/proyecto 2.1

MONITORIZACIÓN DE LAS EMOCIONES

IES Ribera del Tajo
Enseñanza bilingüe

/Enunciado

Hemos conseguido un contrato con una empresa automovilística para realizar un modelo de detección de emociones para controlar el estado de ánimo de un conductor:

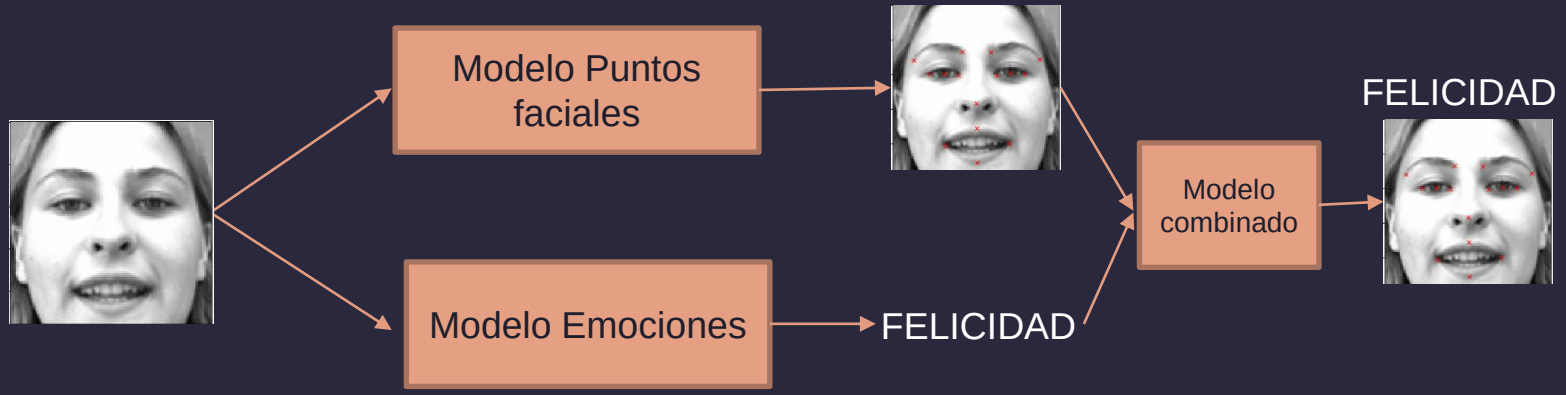
> Nos proporcionan unas 2000 imágenes con los puntos faciales claves marcado y unas 20000 imágenes con sus etiquetas de emoción y los siguientes datasets.

data.csv: Contiene las coordenadas de los puntos faciales clave. Cada fila tiene 31 columnas donde las 30 primeras columnas corresponden a las coordenadas de los puntos faciales clave para detector las emociones y la última columna corresponde a la imagen de entrenamiento. (El dataset data.csv contiene las coordenadas x,y de 15 puntos faciales clave y las imágenes de entrada son de 96x96 píxeles y un solo canal de color)

icml_face_data.csv: Contiene una imagen junto con su emoción {0:'Ira', 1:'Asco', 2:'Tristeza', 3:'Felicidad', 4: 'Sorpresa'}. En este caso las imágenes son de 48x48

/Enunciado

Combinaremos los dos modelos para obtener una predicción de los puntos faciales clave y su emoción:



Desplegaremos los modelos con tensorflow Serving una MV para dar servicio a través de API REST

