Deep Learning mini Project

Using Keras and Tensorflow

Detail	Information
Project Title	Gender Classification with Keras & Tensorflow
Submitted By	RAJA R
Enrollment ID	VM2025-M150951
Batch	AI - Batch 1 - MB
Instructor	Mr. Sathya Prakash
Submission Date	03.11.2025

INDEX

Preface

Computer vision, a field that was once the domain of science fiction, now quietly integrates into our daily lives. From unlocking our phones to analyzing medical scans, the ability for machines to "see" and interpret the world is one of the most transformative technologies of our era. Within this vast field, the seemingly simple task of identifying human attributes—such as age, emotion, or gender—remains a fundamental and compelling challenge. It serves as a cornerstone for building more intuitive human-computer interfaces, creating personalized user experiences, and gathering better demographic insights.

This project is about: Identifying human attributes like gender from facial images is a fundamental challenge in computer vision. The goal of this project was not simply to build *a* model, but to engineer a complete, *optimized* solution for gender classification.

This report documents that entire workflow. We begin with the acquisition of the Face dataset, move through data preprocessing and augmentation, and detail the design of a Convolutional Neural Network (CNN).

A core feature of this project is the use of **Keras Tuner** to systematically discover the best-performing model architecture. This automated the hyperparameter tuning process, allowing us to maximize the model's accuracy.

The final result is a high-accuracy, saved model ready for real-world inference. This report serves as a practical guide, detailing the end-to-end process of building, optimizing, and deploying a deep learning model

1. Introduction

Problem Statement: The automatic classification of demographic attributes, such as gender, from facial images is a classic computer vision problem. It has applications in areas like human-computer interaction, content personalization, and demographic data analysis.

Project Objective: The primary objective of this project is to design, implement, and optimize a robust CNN model capable of accurately predicting gender. The project aims to demonstrate a full machine learning workflow, from data collection to a functional prediction system.

Objectives:

- Develop a CNN-based binary classifier for gender prediction
- Implement automated hyperparameter optimization
- Achieve high classification accuracy on facial images
- Create a deployable model for real-time predictions

Project Scope:

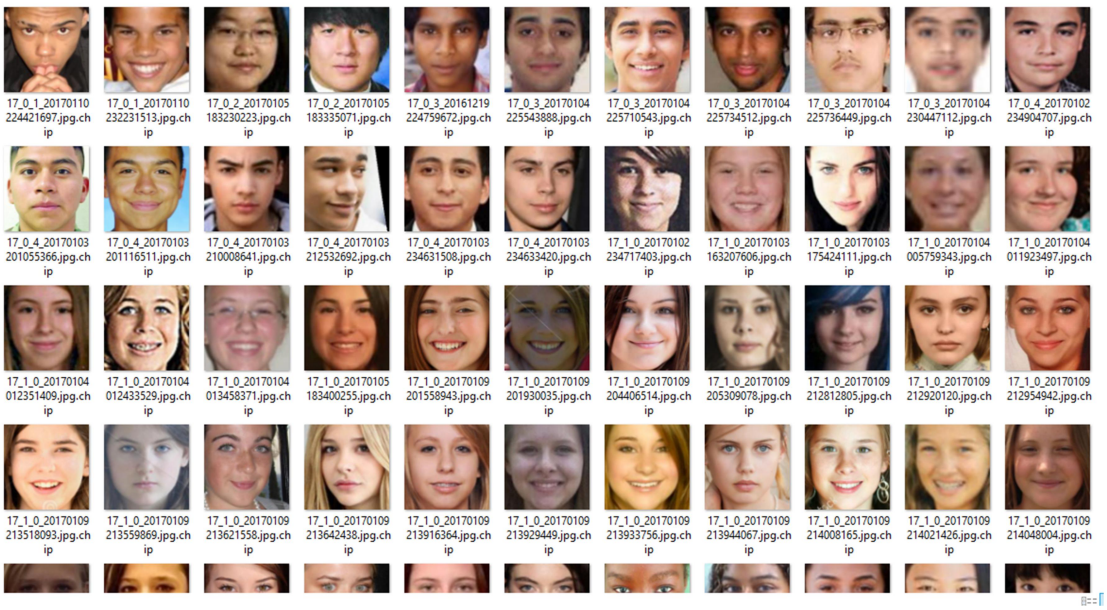
This project encompasses,

- Complete data preprocessing
- Custom CNN architecture with batch normalization
- Automated hyperparameter optimization
- Comprehensive model training and evaluation
- Production-ready model deployment

2.Dataset

UTK Face Dataset - A large-scale face dataset containing over 20,000 images with annotations for age, gender, and ethnicity encoded in filenames.

