TensorFlow, Keras with ResNet50: People Age-Image Regressor (Part 1)

In this project, a model is trained to predict the age of people in pictures. The pretrained model 'ResNet50' is used. This document is the first part of the whole training process.

The dataset can be found in:

Other useful datasets:

https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset

Iteration 1: Model creation and training (learning_rate=1e-3) without data augmentation (no fine-tuning yet)

```
# (height, width, channels)
input_shape = (224, 224, 3)
batch_size = 8
learning_rate = 1e-3
neurons = 128
path_dataset = '../dataset'
folder_models = '../models'

import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint
```

```
2025-09-30 23:20:52.540240: I tensorflow/tsl/cuda/cudart_stub.cc:28]
Could not find cuda drivers on your machine, GPU will not be used.
2025-09-30 23:20:58.139651: I tensorflow/tsl/cuda/cudart_stub.cc:28]
Could not find cuda drivers on your machine, GPU will not be used.
2025-09-30 23:20:58.147960: I
tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2025-09-30 23:21:07.170650: W
```

```
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning:
Could not find TensorRT
```

```
# Find how many images exist
imgs = os.listdir(path_dataset)
num_imgs = len(imgs)
print(f'Total images found: {num_imgs}')
```

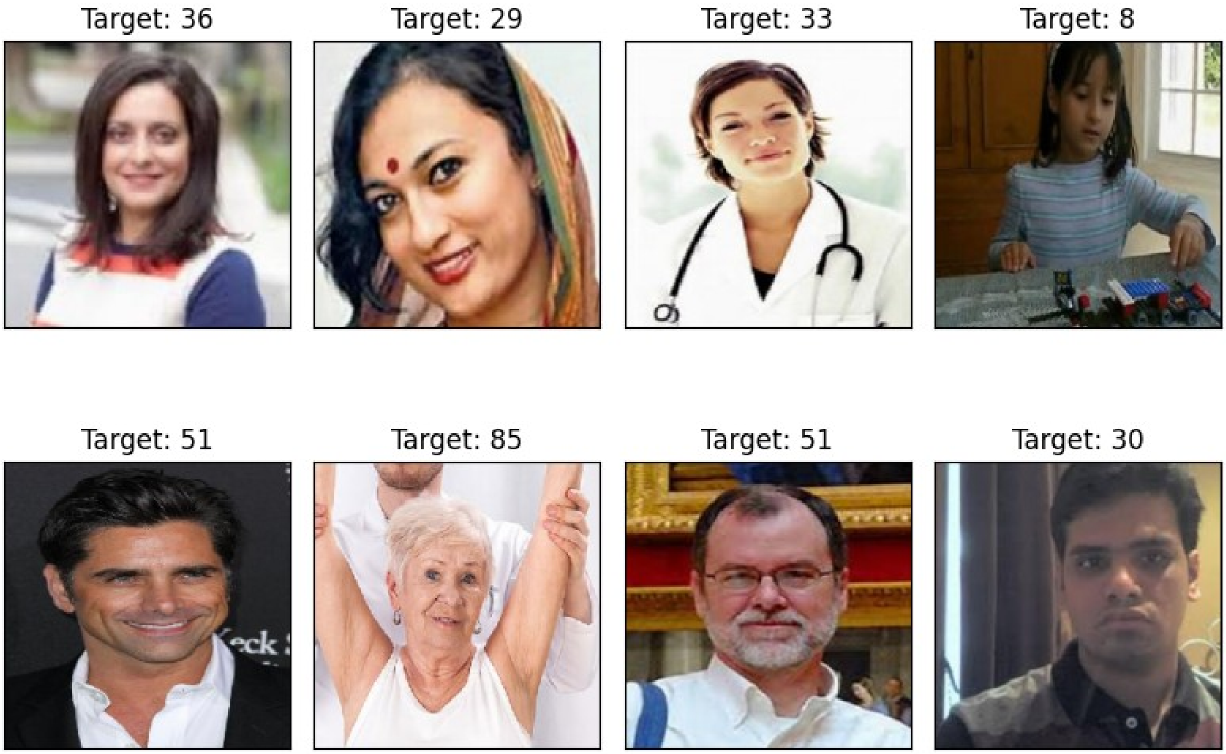
```
Total images found: 13389
```

No Data augmentation

```
def extract_age_from_filename(filename):
    """Extrae la edad desde el primer número del nombre del
archivo."""
    return float(filename.split('_')[0])

def create_dataframe_from_directory(directory):
    """Crea un DataFrame con nombres de archivo y edades como
etiquetas."""
    filenames = [f for f in os.listdir(directory) if
f.endswith('.jpg')]
    ages = [extract_age_from_filename(f) for f in filenames]
    df = pd.DataFrame({'filename': filenames, 'age': ages})
    return df

def load_regression_data(path, input_shape=input_shape,
batch_size=batch_size, seed=123, validation_split=0.2):
    """Crea generadores de imágenes para regresión lineal."""
    height, width = input_shape[:2]
    df = create_dataframe_from_directory(path)

    # Separar en entrenamiento y validación
    train_df = df.sample(frac=1 - validation_split, random_state=seed)
    val_df = df.drop(train_df.index)

    datagen = ImageDataGenerator(rescale=1.0/255, zoom_range=0,
        horizontal_flip=False, vertical_flip=False,
        height_shift_range=0, width_shift_range=0,
        brightness_range=(0.99, 1.0), rotation_range=0)

    train_data = datagen.flow_from_dataframe(
        dataframe=train_df,
        directory=path,
        x_col='filename',
        y_col='age',
        target_size=(height, width),
        batch_size=batch_size,
        class_mode='raw',  # For regression
        seed=seed
    )

    datagen_val = ImageDataGenerator(rescale=1.0/255)
    val_data = datagen_val.flow_from_dataframe(
        dataframe=val_df,
        directory=path,
        x_col='filename',
        y_col='age',
        target_size=(height, width),
        batch_size=batch_size,
        class_mode='raw',
        seed=seed
    )

    return train_data, val_data

# Split training and validation datasets
train, val = load_regression_data(path_dataset)

print(f"Training images: {train.samples}")
