**Asian Facial Age Classification**

1. **Abstract**

Human age prediction has been one of the hardest tasks among image based machine learning problems. To determine one’s age through a single human facial image is a complex task, even for humans, let alone artificial intelligence. There are many subtle traits we might want to take into consideration when it comes to facial age, such as wrinkles, hairline, skin color. All of the traits above vary slightly from person to person, making the task even harder for classifiers. Despite the difficulty of the problem, there are many well defined public datasets on the facial age subject, for example UTKFace, MORPH, …etc. Within the past decade, profound researches have made great contribution to the improvement of age detection algorithms. Microsoft’s work in 2017 “How old am I” stands as a great example of how the interesting issue has generated much discussion in the researching field in recent years.

This work narrows down the original facial age prediction problem to a classification problem. I collected facial image of Asian people with age ranging from 20 to 60 years old, and partitioned them into different age groups. The job of the AI classifier is to figure out which age group the testing image belongs. Viewing the age prediction task from another aspect, since existing datasets are composed of mainly western facial images, the models trained on these dataset yields a poor accuracy when detecting facial age of Asians. Therefore, I set the goal of my manually collected dataset to be limited to only Asian faces, training the model to recognize Asian faces.

To complete the task, 2 supervised algorithms and 1 unsupervised method are used. For supervised method, I choose to use a basic logistic regression classifier and a DNN based ResNet-50 model architecture. As for the unsupervised method, I choose to use the K-Mean algorithm to partition training images. Experiments are then carried out to test the different results when data manipulation methods, such as data augmentation, different sized dataset, and PCA Eigen face extraction are used. The result yields an interesting relation between data manipulation and training accuracy, which will be further discussed in following sections.

1. **Dataset**

The dataset collected contains 800 facial images of 160 subjects (public figures). With a labeled age span ranging from 20 to 60 years old, each subject is captured five times, and labeled according to the corresponding age of the subject. Images are stored under directories named with its labeled age, 4 subjects are selected for every age, making the dataset balanced for every age group. The data collection work could be partitioned into 3 main process: age survey, image capture, image resizing.

|  |  |
| --- | --- |
| D:\bosyu\codes\SS24AI-Capstone\facial age detection\dataset\20\Screenshot 2024-03-11 112103.png | D:\bosyu\codes\SS24AI-Capstone\facial age detection\dataset\26\Screenshot 2024-03-11 214125.png |
| 20 | 26 |

1. Age Survey

To find a subject at a certain age, I referenced the Wikipedia page to sort out subjects by age. The choice of subjects mainly falls in but not limited to the categories of celebrities, actors /actresses, politicians, athletes, and Youtubers. I target subjects that are well known to the majority of Taiwanese people, such that there are more facial images of them to choose from.

1. Image Capture

After deciding the subject, I search for their social media platforms such as Instagram, Facebook, Weibo, and download images capturing their faces from different angle and lightning conditions that are uploaded in the past 6 months. These images are then saved under the label of the subjects age, for example, if a subject is born in 1990, its photo will be labeled under the age 34 directory.

1. Image Resize

For facial image classification, we want all input images to be under similar settings. Therefore, that last step of data processing is resizing, cropping subject faces from the original photo, possibly capturing all facial features (eyes, nose, hairline…) while maintaining a 3 : 4 width height racial. Last, all image will be resized and saved as a 120x160 image.

|  |  |
| --- | --- |
|  |  |
| Image Capture | Image Resize |

1. **Method**

In this project I consider three different approaches to the age prediction task. For the two supervised method, I choose logistic regression and ResNet-50 (convolutional neural network). As for unsupervised method, I implemented the K-Means algorithm to partition data samples and produce predictions. In the following section, I will discuss the implementation details of the three algorithms.

