THE UNIVERSITY OF TEXAS AT EL PASO

# ML Project – Team 2: Face Recognition using Convolutional Neural Networks

Fernando Sepulveda, Derek Cisneros, Marco Garcia Fernandez, Francisco Parra

CRN 25011, Computer Science
The University of Texas at El Paso
El Paso, TX
April 22, 2024

# Overview

- **Problem Statement:** Background and Research Problem
- **Data:** Data Prep and Training
- **Methodology:** Architecture, Training
- **Features:** Highlights and Problems
- **Results:** Mapping and evaluation
- **Summary:** Review, Impact, Future Work

# Problem Statement: Background

- Face recognition purpose:
- Identification or verification of individuals
- Applications:
- surveillance, access control, authentication, and personalized services

FS

# Why ML?

- Traditional methods of face recognition often rely on handcrafted features
- CNN:
  - learn and extract complex patterns from vast amounts of data
  - excel in feature learning and abstraction,

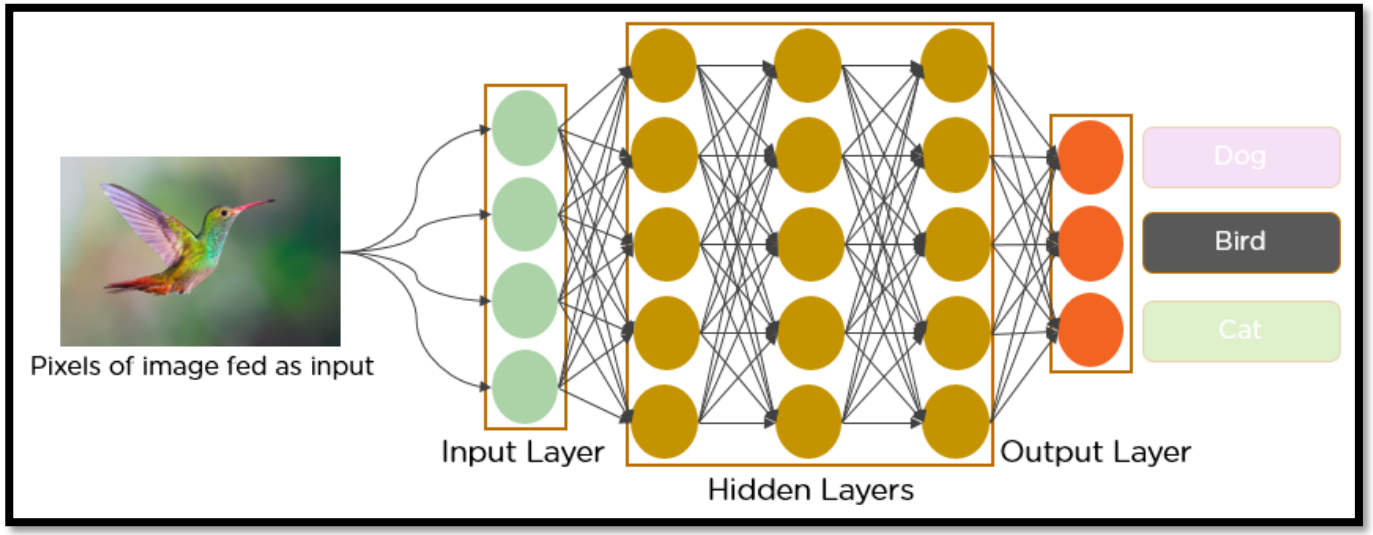



**Fig. 3:** CNN Process Example

# Data Preparation

| | pixel_0 | pixel_1 | pixel_2 | pixel_3 | pixel_4 | pixel_5 | pixel_6 | pixel_7 | pixel_8 | pixel_9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 117 | 97 | 84 | 72 | 66 | 60 | 67 | 72 | 71 | 62 | ... |
| 1 | 254 | 253 | 254 | 254 | 254 | 255 | 253 | 249 | 214 | 207 | ... |
| 2 | 252 | 255 | 255 | 252 | 255 | 254 | 251 | 237 | 178 | 99 | ... |
| 3 | 253 | 254 | 233 | 244 | 254 | 254 | 253 | 254 | 253 | 253 | ... |
| 4 | 247 | 247 | 247 | 246 | 245 | 245 | 243 | 241 | 237 | 227 | ... |

**Fig. 2:** Header Data Set

```python
import os
import pandas as pd
from tqdm import tqdm
import numpy as np
from PIL import Image

# Define folder names
folders = ['derek1000']

# Define path to the folder containing subfolders
main_path = '/Users/magarciafer/Desktop/ml/Cnn/Dataset_Creation/Datasets'

# Define output CSV filename
filename = 'derek100test.csv'

# Target dimensions for the images
dim = (100, 100)

cls = 0  # Initialize to 0, which will be incremented inside the loop

# Create an empty DataFrame
df = pd.DataFrame(columns=[f'pixel_{i}' for i in range(dim[0] * dim[1])] + ['class'])

# Find the minimum number of images in any folder
min_images = float('inf')
for folder in folders:
    folder_path = os.path.join(main_path, folder)
    files = os.listdir(folder_path)
    if len(files) < min_images:
        min_images = len(files)

# Process images ensuring equal number per class
for folder in folders:
    cls += 1  # Increment class label for each folder
    folder_path = os.path.join(main_path, folder)
    files = os.listdir(folder_path)
    # Use min_images to limit the number of processed files
    for i in tqdm(range(min_images)):
        img = Image.open(os.path.join(folder_path, files[i])).convert('L')  # Convert image to grayscale
        img_resized = img.resize(dim)  # Resize image to 100x100
        img_array = np.array(img_resized).flatten()  # Convert image to array and flatten
        df.loc[len(df)] = list(img_array) + [cls]  # Append flattened image data and class to the DataFrame

# Set the index to start from 0
df.index.name = ''

# Save DataFrame to CSV
df.to_csv(filename, index=True)
print('Task Completed')
```