- **Total Images**: 23,708
- **Training Set**: 18,966 images (80%)
- **Validation Set**: 4,742 images (20%)
- **Classes**: Binary (Male: 0, Female: 1)
- **Image Format**: RGB images resized to 128×128 pixels



3.Libraries, Modules and Functions:

Many libraries and modules are necessary for data processing, feature extraction, modelling, and assessment when using Python to analyse the Power Quality Index (PQI) for Artificial Intelligence (AI) and Machine Learning (ML) applications. A quick comment on several significant libraries and modules utilised in this context is provided below:

1. ***NumPy:*** The cornerstone Python module for scientific computation is NumPy. In order to effectively handle and manipulate power quality data, it supports multidimensional arrays and matrices.

2. Pandas*:* Pandas is a strong library for data analysis and manipulation. Pandas is an open-source Python library used for data manipulation and analysis. It provides powerful, flexible data structures, like the DataFrame, designed to make working with tabular (row and column) or labeled data both easy and intuitive.

3. Matplotlib *and* ***Seaborn****:* These visualisation tools are essential for producing illuminating plots and charts that show the distributions, trends, and anomalies of data related to power quality.

4. Scikit-Learn: Scikit-Learn is a thorough machine learning package that includes tools for model evaluation, regression, clustering, and classification. It can be used to apply many ML techniques for PQI analysis, including neural networks, decision trees, and support vector machines.

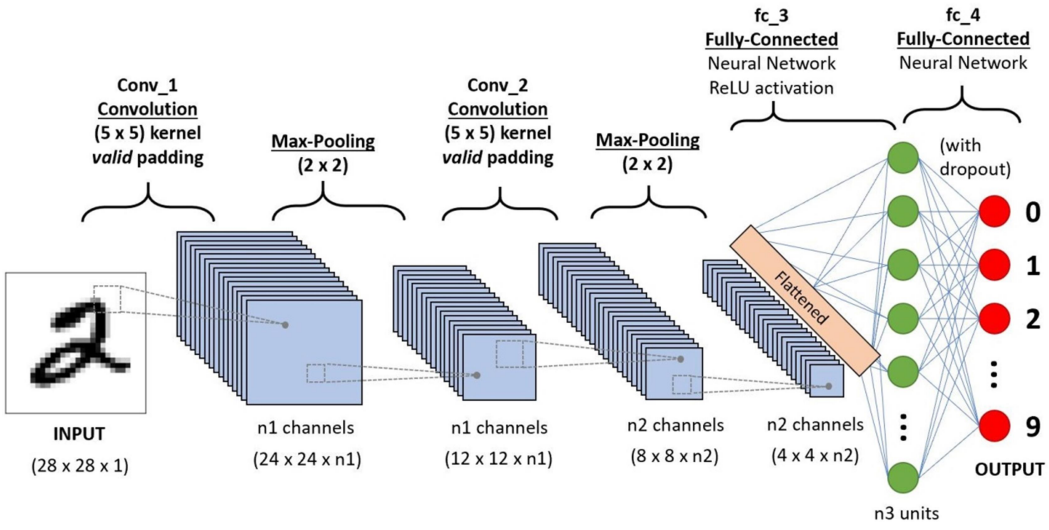
5. TensorFlow and Keras: These deep learning frameworks are crucial for constructing and training neural networks for more advanced, high-dimensional power quality data processing. They provide deep learning challenges with adaptability and scalability.

