

**University of
South Wales**
Prifysgol
De Cymru

Module Title: Deep Learning
Student Name: Salma Javid
Module Leader/Tutor: Dr. Mabrouka Abuhmida
Assessment Type: Practical Assessment
Module Code: CS4S772

Predicting Age from Image — A Deep Learning Approach
[Web Deployment with Anvil's Interface]

|
|
|
|
|
|
|
|
|
|

Student Name: Salma Javid
Student ID: 30107961

Table of Contents

| | |
|--|-----------|
| Abstract | 3 |
| 1. Introduction | 3 |
| – Dataset Description: Insights & Revelations | 3 |
| – Inspecting the Image Folder – The Statistics | 3 |
| – Checking the CSV File | 4 |
| – Data Segmentation | 4 |
| 2. Data Preparation | 4 |
| 3. Data Visualization | 4 |
| – Plotting a Pie Chart to Check Distribution of the Class feature | 4 |
| – Displaying the Dataset [Before Augmentation] | 5 |
| 4. Data Preprocessing | 5 |
| – Label encoding | 5 |
| – Standardizing Image Format | 5 |
| – Data Splitting: For Model Training & Testing | 6 |
| – Dataset Resize | 6 |
| – Normalization or Standardization | 6 |
| – Data Augmentation | 7 |
| – Displaying the Dataset [After Augmentation] | 7 |
| 5. Building a Deep Learning Model: Convolutional Neural Networks (CNNs) | 7 |
| – Test 1: Training CNN Model on Unaugmented Dataset | 7 |
| 6. Data Augmentation | 8 |
| – Test 2: Convolutional Neural Network (CNN) on Augmented Dataset | 8 |
| – Test 3: CNN on Augmented Dataset with Increased Epochs | 8 |
| 7. Converting all images to GrayScale | 8 |
| – Test 4: Testing CNN Model on Grayscale Images | 8 |
| Experimenting with Hyperparameter Tuning... #1 | 9 |
| – Test 5: CNN Hypertuned Model | 9 |
| Experimenting with Hyperparameter Tuning... #2 | 9 |
| – Test 6: CNN Hypertuned Model | 9 |
| Experimenting with Hyperparameter Tuning... #3 | 9 |
| – Test 7: CNN Hypertuned Model | 10 |
| 8. Result Visualization | 11 |
| – Confusion Matrix | 11 |
| – Visualizing Scores | 11 |
| – Comparing All CNN Models | 12 |
| 9. Testing My Model [Making Predictions] | 12 |
| 10. Literature Review | 13 |
| 11. Discussion: Critical Analysis of the Model's Performance, Solutions & Limitations | 13 |
| 12. Anvil Web Deployment Process | 15 |
| – SJK Age Prediction App | 15 |
| – Future GUI Plans | 16 |
| References | 16 |

Abstract

This report presents a detailed account of the development, findings, and results of an age prediction model, emphasizing its deployment through Anvil's web interface. The study utilized the provided dataset for training and testing the model.

The report begins with an overview of the project objectives and an in-depth analysis of the dataset characteristics. The methodology section outlines the steps taken to preprocess the data, visualize it, augment it, and split it for training and testing the predictive model. Special attention is given to the model architecture, hyperparameter tuning, and the validation process to ensure robust performance.

I have also included a comprehensive understanding of the model's training and development. The findings section showcases the model's accuracy on unaugmented dataset, augmented dataset and hypertuned model for both training and test sets. Additionally, insights into the model's prediction performance is also presented, shedding light on potential variations in predictive accuracy.

The model's deployment through Anvil's web interface is discussed next, highlighting the accessibility, and overall efficiency of the platform in delivering predictions to end-users. The report explores potential challenges encountered during deployment, outlines strategies employed to overcome them and presents future GUI plans for it.

In conclusion, this report offers a thorough exploration of the age prediction model's development, findings, and results, emphasizing its seamless deployment through Anvil's web interface. The comprehensive analysis provided here serves as a valuable resource for understanding the model's capabilities, limitations, and real-world application potential.

1. Introduction

Problem Statement: Develop a deep learning model that predicts a person's age from an RGB image, and deploy it using Anvil's web interface.

– Dataset Description: Insights & Revelations

– Inspecting the Image Folder – The Statistics

The dataset schema is images of Indian cinema actors and actresses belonging to Bollywood, Tollywood, etc. Listed below are statistics of the dataset:

- Image type includes the vintage cinema stars and the current ones as well.
- Total count of images: 19,906
- Color Diversity in images: Grayscale and color
- Dimensions: Range from postage stamp size (as small as 8 x 11) to poster size (as large as 543 x 616)
- Additional Information: Contains a few duplicates. [eg: ID: 19605 and 20597 are similar]
- Classes (refers to the age group): 3 – Young, Middle, Old.
- Image format: .jpg

– Analyzing the Dataset Details

- This is a labelled dataset with all images assigned an ID as their filename and has a corresponding class in the .CSV file.
- The images show both genders, males and females, belonging to all age groups (young, middle and old).
- Expression/Emotion-wise: The images show many expressions/emotions including happiness, sadness, anger, anxiety, surprise, etc.
- Action-wise: Few people in the images are winking while others are smiling, sleeping, weeping, crying, etc.
- Additional aspects of images include wearing of spectacles, men with moustaches, women with earrings or nosepin, bindi, etc.

