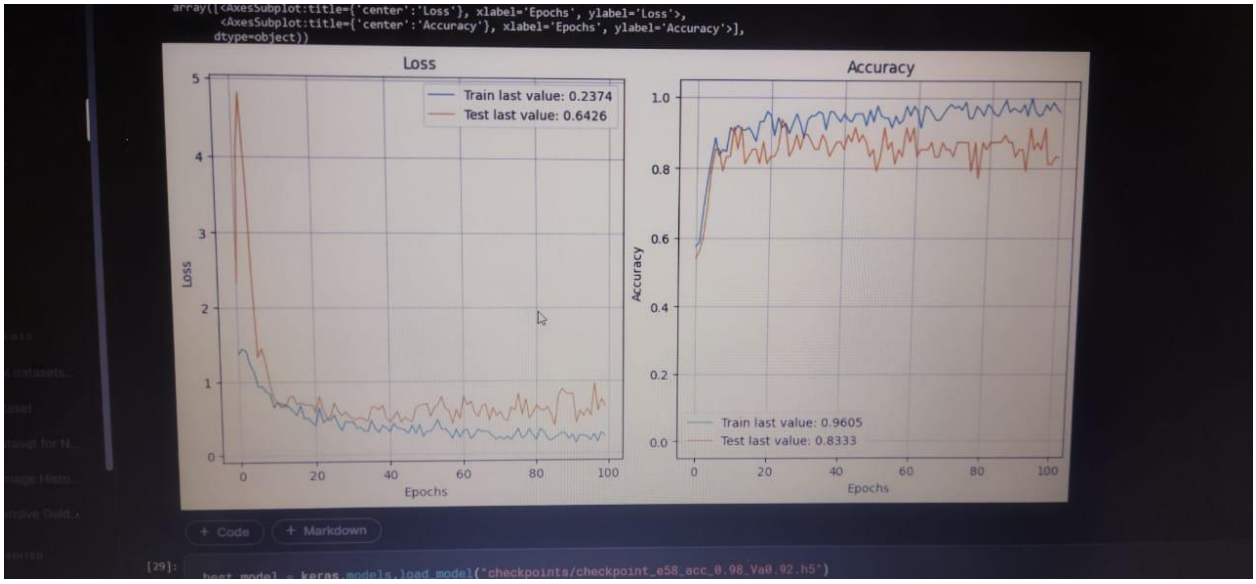
After analyzing the confusion matrix it is found that all the misclassifications belongs to those image dataset that are fetched from another dataset/internet.

Misclassified images:

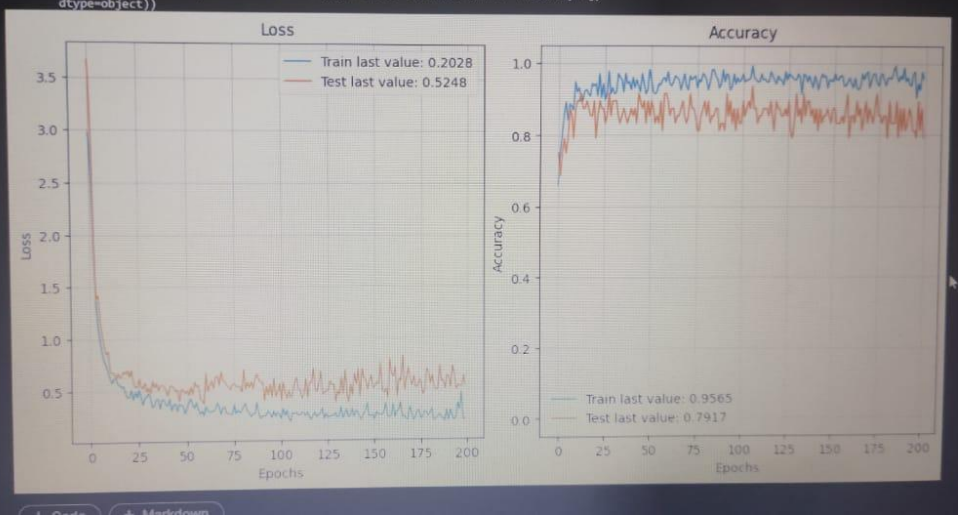
Hyper-Parameter Tunning and changing Model Architecture

Upon changing the model architecture and doing hyper-parameter tunning, we cannot achieved validation accuracy more that 85% and validation loss less than 0.3.





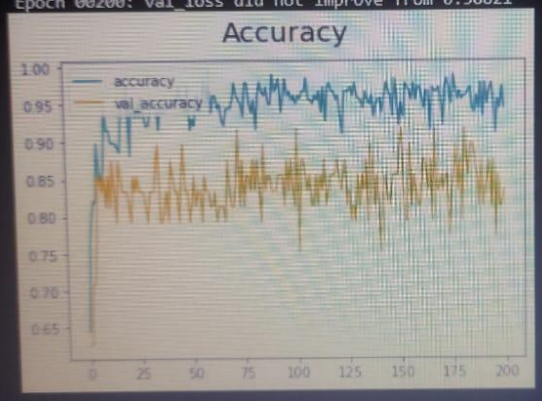
[21]: (<Figure size 1000x500 with 2 Axes>,
array([<AxesSubplot:title={'center':'Loss'}, xlabel='Epochs', ylabel='Loss'>,
<AxesSubplot:title={'center':'Accuracy'}, xlabel='Epochs', ylabel='Accuracy'>],
dtype=object))



23/23 [==========] - 4s 167ms/step - loss: 0.2201 - accuracy: 0.9728 - val_loss: 0.

Epoch 00199: val_loss did not improve from 0.38821
Epoch 200/200
23/23 [==========] - 4s 163ms/step - loss: 0.2221 - accuracy: 0.9435 - val_loss: 0.

Epoch 00200: val_loss did not improve from 0.38821



+ Code + Markdown