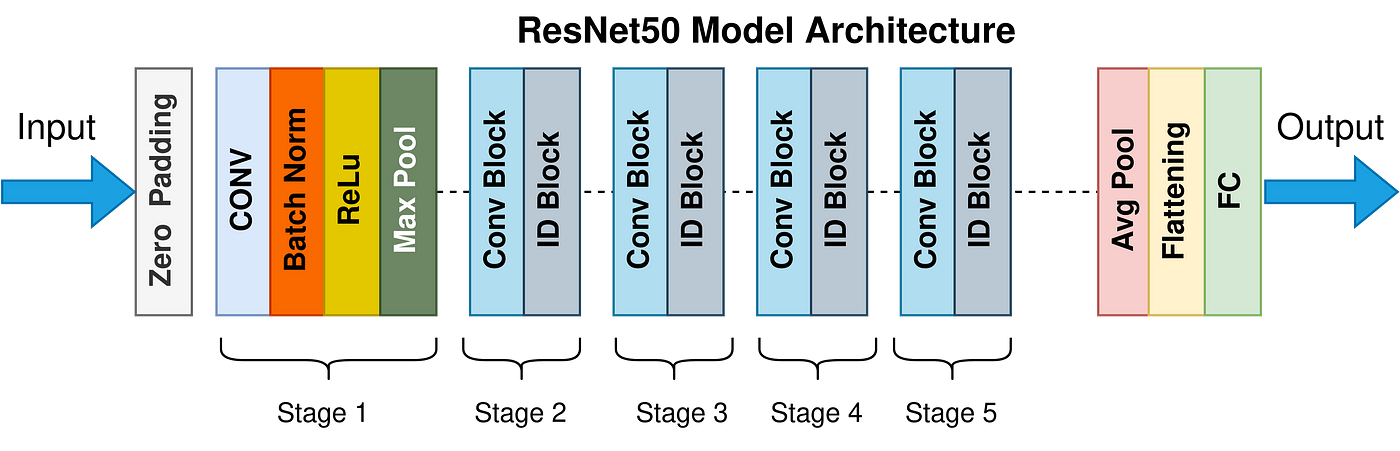
1. Logistic Regression

Logistic Regression is a basic machine learning model that models the odds of one event as a linear combination of features, suitable for classification problems, and for this specific case, the loss function is set to cross-entropy loss to optimize classification accuracy.

For the main model architecture, I imported the Logistic Regression model from Scikit-learn. There is a built in check for convergence on training sample in the imported package, and after try and error testing, the max-iteration of logistic regression is set to 2000 epochs.

1. ResNet-50

Convolution neural network is well known for its outstanding performance when dealing with image classification problems. For facial age classification, I choose to use ResNet-50 as my backbone network. ResNet-50 is recognized for its deep learnable parameters, and residual design to resolve the vanishing gradient issue in deep neural networks. Below is a graph of the architecture of ResNet-50, for implementation simplicity, the ResNet-50 model is adopted from Keras, with pre trained weights from ImageNet. To perform classification, a dense layer is added at the end of the model to produce probability distribution on classes.



1. K-Means

K-means is a popular clustering algorithm used in machine learning and data mining. It's an unsupervised learning algorithm. The primary goal of the K-means algorithm is to partition a dataset into clusters, where each data point belongs to the cluster with the nearest mean.

K-Mean algorithm is also imported from Scikit-learn, to simplify input data vector (originally 120x160 for a single image) PCA feature extraction is applied to the data points before K-Mean clustering, which is also imported from the Scikit-learn library.

1. **Experiment**
2. Performance Comparison

First, we compare the performance of the three different algorithms. Supervised methods are all validated under the condition of 1/10 cross validation, meaning 720 images used for training and 80 for validation. The task is set to classify which decade age span the image falls in, dividing the 20-60 age span dataset into 4 different age bins, each containing 180 training samples.

From the evaluation metrics we can observe that neural network based method has an immense performance gain over the two other traditional machine learning algorithms. Observed from the confusion matrix, it is easier for AI to differentiate facial image of age group 20-29 and age group 50-59. This doesn’t come as a surprise since medical proof states that the aging process starts from the age of 30, and many suffer from menopause around the age of 50, both of these impact facial appearance of humans.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Logistic Regression | | K-Mean | ResNet-50 |
| Accuracy | 33.6% | | 31.8% | 50.5% |
| Confusion Matrix | | | | |
| D:\bosyu\codes\SS24AI-Capstone\facial age detection\results\LR\CM_cv_10.png | | D:\bosyu\codes\SS24AI-Capstone\facial age detection\results\KMean\CM_no_PCA.png | | |
| Logistic Regression | | K-Mean | | |
|  | |
| ResNet-50 | |

1. Unsupervised Clustering Relation with PCA and Data Augmentation

To evaluate the clustering performance of K-Mean algorithm, I calculated the Adjusted Rand Index (ARI) score of the partitions. ARI measures the similarity between two clustering, considering all pairs of samples and counting pairs that are assigned to the same or different clusters in the predicted and true clustering.