– Checking the CSV File

- CSV file has only two columns namely ID and Class (representing age category)
 - Count of rows/records: 19,907 (including headers)
 - Count of Columns: 02 [ID, Class]
 - Unique Classes: 'MIDDLE' 'YOUNG' 'OLD'
- Please Note: The IDs and the Class columns are set in random order. There is no certain order.

– Data Segmentation

- Middle: 10804
- Young: 6706
- Old: 2396

– Testing Two Sorting Scenarios – My Observations

Trying to reset the CSV data... arranging it in an order:

1. Let's sort the IDs in ascending order (checking if it arranges the age in ascending order too).
Result: No certain order found in ages. It is still a mix of young, middle, and old age class.
2. Sort the age (Class) in alphabetical order.
Result: No particular order in ID listings found.

Conclusion: The IDs are not in any continuous order, but skip from 1 to 100 and then 10,000. Either the data is manipulated or we are provided only selected records from a large database. On the bright side, the labels corresponding to images are correctly classed. ***Phew!***

2. Data Preparation

– Importing Required Libraries

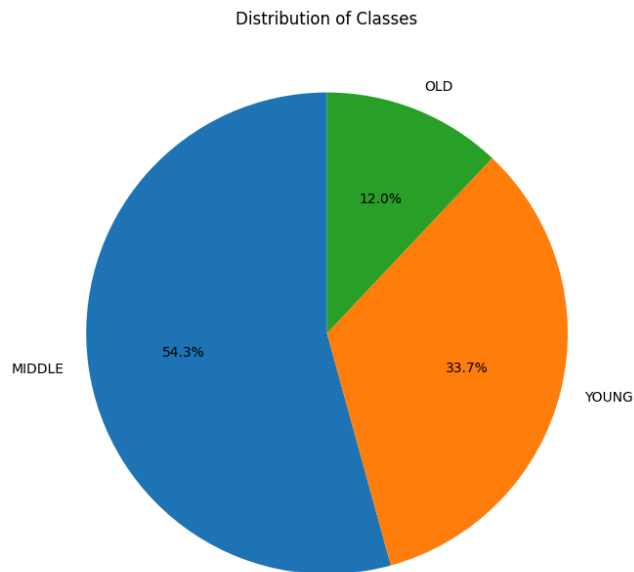
**please check the code provided for details on this, thank you.*

– Extracting Zipped Files

Load the CSV File into a Pandas DataFrame to associate image IDs with their corresponding classes.

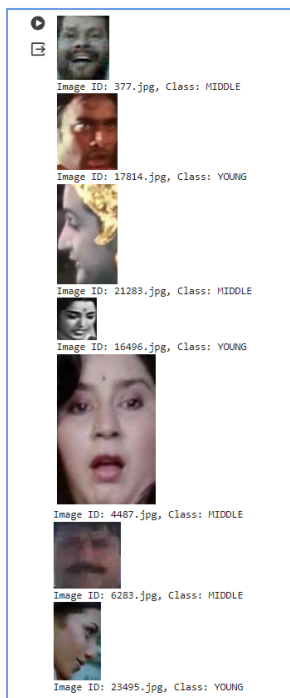
3. Data Visualization

— Plotting a Pie Chart to Check Distribution of the Class feature



A Brief Explanation of the Pie Chart: The Middle age group (or class) represented in **Blue** is the largest group with approximately 54.3% of the dataset. The Young group represented in **Orange** color makes up around 33.7% of the dataset. While the oldies (represented in **Green**) are the smallest set, accounting for only 12% of the dataset.

— Displaying the Dataset [Before Augmentation]



Result: Each image is of a different size & scale.

4. Data Preprocessing

An analysis of the dataset reveals that it has:

- No missing data
- No outliers as all 3 classes are predefined

— Label encoding

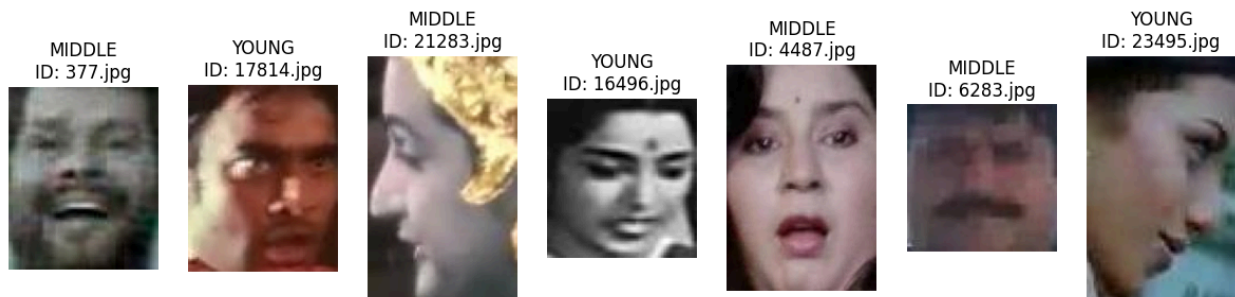
Let's convert the categorical 'Class' feature to numerical through label encoding method.

