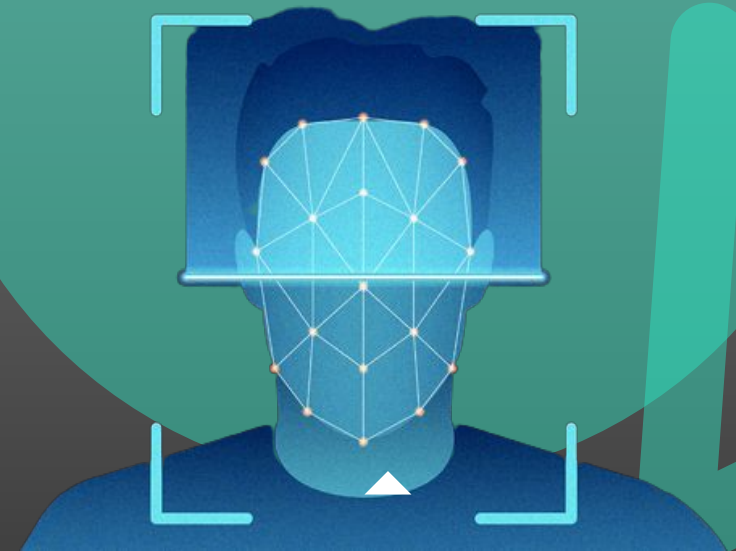
RECONOCIMIENTO FACIAL

Guillermo Andres Daza Meneses 2200143

Ivan Leonardo Niño Villamil 2191968



Contenido de la presentación:

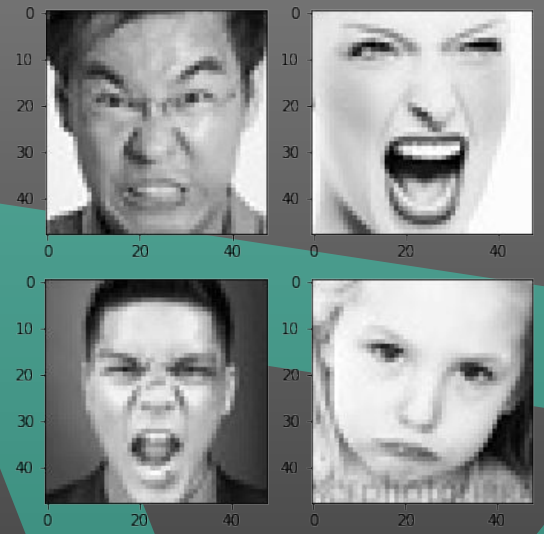
- Resumen del dataset con sus características.
- Estimadores utilizados con su implementación y sus respectivos resultados.
- Conclusiones.
- Observaciones.

Descripción del dataset:

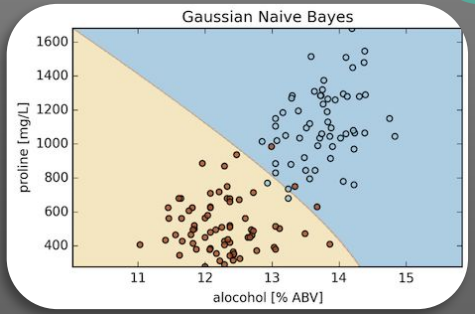
El dataset cuenta con 35,887 imágenes repartidas en 7 categorías:

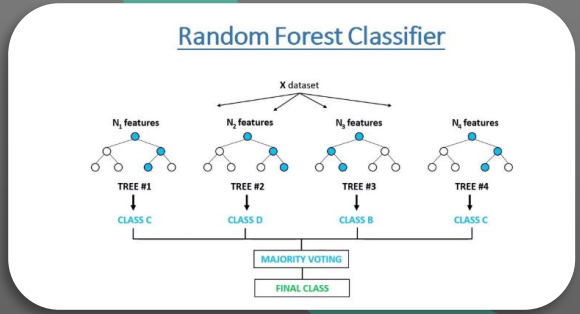
Train: 28.821
Angry: 3.993
Disgust: 436
Fear: 4.103
Happy: 7.164
Neutral: 4.982
Sad: 4.938
Surprise: 3.205

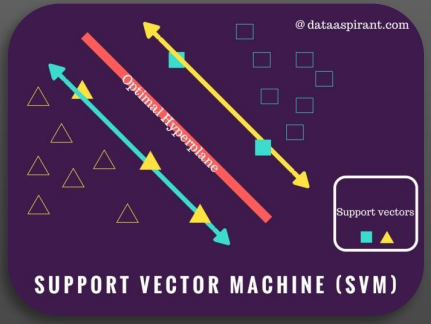
Validation: 7.066
Angry: 960
Disgust: 111
Fear: 1.018
Happy: 1.825
Neutral: 1.216
Sad: 1.139
Surprise: 797

