Exploring the Influence of Using Fingerprints to Determine Gender

Ailynne Hartsell, Aman Mehmood, Pravin Raja, and Taylor Payne
ahartsel@gmu.edu, amehmoo@gmu.edu, praja2@gmu.edu, and tpayne24@gmu.edu

AIT 736-010

George Mason University

Introduction and Problem Formulation

GEORGE MASON UNIVERSITY

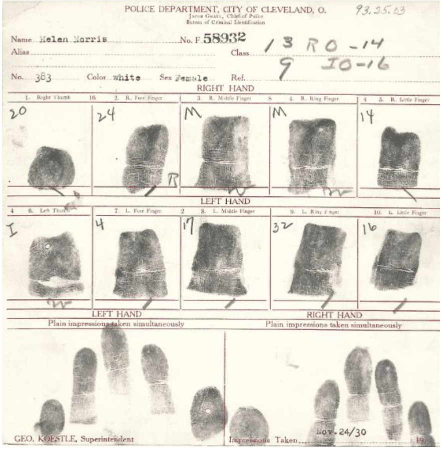
Scope of Study: Background of Fingerprints

- Fingerprints were used by Ancient Civilizations
  - Purpose:
    - Business (Contracts)
    - Identification Purposes
- 1880: Intended to help solve crimes
- 1896: Establishment of a global standard
- Now it is used for:
  - Criminal Identification
  - Access Control

Scope of Study: Why Fingerprints?

- Reliable, convenient and accurate
- Is there a link???
  - Between fingerprints and gender
  - Ultimately productive to criminal investigations

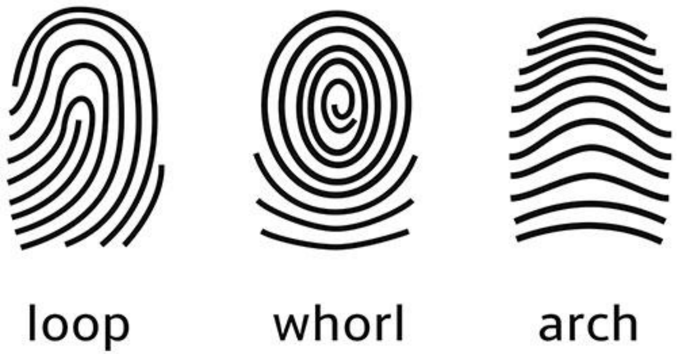
- Intentions: create a machine learning model that can categorize fingerprints based on gender.



MASON

Scope of Study: The Focus of Fingerprints

- Fingerprint Patterns:
  - Ridge and Furrow structure
  - Arch, loop and whorl
- Minutiaes
- Core
- Delta
- Pores and Sweat Glands



Scope of Study: Our Motivation

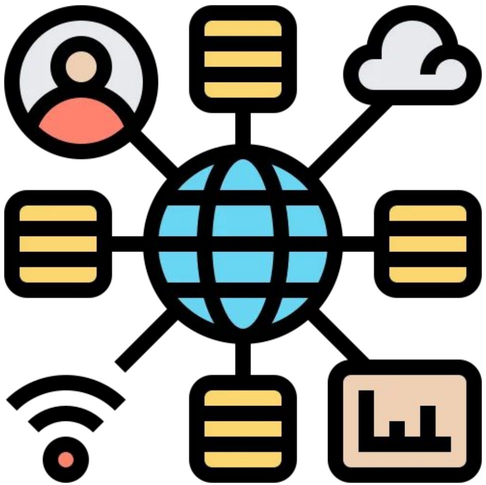
- Investigate the interaction between biometrics and gender studies
- Enhance criminal investigation procedures
- Contribute to forensic science developments
- Relation to other things include:
  - Hereditary
  - Race
  - Health

Goals

- Develop a reliable fingerprint-based gender classification model
- Analyze the accuracy and reliability of gender prediction
- Investigate biases and limitations

Challenges: General Issues

- Data quality
- Variability of Fingerprints
- Technical issues
- Dataset issues
- Algorithmic challenges



Challenge: Good vs. Bad Models

Best Model: CNN

Other Models:

- SVM: Support Vector Machine
- Random Forests
- ResNet 50

Bad Models:

- Linear Models
- Naive Bayes
- Decision Trees
- K-Means
- PCA: Principal Component Analysis

Challenges: SVM – a Potential Model

Classification Report Summary:

- Precision: 63% predictions were correct
- Recall: 63-64% of all instances were correctly identified
- F1-Score: 63% overall performance
- Support: 400 samples total (200 for each)
- Accuracy: 63%

Confusion Matrix Summary:

- Dark blue: represent correct predictions
- Light blue: represent misclassifications

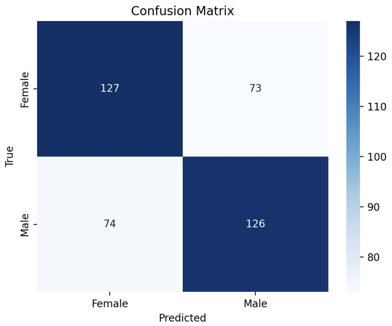
ROC Curve

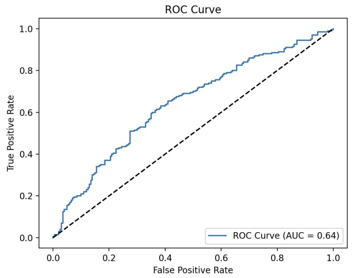
- AUC = 0.64

Takeaways:

- Some patterns may be related but here it is not distinct enough for a high accuracy
- Better luck with another model

```
Loaded 2000 images.
Gender distribution: Female=1000, Male=1000

Classification Report:
              precision    recall  f1-score   support

      Female       0.63      0.64      0.63       200
        Male       0.63      0.63      0.63       200

    accuracy                           0.63       400
   macro avg       0.63      0.63      0.63       400
weighted avg       0.63      0.63      0.63       400
```





Dataset Description

Dataset Description

Source: Kaggle's Sokoto Coventry Fingerprint Dataset (SOCOFing)

Content:

- Real fingerprints
- Altered fingerprints
  - Image format: BMP files
  - Gender information: Embedded in filenames
  - Balanced dataset: Equal representation of male and female samples

Fig. 1. Sample illustration of five left hand fingerprints belonging to the same subject.