**Fig. 3:** Script Convert to CSV

```python
# Importing Modules guu
import cv2
import numpy as np
import os
from tqdm import tqdm
#marco was here
def Crete_Folder_Images(name, directory, tn):
    # Creating directory in the file
    if not os.path.exists(directory):
        os.makedirs(directory)
    else:
        print('DIRECTORY with name %s EXIST'%name)
    # Opening vedio mode with frontal face detector
    faceDetect=cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml") #cv2.Cascad
    cam=cv2.VideoCapture(0)
    size =(100, 100)
    for i in tqdm(range(1, 1+ tn)):
        ret,img=cam.read()
        gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)  #for GRAY SCALE IMAGES
        faces=faceDetect.detectMultiScale(gray,1.3,5)
        # faces=faceDetect.detectMultiScale(img,1.3,5)    # FOR COLOR IMAGES
        for(x,y,w,h) in faces:
            res = cv2.resize(gray[y:y+h,x:x+w], size, interpolation = cv2.INTER_AREA)
            cv2.imwrite('Datasets/'+str(name)+'/'+str(name)+'_'+str(i)+'.jpg', res)  #for GRAY SCALE IMAGES

            # cv2.imwrite(directory+'/'+str(name)+'-'+str(sn)+'.jpg'
            #             ,img[y:y+h,x:x+w])  # FOR COLOR IMAGES
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
            cv2.waitKey(200)
        cv2.imshow('DAtaset Creator',img)
        cv2.waitKey(1)
    cam.release()
    cv2.destroyAllWindows()
    print('TASK COMPLETED')

# Taking inputs

name=input('\nEnter your name: ')

directory='Datasets/'+name
tn= int(input('Enter no.of images to be taken: '))
Crete_Folder_Images(name, directory, tn)
```

**Fig. 4:** Script Photo Automation

- Run a script to take 2,500 photos per class (Member team)
- Pictures are resized to a standard 100 * 100 pixels
- Labels are one-hot encoded into a csv
- Data is split into training and testing sets

DC

# Training Process

**Statistics of the Dataset**

- Has a leader dynamic
- count 6796.000000
- mean 2.500000
- std 1.118116
- min 1.000000
- 25% 1.750000
- 50% 2.500000
- 75% 3.250000
- max 4.000000
- Name: class, dtype: float64

Class Labels: [1, 2, 3, 4]