print(f"Validation images: {val.samples}")
```

```
Found 10710 validated image filenames.
Found 2677 validated image filenames.
Training images: 10710
Validation images: 2677
```

```
# Obtain images and target
images, labels = next(train)

# Show 8 training images (batch_size=8)
figure, axes = plt.subplots(nrows=2,ncols=4, figsize=(8, 6))
for item in zip(axes.ravel(), images, labels):
    axes, image, target = item
    axes.imshow(image)
    axes.set_title(f'Target: {target:.0f}')
    axes.set_xticks([])
    axes.set_yticks([])
plt.tight_layout()
plt.show()

# Image dimensions
print(images.shape)
```



```
(8, 224, 224, 3)
```

Model training

```
def create_resnet_model(input_shape=input_shape, neurons=neurons,
                        learning_rate=learning_rate):
    """Function to create the model using the pretrained model
    'ResNet50' and adding some final layers. The backbone is
'ResNet50',
    but it is freezed (not trained) in this iteration."""

    backbone = ResNet50(weights='imagenet', input_shape=input_shape,
                        include_top=False)

    # Freeze ResNet50 without the top
    backbone.trainable = False
    model = Sequential()
    model.add(backbone)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(neurons, activation='relu'))
    model.add(Dense(1, activation='relu'))  # For regression, Single
output
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='mse', metrics=['mae'])
    return model
```

```
def train_model(model, train_data, val_data, epochs, version_model):
    """Function to train the model and save the best one
    according to the min MAE."""
    file_name =
os.path.join(folder_models,f'regression_model_v{version_model}.h5')

    callbacks = [
        EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True, verbose=0),
        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=1e-6, verbose=0),
        ModelCheckpoint(file_name, monitor='val_mae',
save_best_only=True, mode='min', verbose=1)
    ]

    history = model.fit(train_data, validation_data=val_data,
              epochs=epochs, callbacks=callbacks, verbose=2)

    return model, history

epochs = 10
version_model = 1
print(f"Parameters: batch_size = {batch_size}, learning_rate =
{learning_rate}, neurons = {neurons}, epochs = {epochs}")
```

```
Parameters: batch_size = 8, learning_rate = 0.001, neurons = 128,
epochs = 10
```

```
# Create and train the model v1
model = create_resnet_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712  
                                                                 
 global_average_pooling2d (  (None, 2048)              0         
 GlobalAveragePooling2D)                                         
                                                                 
 dense (Dense)               (None, 128)               262272    
                                                                 
 dense_1 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 23850113 (90.98 MB)
Trainable params: 262401 (1.00 MB)
```

```
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

```python
print(f"TensorFlow Version: {tf.__version__}")

# Ensure GPU is available
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0],True)
    print("GPU is available and memory growth is enabled.")
else:
    print("GPU not available, training will be on CPU.")
```

