# Instituto Tecnológico y de Estudios Superiores de Monterrey

**Campus Monterrey**

**Inteligencia artificial avanzada para la ciencia de datos II**
**TC3007C.501**

# Implementación de un modelo de Deep Learning.

Rodolfo Sandoval Schipper A01720253

1 nov 2023

Se ha desarrollado un sistema que involucra llevar un control efectivo de la participación y asistencia de los estudiantes en un entorno de clases virtuales, se emplean modelos de inteligencia artificial como el reconocimiento facial y la detección de poses. La integración de estas tecnologías permite construir un sistema automatizado de monitoreo y análisis. Sin embargo, nuestra implementación implica la utilización de dos modelos. Un framework de Face recognition con OpenCV y Pose Detection con Yolo. Estos modelos se aplican de la siguiente manera.

- **Pose Detection:** Esta etapa detecta la postura de una persona. Una vez que se genera una nueva pose, se activa el modelo de reconocimiento facial para identificar al estudiante y registrar su asistencia. Además, se genera un cuadro delimitador (bounding box) utilizando la pose detectada para limitar el área de interés en la cual se ejecutará el modelo de reconocimiento facial. Esto es una estrategia eficiente para reducir el área de análisis y mejorar el rendimiento del modelo de reconocimiento facial.

- **Atributos para el Label:** Se han capturado los atributos necesarios para asignar un etiquetado (label) a la imagen incrustada. Esto es crucial para asociar la identidad del estudiante con la imagen reconocida por el modelo de reconocimiento facial.

- **Enfoque en Ejecutar Modelos:** La lógica del sistema se centra en ejecutar los modelos pertinentes basados en las acciones consideradas por el modelo de detección de poses. Esto optimiza el proceso, ya que los modelos se activan específicamente en respuesta a ciertas acciones o poses detectadas.

- **Propuesta de Cambio:** La propuesta para esta entrega implica la implementación de una arquitectura de clasificación de imágenes mediante deep learning. Esto podría significar un cambio en la estrategia actual, donde en lugar de correr el modelo de poses para detectar un alumno y activar el modelo de reconocimiento facial, el sistema podría clasificar y reconocer una persona específica dentro del entorno para así tomar la asistencia de dicho alumno.

A continuación se estará utilizando el dataset de https://www.kaggle.com/datasets/hereisburak/pins-face-recognition para entrenar, probar y validar la arquitectura de un modelo pre-entrenado transfer learning VGG 16 con el uso de Keras, hacer fine tuning, agregar capas, e implementar una solución más óptima para el proyecto a entregar.

Link del Google Colab: https://colab.research.google.com/drive/1kJurW5-7179pK3qyCj7Y4B51TmKqPa__?usp=sharing

# Entrega retroalimentación individual: implementación de un modelo de deep learning

Liga del dataset: https://www.kaggle.com/datasets/hereisburak/pins-face-recognition