/FASE 1 : DETECCIÓN DE PUNTOS FACIALES

/01 /VISUALIZACIÓN DE LOS DATOS

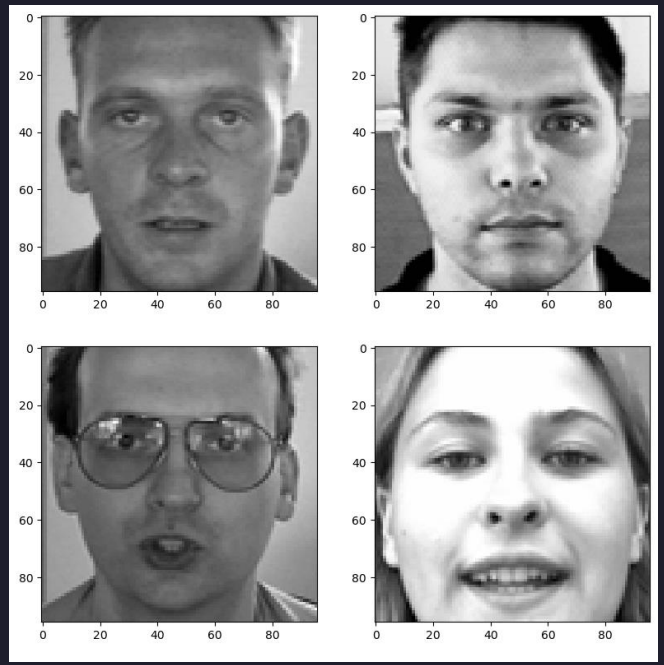
/02 /AUMENTACIÓN Y NORMALIZACIÓN

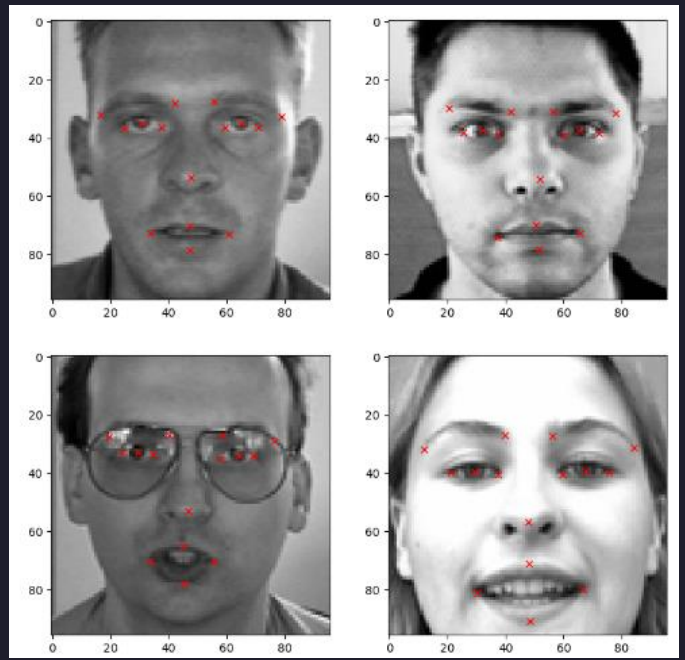
/03 /DEEP LEARNING: RESNET vs CNN

/04 /EVALUACIÓN

/01 /VISUALIZACIÓN DE LOS DATOS

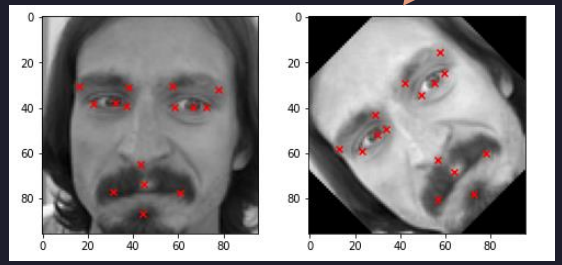
Entended y visualizad la información de los dataframes. Haced un “sanity check” para ver que el conjunto de datos está completo.