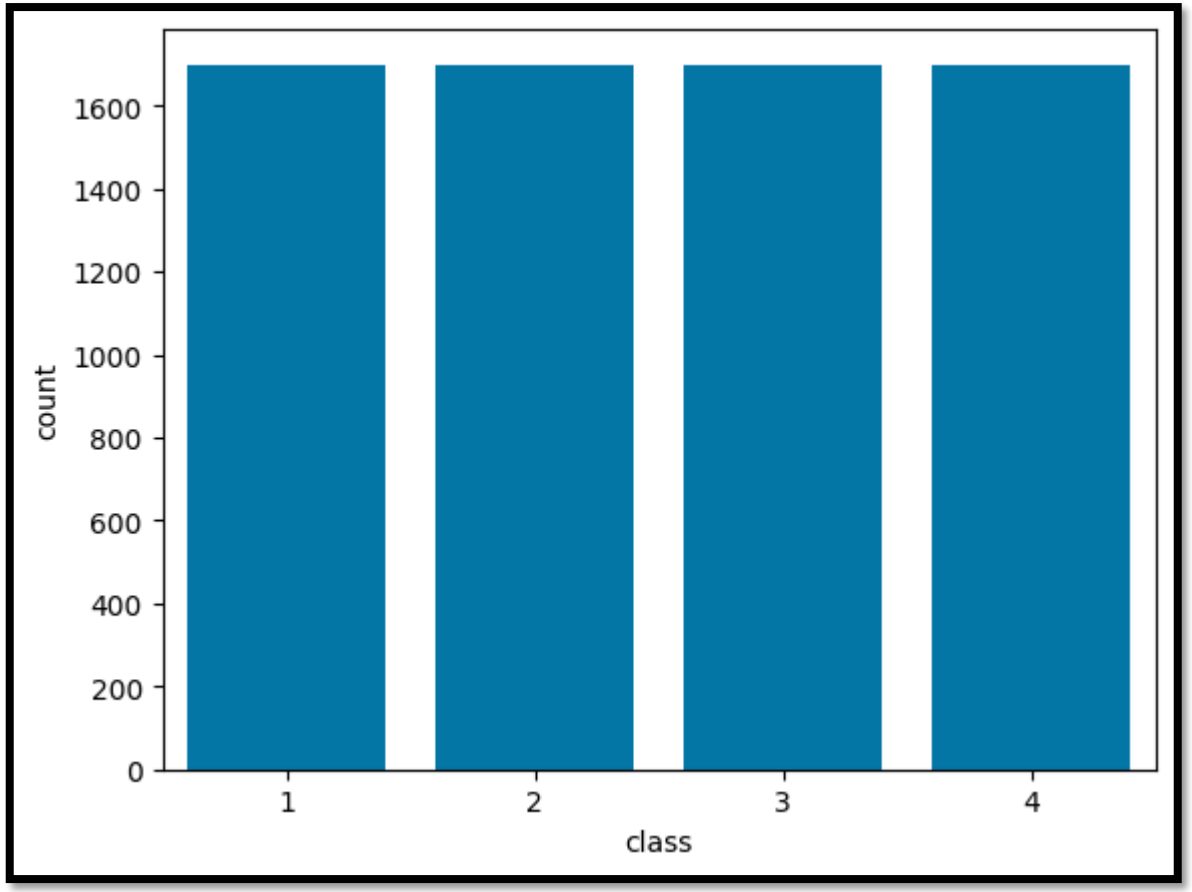
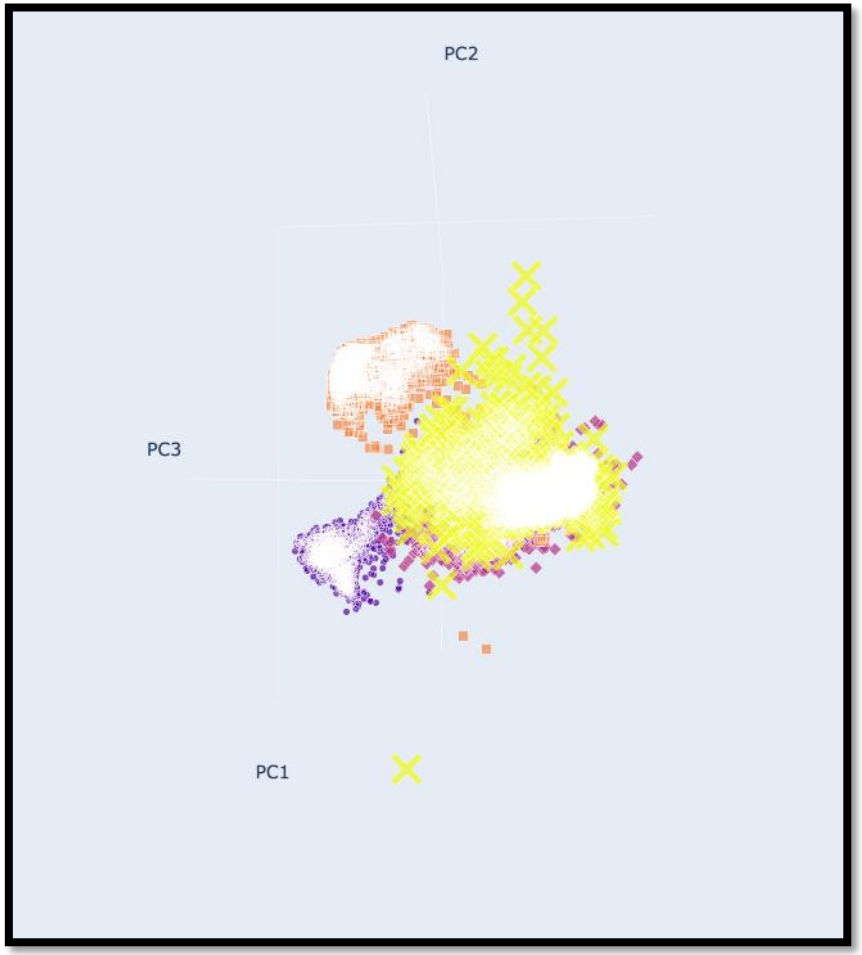
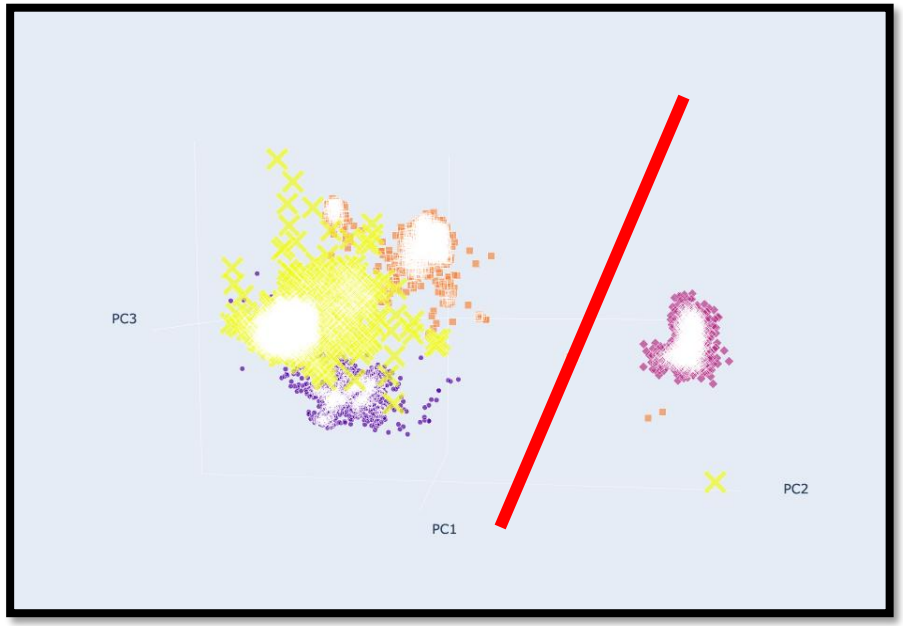