Data Collection and Processing

Image loading:

- From 'Real' and 'Altered' folders
- Using Keras' load_img function

Image preprocessing:

- Resizing to 128x128 pixels
- Conversion to grayscale
- Normalization of pixel values to [0, 1] range

Feature extraction:

- Potential for ridge detection and minutiae extraction (not implemented)

Label encoding:

- Female: 0, Male: 1
- Conversion to categorical format for multi-class classification

Deal with Missing Data and Outliers

Error handling:

- Try-except blocks for file loading errors
- Skipping corrupted or unreadable images

Data limitation:

- Maximum number of images per gender to manage memory usage
- Helps balance the dataset and prevent class imbalance

Quality control

Outlier detection

Partitioning the Training and Testing Set

Split ratio: 80% training, 20% testing

Method: train_test_split from scikit-learn

Stratification:

- Ensures balanced class distribution in both training and testing sets
- Based on gender labels

Random state:

- Set for reproducibility (random_state=42)
- Validation strategy:
    - Additional 20% validation split from training data during model fitting

Data augmentation:

- Applied only to training data
- Includes rotation, width/height shift, zoom, and horizontal flip

Undersampling

- Used to account for overrepresentation in original data

Developing the Model

Describe the Model: Framework

Convolutional Neural Network (CNN)

Model architecture:

- Input layer: 128x128x1 (grayscale images)
- Convolutional layers with ReLU activation
- Max pooling layers
- Flatten layer
- Dense layers with ReLU activation
- Dropout layer for regularization
- Output layer with softmax activation for binary classification

Describe the Model: Computation

- The model is built with the Adam optimizer and the categorical cross entropy loss function.

Dataset Description:

- The project makes advantage of Kaggle's Sokoto Coventry Fingerprint Dataset
- (SOCOFing). This collection contains both real and manipulated fingerprints.

Describe the Model: Formula

- Convolutional layers: Apply learned filters to extract features
    - Formula: conv(x, W) + b
    - x: input, W: filter weights, b: bias
- ReLU activation: Introduce non-linearity
    - Formula: max(0, x)
- Max pooling: Downsample feature maps
    - Formula: max(x) within each pooling window
- Flatten: Convert 2D feature maps to 1D vector
- Dense layers: Fully connected layers for classification
    - Formula: dot(x, W) + b
    - x: input, W: weights, b: bias
- Softmax activation: Convert output to probability distribution
    - Formula: exp(x_i) / sum(exp(x_j)) for each class i

Describe the Model: Loss Function

- Categorical cross entropy loss function
- Measures dissimilarity between predicted and true probability distributions
- Formula: -sum(y_true * log(y_pred))
  - y_true: true label distribution
  - y_pred: predicted probability distribution
- Suitable for multi-class classification problems

Describe the Model: Training Strategy

- Train-test split: 80% training, 20% testing
- Data augmentation using ImageDataGenerator
  - Rotation, width/height shift, zoom, horizontal flip

Callbacks:

- EarlyStopping: Monitor validation loss, stop training if no improvement
- ReduceLROnPlateau: Reduce learning rate if validation loss plateaus

Training parameters:

- Epochs: 50
- Batch size: 32
- Validation split: 0.2

Describe the Model: CNN Backpropagation

Forward Pass

- Input → Convolutional layers (feature extraction) → Pooling layers (dimension reduction) → Fully connected layers (predictions).

Loss Calculation

- Compare output to true labels using a loss function to quantify error.

Backward Pass (Backpropagation)

- Compute gradients using the chain rule and propagate errors backward.

Parameter Update

- Use gradients to update weights and biases with optimization algorithms (e.g., SGD, Adam).

Key Features of CNNs

- Update filter weights in convolutional layers.
- Backpropagate gradients in pooling layers.

Additional Techniques

- Apply regularization (e.g., L2) to prevent overfitting.
- Adjust learning rate dynamically during training.
- Train using mini-batches for efficiency.

Visualizing the Model

Confusion Matrix:

- Heatmap visualization of true vs. predicted labels
- Assess model's performance in each class

ROC Curve:

- Plot of True Positive Rate (TPR) vs. False Positive Rate (FPR)
- Evaluate model's discriminative power
- Area Under the Curve (AUC) as a performance metric

Training and Validation Accuracy/Loss Curves:

- Plot accuracy and loss over epochs
- Monitor model's learning progress and overfitting/underfitting

Alternative CNN Model Framework

- Binary Classification
- Two convolutional layers using 32 and 64 filters
- Max-pooling after each convolutional layer
- Flattened output of layers passed through fully connected layer
- Output layer: Sigmoid Activation
- Class weights
- Early Stopping
- Adam Optimizer and Binary Crossentropy Loss

Alternative CNN Model: Computation

- Adam Optimizer
  - Adjusts model weights using adaptive learning rates.
- Binary Cross Entropy Loss Function
  - Measures the difference between predicted probability and true binary labels.
  - More appropriate for binary data.

Alternative CNN Model: Training Strategy

- Train-test split: 80% training, 20% testing
- Females were undersampled, males were oversampled.

Callbacks:

- EarlyStopping: Monitor validation loss, stop training if no improvement

Training parameters:

- Epochs: 50
- Batch size: 32
- Validation split: 0.2

Alternative Model: Transfer Deep Learning

- Transfer model with ResNet50
  - Leveraged deep learning architecture and ability to extract complex features
- Custom top layers added:
  - Global average pooling layer
  - Dropout layer (rate = 0.5)
- Dense output layer with sigmoid activation

MASON

Alternative Model: Preprocessing and Augmentation

- Image resizing to match ResNet50 requirements
- Data augmentation:
  - Rescaling
  - Rotation
  - Width and height shifts
  - Shear and zoom transformations
- Goal:
  - Increase data variability to enhance model generalization

Alternative Model: Training

- Parameters:
  - Optimizer: Adam (learning rate = 0.001)
  - Loss function: binary crossentropy
  - Evaluation metrics: accuracy, precision, recall, AUC
- Training strategy:
  - Initial training:
    - Base layers frozen
    - Trained custom top layers (10 epochs)
  - Fine-tuning:
    - Unfroze last 10 layers
    - Reduce learning rate (1e-5)
    - Train for additional 5 epochs

Evaluation Results

Final Results

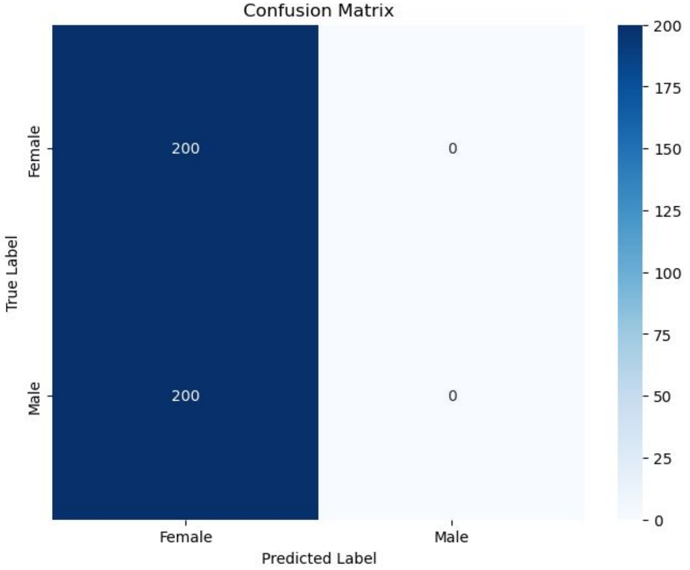
Model Performance:
- Overall accuracy on test set
- Gender-specific accuracy (Male vs. Female)