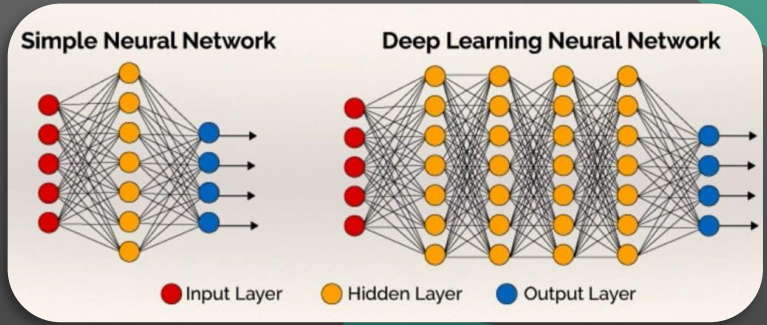


Estimadores utilizados









GaussianNB

GaussianNB

```
#@title GaussianNB
from sklearn.naive_bayes import GaussianNB
inicio = time.time()
estimador = GaussianNB()

estimador.fit(X,y)
y_pred = estimador.predict(X_test)
score = cross_val_score(estimador, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
fin = time.time()
print("accuracy score: %.3f (+/- %.5f)"%(np.mean(score), np.std(score)))
print(fin-inicio)
print(classification_report(y_test, y_pred))
```

```
accuracy score: 0.206 (+/- 0.01394)
24.073458671569824
              precision    recall  f1-score   support

           0       0.44      0.14      0.21       888
           1       0.16      0.04      0.07       512
           2       0.03      0.18      0.06        92
           3       0.25      0.39      0.30       609
           4       0.24      0.06      0.10       560
           5       0.26      0.17      0.21       587
           6       0.19      0.59      0.29       395

    accuracy                           0.21      3643
   macro avg       0.22      0.23      0.18      3643
weighted avg       0.27      0.21      0.19      3643
```

DecisionTreeClassifier

DecisionTreeClassifier

```
[15] #@title DecisionTreeClassifier
     from sklearn.tree import DecisionTreeClassifier
     inicio = time.time()
     estimador = DecisionTreeClassifier(max_depth=20)

     estimador.fit(X, y)
     y_pred = estimador.predict(X_test)
     score = cross_val_score(estimador, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
     fin = time.time()
     print("accuracy score: %.3f (+/- %.5f)"%(np.mean(score), np.std(score)))
     print(fin-inicio)
     print(classification_report(y_test, y_pred))
```

```
accuracy score: 0.282 (+/- 0.00969)
3313.314398765564
              precision    recall  f1-score   support

           0       0.40      0.38      0.39       912
           1       0.19      0.20      0.19       491
           2       0.28      0.35      0.31       117
           3       0.23      0.24      0.24       576
           4       0.22      0.20      0.21       523
           5       0.28      0.28      0.28       624
           6       0.40      0.41      0.40       400

    accuracy                           0.29      3643
   macro avg       0.29      0.29      0.29      3643
weighted avg       0.29      0.29      0.29      3643
```

RandomForestClassifier

RandomForestClassifier

```
[ ]  #@title RandomForestClassifier
     from sklearn.ensemble import RandomForestClassifier
     inicio = time.time()
     estimador = RandomForestClassifier()

     estimador.fit(X, y)
     y_pred = estimador.predict(X_test)
     score = cross_val_score(estimador, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))

     fin = time.time()
     print("accuracy score1 %.3f (+/- %.5f)"%(np.mean(score), np.std(score)))
     print(fin-inicio)
     print(classification_report(y_test, y_pred))
```

```
accuracy score1 0.420 (+/- 0.01290)
1471.074490070343
              precision    recall  f1-score   support

           0       0.43      0.76      0.55       888
           1       0.43      0.18      0.25       512
           2       1.00      0.30      0.47        92
           3       0.32      0.32      0.32       609
           4       0.50      0.22      0.30       560
           5       0.37      0.38      0.37       587
           6       0.59      0.54      0.56       395

    accuracy                           0.43      3643
   macro avg       0.52      0.39      0.40      3643
weighted avg       0.44      0.43      0.40      3643
```

Support Vector Classifier(SVC)

Implementación de SVC

```
[ ]  #@title Implementación de SVC
     from sklearn.svm import SVC
     inicio = time.time()
     kernels = ['linear', 'poly', 'rbf']
     for i in kernels:
       print(i)
       estimador = SVC(kernel=i)
       estimador.fit(X, y)
       predicciones = estimador.predict(X_test)
       score = cross_val_score(estimador, X, y, cv=KFold(10, shuffle=True), scoring=make_scorer(accuracy_score))
       fin = time.time()
       print("accuracy score %.3f (+/- %.5f)"%(np.mean(score), np.std(score)))
       print(classification_report(y_test, y_pred))
       print(""))
     print(fin-inicio)
```

```
linear
accuracy score 0.275 (+/- 0.01894)
poly
```