*Utilizamos 25 actores/clases debido a que la cantidad de datos en el dataset*

*completo es bastante grande para subir en la nube*

Documentación Keras: https://keras.io/api/applications/

Objetivo: Probar y validar la arquitectura de un modelo pre-entrenado transfer learning VGG 16 con el uso de Keras, hacer fine tuning e implementar una solución más óptima para el proyecto a entregar.

```
# Importamos el dataset desde la nube
from google.colab import drive
drive.mount('/content/drive')

# Path
source_dir = "/content/drive/MyDrive/DataSetTransferLearning/105_classes_pins_dataset"
```

```
Mounted at /content/drive
```

```
# Hacemos un split del dataset en subconjuntos de train y test y los
asignamos a un nuevo folder import os
import shutil
from sklearn.model_selection import train_test_split

dest_dir = "/content/drive/MyDrive/DataSetTransferLearning/split_data_set"

os.makedirs(dest_dir, exist_ok=True)
os.makedirs(os.path.join(dest_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(dest_dir, 'test'), exist_ok=True)

actors = os.listdir(source_dir)

# Split de los datos de cada actor a sets de train y test
for actor in actors:
    actor_dir = os.path.join(source_dir, actor)
    images = os.listdir(actor_dir)

    # Split de las imagenes a train y test
    train_images, test_images = train_test_split(images, test_size=0.2, random_state=42)

    # Crear directorio de cada actor en train y test
    os.makedirs(os.path.join(dest_dir, 'train', actor), exist_ok=True)
    os.makedirs(os.path.join(dest_dir, 'test', actor), exist_ok=True)

    # Mover las imagenenes a su directorio
    for image in train_images:
        shutil.copy(os.path.join(actor_dir, image), os.path.join(dest_dir, 'train', actor, image))

    for image in test_images:
        shutil.copy(os.path.join(actor_dir, image), os.path.join(dest_dir, 'test', actor, image))
```

```
from keras.applications import VGG16
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout, Conv2D, Activation, MaxPooling2D
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping,ModelCheckpoint


# Ruta de los datos ya divididos en train y test
train_data_dir = '/content/drive/MyDrive/DataSetTransferLearning/split_data_set/train'
test_data_dir = '/content/drive/MyDrive/DataSetTransferLearning/split_data_set/test'

# Parametros
img_width, img_height = 224, 224
batch_size = 32
num_classes = 25 # Cantidad de actores y clasificaciones
epochs = 50

# Cargamos el modelo pre-entrenado VGG16 sin incluir las capas densas
vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))



# No corremos las capas
for layer in vgg_model.layers:
    layer.trainable = False

model = Sequential()

# Añadimos el modelo VGG16 pre-entrenado
model.add(vgg_model)

# Capas personalizadas
model = Sequential()
model.add(vgg_model)

# Capas personalizadas
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))# Capa de salida, numero de clases

# Compilamos el modelo
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Data augmentation y preparacion de los datos
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_data_dir,
                            target_size=(img_width, img_height),
                                                batch_size=batch_size,
                                                class_mode='categorical')

test_generator = test_datagen.flow_from_directory(test_data_dir,
                                      target_size=(img_width, img_height),
                                                batch_size=batch_size,
                                                class_mode='categorical')
```

```
# Entrenamiento del modelo
model.fit(train_generator,
          steps_per_epoch=train_generator.samples // batch_size,
          epochs=epochs,
          validation_data=test_generator,
          validation_steps=test_generator.samples // batch_size)

# Guardar el modelo entrenado
model.save('/content/drive/MyDrive/DataSetTransferLearning/actor_recognition_model.h5')
```

```
Found 3118 images belonging to 25 classes.
Found 767 images belonging to 25 classes.