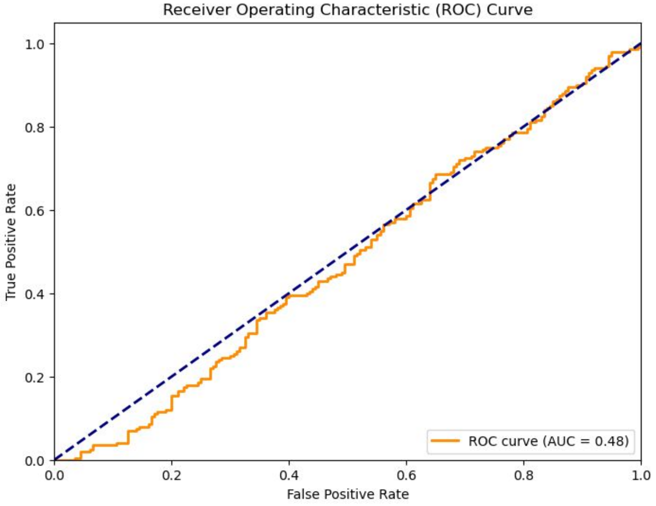
Training History:
- Training and validation accuracy over epochs
- Training and validation loss over epochs

Key Findings:
- Effectiveness of CNN for fingerprint-based gender classification
- Potential biases or limitations identified
- Comparison with existing literature or benchmarks (if available)

CNN Model Results

```
Classification Report:
              precision    recall  f1-score   support

      Female       0.50      1.00      0.67       200
        Male       0.00      0.00      0.00       200

    accuracy                           0.50       400
   macro avg       0.25      0.50      0.33       400
weighted avg       0.25      0.50      0.33       400
```





Alternative CNN Model: Results

Model Performance:

- 91% accuracy on test set.

Training History:

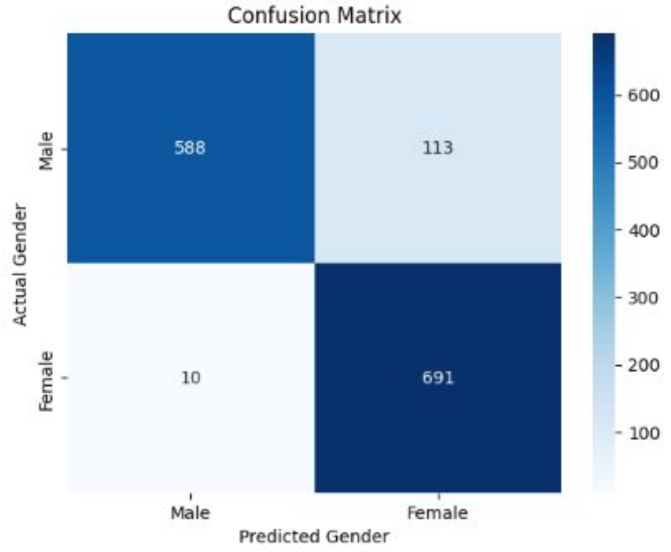
- Training accuracy increased from 50.74% to 99.83%
- Test accuracy increased from 59.99% to 91.23%.
- Training loss decreased from 1.16 to 0.0083.
- Test loss decreased from 0.6867 to 0.6094.

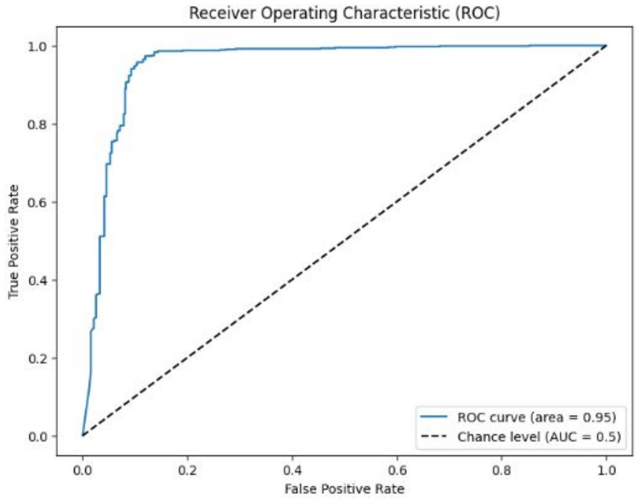
Key Findings:

- Model performed better than original CNN model.

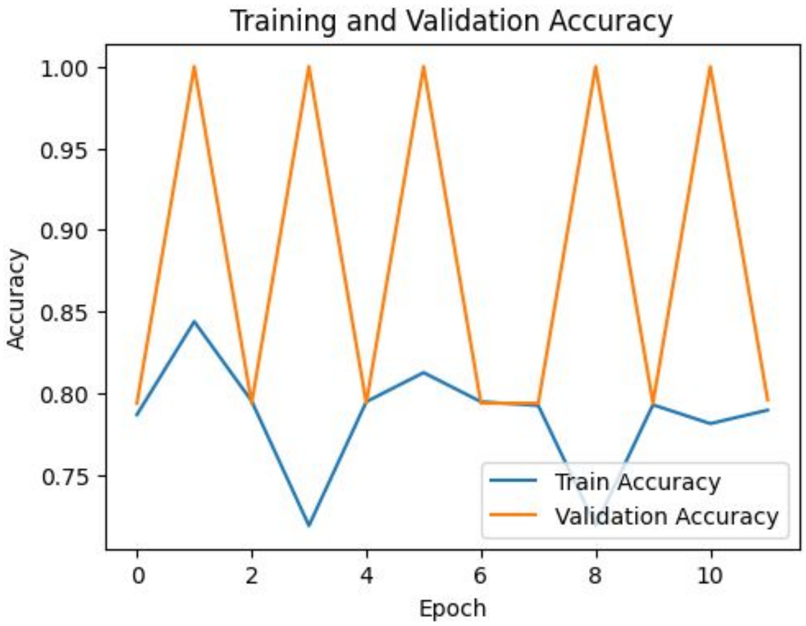
Alternative CNN Model Results

	precision	recall	f1-score	support
Male	0.98	0.84	0.91	701
Female	0.86	0.99	0.92	701
accuracy			0.91	1402
macro avg	0.92	0.91	0.91	1402
weighted avg	0.92	0.91	0.91	1402