Result:

- Middle is encoded as 0
- Young is encoded as 2
- Old is encoded as 1

— Standardizing Image Format

Converting Mixed RGB and Grayscale Images to RGB in the Dataset. The given image dataset is a mix of RGB and Grayscale. Let's convert all of them to RGB... that's the ask!



— Data Splitting: For Model Training & Testing

I'm splitting my dataset into **70:30 ratio** training and testing sets to evaluate the model's performance.

- 70% training and 30% testing set
- Number of training images: 13934
- Number of testing images: 5972

Distribution of classes in the training set

- Class 0: 7563 images
- Class 1: 1677 images
- Class 2: 4694 images

Distribution of classes in the testing set

- Class 0: 3241 images
- Class 1: 719 images
- Class 2: 2012 images

— Dataset Resize

Refers to adjusting the visual size of images. I will apply a single physical dimension, a target size of width 64 x 64 height [square images with powers of 2 dimensions] to all the images in the dataset.

Reasons for resizing and choosing the target size:

- **Consistency:** Resizing all images to a consistent size ensures uniform input dimensions for the neural network.
- **Computational Resources:** I have intentionally chosen a smaller size to decrease computational expenses as larger images require more computational resources, both in terms of memory and processing power.
- **Quicker Results:** Small size will also get the results quickly.
- **Training Time:** Larger images need longer training times. Smaller images often result in faster training times, allowing for quicker iterations and experimentation with different models and hyperparameters. As I'm working with time constraints, a smaller size is my choice.
- **Experimentation:** I may need to experiment with different sizes – technically depends on model performance evaluation. Ultimately, the choice of image size is a trade-off between computational efficiency, model requirements, and the characteristics of the dataset.

– Normalization or Standardization

It refers to rescaling the pixel values of images to a range between 0 and 1 to facilitate training.

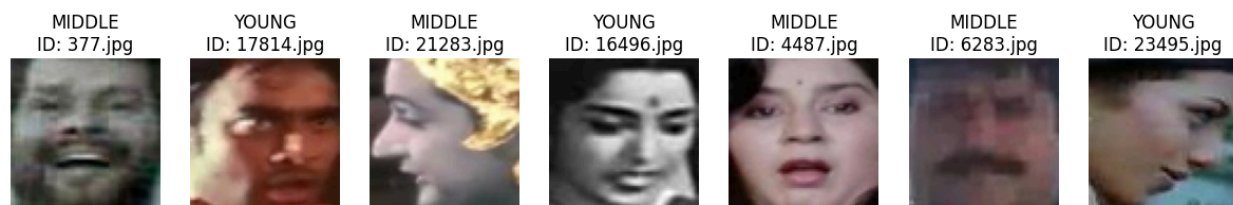
Reasons for rescaling are explained below:

- To ensure that all the image pixel values have a consistent scale. It is important for machine learning algorithms.
- Rescaling is used when input images have pixel values in varying scales. They need to be on a similar scale to prevent certain features from dominating others during model training.

– Data Augmentation

I am using data augmentation technique – `ImageDataGenerator` for data augmentation, specifying various transformations like rotation, shifting (height and width), shearing, zooming, and flipping (horizontally and vertically) along with the fill mode set to 'nearest' to generate additional training samples, improving the model's robustness and generalization.

– Displaying the Dataset [After Augmentation]

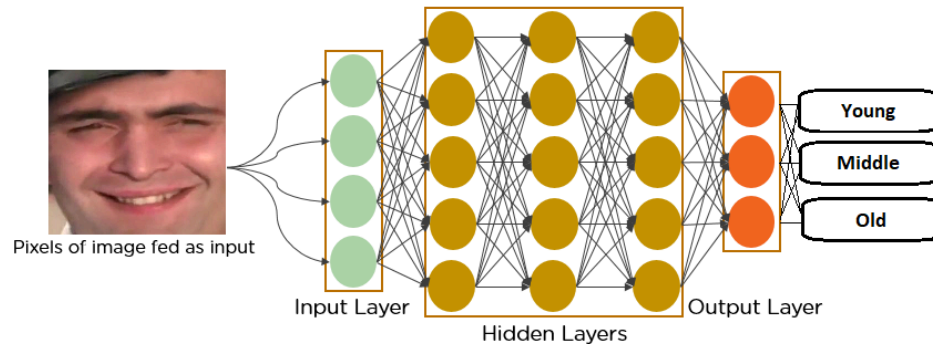


Result: Now all images are of the same size and scale.