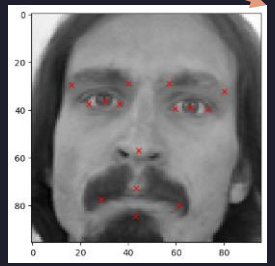


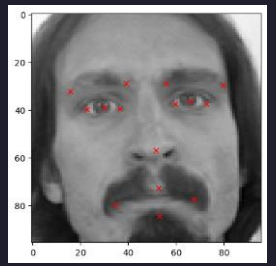


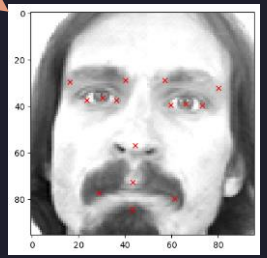
/02 /AUMENTACIÓN Y NORMALIZACIÓN

Como 2000 muestras no parecen ser demasiadas, podemos realizar procesos de aumentación de datos para, por ejemplo, rotar una imagen, girar los ejes (espejo), cambiarle el brillo, hacerlo borroso, etc.









Podéis pedir, al coordinador del proyecto código en Python para realizar estas aumentaciones de los datos.

Normalizad los datos para que cada valor de cada pixel esté en el rango 0..1

/03 /DEEP LEARNING: RESNET vs CNN

Aquí tenéis dos tareas importantes que realizar:

- Una de investigación: Nos han contado que este tipo de problemas se resuelve muy bien con una RESNET. Nos han proporcionado un código para crear este tipo de RNAs en TensorFlow

- Una de evaluación: Nosotros conocemos muy bien las redes neuronales convolucionales, pero no tanto las RESNET. Vamos a utilizar nuestro propio modelo de TensorFlow con una RNA convolucional y vamos a ver cuál de los dos modelos converge mejor

Pídele el código de ambos modelos al coordinador de tu proyecto y entrena el conjunto de datos con las dos RNAs diferentes. Elegid la que penséis que **puede dar mejores resultados**

RESNET:

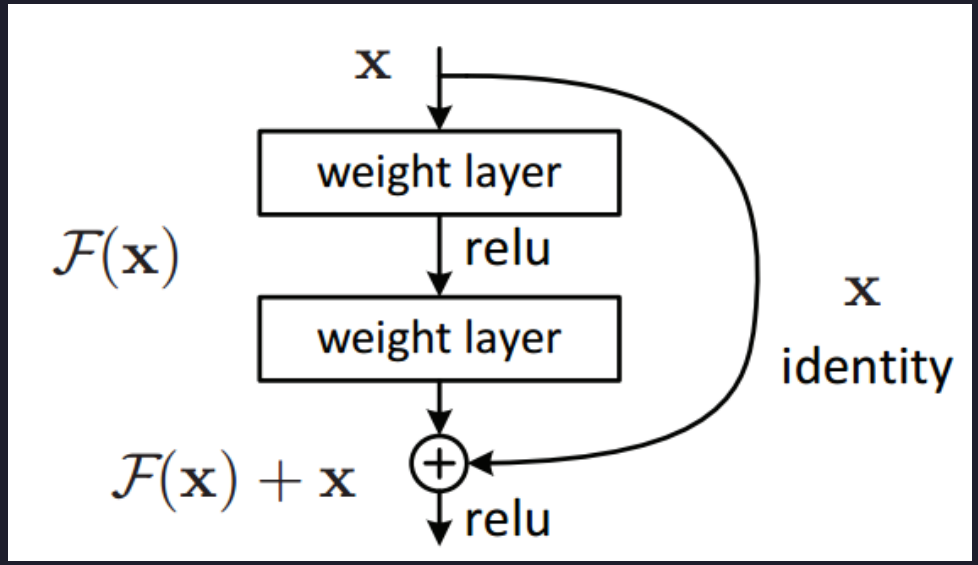
A medida que las RNAs convolucionales se hacen más grandes y más complejas surge un problema conocido como el “desvanecimiento del gradiente” o **vanishing gradient** que impacta negativamente en el rendimiento de la RNA.

El vanishing gradient sucede cuando el gradiente se propaga hacia atrás en redes muy extensas, puesto que al atravesar muchas capas, tiende a 0.