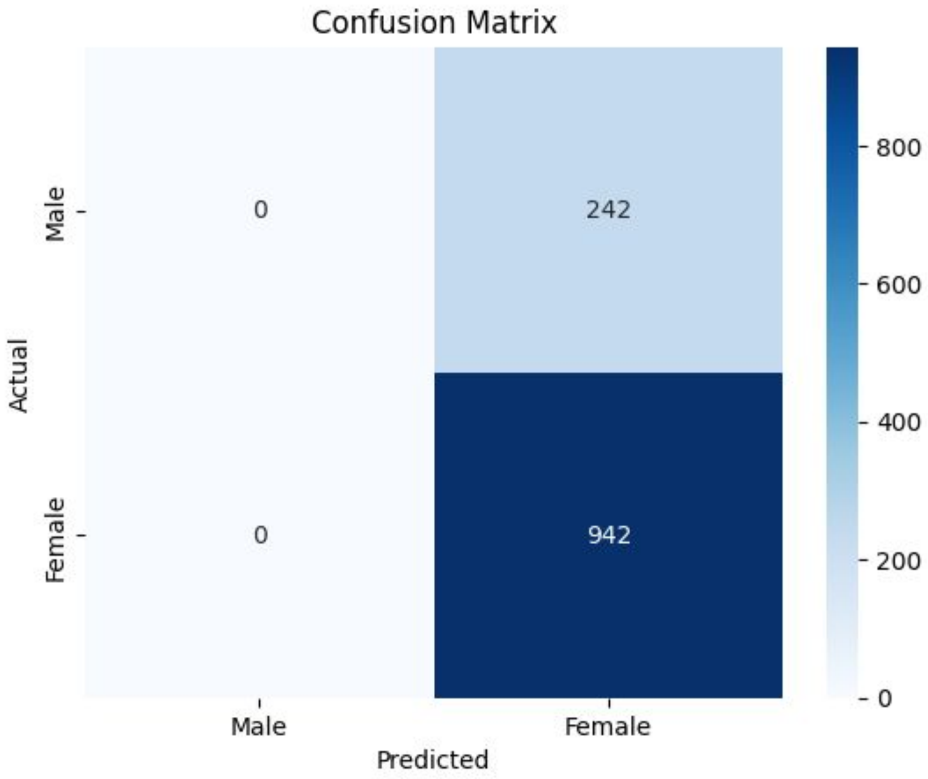




Alternative Model ResNet50: Evaluation Metrics



Alternative Model ResNet50: Imbalanced Classes



	precision	recall	f1-score	support
Male	0.00	0.00	0.00	242
Female	0.80	1.00	0.89	942
accuracy			0.80	1184
macro avg	0.40	0.50	0.44	1184
weighted avg	0.63	0.80	0.71	1184

Run the Model on a Test Set

Test Set Preparation:
- 20% of data reserved for testing (stratified split)
- Preprocessing applied consistently with training data

Model Prediction:
- Use trained model to predict gender on test set
- Generate probability scores for each class

Output Format:
- Binary classification (Male/Female)
- Probability scores for each class

Using Evaluation Metrics to Measure the Correctness of the Classifier

Classification Report:

- Precision: Accuracy of positive predictions
- Recall: Fraction of positives correctly identified
- F1-score: Harmonic mean of precision and recall
- Support: Number of occurrences of each class

Confusion Matrix:

- True Positives, False Positives, True Negatives, False Negatives
- Visualization using heatmap for easy interpretation
- ROC Curve and AUC:
- Plot of True Positive Rate vs. False Positive Rate
- Area Under the Curve (AUC) as a measure of model performance

Summary

Classification Report:

- Precision: Accuracy of positive predictions
- Recall: Fraction of positives correctly identified
- F1-score: Harmonic mean of precision and recall
- Support: Number of occurrences of each class

Confusion Matrix:

- True Positives, False Positives, True Negatives, False Negatives
- Visualization using heatmap for easy interpretation

ROC Curve and AUC:

- Plot of True Positive Rate vs. False Positive Rate
- Area Under the Curve (AUC) as a measure of model performance

Data Handling:

- Preprocessed and normalized fingerprint images
- Implemented data augmentation techniques
- Stratified train-test split for balanced representation

Evaluation Metrics:

- Accuracy, precision, recall, and F1-score
- Confusion matrix and ROC curve analysis

Ethical Considerations:

- Addressed potential biases in the dataset and model
- Discussed privacy concerns and implications of gender classification

Future Directions:

- Explore advanced CNN architectures (e.g., ResNet, VGG)
- Investigate the impact of fingerprint traits on classification accuracy
- Consider expanding to additional demographic factors

MASON

Final Thoughts

Key Takeaways

- Gender can be determined from a fingerprint with reasonable accuracy. This has implications in many areas including:
  - Forensic Science and Law Enforcement
  - Biometric Security
  - Medical and Research Application

Further Exploration

- Exploring specific fingers and if there are fingers that are better at indicating gender.
- Opening research to different races.
- Exploring if fingerprints can indicate other characteristics, such as age.

References

References

- Alabama State Department of Education. (n.d.). Fingerprints and forensic science. ACCESS Virtual Learning. Retrieved from https://accessdl.state.al.us/AventaCourses/access_courses/forensic_sci_ua_v22/03_unit/03-05/03-05_learn_text.htm
- Alamy. (n.d.). Vintage fingerprints stock photos. Retrieved from https://www.alamy.com/stock-photo/vintage-fingerprints.html?sortBy=relevant
- Geggel, L. (2016). Why do humans have fingerprints? Live Science. Retrieved from https://www.livescience.com/why-do-humans-have-fingerprints.html
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. [ISBN: 978-0-262-03561-3]
- Gnanasivam, P., & Muttan, S. (2013). Gender classification using fingerprint through frequency domain analysis. European Journal of Scientific Research, 59(2), 191-199. https://doi.org/10.1109/ICICIP.2012.6391490
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. Retrieved from https://numpy.org/
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Retrieved from https://matplotlib.org/
- Kindt, E. J. (2013). Privacy and data protection issues of biometric applications. Springer. https://doi.org/10.1007/978-94-007-7522-0
- Keras. (n.d.). Keras. Retrieved from https://keras.io/
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105. https://doi.org/10.1145/3065386
- Maltoni, D., Maio, D., Jain, A. K., & Prabhakar, S. (2009). Handbook of fingerprint recognition. Springer Science & Business Media. [ISBN: 978-1-84882-254-2]
- Marasco, E., & Lugini, L. (2014). Exploiting quality and texture features to estimate age and gender from fingerprints. SPIE Defense+ Security, 9075, 90750F. https://doi.org/10.1117/12.2054201
- McKinney, W. (2010). Data structures for statistical computing in Python. Retrieved from https://pandas.pydata.org/
- Mordini, E., & Tzovaras, D. (2012). Second generation biometrics: The ethical, legal and social context. Springer Science & Business Media. [ISBN: 978-94-007-3892-1]
- Mysterious 3000-year-old fingerprints found at ancient site. (2019). Retrieved from https://nypost.com/2019/06/04/mysterious-3000-year-old-fingerprints-found-at-ancient-site/
- OpenCV. (2021). Fingerprint recognition OpenCV Python tutorial. Retrieved from https://www.phddirection.com/fingerprint-recognition-opencv-python/
- PackIOT. (n.d.). Data hub: Unified Namespace (UNS). Retrieved from https://packiot.com/data-hub-uns-unified-namespace/
- Prabhakar, S., Pankanti, S., & Jain, A. K. (2003). Biometric recognition: Security and privacy concerns. IEEE Security & Privacy, 1(2), 33-42. https://doi.org/10.1109/MSECP.2003.1193209
- Ruiz Gara, A. (n.d.). Sokoto Coventry Fingerprint Dataset (SOCOFing). Kaggle. Retrieved from https://www.kaggle.com/ruizgara/socofing
- Scikit-learn Developers. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/
- Shehu, Y.I., Ruiz-Garcia, A., Palade, V., James, A. (2018) "Detection of Fingerprint Alterations Using Deep Convolutional Neural Networks" in Proceedings of the In ternational Conference on Artificial Neural Networks (ICANN 2018), Rhodes – Greece, 5th - 7th October 2018. Springer-Verlag Lecture Notes in Computer Science.
- *Sokoto Coventry Fingerprint Dataset (SOCOFing)*. (2018). Semantic Scholar. Retrieved November 25, 2024, from https://www.semanticscholar.org/paper/Sokoto-Coventry-Fingerprint-Dataset-Shehu-Ruiz-Garcia/88f13170c952860878b2be2767578f7fee0d2261.
- TensorFlow. (2021). Convolutional neural network (CNN). Retrieved from https://www.tensorflow.org/tutorials/images/cnn
- Towards Data Science. (2019). A comprehensive guide to convolutional neural networks. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- Waskom, M. L. (2021). Seaborn: Statistical Data Visualization. Retrieved from https://seaborn.pydata.org/