5. Building a Deep Learning Model: Convolutional Neural Networks (CNNs)

My chosen Architecture – CNN - Best for image classification tasks. These are effective at extracting spatial features from color images.

Rationale Detailed: I am employing Convolutional Neural Networks [CNNs] because they can automatically extract meaningful spatial features from images and also because it excels at learning hierarchical features like shapes, edges, textures etc. enabling accurate object recognition in images. Both the above reasons position CNNs as powerful tools for accurately predicting age from diverse image datasets.



— Test 1: Training CNN Model on Unaugmented Dataset

Here, the original dataset is initially trained and tested using Convolutional Neural Networks (CNNs) without applying any augmentations. This step helps establish a baseline performance and assess the model's learning ability from the original dataset.

Evaluation Metric: My model's evaluation metric is accuracy. It will provide us with insights into the model performance on both the training & testing datasets.

Model: "sequential_1"— Results after 7 Epochs:

Training Accuracy: 77.57%

Testing Accuracy: 67.78%

Result: Good! But not very impressive!

6. Data Augmentation

Let me now try and optimize my dataset and subsequently model performance through image data augmentation strategies... Let's go ahead with Data Augmentation.

I'm setting the rotation range [90], width and height shift range [0.3], shear_range [0.3], Zoom_range [0.3], horizontal and vertical_flip [True] with fill_mode as 'nearest'.

— Test 2: Convolutional Neural Network (CNN) on Augmented Dataset

Model: "sequential_2"— Results after 7 Epochs:

Training Accuracy: 57.22%

Testing Accuracy: 57.80%

Result: Not very good!

Maybe I need to experiment with more epochs... Let's test with 33.

– Test 3: CNN on Augmented Dataset with Increased Epochs

Model: "sequential_3"— Results after 33 Epochs:

Training Accuracy: 60.26%

Testing Accuracy: 60.63%

Result: Very disappointing! Let's try once again with Grayscale images.

7. Converting all images to GrayScale

(The intention is to increase accuracy)



In this phase, the dataset is preprocessed to convert all RGB images to grayscale. This phase examines the

model's performance when trained on grayscale images and explores the trade-offs between color information and training efficiency.

Rationale for using grayscale images:

- Grayscale images need fewer computational resources
- They potentially speed up the training times
- Take less memory when compared to coloured images
- Grayscale images have a single channel compared to the three channels in RGB images, which can affect the model's ability to extract features.

– Test 4: Testing CNN Model on Grayscale Images

Model: "sequential_4" – Results after 7 Epochs:

Training Accuracy for Grayscale Model: 57.98%

Testing Accuracy for Grayscale Model: 58.49%

Results: Nope! Not working. Results are not very impressive.

Experimenting with Hyperparameter Tuning... #1

Let us now experiment with hyperparameter tuning to optimize the model's performance. The changes I made include:

1. Changing the data augmentation parameters
2. Changing the split size to 80:20 ratio (80% training and testing 20%)
3. Reducing batch size to 16
4. Increasing epochs to 11
5. Adding a dense layer with 128 neurons and applying the ReLU activation function to introduce non-linearity to the model. This layer is often used in hidden layers of neural networks to capture complex patterns in the data.

– Test 5: CNN Hypertuned Model

Model: "sequential_5" – Results after 11 Epochs:

Training Accuracy: 54.28%

Testing Accuracy: 54.27%

Results: That didn't Work! Definitely needs more hyperparameter tuning.

Experimenting with Hyperparameter Tuning... #2

Here I'm defining a custom learning rate schedule called `lr_schedule`. It adjusts the learning rate during the training of a neural network based on the number of epochs. The purpose of adjusting the learning rate during training is to fine-tune the model's optimization process. It allows for a larger learning rate initially for faster convergence, and then a smaller learning rate later to fine-tune the model and avoid overshooting the minimum of the loss function. I'm using this technique to achieve better performance, let's see.

— Test 6: CNN Hypertuned Model

Model: "sequential_6" — Results after 7 Epochs:

Training Accuracy: 54.28%

Testing Accuracy: 54.27%

Results: Didn't Work again! This is definitely a tough one to crack!