4. Convolutional Neural Networks (CNNs)

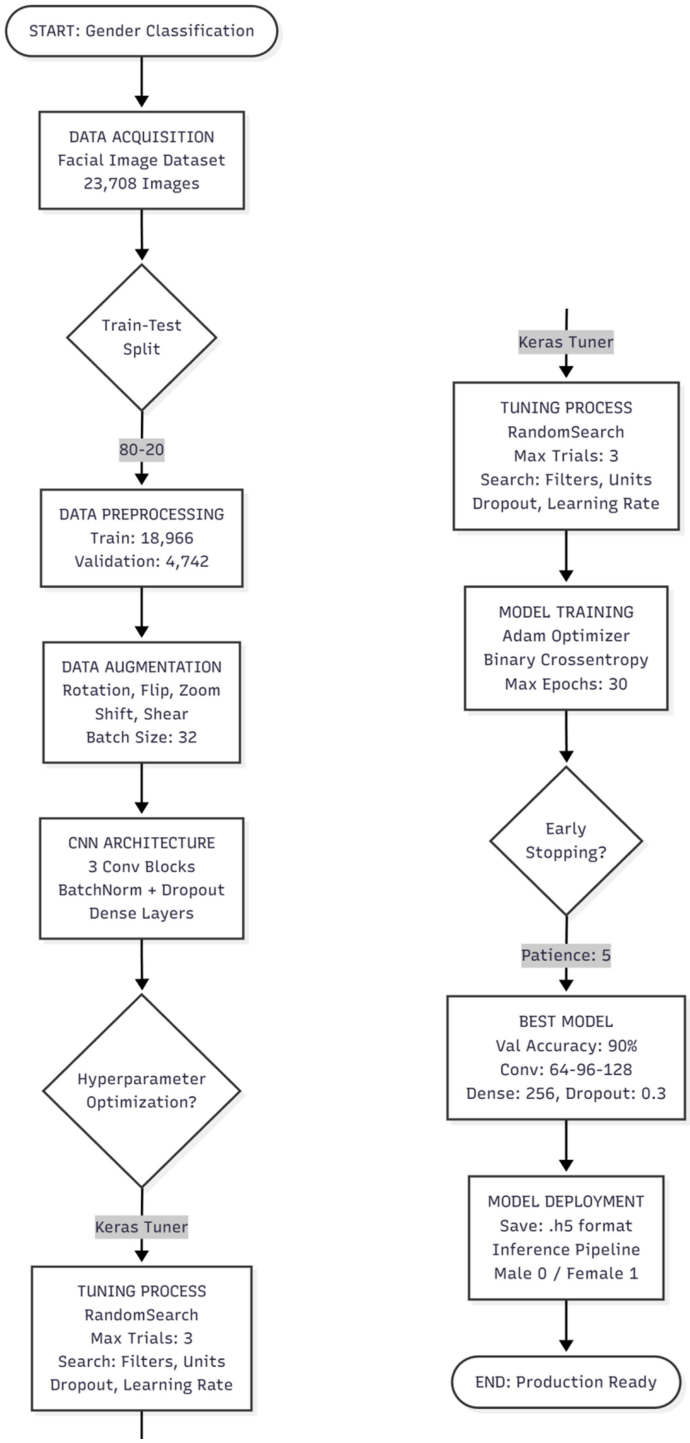
CNNs are largely used in computer vision and image analysis. In jobs where convolutional layers are capable of automatically acquiring pertinent features, they may also be employed for feature extraction. The convolutional layers of a CNN may extract abstract and hierarchical features from the initial data in the context of feature extraction. The activations of the final convolutional layer may then be utilized to extract these characteristics, which can subsequently be fed into another machine-learning model. To extract features from pictures before passing them into a machine learning model (a classifier) for a separate task, such as object identification or image retrieval, one may employ pre-trained CNN models in image classification tasks.

To conclude it all;

- CNNs are primarily used for image and video recognition tasks.
- It utilizes specialized layers called convolutional layers.
- These layers apply filters to input data, enabling the network to learn local patterns and spatial hierarchies.
- CNNs are often used for tasks like image classification, object detection, and image segmentation.