```
TensorFlow Version: 2.13.1
GPU not available, training will be on CPU.
```

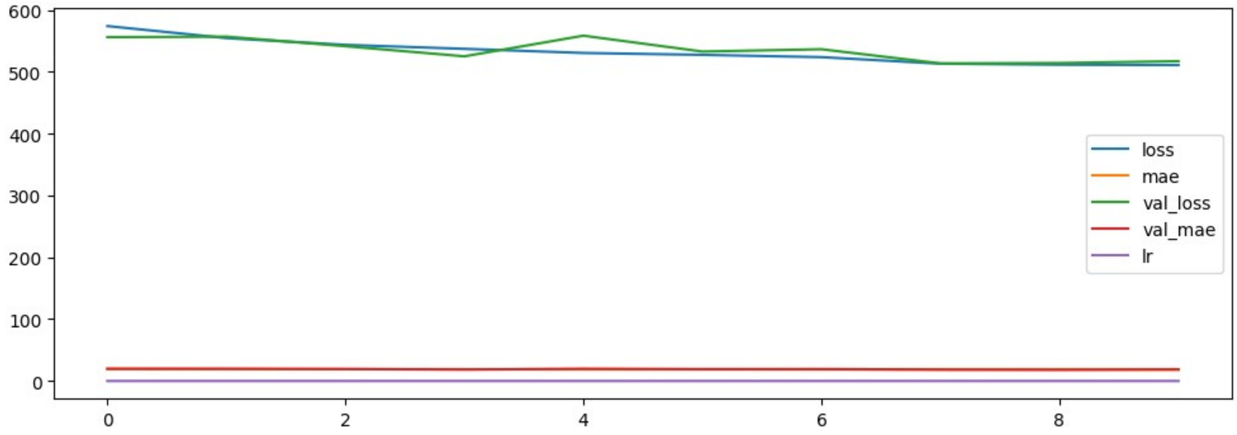
```python
# Train the model
model, history_stage1 = train_model(model, train, val, epochs=epochs,
version_model=version_model)
```

```
Epoch 1/10

2025-09-30 23:22:59.933200: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-30 23:23:00.844435: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-30 23:23:00.876807: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
26615808 exceeds 10% of free system memory.
2025-09-30 23:23:00.909189: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-30 23:23:00.924452: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.


Epoch 1: val_mae improved from inf to 19.70578, saving model to
../models/regression_model_v1.h5

/home/ant/TensorFlow-Keras-ResNet50-HuggingFace/env/lib/python3.8/
site-packages/keras/src/engine/training.py:3000: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(

1339/1339 - 981s - loss: 573.8323 - mae: 19.7482 - val_loss: 555.9131
- val_mae: 19.7058 - lr: 0.0010 - 981s/epoch - 733ms/step
```

```
Epoch 2/10

Epoch 2: val_mae did not improve from 19.70578
1339/1339 - 901s - loss: 554.4229 - mae: 19.3387 - val_loss: 556.7506
- val_mae: 19.8103 - lr: 0.0010 - 901s/epoch - 673ms/step
Epoch 3/10

Epoch 3: val_mae improved from 19.70578 to 19.37602, saving model
to ../models/regression_model_v1.h5
1339/1339 - 894s - loss: 543.4223 - mae: 19.1221 - val_loss: 541.3098
- val_mae: 19.3760 - lr: 0.0010 - 894s/epoch - 667ms/step
Epoch 4/10

Epoch 4: val_mae improved from 19.37602 to 18.51366, saving model
to ../models/regression_model_v1.h5
1339/1339 - 924s - loss: 537.0416 - mae: 18.9819 - val_loss: 524.9328
- val_mae: 18.5137 - lr: 0.0010 - 924s/epoch - 690ms/step
Epoch 5/10

Epoch 5: val_mae did not improve from 18.51366
1339/1339 - 898s - loss: 530.3484 - mae: 18.8205 - val_loss: 558.3091
- val_mae: 19.8452 - lr: 0.0010 - 898s/epoch - 671ms/step
Epoch 6/10

Epoch 6: val_mae did not improve from 18.51366
1339/1339 - 882s - loss: 527.3073 - mae: 18.7780 - val_loss: 532.6477
- val_mae: 19.1653 - lr: 0.0010 - 882s/epoch - 659ms/step
Epoch 7/10

Epoch 7: val_mae did not improve from 18.51366
1339/1339 - 911s - loss: 523.6089 - mae: 18.6823 - val_loss: 536.3851
- val_mae: 19.2652 - lr: 0.0010 - 911s/epoch - 680ms/step
Epoch 8/10

Epoch 8: val_mae improved from 18.51366 to 18.45678, saving model
to ../models/regression_model_v1.h5
1339/1339 - 878s - loss: 513.0731 - mae: 18.4883 - val_loss: 513.4663
- val_mae: 18.4568 - lr: 2.0000e-04 - 878s/epoch - 656ms/step
Epoch 9/10

Epoch 9: val_mae improved from 18.45678 to 18.22381, saving model
to ../models/regression_model_v1.h5
1339/1339 - 892s - loss: 511.4128 - mae: 18.4569 - val_loss: 514.0538
- val_mae: 18.2238 - lr: 2.0000e-04 - 892s/epoch - 666ms/step
Epoch 10/10

Epoch 10: val_mae did not improve from 18.22381
1339/1339 - 880s - loss: 510.7593 - mae: 18.4412 - val_loss: 517.0651
- val_mae: 18.6811 - lr: 2.0000e-04 - 880s/epoch - 658ms/step
```

Result 1: val_mae=18.68.

```
pd.DataFrame(history_stage1.history).plot(figsize=(12, 4))
plt.show()
```



```
# Save model
#
model.save(os.path.join(folder_models,f'binary_model_v{version_model}.
keras'))
```

In the next iteration, the model will be retrained, data augmentation and fine-tuning will be performed.