Experimenting with Hyperparameter Tuning... #3

1. Changing augmentation techniques again
2. Resetting batch size to 32 again
3. Epochs are set to 33 now
4. Change made in model architecture: The neural network architecture consists of three dense layers: the first layer has 512 neurons with Rectified Linear Unit (ReLU) activation. It has a dropout layer with a rate of 50%, aiming to prevent overfitting by randomly deactivating neurons during training. The second dense layer comprises 256 neurons with ReLU activation, accompanied by another dropout layer with a 30% dropout rate. The final layer, with three neurons and Softmax activation, is responsible for the classification task, producing probabilities for each class. The model is then compiled with the Adam optimizer with a specified learning rate of 0.0001, sparse categorical crossentropy as the loss function suitable for integer-encoded labels, and accuracy as the evaluation metric during training. This configuration aims to strike a balance between model complexity, regularization, and optimization for effective multi-class classification.

— Test 7: CNN Hypertuned Model

Model: "sequential_7" — Results after 33 Epochs:

Training Accuracy: 77.75%

Testing Accuracy: 76.90%

Results: Excellent! It worked well this time!

Detailing Model Architecture:

```

# Defining the CNN model Architecture
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (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(256, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()

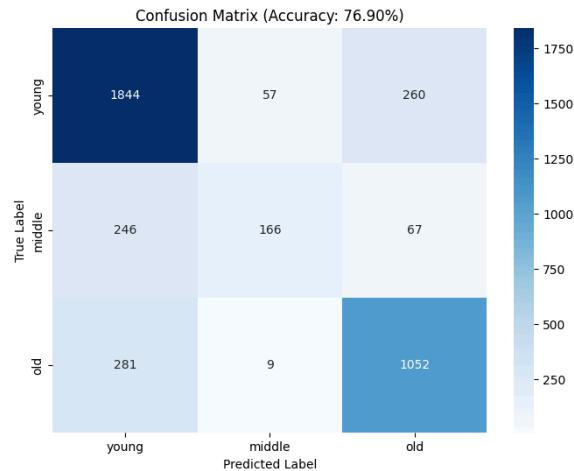
```

Let me explain each line of the model architecture:

- First, I'm initializing a sequential model which is a linear stack of layers.
- Then adding a 2D-convolutional layer with 64 filters (size 3x3), a ReLU activation function along with input shape of (64, 64, 3). This layer helps in extracting features from the input images.
- Then adding a 2D max-pooling layer with a pool size of 2x2. It reduces the spatial dimensions of the representation and retains the most important information.
- Next, adding another 2D-convolutional layer consisting 128 filters of size 3x3 with ReLU activation. This layer further refines the extracted features.
- Then adding another 2D-max-pooling layer of the pool size 2x2.
- Adding a third 2D-convolutional layer of 256 filters of 3x3 size with ReLU activation.
- Adding another 2D-max-pooling layer of a 2x2 pool size.
- Using Flatten() to flatten the input – necessary before passing it to a fully connected layer.
- Next adding another fully connected layer with 512 neurons and ReLU activation.
- Adding a dropout layer with a 0.5 rate to prevent overfitting.
- Adding another fully connected layer with 256 neurons and ReLU activation.
- Adding another dropout layer with a dropout rate of 0.3.
- Adding the output layer with 3 neurons (as mine is a classification task with 3 classes) and a softmax activation function, which is appropriate for multi-class classification problems.
- Next, I'm compiling the model, specifying the optimizer (Adam with a learning rate of 0.0001), the loss function (sparse categorical crossentropy for multi-class classification), and the metric to monitor during training (accuracy).
- Next line displays a summary of the model architecture, including the type of layers, output shape, and the number of parameters in each layer. This is useful for reviewing the model's structure and parameter count.

8. Result Visualization

– Confusion Matrix



Let's build a Confusion Matrix for our Model. In the context of an age prediction model with three age classes (young, middle, and old), the confusion matrix is a valuable tool for evaluating the performance of the model in terms of classification accuracy. It provides a detailed breakdown of the model's predictions and actual labels, helping to identify areas of strengths and weaknesses.

1. True Positives (TP): The diagonal elements of the matrix represent the instances where the model correctly predicted the age class. For example, the element at (1,1) indicates the number of true predictions for the middle age class.
2. False Positives (FP): Represent instances where the model predicted a certain age class, but the true label was different. For example, the element at (2,1) indicates the number of instances where the model incorrectly predicted the middle age class when the true label was young.
3. False Negatives (FN): Represents instances where the model failed to predict a certain age class, and the true label was that class. For example, the element at (1,2) indicates the number of instances where the model failed to predict the middle age class when the true label was middle.
4. True Negatives (TN): All other elements not in the specified row or column are true negatives, representing instances where the model correctly predicted a class other than the one specified.