- Shehu, Y.I., Ruiz-Garcia, A., Palade, V., James, A. (2018) "Detection of Fingerprint Alterations Using Deep Convolutional Neural Networks" in Proceedings of the In ternational Conference on Artificial Neural Networks (ICANN 2018), Rhodes – Greece, 5th - 7th October 2018. Springer-Verlag Lecture Notes in Computer Science.

- "Backward Pass in Convolutional Neural Network (CNN) Explained." *PyCodeMates*, 13 July 2023, www.pycodemates.com/2023/07/backward-pass-in-convolutional-neural-network-explained.html. Accessed 1 Dec. 2024.

MASON

Appendix

Finger Print Model

FingerPrint.py – Refer the GitHub link - AIT736_Fingerprint_Analysis_MachineLearning/fingerprint_classify.ipynb at main · Pravinraja/AIT736_Fingerprint_Analysis_MachineLearning

Results & output

fingerprint.ipynb

File Edit View Run Kernel Tabs Settings Help

Code

```
# Evaluate model
classifier.evaluate_model(X_test, y_test)
```

```
Loading and preprocessing fingerprint images...

Dataset Statistics:
Total images: 2000
Gender distribution: Female=1000, Male=1000
\\?\C:\Users\raja\AppData\Roaming\jupyterlab-desktop\jlab_server\Lib\site-packag
efer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model Architecture:
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 126, 126, 32)	320
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
conv2d_10 (Conv2D)	(None, 124, 124, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 124, 124, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 62, 62, 32)	0
dropout_3 (Dropout)	(None, 62, 62, 32)	0
conv2d_11 (Conv2D)	(None, 60, 60, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 64)	256
conv2d_12 (Conv2D)	(None, 58, 58, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 58, 58, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 29, 29, 64)	0
dropout_4 (Dropout)	(None, 29, 29, 64)	0
conv2d_13 (Conv2D)	(None, 27, 27, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 27, 27, 128)	512
conv2d_14 (Conv2D)	(None, 25, 25, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 25, 25, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_5 (Dropout)	(None, 12, 12, 128)	0
flatten_3 (Flatten)	(None, 18432)	0
dense_6 (Dense)	(None, 512)	9,437,696
batch_normalization_6 (BatchNormalization)	(None, 512)	2,048
dropout_6 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 256)	131,328
batch_normalization_7 (BatchNormalization)	(None, 256)	1,024
dropout_7 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 2)	514

Total params: 9,860,834 (37.62 MB)

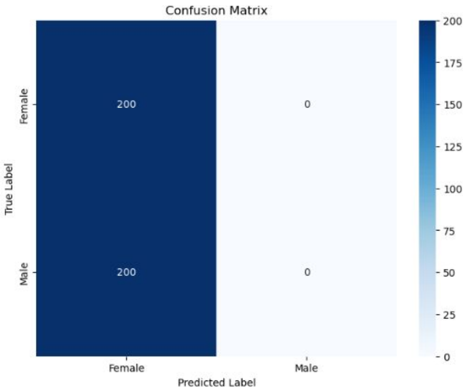
Trainable params: 9,858,402 (37.61 MB)

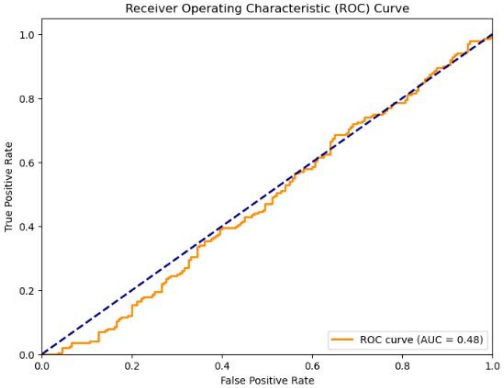
Non-trainable params: 2,432 (9.50 KB)