**Fig. 5:** Class Photos

# Clusters

Some clusters are linearly separable in certain dimensions



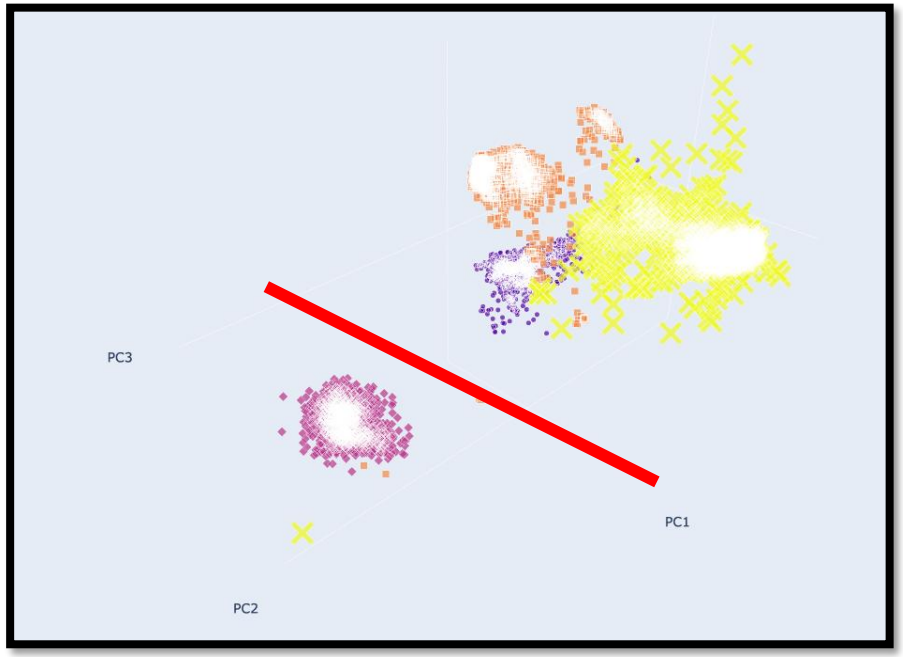




**Fig. 6,7,8:** Cluster of 4 classes

MG

# Model Proposed

```
model = Sequential(name = model_name)

model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(100, 100, 1)))
model.add(BatchNormalization()) #-----------------
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization()) #-----------------
model.add(Conv2D(64, kernel_size=5, padding='same', activation='relu'))
model.add(BatchNormalization()) #-----------------
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2)) #-----------------

model.add(Conv2D(128, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size=5, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(256, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Dense(5, activation='softmax'))

model.summary()
```

**Fig. 9:** Script Photo Automation

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 98, 98, 64) | 640 |
| batch_normalization (BatchNo | (None, 98, 98, 64) | 256 |
| conv2d_1 (Conv2D) | (None, 96, 96, 64) | 36928 |
| batch_normalization_1 (Batch | (None, 96, 96, 64) | 256 |
| conv2d_2 (Conv2D) | (None, 96, 96, 64) | 102464 |
| batch_normalization_2 (Batch | (None, 96, 96, 64) | 256 |
| max_pooling2d (MaxPooling2D) | (None, 48, 48, 64) | 0 |
| dropout (Dropout) | (None, 48, 48, 64) | 0 |
| conv2d_3 (Conv2D) | (None, 46, 46, 128) | 73856 |
| batch_normalization_3 (Batch | (None, 46, 46, 128) | 512 |
| conv2d_4 (Conv2D) | (None, 44, 44, 128) | 147584 |
| batch_normalization_4 (Batch | (None, 44, 44, 128) | 512 |
| conv2d_5 (Conv2D) | (None, 44, 44, 128) | 409728 |
| batch_normalization_5 (Batch | (None, 44, 44, 128) | 512 |
| max_pooling2d_1 (MaxPooling2 | (None, 22, 22, 128) | 0 |
| dropout_1 (Dropout) | (None, 22, 22, 128) | 0 |
| conv2d_6 (Conv2D) | (None, 20, 20, 256) | 295168 |
| batch_normalization_6 (Batch | (None, 20, 20, 256) | 1024 |
| max_pooling2d_2 (MaxPooling2 | (None, 10, 10, 256) | 0 |
| dropout_2 (Dropout) | (None, 10, 10, 256) | 0 |
| flatten (Flatten) | (None, 25600) | 0 |
| dense (Dense) | (None, 256) | 6553856 |
| batch_normalization_7 (Batch | (None, 256) | 1024 |
| dense_1 (Dense) | (None, 128) | 32896 |
| batch_normalization_8 (Batch | (None, 128) | 512 |
| dense_2 (Dense) | (None, 5) | 645 |

Total params: 7,658,629
Trainable params: 7,656,197
Non-trainable params: 2,432

**Fig. 10:** Script Photo Automation

# Problem Encounter: Overfitting

- Complex Model: With its 7.6 million parameters, the complex model is learning very detailed features from the data.

- high level of detail can be a disadvantage if the training dataset is not large or diverse enough.

- The model starts learning noise and errors in the data as if they were significant features, leading to overfitting.

- It might perform well on the training data, it could perform poorly on new, unseen data because it has learned to fit the training data too closely.