– Classification Report

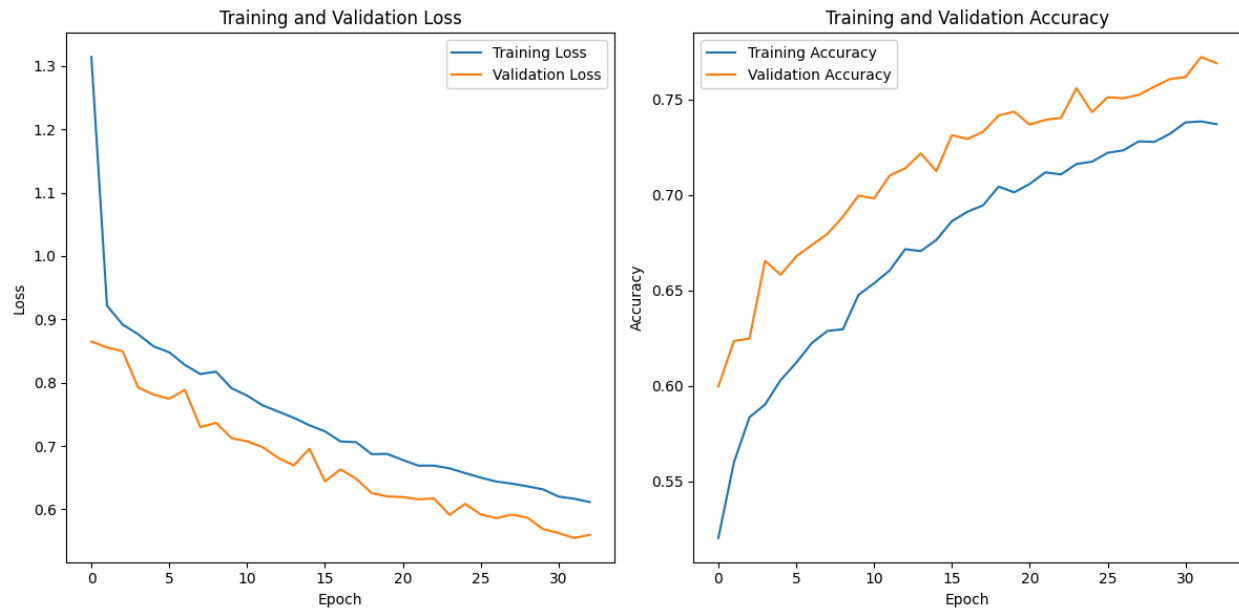
The classification report provides a detailed summary (comprehensive overview) of the performance metrics (in terms of precision, recall, F1-score, accuracy, etc.) for each class in a multiclass classification problem. Let's break down the key components:

| Classification Report: | | | | |
|------------------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| young | 0.78 | 0.85 | 0.81 | 2161 |
| middle | 0.72 | 0.35 | 0.47 | 479 |
| old | 0.76 | 0.78 | 0.77 | 1342 |
| accuracy | | | 0.77 | 3982 |
| macro avg | 0.75 | 0.66 | 0.68 | 3982 |
| weighted avg | 0.77 | 0.77 | 0.76 | 3982 |

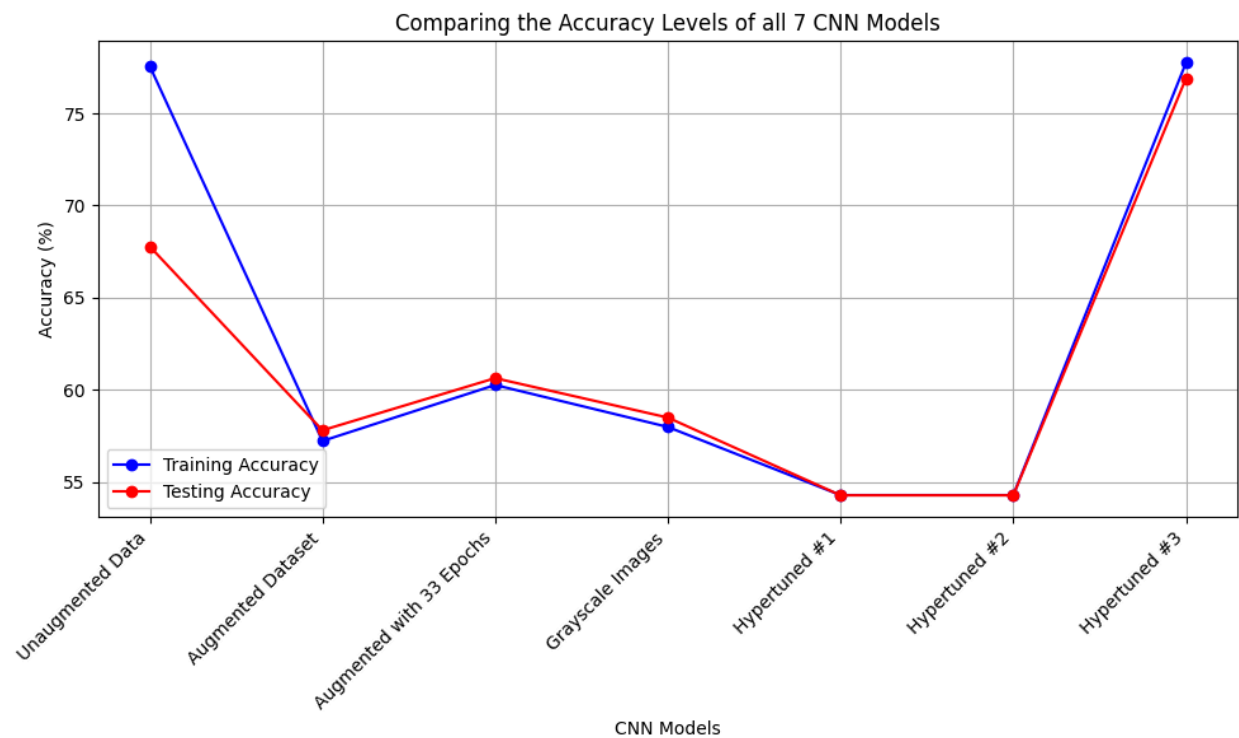
1. Precision for the young class is 0.78, indicating the model predictions are correct 78% of the time. Similarly, precision for the middle class is 72% and old is 76%.
2. Recall for the young class is 85%. Middle is only 35% while for the old, it is 78%.
3. F1-score for the "young" class is 81%, Middle scored 47% and old is 77%.
4. Overall accuracy of the model across all classes is 77%.