```
\\?\C:\Users\raja\AppData\Roaming\jupyterlab-desktop\jlab_server\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
`**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - accuracy: 0.5874 - loss: 1.2644 - val_accuracy: 0.5000 - val_loss: 0.7602 - learning_rate: 0.0010
Epoch 2/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.6534 - loss: 0.8480 - val_accuracy: 0.5000 - val_loss: 2.3165 - learning_rate: 0.0010
Epoch 3/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 79s 2s/step - accuracy: 0.6215 - loss: 0.8000 - val_accuracy: 0.5000 - val_loss: 1.1506 - learning_rate: 0.0010
Epoch 4/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 74s 1s/step - accuracy: 0.6494 - loss: 0.7957 - val_accuracy: 0.5000 - val_loss: 1.5451 - learning_rate: 0.0010
Epoch 5/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 77s 2s/step - accuracy: 0.6227 - loss: 0.7749 - val_accuracy: 0.4900 - val_loss: 0.8197 - learning_rate: 0.0010
Epoch 6/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.6663 - loss: 0.6950 - val_accuracy: 0.5000 - val_loss: 0.9620 - learning_rate: 0.0010
Epoch 7/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.6582 - loss: 0.6899 - val_accuracy: 0.4550 - val_loss: 0.9066 - learning_rate: 1.0000e-04
```

Results & output

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.6582 - loss: 0.6899 - val_accuracy: 0.4550 - val_loss: 0.9066 - learning_rate: 1.0000e-04
Epoch 8/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 75s 2s/step - accuracy: 0.6721 - loss: 0.6428 - val_accuracy: 0.4650 - val_loss: 0.9676 - learning_rate: 1.0000e-04
Epoch 9/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.6985 - loss: 0.6340 - val_accuracy: 0.4925 - val_loss: 1.0975 - learning_rate: 1.0000e-04
Epoch 10/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 74s 1s/step - accuracy: 0.6811 - loss: 0.6330 - val_accuracy: 0.5050 - val_loss: 1.2704 - learning_rate: 1.0000e-04
Epoch 11/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 77s 2s/step - accuracy: 0.6737 - loss: 0.6427 - val_accuracy: 0.5050 - val_loss: 1.1770 - learning_rate: 1.0000e-04
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 360ms/step

Classification Report:
              precision    recall  f1-score   support

      Female       0.50      1.00      0.67       200
        Male       0.00      0.00      0.00       200

    accuracy                           0.50       400
   macro avg       0.25      0.50      0.33       400
weighted avg       0.25      0.50      0.33       400

\\?\C:\Users\raja\AppData\Roaming\jupyterlab-desktop\jlab_server\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
\\?\C:\Users\raja\AppData\Roaming\jupyterlab-desktop\jlab_server\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