Las redes neuronales residuales incluye la función “omisión de conexión” que permite el entrenamiento de hasta 152 capas sin el problema del desvanecimiento del gradiente.

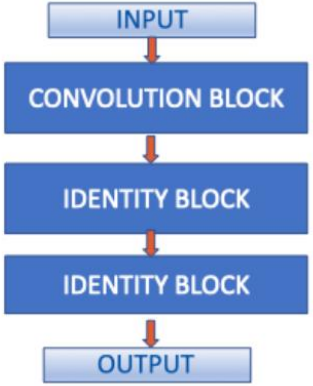
Resnet funciona agregando “asignaciones de identidad” en la parte superior de una CNN.

Por ejemplo, imagenet (image-net.org) ha entrenado 11 millones de imágenes agrupándolas en 11000 categorías con una RESNET



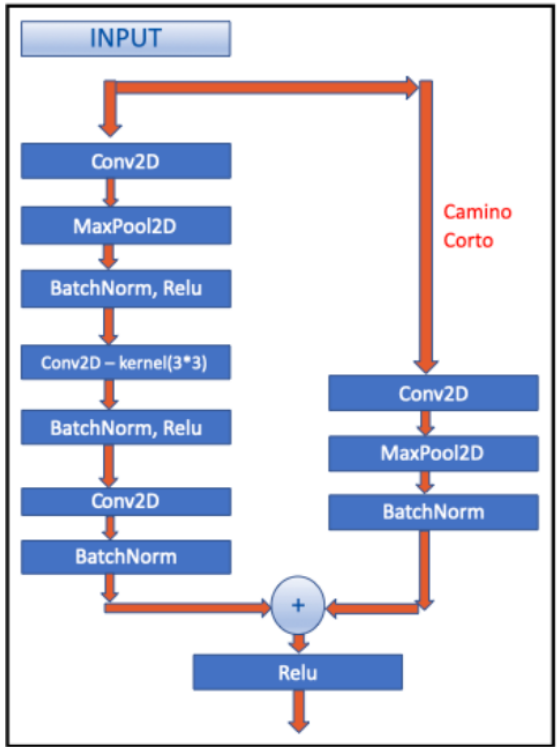
RESNET

RES-BLOCK

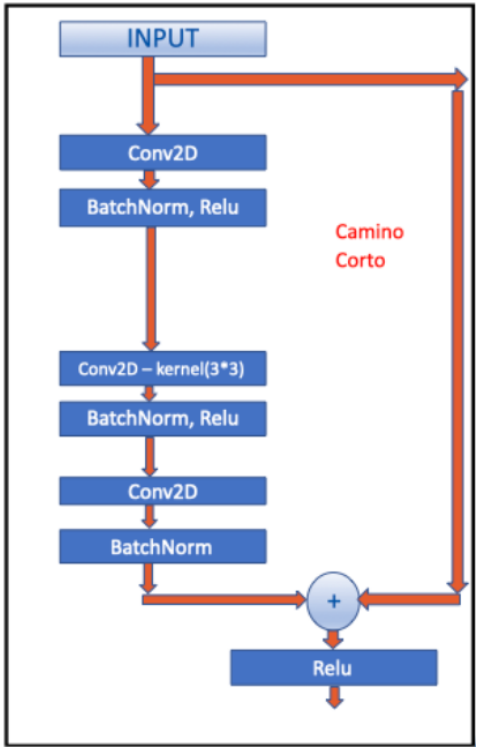


Tenemos el código que nos han dado en **resnet_puntos_faciales.py**