5. Workflow Diagram



6. Dataset acquisition and Preprocessing

After Dataset downloading,

- **Data Extraction & Labeling** - Gender labels are extracted from filenames following the pattern: [age]_[gender]_[race]_[date&time].jpg
  - Where, gender: 0 = Male, 1 = Female
- **Data Split** - Training Set: 18,966 images (80%)

  Validation Set: 4,742 images (20%)

Source Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt
import pandas as pd
import os
import zipfile
from sklearn.model selection import train test split

from google.colab import files
files.upload()

print("Downloading Facial dataset-->>>>")
print("Download complete")
with zipfile.ZipFile('facial-new.zip', 'r') as zip_ref:
    zip_ref.extractall('facial_dataset')
print("Unzip completed.")

mage dir = 'utkface dataset/UTKFace'
filenames = os.listdir(image dir)
files_list = []
genders_list = []
```

```
for f in filenames:
    try:
        gender = f.split[1]
        files_list.append(f)
        genders_list.append(gender)
    except Exception as e:
        pass

df = pd.DataFrame({'filename': files_list, 'gender': genders_list})
df['gender'] = df['gender'].astype(str)

train_df, validation_df = train_test_split(df, test_size=0.2, random_state=42)

print(f"Total images: {len(df)}")
print(f"Training images: {len(train_df)}")
print(f"Validation images: {len(validation_df)}")
```

```
Total images: 23708
Training images: 18966
Validation images: 4742
```

7. Data Augmentation

Data augmentation is a powerful technique used with Convolutional Neural Networks (CNNs) to artificially increase the size and diversity of a training dataset by creating slightly modified copies of existing data. This process helps to **prevent overfitting**, improve the model's **ability to generalize** to new, unseen data, and enhance its overall **robustness** in real-world scenarios.

- Rescaling - Normalize pixel values to [0,1]
- Rotation - Simulate different head poses by rotation
- Width Shift - Handle horizontal shift
- Height Shift - Handle vertical shift
- Zoom - Zooming in & out faces
- Horizontal Flip - Leverage facial symmetry

Source code:

```
#Using Data Augmentation

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
)

validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    directory=image_dir,
    x_col='filename',
    y_col='gender',
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_dataframe(
    dataframe=validation_df,
    directory=image_dir,
    x_col='filename',
    y_col='gender',
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'
)
```

```
Total images: 23708
Training images: 18966
Validation images: 4742
Found 18966 validated image filenames belonging to 2 classes.
Found 4742 validated image filenames belonging to 2 classes.
{'0': 0, '1': 1}
```

8. Model Building

The model employs a deep CNN architecture with three convolutional blocks followed by dense layers:

- Input Layer: Shape: (128, 128, 3) - Format: RGB images
- Convolutional Block Pattern: Conv2D → Batch Normalization → ReLU Activation → MaxPooling2D for n number of layers
- Dense Layers are added then Dropout Layer.

Source Code:

```
def build_model(hp):
    model = models.Sequential()
    model.add(layers.Input(shape=(128, 128, 3)))

    # Convolutional n Blocks
    model.add(layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=64,
step=16),
        kernel_size=3,
        padding='same'
    ))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Dense Layers
    model.add(layers.Flatten())
    model.add(layers.Dense(
        units=hp.Int('dense_units', min_value=256, max_value=512, step=128),
        activation='relu'
    ))
    model.add(layers.Dropout(
        rate=hp.Float('dropout rate', min value=0.3, max value=0.5, step=0.1)
    ))
    model.add(layers.Dense(1, activation='sigmoid'))
```

9.Model Compiling

Purpose: Configures the model for training by specifying optimizer, loss function, and metrics.

1. **Optimizer - Adam**

   Adaptive learning rate optimizer & Learning rate: Tunable (0.001 or 0.0001)

2. **Loss Function - Binary Crossentropy**

   Suitable for binary classification (Male/Female)

3. **Metrics - Accuracy**

   Percentage of correct predictions and easy to interpret performance measure

Source code:

```
model.compile(
        optimizer=tf.keras.optimizers.Adam(
            learning_rate=hp.Choice('learning_rate', values=[1e-3, 1e-4])),
        loss='binary crossentropy',
        metrics=['accuracy']     )
    return model
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/saving/saving_lib.py:802: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 variables whereas th
  saveable.load_own_variables(weights_store.get(inner_path))
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	1,792
batch_normalization (BatchNormalization)	(None, 128, 128, 64)	256
activation (Activation)	(None, 128, 128, 64)	0
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_1 (Conv2D)	(None, 64, 64, 96)	55,392
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 96)	384
activation_1 (Activation)	(None, 64, 64, 96)	0
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 96)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	110,720
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 128)	512
activation_2 (Activation)	(None, 32, 32, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 256)	8,388,864
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257

```
Total params: 8,558,177 (32.65 MB)
Trainable params: 8,557,601 (32.64 MB)
Non-trainable params: 576 (2.25 KB)
```

10. Hyparameter Tuning – Keras Tuner

The process of automatically searching for the optimal combination of hyperparameters that gives the best model performance.