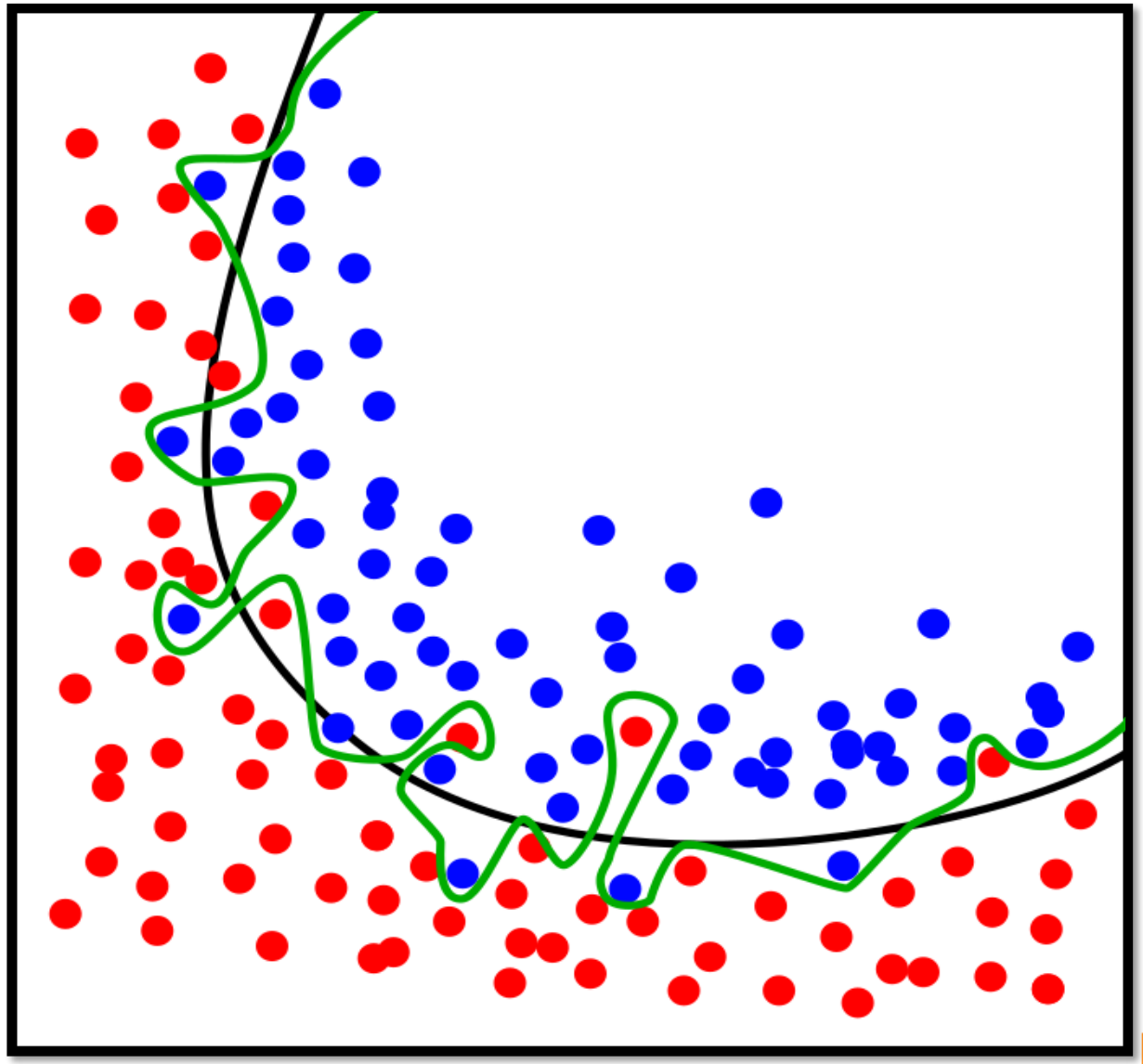


**Fig. 11:** Script Photo Automation

MG

```
Model: "simple_cnn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 98, 98, 64)        640

 batch_normalization_35 (Ba  (None, 98, 98, 64)        256
 tchNormalization)

 max_pooling2d_11 (MaxPooli  (None, 49, 49, 64)        0
 ng2D)

 dropout_15 (Dropout)        (None, 49, 49, 64)        0

 flatten_3 (Flatten)         (None, 153664)            0

 dense_11 (Dense)            (None, 5)                 768325

=================================================================
Total params: 769221 (2.93 MB)
Trainable params: 769093 (2.93 MB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
```

**Fig. 12: Edited Model**

# Model Architecture

- TensorFlow's Keras API, .
- Input Layer: The first layer is a 2D convolutional layer (Conv2D) with 64 filters, each with a kernel size of 3x3.
- Activation function: ReLU
- Batch Normalization:  maintains the mean output close to 0 and the output standard deviation close to 1.
- Max Pooling: A 2D layer follows, with a pool size of 2x2.
- Dropout: This layer randomly sets a fraction (20% in this case) of the input units to zero at each update during training time. This helps prevent overfitting.
- Flatten:  flattens the output of the previous layers into a single long feature vector
- Output Layer: 5 units and a softmax activation function. where each of the 5 units corresponds our class categorization problem

# Training


**Fig. 13:** Classes Display

**Fig. 14:** Eigen Faces of the Dataset

# Results

```
32/32 [==============================] - 1s 16ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       265
           2       1.00      1.00      1.00       258
           3       1.00      1.00      1.00       256
           4       1.00      1.00      1.00       241

    accuracy                           1.00      1020
   macro avg       1.00      1.00      1.00      1020
weighted avg       1.00      1.00      1.00      1020
```