CONVOLUTION BLOCK

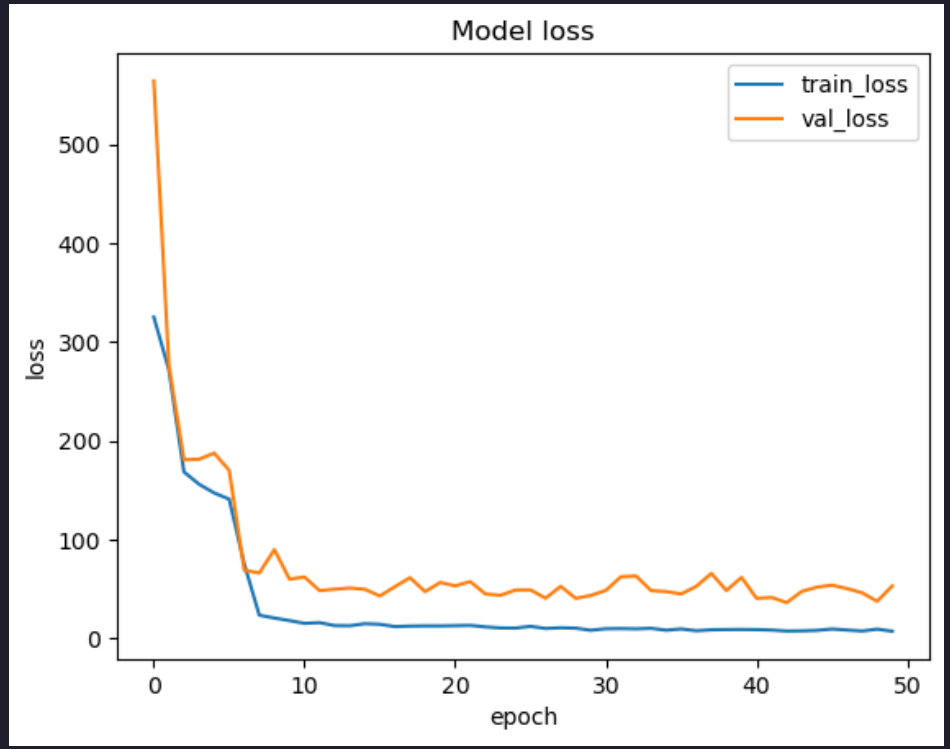


IDENTITY BLOCK



/04 /EVALUACIÓN

Evaluad el rendimiento de los modelos y comparadlo con una red CNN como las que estamos acostumbrados a utilizar:

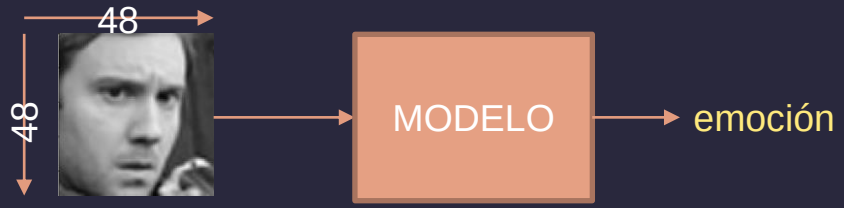


RED Convolucional

```
model = Sequential()
model.add(Convolution2D(32, (5, 5), input_shape=(96,96,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Convolution2D(30, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(30))
```

optimizer = 'adam', loss = 'mean_squared_error',
metrics = ['accuracy']