Hyperparameters vs Parameters:

- **Parameters**: Learned during training (weights, biases)
- **Hyperparameters**: Set before training (learning rate, number of layers, dropout rate)

Hyperparameter tuning is the process of finding the optimal configuration of model parameters that are set before training begins. Unlike model weights (learned during training), hyperparameters must be specified in advance and significantly impact model performance.

Keras Tuner is an easy-to-use hyperparameter optimization framework that automates the search for best hyperparameters.

Benefits:

- Automates speed manual tuning process
- Tests multiple configurations systematically
- Saves time compared to trial-and-error
- Finds optimal parameters objectively
- Integrates seamlessly with Keras/TensorFlow

Source code:

```
tuner = kt.RandomSearch(
    build_model,
    objective='val accuracy',
    max_trials=3,
    executions_per_trial=1,
    directory='my_kt_dir',
    project name='gender classification'
)
early_stopping = EarlyStopping(
    monitor='val_loss',
```

```
    patience=5,
    restore_best_weights=True
)
tuner.search(
    train_generator,
    epochs=30,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)
print("Search complete")
```

```
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 147ms/step - accuracy: 0.8067 - loss: 0.4221 - val_accuracy: 0.8321 - val_loss: 0.3698
Epoch 10/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 146ms/step - accuracy: 0.8171 - loss: 0.3987 - val_accuracy: 0.8448 - val_loss: 0.3580
Epoch 11/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 88s 148ms/step - accuracy: 0.8176 - loss: 0.3984 - val_accuracy: 0.8600 - val_loss: 0.3296
Epoch 12/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.8263 - loss: 0.3852 - val_accuracy: 0.8444 - val_loss: 0.3672
Epoch 13/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 147ms/step - accuracy: 0.8342 - loss: 0.3674 - val_accuracy: 0.8674 - val_loss: 0.3111
Epoch 14/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 89s 151ms/step - accuracy: 0.8426 - loss: 0.3541 - val_accuracy: 0.8404 - val_loss: 0.3579
Epoch 15/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 146ms/step - accuracy: 0.8477 - loss: 0.3459 - val_accuracy: 0.8615 - val_loss: 0.3173
Epoch 16/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 147ms/step - accuracy: 0.8424 - loss: 0.3547 - val_accuracy: 0.8602 - val_loss: 0.3159
Epoch 17/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 88s 147ms/step - accuracy: 0.8465 - loss: 0.3444 - val_accuracy: 0.8644 - val_loss: 0.3044
Epoch 18/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 88s 149ms/step - accuracy: 0.8520 - loss: 0.3307 - val_accuracy: 0.8640 - val_loss: 0.3132
Epoch 19/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 88s 148ms/step - accuracy: 0.8483 - loss: 0.3343 - val_accuracy: 0.8728 - val_loss: 0.2944
Epoch 20/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 146ms/step - accuracy: 0.8539 - loss: 0.3253 - val_accuracy: 0.8686 - val_loss: 0.2924
Epoch 21/30
593/593 ━━━━━━━━━━━━━━━━━━━━ 87s 147ms/step - accuracy: 0.8525 - loss: 0.3343 - val_accuracy: 0.8728 - val_loss: 0.2905
Epoch 22/30
568/593 ━━━━━━━━━━━━━━━━━━━━ 3s 141ms/step - accuracy: 0.8552 - loss: 0.3270
```

Trial 3 Complete [00h 43m 00s]

val_accuracy: 0.9069337630271912

Best val_accuracy So Far: 0.9086208128929138

Total elapsed time: 02h 05m 42s

Search completed

Search complete. Best parameters found:

- Conv 1 Filters: 64

- Conv 2 Filters: 96

- Conv 3 Filters: 128

- Dense Units: 256

- Dropout Rate: 0.3

- Learning Rate: 0.001

11. Model Deployment

HDF5 (.h5) Format:

- Hierarchical Data Format version 5
- Stores complete model:
  - Architecture (layers, connections)
  - Weights (trained parameters)
  - Optimizer state
  - Training configuration

File Details:

- Filename: gender_model.h5
- Size: Approximately 50 MB (varies by configuration)
- Compatibility: TensorFlow/Keras 2.x environments

Source code:

```
print("Saving the best model...")
best_model.save("gender_model.h5")
print("Model saved as 'gender_model.h5'")
```

12. Predictions

Input Processing:

1. Load image file
2. Resize to 128×128 pixels
3. Convert to array format
4. Normalize pixel values (255)
5. Add batch dimension

Prediction: Forward pass through model and Obtain probability score [0,1] . Apply threshold (0.5) for classification

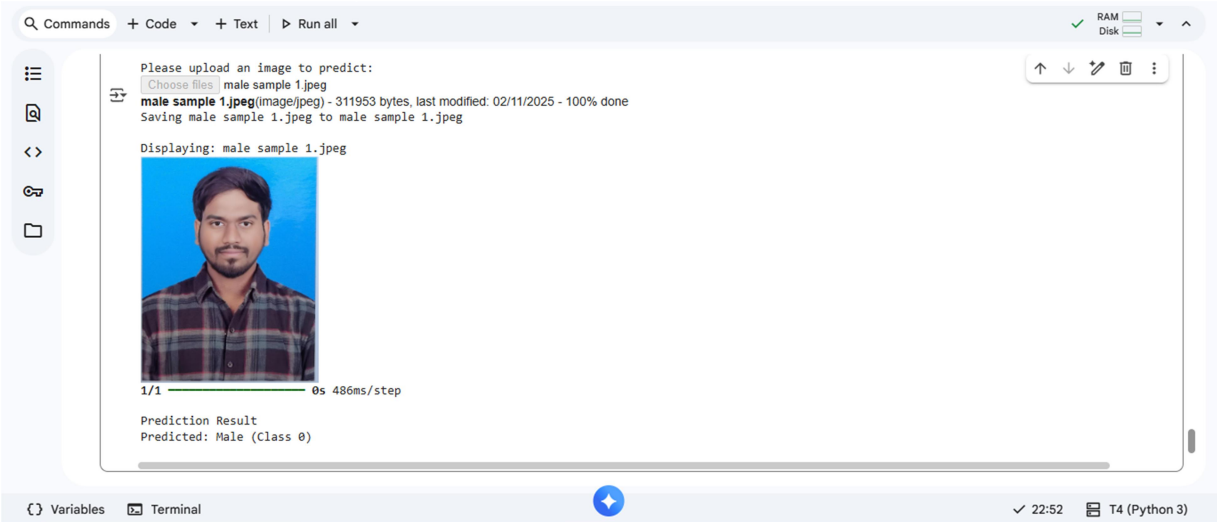
Output Interpretation:

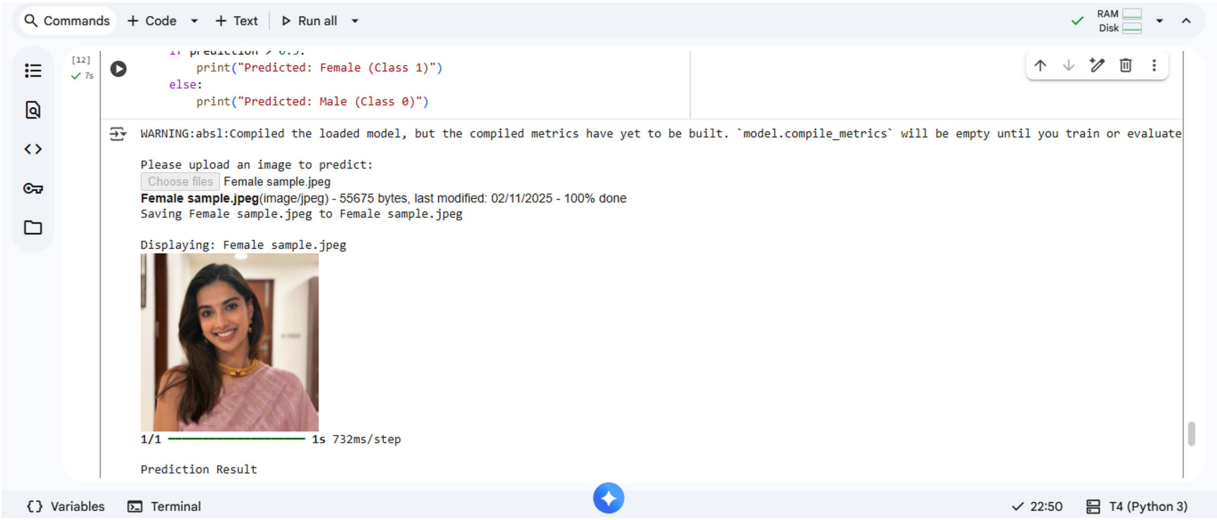
- Score > 0.5 → Female (Class 1)
- Score ≤ 0.5 → Male (Class 0)