**Fig. 4:** Automation Levels

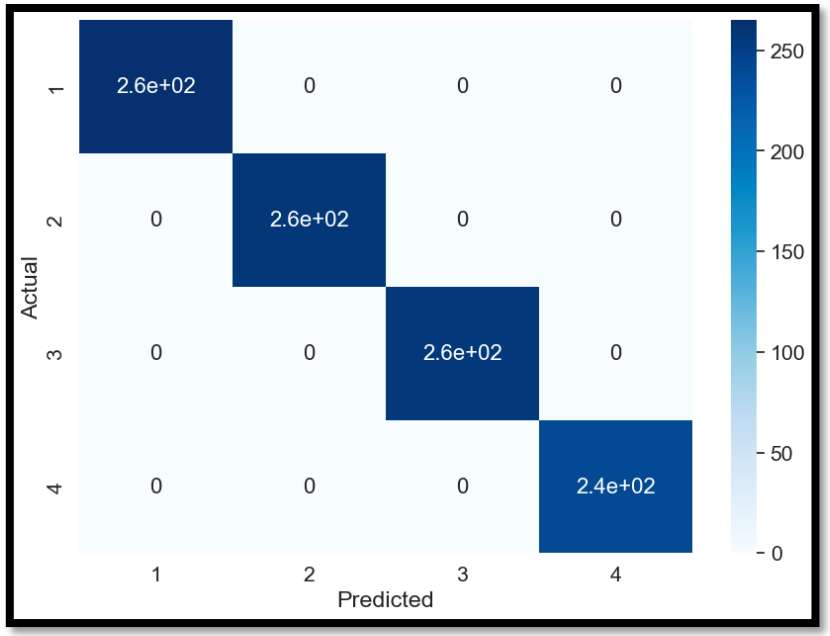



**Fig. 11:** Confusion Matix Accuracy

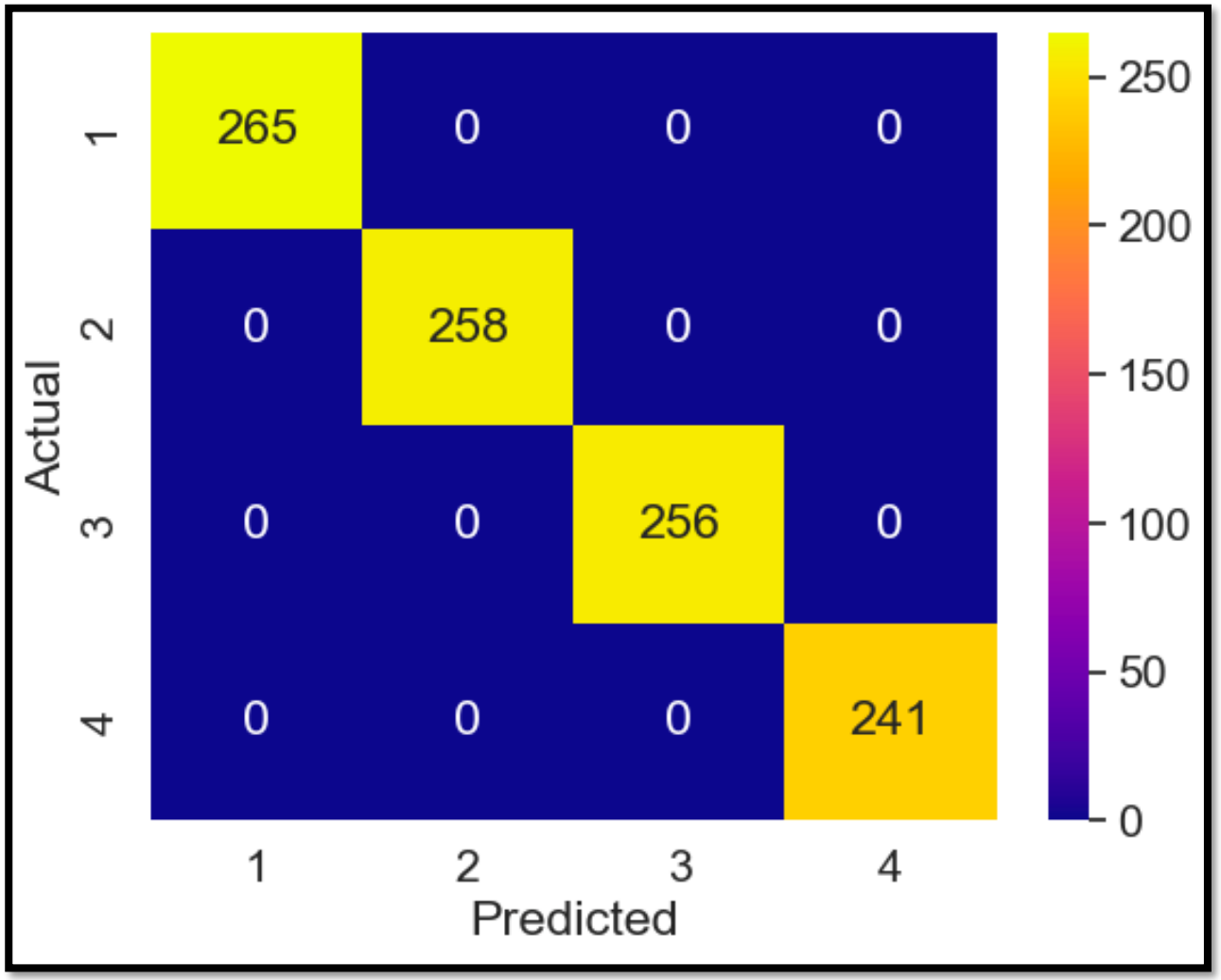



**Fig. 3:** Script Photo Automation

FP

# Results

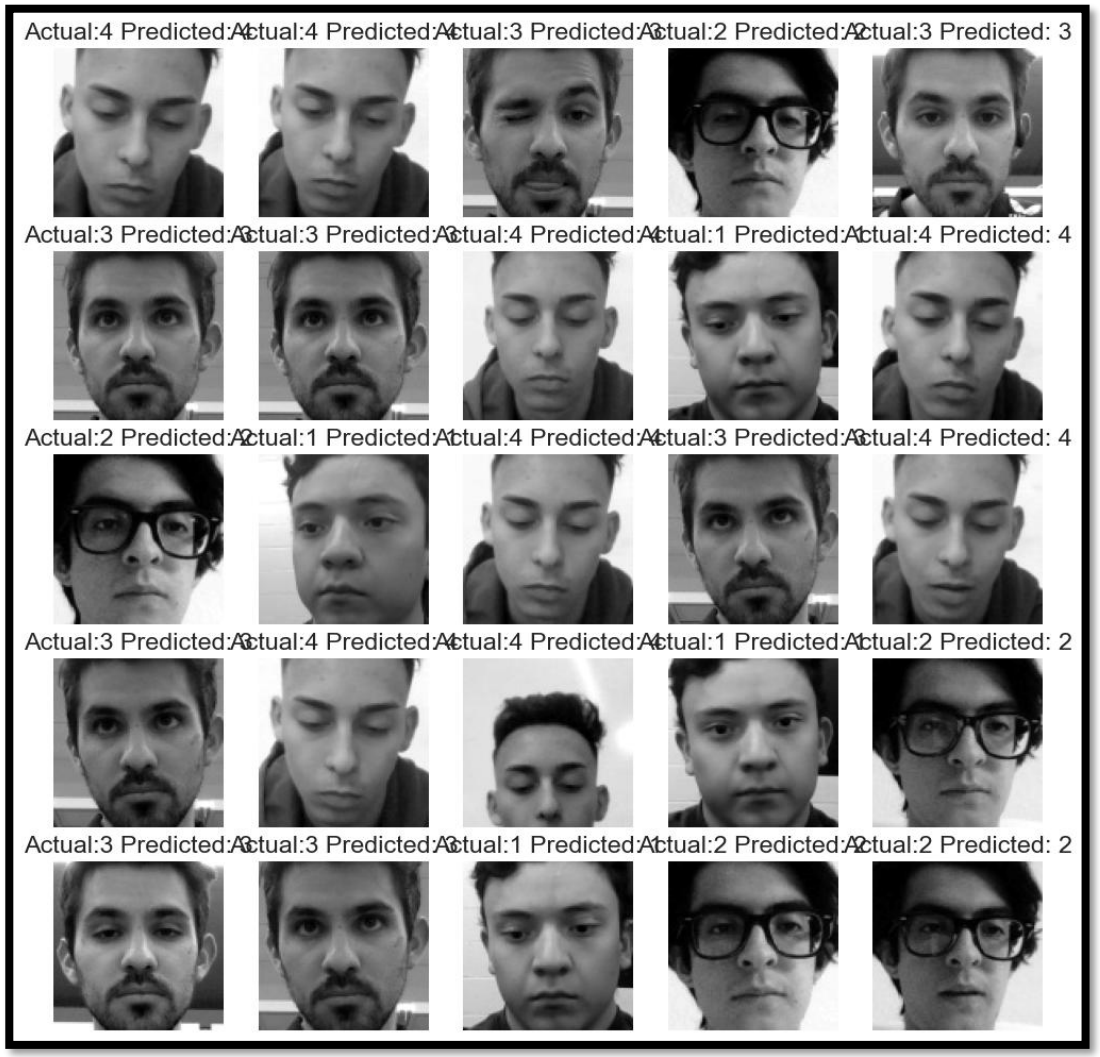



**Fig. 12:** Sampe Results

# Gradient-weighted Class Activation Mapping

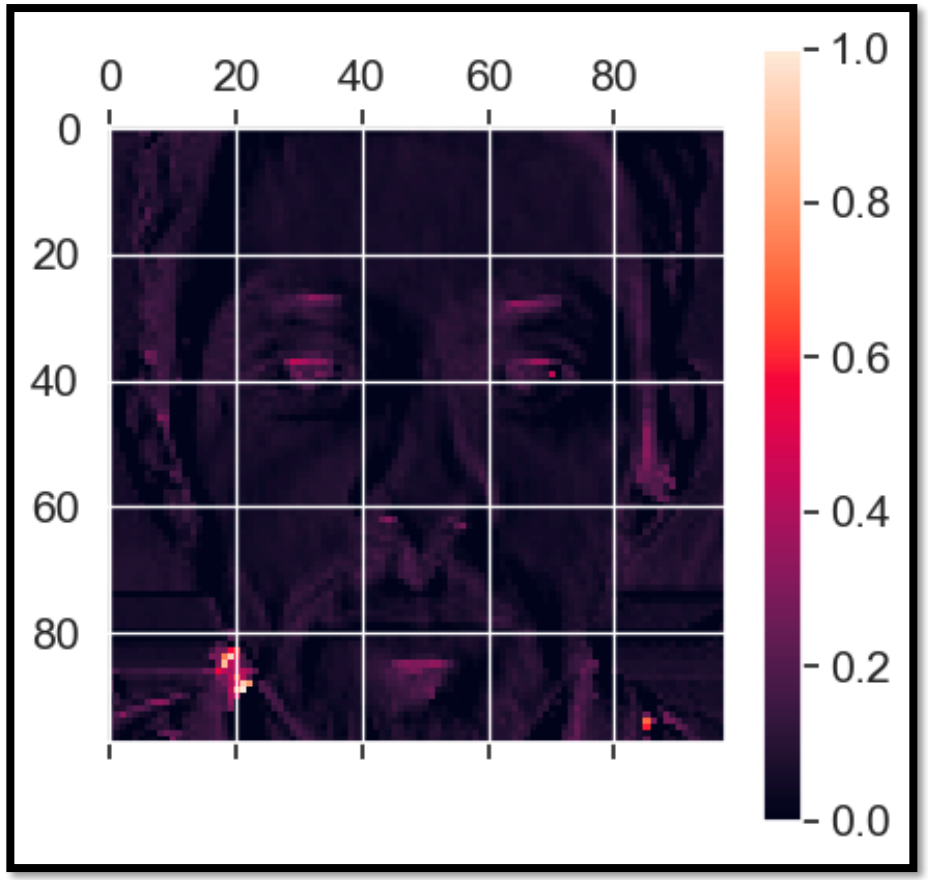



**Fig. 14:**Heatmap Weight


**Fig. 15:** Weight Calculation Classes

# Gradient-weighted Class Activation Mapping

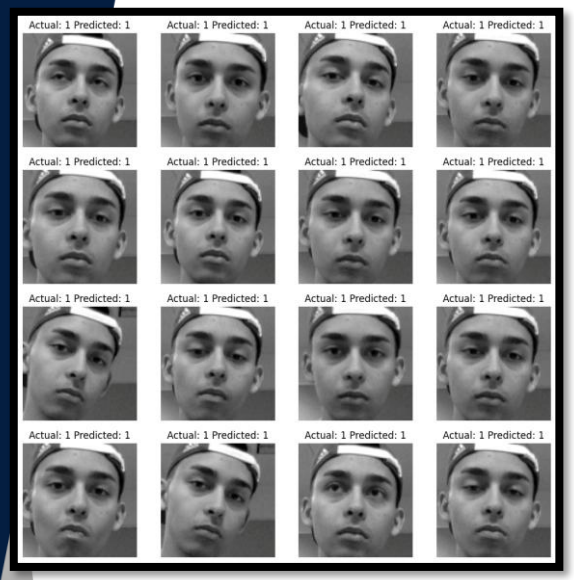



**Fig. 16:** Derek + Hat

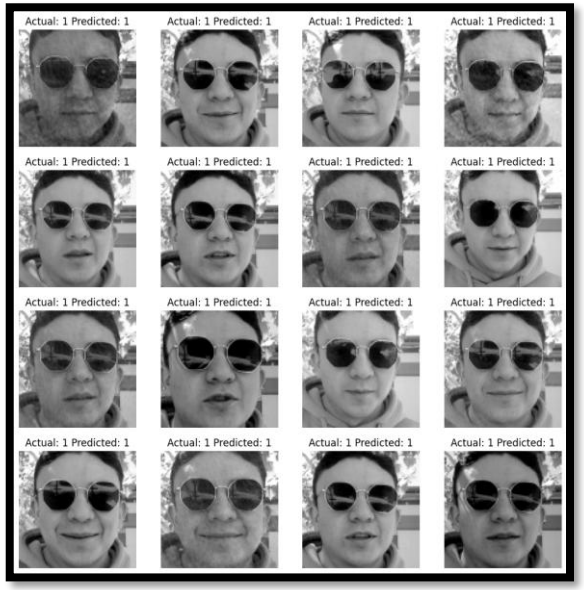



**Fig. 17:** Fernando + Glasses

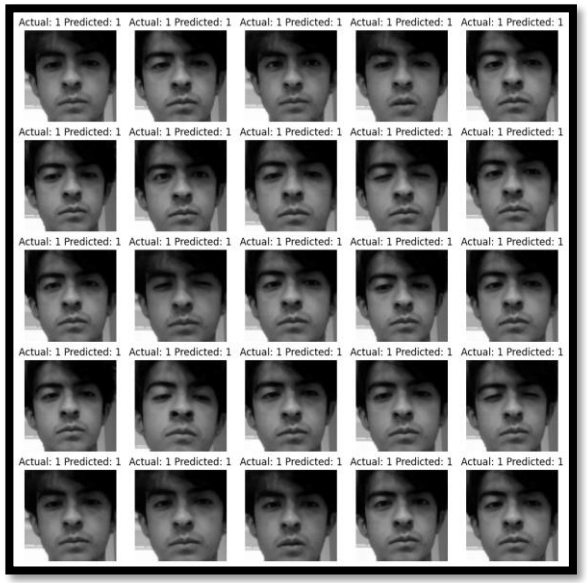