Since clustering methods depend fully on feature similarity of input samples, I think it would be interesting to test the feature extraction method PCA to decrease dimensions of input data, and augment samples to see how it affects the clustering result of K-Means.

|  |  |
| --- | --- |
| Method | Accuracy |
| K-Mean (baseline) | 31.8% |
| K-Mean + PCA | 31.5% |
| K-Mean + PCA + Rotate | 24.8% |
| K-Mean + PCA + Flip | 26.3% |
| K-Mean + PCA + Gaussian Noise | 31.5% |

For the output dimension of PCA, I tested through dimensions 1-100 to find the dimension that produce the best accuracy, and found that dimension 31 best fits our condition with an accuracy of 31.5%. From the result, we can see that although the performance after PCA is slightly decreased, the result is still reasonable, considering that its clustering input is decreased from 120 \* 160 to only 31.

To test the relation between data augmentation and clustering, three different kinds of data augmentation methods are used. All input samples are augmented with an augmentation duplicate. It can be seen that flipping and rotating doesn’t help the performance of clustering, and the Gaussian Blur doesn’t effect the result, which may be due to the PCA dimension reduction after the augmentation.

1. Decreased Training Data

To evaluate how the amount of training data effects training results. I alter the cross validation batch size parameter, originally the dataset is partitioned into 10 batches and validated 10 times, each time training with 9/10 of its original data. By changing the batch number to 5, 2 the amount of training data will decrease.

The same cross validation change is tested on ResNet-50 model, and to analyze how the amount of data effects other hyper parameters, ResNet-50 models are tested when trained on 30 and 50 epochs. This shows how the amount of data may effect the time of training convergence.

From the result, we can see when the amount of data increase and trained on more epochs, the model does indeed yield a better performance. Although the accuracy gain may be small due to the fact that this manually collected dataset may be biased and lack sufficient amount of samples, we can still observe the trend of performance gain with the increasing amount of data. I expect the model to produce better performance when dealing with larger datasets.

1. Age Group partition

In this experiment, the age prediction problem is simplified from a regression problem for predicting an accurate age of human to classifying which age group a person belongs. The amount of age groups is strongly related to the performance of models. I test models’ performance on 20 years old age group, 10 years age group, 5 years age group, and 2 years age group.

For age groups within 2 years’ gap, the model would be better supervised under MAE loss than cross-entropy loss. Since we train our classifier on cross-entropy criteria, it is not possible for them to distinguish “how close a guess might be” when its prediction falls near the true label, therefore, making it harder to train a reasonable model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group Length | 20 | 10 | 5 | 2 |
| Accuracy | 70.0% | 50.5% | 26.3% | 21.3% |

|  |  |
| --- | --- |
| Confusion Matrix | |
|  | CM_cv10_epoch50 |
| 20 years age group | 10 years age group |
|  |  |
| 5 years age group | 2 years age group |

1. **Discussion**
2. Data Overfitting

Through observing the ResNet-50 training process, it can be seen that the accuracy on training samples after training for 10 epochs, could raise to around 90%. However, the performance of the classifier used on testing samples only shows accuracy around 50%. This showcase the issue of training data overfitting, and the learning potential of the model may not be fully released due to the biased and insufficient amount of training samples. I expect to see performance gain if trained on a larger and more general facial age dataset.

1. Classification or Regression

Due to implementation simplicity, the age prediction problem was narrowed down to a classification problem. However, this neglects the relation between close age groups, for example the algorithm will be punished with the same amount of loss when predicting a 49 years old person as the 50-59 age group or predicting it as the 20-29 age group. This slows the convergence of training, and damage the performance when the predictions are evaluated under different criteria such as MAE between prediction and true label.

1. Repeated Subject in Training and Testing Set

As described in the dataset section, each subject will be sampled 5 times, and categorized under the same label. Although I tried my best to choose 5 images as different to each other as possible, under different lighting, angles, appearance, there is still the possibility of models recognizing the subject from the learned training set, instead of actually making a prediction under general ageing traits of the facial image. Therefore, when the classifier face a new subject not included in the training set, it may produce poor classification results. A better approach may be collecting testing subjects differently from training subjects, or collecting ample amount of subjects, so that it would not be feasible for the model to recognize individual subjects.