/FASE 2 : DETECCIÓN DE EMOCIONES



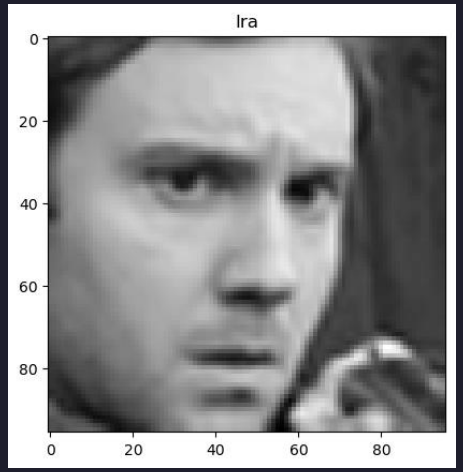
/01 /VISUALIZACIÓN DE LOS DATOS

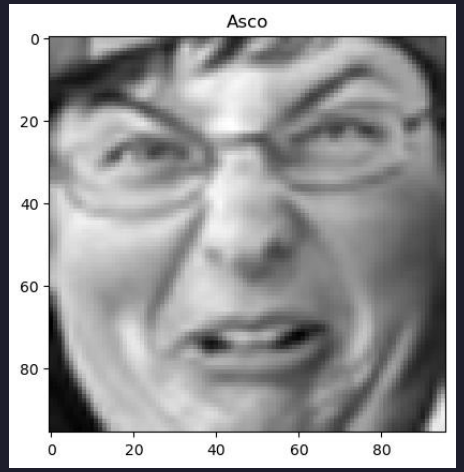
/02 /AUMENTACIÓN

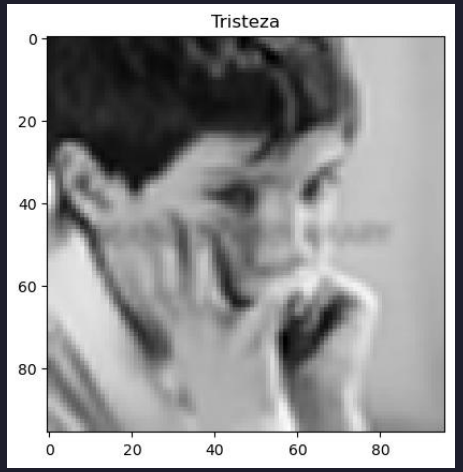
/03 /DEEPLEARNING + EVALUACIÓN

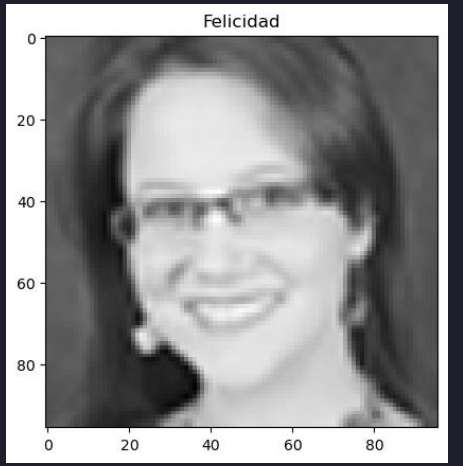
/04 /COMBINACIÓN DE MODELOS Y DESPLIEGUE

/01 /VISUALIZACIÓN DE LOS DATOS









{0:'Ira', 1:'Asco', 2:'Tristeza', 3:'Felicidad', 4: 'Sorpresa'}.