Epoch 1/50
97/97 [==============================] - 60s 585ms/step - loss: 3.1742 - accuracy: 0.0632 - val_loss:
2.9590 - val_accuracy: 0.1698 Epoch 2/50
97/97 [==============================] - 54s 557ms/step - loss: 2.9347 - accuracy: 0.1131 - val_loss:
2.6458 - val_accuracy: 0.2065 Epoch 3/50
97/97 [==============================] - 55s 568ms/step - loss: 2.7118 - accuracy: 0.1692 - val_loss:
2.3920 - val_accuracy: 0.3016 Epoch 4/50
97/97 [==============================] - 53s 550ms/step - loss: 2.5340 - accuracy: 0.2233 - val_loss:
2.2156 - val_accuracy: 0.3546 Epoch 5/50
97/97 [==============================] - 53s 543ms/step - loss: 2.3771 - accuracy: 0.2576 - val_loss:
2.0211 - val_accuracy: 0.4022 Epoch 6/50
97/97 [==============================] - 56s 575ms/step - loss: 2.2653 - accuracy: 0.2955 - val_loss:
1.9459 - val_accuracy: 0.4470 Epoch 7/50
97/97 [==============================] - 51s 526ms/step - loss: 2.1508 - accuracy: 0.3325 - val_loss:
1.8167 - val_accuracy: 0.4715 Epoch 8/50
97/97 [==============================] - 55s 565ms/step - loss: 2.0520 - accuracy: 0.3526 - val_loss:
1.6919 - val_accuracy: 0.4891 Epoch 9/50
97/97 [==============================] - 54s 552ms/step - loss: 1.9664 - accuracy: 0.3869 - val_loss:
1.6189 - val_accuracy: 0.5149 Epoch 10/50
97/97 [==============================] - 55s 567ms/step - loss: 1.9087 - accuracy: 0.3957 - val_loss:
1.5793 - val_accuracy: 0.5353 Epoch 11/50
97/97 [==============================] - 51s 527ms/step - loss: 1.8032 - accuracy: 0.4355 - val_loss:
1.5480 - val_accuracy: 0.5312 Epoch 12/50
97/97 [==============================] - 53s 543ms/step - loss: 1.7625 - accuracy: 0.4595 - val_loss:
1.4695 - val_accuracy: 0.5666 Epoch 13/50
97/97 [==============================] - 55s 560ms/step - loss: 1.7148 - accuracy: 0.4598 - val_loss:
1.3953 - val_accuracy: 0.5761 Epoch 14/50
97/97 [==============================] - 53s 542ms/step - loss: 1.6712 - accuracy: 0.4825 - val_loss:
1.3919 - val_accuracy: 0.5802 Epoch 15/50
97/97 [==============================] - 51s 521ms/step - loss: 1.6170 - accuracy: 0.5006 - val_loss:
1.3079 - val_accuracy: 0.6060 Epoch 16/50
97/97 [==============================] - 53s 545ms/step - loss: 1.5513 - accuracy: 0.5123 - val_loss:
1.3027 - val_accuracy: 0.6005 Epoch 17/50
97/97 [==============================] - 51s 524ms/step - loss: 1.5166 - accuracy: 0.5243 - val_loss:
1.2643 - val_accuracy: 0.6155 Epoch 18/50
97/97 [==============================] - 53s 543ms/step - loss: 1.4792 - accuracy: 0.5266 - val_loss:
1.2930 - val_accuracy: 0.5842 Epoch 19/50
97/97 [==============================] - 52s 531ms/step - loss: 1.4485 - accuracy: 0.5327 - val_loss:
1.2302 - val_accuracy: 0.6304 Epoch 20/50
97/97 [==============================] - 53s 544ms/step - loss: 1.4310 - accuracy: 0.5405 - val_loss:
1.2309 - val_accuracy: 0.6182 Epoch 21/50
97/97 [==============================] - 53s 543ms/step - loss: 1.3918 - accuracy: 0.5697 - val_loss:
1.1633 - val_accuracy: 0.6345 Epoch 22/50
97/97 [==============================] - 54s 558ms/step - loss: 1.3489 - accuracy: 0.5752 - val_loss:
1.1379 - val_accuracy: 0.6562 Epoch 23/50
97/97 [==============================] - 51s 521ms/step - loss: 1.3304 - accuracy: 0.5843 - val_loss:
1.1453 - val_accuracy: 0.6562 Epoch 24/50
97/97 [==============================] - 52s 534ms/step - loss: 1.3076 - accuracy: 0.5807 - val_loss:
1.1468 - val_accuracy: 0.6440 Epoch 25/50
97/97 [==============================] - 52s 528ms/step - loss: 1.2817 - accuracy: 0.5949 - val_loss:
1.0696 - val_accuracy: 0.6848 Epoch 26/50
97/97 [==============================] - 52s 534ms/step - loss: 1.2147 - accuracy: 0.6209 - val_loss:
1.0147 - val_accuracy: 0.6902 Epoch 27/50
97/97 [==============================] - 53s 548ms/step - loss: 1.1894 - accuracy: 0.6218 - val_loss:
1.0410 - val_accuracy: 0.6902 Epoch 28/50
```

```
import os
# Creamos diccionario para los actores
data_dir = '/content/drive/MyDrive/DataSetTransferLearning/split_data_set/train' # Directorio de datos de
```