1. **Reference**

ChatGPT: <https://chat.openai.com/>

Facebook: <https://www.facebook.com/>

Instagram: <https://www.instagram.com/>

Kaggle: <https://www.kaggle.com/>

Keras: <https://keras.io/>

Scikit-learn: <https://scikit-learn.org/>

The Annotated ResNet-50: <https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758>

Wiebo: <https://weibo.com/>

Wikipedia: <https://zh.wikipedia.org/>

1. **Appendix**
2. **LogisticRegression.py**

import numpy as np

import cv2

import os

from sklearn.model\_selection import cross\_val\_predict

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from matplotlib.pyplot import show

# Function to preprocess labels into 10-year bins

def preprocess\_labels(age):

    label = int((age / 10) - 2)  # Group ages into 10-year bins

    return label

# Function to load images from directory

def load\_images\_from\_dir(directory):

    images = []

    labels = []

    for folder in os.listdir(directory):

        age = int(folder)

        print(f"Loading images from {folder}...")

        label = preprocess\_labels(age)

        folder\_path = os.path.join(directory, folder)

        for filename in os.listdir(folder\_path):

            if filename.endswith('.jpg') or filename.endswith('.png'):

# Load image with cv2

                img = cv2.imread(os.path.join(folder\_path, filename))

# Resize image to 120 x 160 with cv2

                img = cv2.resize(img, (120, 160))

# Flatten image to 1D array

                img\_array = np.array(img).flatten()

                images.append(img\_array)

                labels.append(label)  # Use preprocessed label

    return np.array(images), np.array(labels)

# Load images from your dataset directory

dataset\_dir = 'dataset'

X, y = load\_images\_from\_dir(dataset\_dir)

# Initialize logistic regression model

model = LogisticRegression(max\_iter=2000)

# Perform cross-validation and get predicted labels

y\_pred = cross\_val\_predict(model, X, y, cv=2)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(y, y\_pred)

# Print confusion matrix

print("Confusion Matrix:")

print(conf\_matrix)

# Display confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix)

disp.plot()

show()

# Calculate accuracy from confusion matrix

accuracy = np.trace(conf\_matrix) / np.sum(conf\_matrix)

print("Accuracy:", accuracy)

1. **K-Mean.py**

import numpy as np

import cv2

import os

from sklearn.model\_selection import cross\_val\_predict

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from matplotlib.pyplot import show

from skimage.transform import rotate

from sklearn.metrics import adjusted\_rand\_score

from skimage.util import random\_noise

from skimage.transform import rescale

# Function to preprocess labels into 10-year bins

def preprocess\_labels(age):

    label = int((age / 10) - 2)  # Group ages into 10-year bins

    return label

# Function to load images from directory

def load\_images\_from\_dir(directory, augmentation=False):

    images = []

    labels = []

    for folder in os.listdir(directory):

        age = int(folder)

        print(f"Loading images from {folder}...")

        label = preprocess\_labels(age)

        folder\_path = os.path.join(directory, folder)

        for filename in os.listdir(folder\_path):

            if filename.endswith('.jpg') or filename.endswith('.png'):

# Load image with cv2

                img = cv2.imread(os.path.join(folder\_path, filename))

# Resize image to 120 x 160 with cv2

                img = cv2.resize(img, (120, 160))

                images.append(img)

                labels.append(label)  # Use preprocessed label

                # Data augmentation

                if augmentation:

                    # # Rotate image

                    rotated\_img = rotate(img, angle=np.random.uniform(-15, 15), mode='edge')

                    images.append(rotated\_img)

                    labels.append(label)

                    # Flip image horizontally

                    # flipped\_img = cv2.flip(img, 1)

                    # images.append(flipped\_img)

                    # labels.append(label)

                    # # Add Gaussian noise

                    # noisy\_img = random\_noise(img, var=0.01\*\*2)

                    # noisy\_img = (255\*noisy\_img).astype(np.uint8)

                    # images.append(noisy\_img)

                    # labels.append(label)

                    # # Rescale image

                    # scaled\_img = rescale(img, scale=np.random.uniform(0.8, 1.2), mode='constant')

                    # images.append((scaled\_img \* 255).astype(np.uint8))

                    # labels.append(label)

    return np.array(images), np.array(labels)

# Load images from your dataset directory

dataset\_dir = 'dataset'

X, y = load\_images\_from\_dir(dataset\_dir)