/02 /AUMENTACIÓN

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode='nearest',
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    interpolation_order=1,
    dtype=None
)
```

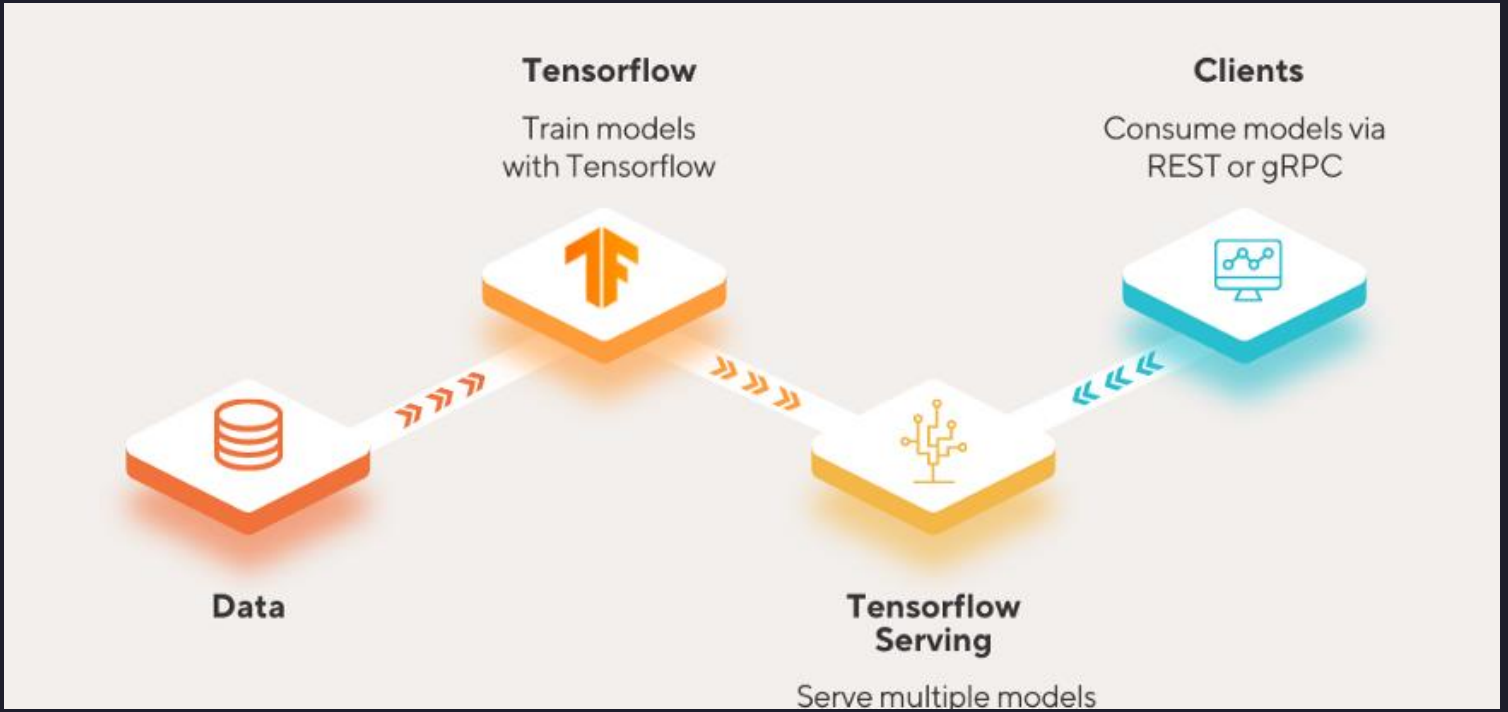
```
train_datagen = ImageDataGenerator(
    rotation_range = 15,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.1,
    zoom_range = 0.1,
    horizontal_flip = True,
    vertical_flip = True,
    brightness_range = [1.1, 1.5],
    fill_mode = "nearest")
```

/03 /DEEPLEARNING + EVALUACIÓN

Crear una resnet, compilar y entrenar el modelo. Probarlo y evaluarlo con una matriz de confusión:

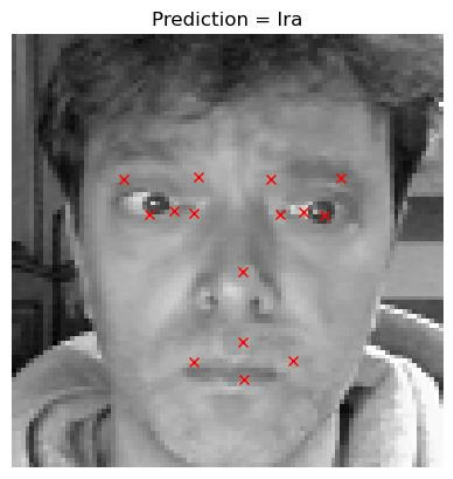
Tenemos el código que nos han dado en **resnet_emociones.py**

/04 /COMBINACIÓN DE MODELOS Y DESPLIEGUE



/04 /COMBINACIÓN DE MODELOS Y DESPLIEGUE

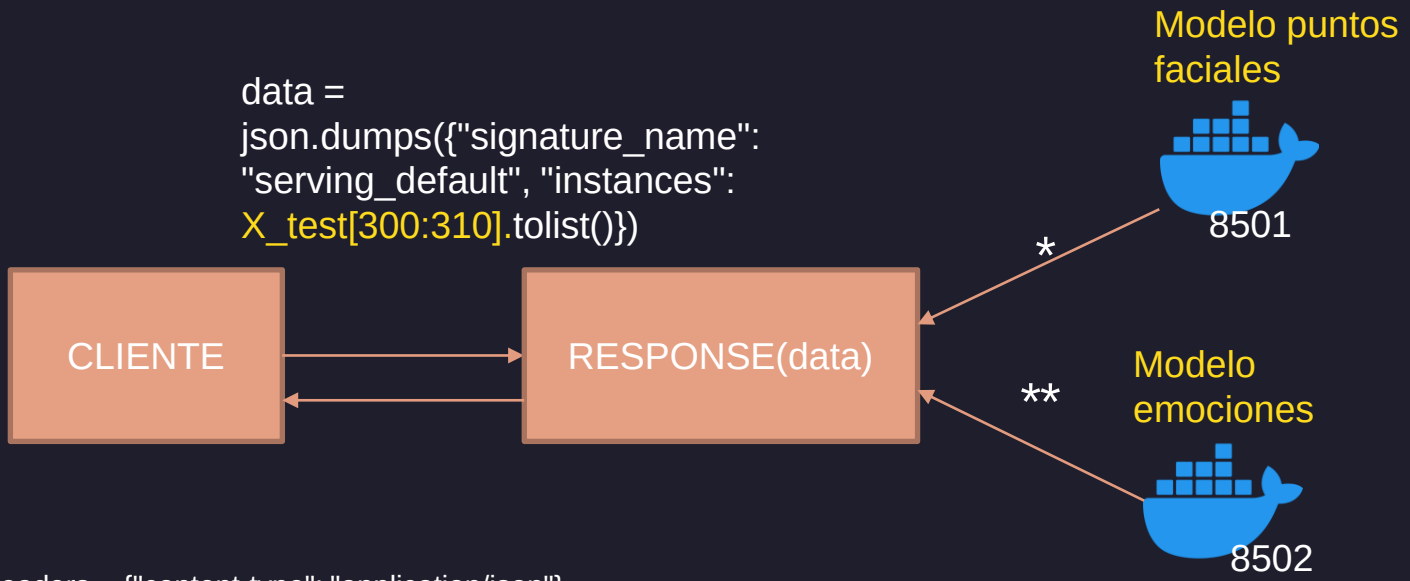
Cuando nuestros modelos estén entrenados y dando buenos resultados habrá que crear una función en Python predict() que devuelva en un dataframe tanto los puntos faciales como la etiqueta de emoción detectada.



Tenemos que implementar este modelo combinado en una página web y para ello usaremos TensorFlow Serving desde docker

```python
def predict(X_test):

    # Hacemos la predicción con el modelo de puntos clave
    df_predict = model_1_facialKeyPoints.predict(X_test)

    # Hacemos la predicción con el modelo de emociones
    df_emotion = np.argmax(model_2_emotion.predict(X_test), axis=-1)

    # Redimensionamos el array de (856,) a (856,1)
    df_emotion = np.expand_dims(df_emotion, axis = 1)

    # Convertimos las predicciones en un dataframe
    df_predict = pd.DataFrame(df_predict, columns= columns)

    # Añadimos la emoción al dataframe de predicciones
    df_predict['emotion'] = df_emotion

    return df_predict
```

/04 /COMBINACIÓN DE MODELOS Y DESPLIEGUE



*
```
headers = {"content-type": "application/json"}
json_response = requests.post('http://localhost:8501/v1/models/facialKeyPoints/versions/1:predict',
        data=data, headers=headers, verify = False)
df_predict = json.loads(json_response.text)['predictions']
```

Código de ayuda en despliegue.py