— Visualizing Scores

Visualizing the training and testing loss, training and testing accuracy results through a line graph





— Comparing All CNN Models



Result: CNN model with unaugmented dataset and the hypertuned #3 versions scored well.

9. Testing My Model [Making Predictions]

| Way – 1: This program asks for an input image from the user, and then predicts its age/class and ID. | Way – 2: In this we specify an image mentioning its path, the program then predicts its age/class and ID. |
|---|---|
| <p>ID: 2439 Predicted Class: MIDDLE</p>  | <p>ID: 673 Predicted Class: MIDDLE</p>  |
| Result: Correct! | Result: Correct! |

10. Literature Review

I came across a post at medium.com where a blogger suggests building a customer profiling system for a store, enabling it to predict age, gender and emotion of people entering their store. It uses CNN based deep learning approach for model building. For this purpose the blogger picks up a publicly available labelled image dataset for building his Age detection model.

As in any other deep learning program, s/he begins with data augmentation techniques and then converts all his images to GrayScale. After just these two processes, he begins with a CNN based deep learning approach for model building by splitting it into 70-30 training and test sets. S/he defines age classes and uses 60 epochs for model fitting. Post this he trains his model and successfully achieves 90% accuracy... that was pretty quick I think. S/he visualizes the results with a confusion matrix with quite impressive results. Later s/he talks about putting this model to proper use by using this in a business application as mentioned initially.

My Review: I have two points of view for this.

First, the positive, I'd say the blogger could go ahead and build a model which not only detects the age, gender and emotion of the customer, but should detect additional characteristics such as their complexion, height, approx. weight, body language, dressing style and so forth and suggest suitable clothes... that's me assuming it's a cloth retail store selling the latest designer clothes. The model should be deployed on an App which could be installed as an interactive screen mirror with all options across it to use and decide. It should suggest suitable colours, patterns, sizes, materials, etc. as well to the customer. There could be more options to choose from like style, occasion, mood, budget, etc. This would make their shopping experience completely autonomous, interesting and easy.

The second view that I have is that it wouldn't be easy to get a 90% accuracy score at the first go in certain cases (as with mine), and definitely not with unlabeled data. The blogger could have kept his/her options open for more possibilities, problems and solutions.

11. Discussion: Critical Analysis of the Model's Performance, Solutions & Limitations

– Critical Analysis of the Model's Performance

It was really challenging yet I had fun (and it was a learning experience) trying to achieve a good accuracy score for my CNN models. It compelled me to experiment with various dataset changes and model configurations. Here is a breakdown of the challenges encountered while training my model on seven distinct CNN models:

1. **Unaugmented Dataset Model: Training Accuracy: 77.57% | Testing Accuracy: 67.78%**
The first model trained on the unaugmented dataset showed promising training accuracy but had a notable drop in testing accuracy, indicating potential overfitting or inadequacy in generalization.
2. **Augmented Dataset Model: Training Accuracy: 57.22% | Testing Accuracy: 57.80%**
Introducing data augmentation aimed to address overfitting issues, but surprisingly, both training & testing accuracies decreased. This raises questions about their impact on the model.
3. **Augmented Dataset with 33 Epochs: Training Accuracy: 60.26% | Testing Accuracy: 60.63%**
In an attempt to improve the augmented model's performance, I trained my model for 33 epochs. While there was a marginal increase in accuracy, it was still below the desired threshold, prompting further exploration.
4. **Grayscale Images Model: Training Accuracy: 57.98% | Testing Accuracy: 58.49%**
Switching to grayscale images was aimed to increase accuracy. However, the results showed a modest (almost negligible) improvement.
5. **Hypertuned Model #1: Training Accuracy: 54.28% | Testing Accuracy: 54.27%**
Experimenting with basic hyperparameter tuning resulted in a model with accuracy way below expectations, revealing the intricate balance required in tuning parameters.
6. **Hypertuned Model #2: Training Accuracy: 54.28% | Testing Accuracy: 54.27%**
The second attempt did not yield any significant improvements either, underscoring the complexity of finding the right combination of hyperparameters to achieve the desired results.
7. **Hypertuned Model #3: Training Accuracy: 77.75% | Testing Accuracy: 76.90%**
Finally, the third hyperparameter-tuned model demonstrated substantial improvement, meeting the target accuracy. This success emphasized the iterative nature of model development, with each attempt providing valuable insights into the dataset characteristics and model architecture.