**Fig. 18:** Francisco – No Glasses


**Fig. 19:** Marco + Glasses

# Summary

Our model has proven to be a reliable implementation of a CNN. However, one significant challenge we've encountered is overfitting, particularly when the dataset is small. This underscores the importance of data size in ensuring the model's robustness. Scaling up the project to a larger dataset would undoubtedly require higher computational power, considering factors such as processing time, data collection time, and analysis time.

This project has promising applications beyond its current scope. For instance, it could be expanded to serve as an identification feature for students during exams or for simple security process.

# **Takeaways:** future & details

1. **Effective CNN Implementation:** Demonstrated CNN's effectiveness for the task.

2. **Overfitting Challenge:** Addressing overfitting with small datasets is crucial.

3. **Data Size's Significance:** Larger datasets improve model robustness, but require more computational resources.

4. **Working as a Team:** Efficient processing.

5. **Future Development:** Continue with a larger dataset.

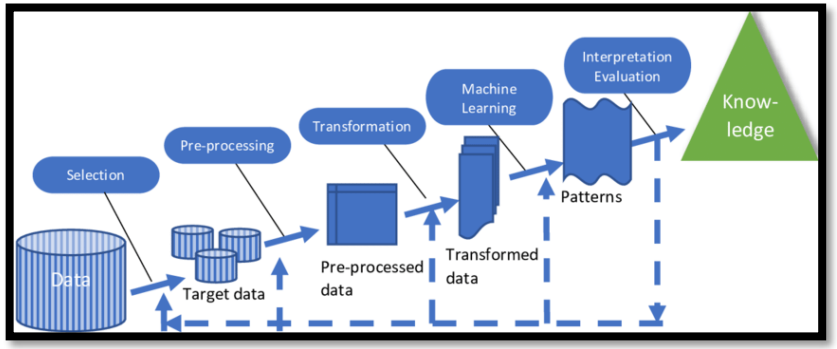



**Fig. 16:**ML Process

# Thank you

## Team 5:

- Fernando Sepulveda
- Derek Cisneros
- Marco Garcia Fernandez
- Francisco Parra

## Questions?