entrenamiento

```
actor_names = {}
class_index = 0

# Obtenemos los nombres de los actores con cada folder
actors = os.listdir(data_dir)

# Creamos el diccionario con los nombres de los actores
for actor in actors:
    actor_names[class_index] = actor
    class_index += 1

print("Diccionario de nombres de actores:", actor_names)
```

```
Diccionario de nombres de actores: {0: 'pins_Adriana Lima', 1: 'pins_Amanda Crew', 2: 'pins_Alex Lawther', 3:
                                                                                    'pins_alycia dabnem carey
```

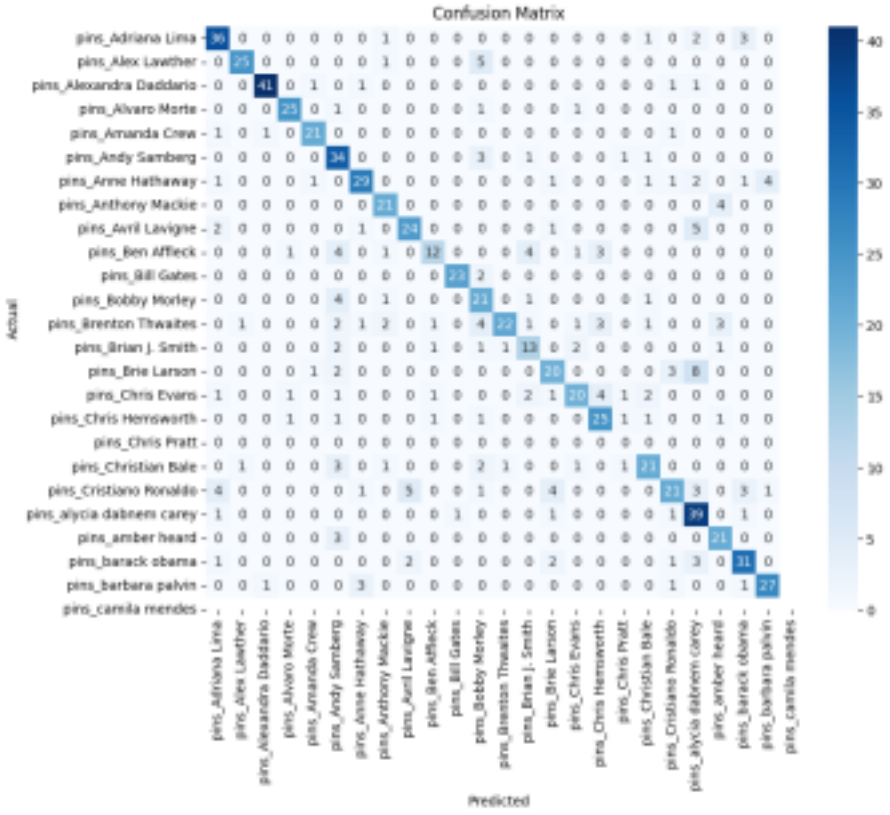
```
from keras.models import load_model
from keras.preprocessing import image
import numpy as np

# Cargamos el modelo entrenado
saved_model_path = '/content/drive/MyDrive/DataSetTransferLearning/actor_recognition_model.h5'
model = load_model(saved_model_path)

# Ruta de la imagen a predecir
image_path = '/content/drive/MyDrive/DataSetTransferLearning/split_data_set/test/pins_Anthony Mackie/Anthony
Mackie132_459.jpg'

# Cargamos la imagen y la preprocesamos para hacer la prediccion
img = image.load_img(image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = img_array / 255.

# Realizamos la prediccion
predictions = model.predict(img_array)

# Decodificamos las predicciones

predicted_class = np.argmax(predictions)
predicted_actor = actor_names[predicted_class]

print("Se predice:", predicted_actor)
```

```
1/1 [==============================] - 0s 322ms/step
Se predice: pins_Anthony Mackie
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Data augmentation y preparacion de los datos de prueba
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(test_data_dir,
                                    target_size=(img_width, img_height),
                                               batch_size=batch_size,
                                               class_mode='categorical',
                                                          shuffle=False)

# Obtenemos las predicciones del modelo en el conjunto de datos de prueba
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
```

```
# Obtenemos los labels reales
true_classes = test_generator.classes

# Calcular el accuracy y el loss
accuracy = accuracy_score(true_classes, predicted_classes)
loss = model.evaluate(test_generator)[0] # Perdida en el conjunto de prueba

print("Accuracy:", accuracy)
print("Loss:", loss)

# Creamos y mostramos la matriz de confusion
confusion = confusion_matrix(true_classes, predicted_classes)

# Obtenemos los nombres de las clases
class_names = list(test_generator.class_indices.keys())

# Mostramos la matriz de confusion
plt.figure(figsize=(10, 8))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Found 767 images belonging to 25 classes.
24/24 [==============================] - 8s 311ms/step
24/24 [==============================] - 3s 120ms/step - loss: 0.8088 - accuracy: 0.745
Accuracy: 0.7457627118644068
Loss: 0.8088144063949585
```



**Conclusiones:**

Hemos realizado varias acciones para desarrollar un modelo de reconocimiento de actores utilizando transfer learning con VGG 16, podemos utilizar este modelo para clasificar los estudiantes que pertenecen un salón dando a fin un resultado en el proyecto que se está elaborando con el socio formador. Se observó que el modelo es pesado y toma bastante procesamiento para entrenar. En este caso se hizo el entrenamiento con 50 epochs y los steps siendo la cantidad de clases entre batch size. Esto resultó tener una alta demanda computacional. Con de acuerdo al proceso, dividimos el conjunto de datos en train y test, y realizamos ajustes para la alimentación de datos en el modelo. Implementamos VGG 16 cómo base pre-entrenada y agregamos capas personalizadas para que se pueda adaptar al reconocimiento de actores. Ajustamos y entrenamos el modelo con los datos provistos y obtuvimos una precisión de 74.58% en el set de validación. Se pudo haber optimizado el modelo a través de la experimentación con diferentes arquitecturas y técnicas de regularización. Esto se puede considerar para otro entrenamiento a futuro. También hicimos unos casos prueba donde ingresamos una imagen de test y en la mayoría de los casos nos predice el resultado correcto. También logramos desplegar una matriz de confusión donde podemos observar que la cantidad de falsos positivos es baja a comparación de los resultados correctos.