To conclude, the process highlighted the importance of a systematic approach, continuous refinement, and a deep understanding of the underlying factors influencing model performance.

– Potential Limitations:

Despite the achieved success in the third hyperparameter-tuned model, it is essential to acknowledge the limitations encountered throughout the process. The challenges with the augmented dataset models underscore the complexity of finding an optimal balance in data augmentation techniques, which might require more sophisticated approaches. The modest improvements seen with grayscale images suggest that simplifying the model may not always lead to substantial gains, emphasizing the need for a nuanced understanding of the dataset and problem domain.

– Proposed Solutions and Future Directions:

Building on the insights gained from the critical analysis and potential limitations, my proposed solution would be to further explore advanced data augmentation techniques to enhance the model's ability to generalize. Or another would be to consider a hybrid dataset that incorporates both color and grayscale images (the model did score well on unaugmented dataset). Additionally, continuous monitoring and adaptation of best hyperparameter tuning strategies can help increase accuracies.

– Conclusion:

This project has given me invaluable insights into image classification using CNN. The challenges I faced taught me a lot, and from where I am, I see a vast space with unlimited possibilities. Each round of model refinement has deepened my understanding of the dataset intricacies and the delicate interplay of model parameters. Rigorous analysis and critical thinking have proven to be indispensable tools for honing model performance. The challenges encountered, especially in the domains of data augmentation and grayscale transformations, have underscored the nuanced nature of model development, urging the adoption of adaptive strategies.

There's huge potential for improvements with more advanced data augmentation techniques, and a continuous adaptation of hyperparameter tuning strategies – I have a roadmap charted for continued exploration. It is exciting to imagine the various fields where I can implement my knowledge for further hobby projects on real-world applications.

12. Anvil Web Deployment Process

Deployment: Explanation of deploying the model using Anvil, including any challenges and solutions.


Age Prediction App GUI Description: It begins with a welcome message. The second section introduces itself as an Age Detection App, requesting the user to upload an image using the 'file loader button' displayed below. Once the user uploads an image, it is displayed and an age prediction is made below it. The App also asks the user's input if it was correct in its prediction (**or not!**)... *this particular section doesn't work right now. I plan to include it in my "Future GUI Plans".*


– SJK Age Prediction App

<https://sjk-age-prediction-app.anvil.app>

Hello there... Welcome!

I am an age-detection App. Please upload an image to test.



 1 file selected

My Prediction: Young

Was I correct?

Yes, you are correct!

No, that was a wrong answer!

– Future GUI Plans

Using the same dataset I intend to make an app that not only predicts the age of a person but also additional characteristics such as genre, gender, skin colour, emotions, expressions, region, category, etc. The App will also be able to collect user feedback if it was correct in judging or not.

– QR Code to Install My Age Detection App



Please scan this QR Code to download the App

References

1. Agarwal, P. (2020) *Age Detection using Facial Images: traditional Machine Learning vs. Deep Learning*. Available at: <https://towardsdatascience.com/age-detection-using-facial-images-traditional-machine-learning-vs-deep-learning-2437b2feeab2> (Accessed: 1 February 2024).
2. Al, S. (2022) 'Age Detection Model using CNN – a complete guide'. *Medium* 23 August. Available at: <https://medium.com/@skillcate/age-detection-model-using-cnn-a-complete-guide-7b10ad717c60> (Accessed: 1 February 2024).
3. Chen, S., Zhang, C., Dong, M., Le, J. and Rao, M. (2017) 'Using Ranking-CNN for Age Estimation'. in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, pp. 742-751. Available at: <http://ieeexplore.ieee.org/document/8099569/> (Accessed: 1 February 2024).
4. Sheoran, V., Joshi, S. and Bhayani, T.R. (2021) 'Age and Gender Prediction Using Deep CNNs and Transfer Learning'. in Singh, S. K., Roy, P., Raman, B., and Nagabhushan, P. (eds.) *Computer Vision and Image Processing*. Communications in Computer and Information Science. Singapore: Springer Singapore, pp. 293-304. Available at: https://link.springer.com/10.1007/978-981-16-1092-9_25 (Accessed: 1 February 2024).