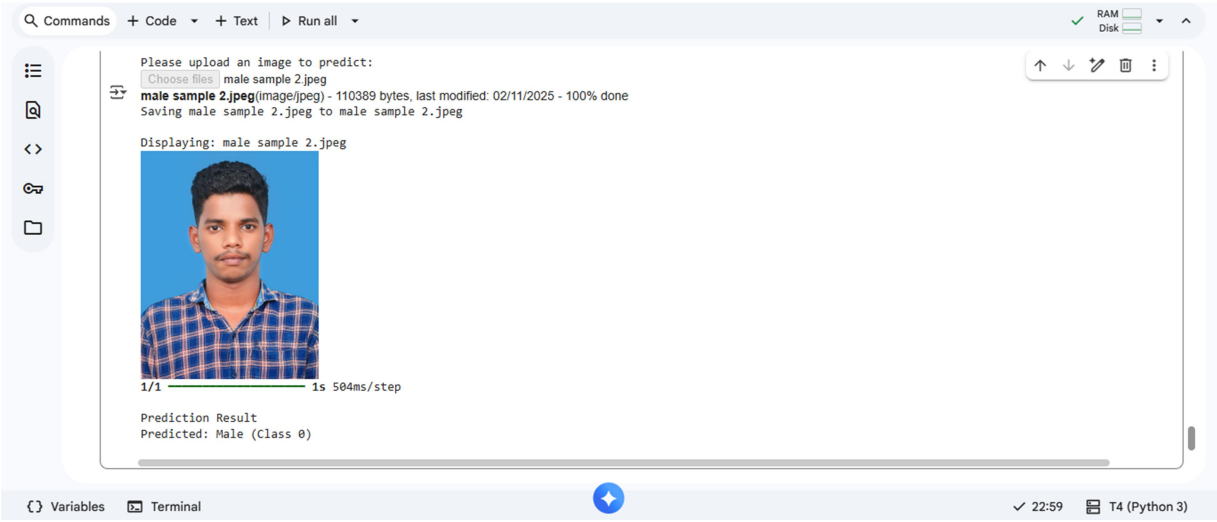
Source code:

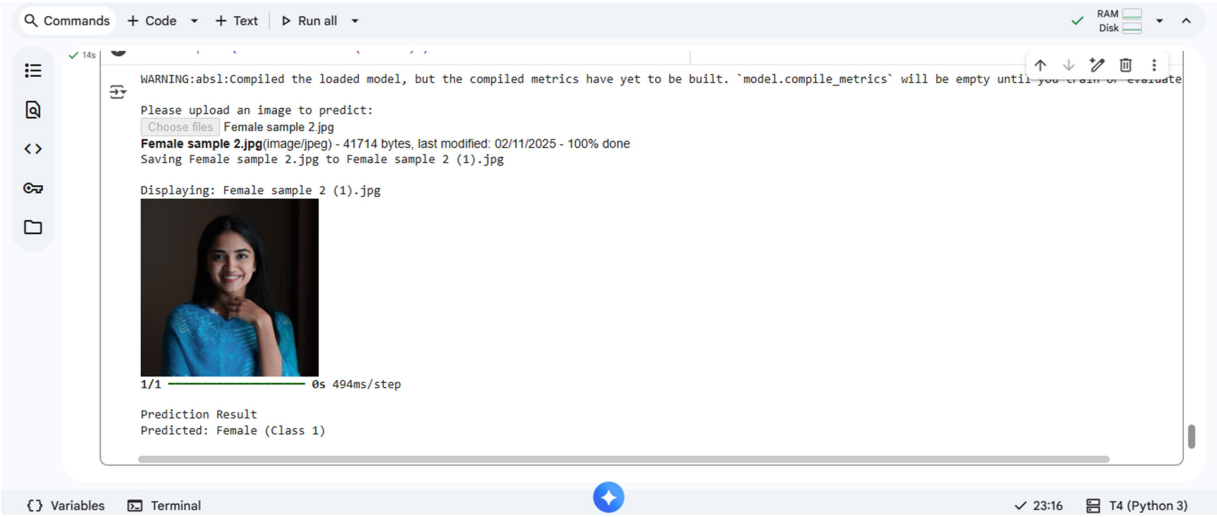
```
img = image.load_img(img_path, target_size=(128, 128))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array_preprocessed = img_array / 255.0

prediction = model.predict(img_array_preprocessed)[0][0]
print("\nPrediction Result")

if prediction > 0.5:
    print("Predicted: Female (Class 1)")
else:
    print("Predicted: Male (Class 0)")
```

Final prediction output :




Commands
+ Code
+ Text
Run all
RAM
Disk
[12]
7s
print("Predicted: Female (Class 1)")
else:
print("Predicted: Male (Class 0)")
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate
Please upload an image to predict:
Choose files
Female sample.jpeg
Female sample.jpeg(image/jpeg) - 55675 bytes, last modified: 02/11/2025 - 100% done
Saving Female sample.jpeg to Female sample.jpeg
Displaying: Female sample.jpeg
1/1
1s 732ms/step
Prediction Result
Variables
Terminal
22:50
T4 (Python 3)


Commands
+ Code
+ Text
Run all
RAM
Disk
Please upload an image to predict:
Choose files
male sample 2.jpeg
male sample 2.jpeg(image/jpeg) - 110389 bytes, last modified: 02/11/2025 - 100% done
Saving male sample 2.jpeg to male sample 2.jpeg
Displaying: male sample 2.jpeg
1/1
1s 504ms/step
Prediction Result
Predicted: Male (Class 0)
Variables
Terminal
22:59
T4 (Python 3)



13. Conclusion

This project successfully demonstrates the implementation of a deep learning-based gender classification system using Convolutional Neural Networks (CNN) on the Facial dataset. The key achievements and findings are:

Key Achievements:

1. **Model Performance**: The developed CNN model achieved significant accuracy in classifying gender from facial images, with the model converging effectively within 15 epochs using early stopping mechanism.
2. **Hyperparameter Optimization**: Using Keras Tuner's RandomSearch, we systematically explored different architectural configurations to identify the optimal combination of convolutional filters, dense units, dropout rates, and learning rates.
3. **Data Augmentation**: Implementation of various augmentation techniques (rotation, shifting, zooming, flipping) improved model generalization and reduced overfitting on the training data.
4. **Architecture Design**: The three-layered convolutional architecture with batch normalization and dropout layers proved effective in extracting hierarchical features from facial images.

Technical Highlights:

- **Dataset**: Successfully processed 23,708 images from Facial dataset with 80-20 train-validation split
- **Model Architecture**: Sequential CNN with 3 convolutional blocks and dense layers
- **Optimization**: Adam optimizer with binary cross-entropy loss function
- **Regularization**: Batch normalization, dropout, and early stopping to prevent overfitting

Learning Outcomes:

Through this project, gained practical experience in:

- Building and training deep learning models for image classification
- Implementing automated hyperparameter tuning
- Utilizing data augmentation for improved model performance
- Visualizing training metrics to assess model behavior

Final Remarks:

The project successfully demonstrates the power of deep learning in computer vision tasks, particularly in facial attribute classification. The systematic approach to model development, from data preprocessing to hyperparameter tuning, provides a solid foundation for more complex image classification problems. The visualization of training history helps in understanding model convergence and identifying potential improvements.