Implementación de Redes neuronales

```python
# Crear un objeto ImageDataGenerator para el conjunto de entrenamiento
train_datagen = ImageDataGenerator(rescale=1./255)

# Crear un objeto ImageDataGenerator para el conjunto de prueba
val_datagen = ImageDataGenerator(rescale=1./255)

# Cargar los conjuntos de entrenamiento y validación
train_generator = train_datagen.flow_from_directory(
    dir_train,
    target_size=(48, 48),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    dir_val,
    target_size=(48, 48),
    batch_size=32,
    class_mode='categorical'
)
```

```python
[ ] # Construir un modelo de red neuronal

    model = keras.Sequential([
        keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 3)),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(64, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(128, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.Flatten(),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.Dense(7, activation='softmax')
    ])
    model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 46, 46, 32)        896

 max_pooling2d_12 (MaxPoolin  (None, 23, 23, 32)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 21, 21, 64)        18496

 max_pooling2d_13 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_14 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_4 (Dropout)         (None, 4, 4, 128)         0

 flatten_4 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 512)               1049088

 dense_9 (Dense)             (None, 7)                 3591

=================================================================
Total params: 1,145,927
Trainable params: 1,145,927
Non-trainable params: 0
_________________________________________________________________
```

```python
# Compilar el modelo
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') > 0.98 ):
            print("\nAlcanzado el 98% de acuracy, se detiene el modelo!")
            self.model.stop_training = True

callbacks=myCallback()

# Entrenar el modelo
history = model.fit(
    train_generator,
    batch_size =32,
    steps_per_epoch=100,
    epochs=300,
    validation_data=val_generator,
    validation_steps=50,
    callbacks = callbacks,
    verbose=1,
    shuffle=True
)
```

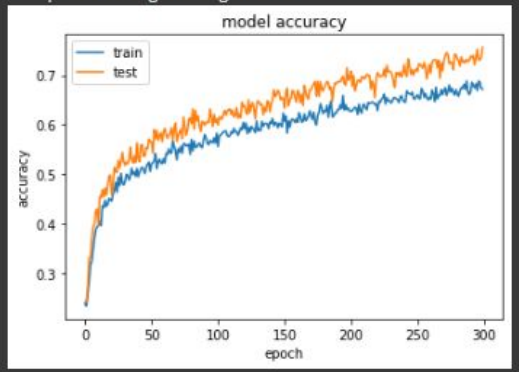
Resultados(Implementación de Redes neuronales)

```
Tiempo de ejecución:
5384.667600631714
```

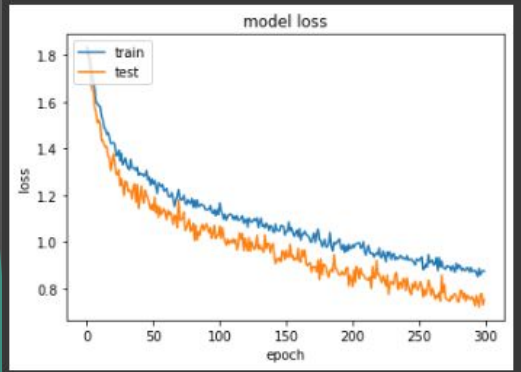
```
Epoch 300/300
100/100 [==============================] - 2s 23ms/step - loss: 0.1854 - accuracy: 0.9309 - val_loss: 1.7875 - val_accuracy: 0.5944
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7ff78c29d790>
```

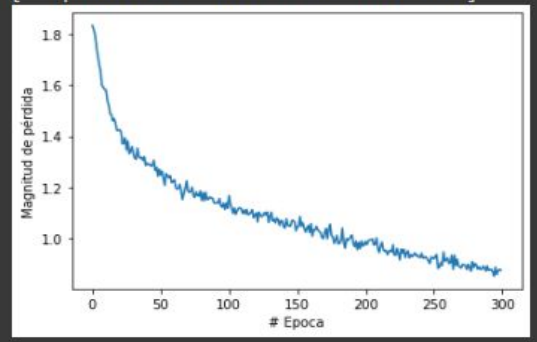



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
plt.xlabel("# Epoca")
plt.ylabel("Magnitud de pérdida")
plt.plot(history.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7ff78c1e3850>]
```



Conclusiones

- El estimador más rápido fue GaussianNB demorando apenas 24 segundos, pero teniendo un accuracy de alrededor del 21%
- El mejor estimador fue RandomForest Classifier con un accuracy del 43%, pero demorando alrededor de 24 minutos.
- El modelo de redes neuronales fue capaz de alcanzar un accuracy del 93% al ejecutarse con 300 épocas.
- Al realizar modificaciones en el objeto ImageDataGenerator como lo son rotar las imágenes, aplicar zoom, o invertir las imágenes con respecto al eje Y, el tiempo de ejecución aumenta de manera drástica, y el accuracy cae alrededor de 63 %.

Observaciones

- Debido a la gran cantidad de imágenes del dataset se tuvo que trabajar solamente con la mitad para el caso de los estimadores de GaussianNB, DecisionTreeClassifier, RandomForestClassifier y SVC, la única categoría de imágenes que no se redujo fue disgust puesto que esta es la que menos imágenes con apenas 547.
- De la misma manera no fue posible trabajar con SVC con ninguno de sus kernel debido a que este estimador demoraba mucho tiempo en ejecutarse, lo que ocasiona que el colab se reiniciará.