\\?\C:\Users\raja\AppData\Roaming\jupyterlab-desktop\jlab_server\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```





Results & output

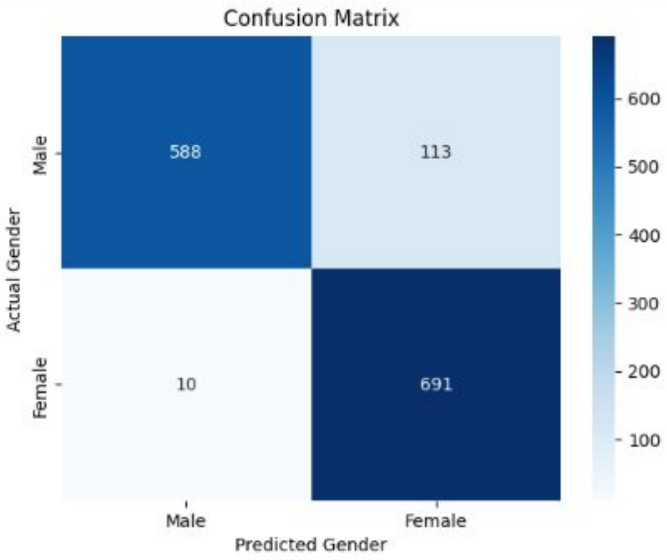
```
Epoch 1/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 62s 247ms/step - accuracy: 0.5074 - loss: 1.1624 - val_accuracy: 0.5999 - val_loss: 0.6867
Epoch 2/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 233ms/step - accuracy: 0.5549 - loss: 0.7050 - val_accuracy: 0.5792 - val_loss: 0.7330
Epoch 3/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 233ms/step - accuracy: 0.6011 - loss: 0.6373 - val_accuracy: 0.6505 - val_loss: 0.6389
Epoch 4/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 235ms/step - accuracy: 0.6960 - loss: 0.5251 - val_accuracy: 0.8181 - val_loss: 0.4306
Epoch 5/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 234ms/step - accuracy: 0.8399 - loss: 0.3384 - val_accuracy: 0.8017 - val_loss: 0.4949
Epoch 6/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 55s 231ms/step - accuracy: 0.9153 - loss: 0.1976 - val_accuracy: 0.8388 - val_loss: 0.4300
Epoch 7/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 233ms/step - accuracy: 0.9429 - loss: 0.1277 - val_accuracy: 0.8859 - val_loss: 0.3569
Epoch 8/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 56s 232ms/step - accuracy: 0.9787 - loss: 0.0574 - val_accuracy: 0.9030 - val_loss: 0.3548
Epoch 9/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 55s 231ms/step - accuracy: 0.9862 - loss: 0.0375 - val_accuracy: 0.9330 - val_loss: 0.4012
Epoch 10/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 82s 230ms/step - accuracy: 0.9942 - loss: 0.0173 - val_accuracy: 0.9394 - val_loss: 0.4316
Epoch 11/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 54s 227ms/step - accuracy: 0.9983 - loss: 0.0165 - val_accuracy: 0.9394 - val_loss: 0.5043
Epoch 12/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 57s 238ms/step - accuracy: 0.9994 - loss: 0.0046 - val_accuracy: 0.9365 - val_loss: 0.4611
Epoch 13/50
239/239 ━━━━━━━━━━━━━━━━━━━━ 55s 231ms/step - accuracy: 0.9983 - loss: 0.0083 - val_accuracy: 0.9123 - val_loss: 0.6094
```

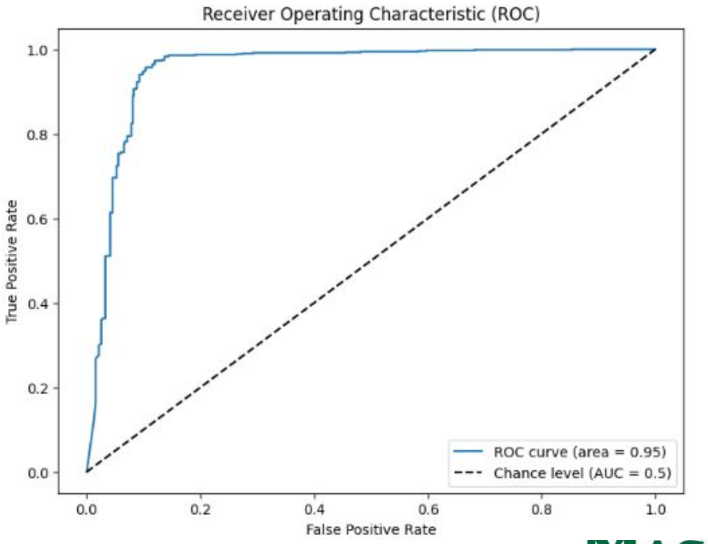
```
44/44 - 3s - 57ms/step - accuracy: 0.9123 - loss: 0.6094
Test accuracy: 91.23%
```

MASON

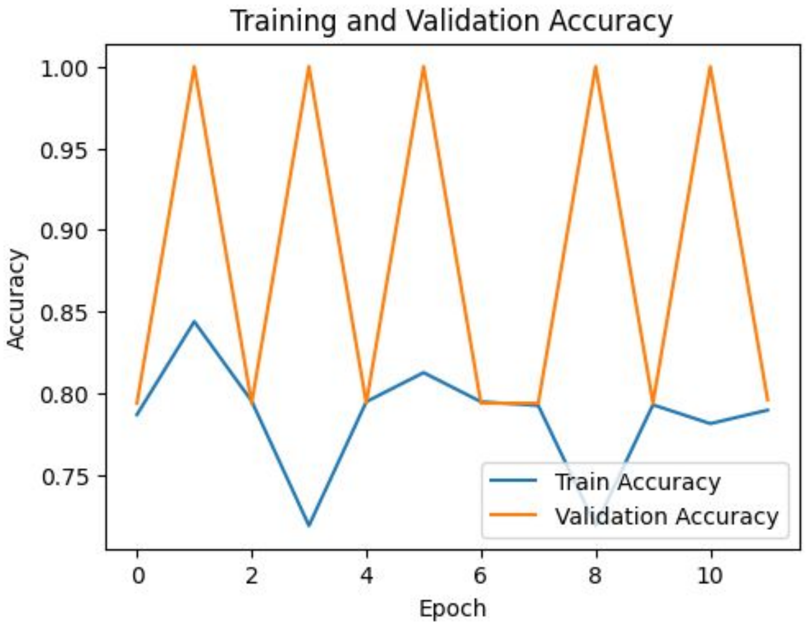
Results & output

	precision	recall	f1-score	support
Male	0.98	0.84	0.91	701
Female	0.86	0.99	0.92	701
accuracy			0.91	1402
macro avg	0.92	0.91	0.91	1402
weighted avg	0.92	0.91	0.91	1402

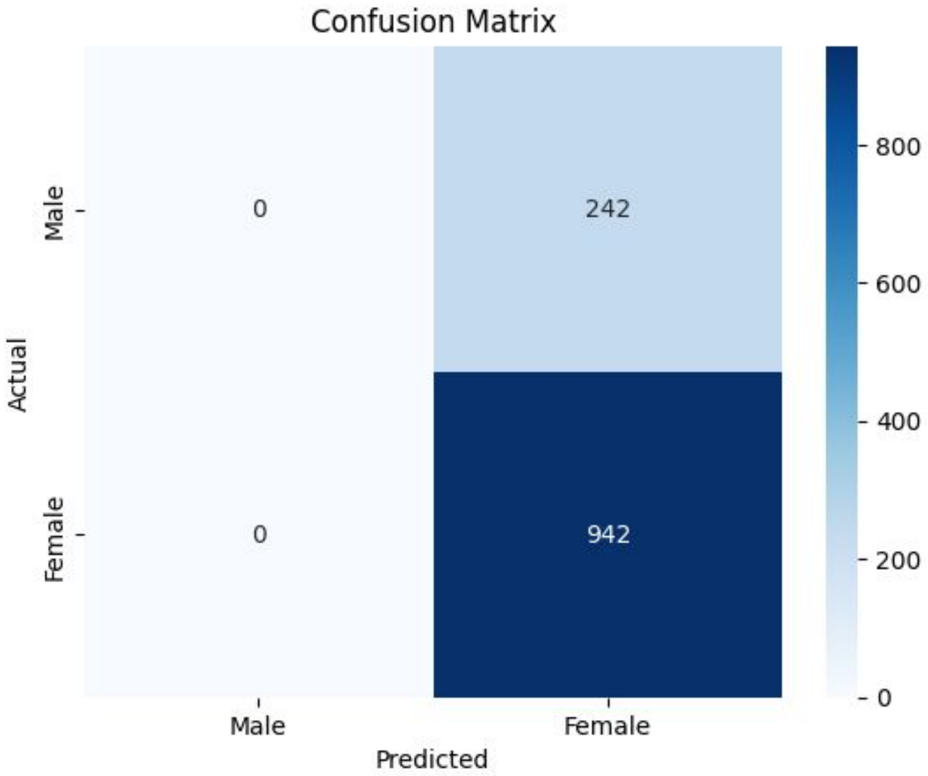




ResNet50: Accuracy, Loss, Precision, Recall



ResNet50: Imbalanced Classes



Backpropagation Code Link

– Refer the GitHub link - AIT736_Fingerprint_Analysis_MachineLearning/fingerprint_backpropagation.ipynb at main · Pravinraja/AIT736_Fingerprint_Analysis_MachineLearning

Backpropagation Results & Output

```
Model Architecture:
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 126, 126, 32)	320
batch_normalization_16 (BatchNormalization)	(None, 126, 126, 32)	128
conv2d_22 (Conv2D)	(None, 124, 124, 32)	9,248
batch_normalization_17 (BatchNormalization)	(None, 124, 124, 32)	128
max_pooling2d_12 (MaxPooling2D)	(None, 62, 62, 32)	0
dropout_13 (Dropout)	(None, 62, 62, 32)	0
conv2d_23 (Conv2D)	(None, 60, 60, 64)	18,496
batch_normalization_18 (BatchNormalization)	(None, 60, 60, 64)	256
conv2d_24 (Conv2D)	(None, 58, 58, 64)	36,928
batch_normalization_19 (BatchNormalization)	(None, 58, 58, 64)	256
max_pooling2d_13 (MaxPooling2D)	(None, 29, 29, 64)	0
dropout_14 (Dropout)	(None, 29, 29, 64)	0
conv2d_25 (Conv2D)	(None, 27, 27, 128)	73,856
batch_normalization_20 (BatchNormalization)	(None, 27, 27, 128)	512
conv2d_26 (Conv2D)	(None, 25, 25, 128)	147,584
batch_normalization_21 (BatchNormalization)	(None, 25, 25, 128)	512
max_pooling2d_14 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_15 (Dropout)	(None, 12, 12, 128)	0
flatten_5 (Flatten)	(None, 18432)	0
dense_12 (Dense)	(None, 512)	9,437,696
batch_normalization_22 (BatchNormalization)	(None, 512)	2,048
dropout_16 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 256)	131,328
batch_normalization_23 (BatchNormalization)	(None, 256)	1,024
dropout_17 (Dropout)	(None, 256)	0
dense_14 (Dense)	(None, 2)	514

```
Total params: 9,860,834 (37.62 MB)
Trainable params: 9,858,402 (37.61 MB)
Non-trainable params: 2,432 (9.50 KB)
```

```
  self._warn_if_super_not_called()
50/50 ━━━━━━━━━━━━━━━━━━━━ 99s 2s/step - accuracy: 0.5635 - loss: 2.9640 - val_accuracy: 0.5000 - val_loss: 3.2761 - learning_rate: 0.0010
Epoch 2/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 114s 2s/step - accuracy: 0.5940 - loss: 2.8738 - val_accuracy: 0.5000 - val_loss: 3.1602 - learning_rate: 0.0010
Epoch 3/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.6365 - loss: 2.7011 - val_accuracy: 0.4950 - val_loss: 3.0920 - learning_rate: 0.0010
Epoch 4/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.6111 - loss: 2.7285 - val_accuracy: 0.5000 - val_loss: 4.2332 - learning_rate: 0.0010
Epoch 5/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.6674 - loss: 2.5664 - val_accuracy: 0.5000 - val_loss: 3.3000 - learning_rate: 0.0010
Epoch 6/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 75s 2s/step - accuracy: 0.6569 - loss: 2.4482 - val_accuracy: 0.5000 - val_loss: 2.5898 - learning_rate: 0.0010
Epoch 7/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.6548 - loss: 2.3217 - val_accuracy: 0.5000 - val_loss: 2.4240 - learning_rate: 0.0010
Epoch 8/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.6834 - loss: 2.1741 - val_accuracy: 0.4850 - val_loss: 2.2755 - learning_rate: 0.0010
Epoch 9/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 74s 1s/step - accuracy: 0.7000 - loss: 2.0636 - val_accuracy: 0.5025 - val_loss: 2.2873 - learning_rate: 0.0010
Epoch 10/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 74s 1s/step - accuracy: 0.6621 - loss: 2.0021 - val_accuracy: 0.4775 - val_loss: 2.2467 - learning_rate: 0.0010
Epoch 11/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 687s 14s/step - accuracy: 0.6864 - loss: 1.8879 - val_accuracy: 0.4900 - val_loss: 2.2926 - learning_rate: 0.0010
Epoch 12/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.7302 - loss: 1.7577 - val_accuracy: 0.4550 - val_loss: 2.0775 - learning_rate: 0.0010
Epoch 13/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 98s 2s/step - accuracy: 0.7328 - loss: 1.6830 - val_accuracy: 0.5675 - val_loss: 1.8628 - learning_rate: 0.0010
Epoch 14/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.7102 - loss: 1.6055 - val_accuracy: 0.5925 - val_loss: 1.7446 - learning_rate: 0.0010
Epoch 15/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 113s 2s/step - accuracy: 0.7011 - loss: 1.5622 - val_accuracy: 0.6500 - val_loss: 1.5797 - learning_rate: 0.0010
Epoch 16/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 88s 2s/step - accuracy: 0.6937 - loss: 1.5012 - val_accuracy: 0.7250 - val_loss: 1.4274 - learning_rate: 0.0010
Epoch 17/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 74s 1s/step - accuracy: 0.7288 - loss: 1.4419 - val_accuracy: 0.6875 - val_loss: 1.4968 - learning_rate: 0.0010
Epoch 18/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.7401 - loss: 1.3904 - val_accuracy: 0.7200 - val_loss: 1.3789 - learning_rate: 0.0010
Epoch 19/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.7413 - loss: 1.3428 - val_accuracy: 0.6850 - val_loss: 1.4188 - learning_rate: 0.0010
Epoch 20/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.7478 - loss: 1.2932 - val_accuracy: 0.7175 - val_loss: 1.3094 - learning_rate: 0.0010
Epoch 21/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.7030 - loss: 1.3013 - val_accuracy: 0.6850 - val_loss: 1.3595 - learning_rate: 0.0010
Epoch 22/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 88s 2s/step - accuracy: 0.7566 - loss: 1.2127 - val_accuracy: 0.6675 - val_loss: 1.3869 - learning_rate: 0.0010
Epoch 23/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.7455 - loss: 1.2243 - val_accuracy: 0.7050 - val_loss: 1.2881 - learning_rate: 0.0010
Epoch 24/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7408 - loss: 1.1683 - val_accuracy: 0.6475 - val_loss: 1.2995 - learning_rate: 0.0010
Epoch 25/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7335 - loss: 1.1572 - val_accuracy: 0.7050 - val_loss: 1.2406 - learning_rate: 0.0010
Epoch 26/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.7279 - loss: 1.1481 - val_accuracy: 0.6875 - val_loss: 1.1847 - learning_rate: 0.0010
Epoch 27/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 127s 3s/step - accuracy: 0.7569 - loss: 1.0997 - val_accuracy: 0.6450 - val_loss: 1.3365 - learning_rate: 0.0010
Epoch 28/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 99s 2s/step - accuracy: 0.7660 - loss: 1.0694 - val_accuracy: 0.6425 - val_loss: 1.3349 - learning_rate: 0.0010
Epoch 29/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 88s 2s/step - accuracy: 0.7603 - loss: 1.0586 - val_accuracy: 0.6225 - val_loss: 1.3005 - learning_rate: 0.0010
Epoch 30/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.7468 - loss: 1.0754 - val_accuracy: 0.6950 - val_loss: 1.1172 - learning_rate: 0.0010
Epoch 31/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.7506 - loss: 1.0280 - val_accuracy: 0.6525 - val_loss: 1.4480 - learning_rate: 0.0010
Epoch 32/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - accuracy: 0.7055 - loss: 1.0742 - val_accuracy: 0.7125 - val_loss: 1.1394 - learning_rate: 0.0010
Epoch 33/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 100s 2s/step - accuracy: 0.7412 - loss: 1.0108 - val_accuracy: 0.5850 - val_loss: 1.6478 - learning_rate: 0.0010
Epoch 34/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 623s 13s/step - accuracy: 0.7605 - loss: 1.0028 - val_accuracy: 0.6425 - val_loss: 1.4830 - learning_rate: 0.0010
Epoch 35/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 65s 1s/step - accuracy: 0.7662 - loss: 0.9931 - val_accuracy: 0.7050 - val_loss: 1.1035 - learning_rate: 0.0010
Epoch 36/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 67s 1s/step - accuracy: 0.7575 - loss: 0.9693 - val_accuracy: 0.6750 - val_loss: 1.2539 - learning_rate: 0.0010
Epoch 37/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.7486 - loss: 0.9850 - val_accuracy: 0.7500 - val_loss: 0.9797 - learning_rate: 0.0010
Epoch 38/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7453 - loss: 0.9856 - val_accuracy: 0.6375 - val_loss: 1.8154 - learning_rate: 0.0010
Epoch 39/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7326 - loss: 0.9781 - val_accuracy: 0.6400 - val_loss: 1.1893 - learning_rate: 0.0010
Epoch 40/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.7485 - loss: 0.9342 - val_accuracy: 0.7325 - val_loss: 0.9187 - learning_rate: 0.0010
Epoch 41/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 61s 1s/step - accuracy: 0.7485 - loss: 0.9479 - val_accuracy: 0.7100 - val_loss: 1.0514 - learning_rate: 0.0010
Epoch 42/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.7861 - loss: 0.9222 - val_accuracy: 0.7450 - val_loss: 0.9481 - learning_rate: 0.0010
Epoch 43/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 61s 1s/step - accuracy: 0.7626 - loss: 0.9306 - val_accuracy: 0.6850 - val_loss: 1.1752 - learning_rate: 0.0010
Epoch 44/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 687s 14s/step - accuracy: 0.7557 - loss: 0.9270 - val_accuracy: 0.7300 - val_loss: 1.0770 - learning_rate: 0.0010
Epoch 45/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 42s 848ms/step - accuracy: 0.7581 - loss: 0.9462 - val_accuracy: 0.7400 - val_loss: 0.9736 - learning_rate: 0.0010
Epoch 46/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 41s 829ms/step - accuracy: 0.7834 - loss: 0.8896 - val_accuracy: 0.7475 - val_loss: 0.9248 - learning_rate: 1.0000e-04
Epoch 47/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 41s 821ms/step - accuracy: 0.8088 - loss: 0.8438 - val_accuracy: 0.7475 - val_loss: 0.9093 - learning_rate: 1.0000e-04
Epoch 48/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 42s 836ms/step - accuracy: 0.7834 - loss: 0.8511 - val_accuracy: 0.7550 - val_loss: 0.9089 - learning_rate: 1.0000e-04
Epoch 49/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 42s 846ms/step - accuracy: 0.8042 - loss: 0.8016 - val_accuracy: 0.7475 - val_loss: 0.9027 - learning_rate: 1.0000e-04
Epoch 50/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 42s 833ms/step - accuracy: 0.8018 - loss: 0.8134 - val_accuracy: 0.7500 - val_loss: 0.8681 - learning_rate: 1.0000e-04
```

Classification Report:

	precision	recall	f1-score	support
Female	0.75	0.76	0.75	200
Male	0.76	0.74	0.75	200
accuracy			0.75	400
macro avg	0.75	0.75	0.75	400
weighted avg	0.75	0.75	0.75	400