# Flatten the images

X\_flat = X.reshape(X.shape[0], -1)

# Perform PCA

pca = PCA(n\_components=31)  # Adjust the number of components as needed

X\_pca = pca.fit\_transform(X\_flat)

# Initialize KMeans model

kmeans = KMeans(n\_clusters=4, random\_state=4, n\_init=10, max\_iter=300, tol=1e-04)

# Fit KMeans model

kmeans.fit(X\_flat)

# Predict clusters

y\_pred = kmeans.predict(X\_flat)

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(y, y\_pred)

# Print confusion matrix

print("Confusion Matrix:")

print(conf\_matrix)

# Display confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix)

disp.plot()

show()

# Calculate accuracy from confusion matrix

accuracy = np.trace(conf\_matrix) / np.sum(conf\_matrix)

print("Accuracy:", accuracy)

ari\_score = adjusted\_rand\_score(y, y\_pred)

print("Adjusted Rand Index (ARI):", ari\_score)

1. **ResNet.py**

import numpy as np

import cv2

import os

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, accuracy\_score

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from matplotlib.pyplot import show

from skimage.transform import rotate

from skimage.util import random\_noise

from skimage.transform import rescale

# Function to preprocess labels into year bins

def preprocess\_labels(age):

    label = int((age / 20) - 1)  # Group ages into year bins

    return label

# Function to load images from directory

def load\_images\_from\_dir(directory, augmentation=False):

    images = []

    labels = []

    for folder in os.listdir(directory):

        age = int(folder)

        print(f"Loading images from {folder}...")

        label = preprocess\_labels(age)

        folder\_path = os.path.join(directory, folder)

        for filename in os.listdir(folder\_path):

            if filename.endswith('.jpg') or filename.endswith('.png'):

                img = cv2.imread(os.path.join(folder\_path, filename))  # Load image with cv2

                img = cv2.resize(img, (168, 224))  # Resize image to 168 x 224 with cv2

                images.append(img)

                labels.append(label)  # Use preprocessed label

                # Data augmentation

                if augmentation:

                    # Rotate image

                    rotated\_img = rotate(img, angle=np.random.uniform(-15, 15), mode='edge')

                    images.append(rotated\_img)

                    labels.append(label)

                    # Flip image horizontally

                    flipped\_img = cv2.flip(img, 1)

                    images.append(flipped\_img)

                    labels.append(label)

                    # Add Gaussian noise

                    noisy\_img = random\_noise(img, var=0.01\*\*2)

                    noisy\_img = (255\*noisy\_img).astype(np.uint8)

                    images.append(noisy\_img)

                    labels.append(label)

                    # Rescale image

                    scaled\_img = rescale(img, scale=np.random.uniform(0.8, 1.2), mode='constant')

                    images.append((scaled\_img \* 255).astype(np.uint8))

                    labels.append(label)

    return np.array(images), np.array(labels)

# Load images from dataset directory

dataset\_dir = 'dataset'

X, y = load\_images\_from\_dir(dataset\_dir)

# Initialize variables for cross-validation, n\_split is the number of folds

skf = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)

y\_pred = []

y\_true = []

# Perform cross-validation

for train\_index, test\_index in skf.split(X, y):

    print("start iteration: " + str(test\_index[0]))

    X\_train, X\_test = X[train\_index], X[test\_index]

    y\_train, y\_test = y[train\_index], y[test\_index]

    # Load pre-trained ResNet50 model without top layers

    base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=(224, 168, 3))

    # Add Global Average Pooling layer

    x = base\_model.output

    x = GlobalAveragePooling2D()(x)

    # Add Dense layer for classification (adjust the number of units according to your needs)

    predictions = Dense(2, activation='softmax')(x)

    # Combine base model and new layers

    model = Model(inputs=base\_model.input, outputs=predictions)

    # Freeze pre-trained layers

    for layer in base\_model.layers:

        layer.trainable = False

    # Compile model

    model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

    # Train the model

    model.fit(X\_train, y\_train, epochs=30, batch\_size=32, verbose=1)

    # Predict on test set

    y\_pred\_fold = np.argmax(model.predict(X\_test), axis=1)

    y\_pred.extend(y\_pred\_fold)

    y\_true.extend(y\_test)

    # Print accuracy of individual test/train set partition

    accuracy = accuracy\_score(y\_true, y\_pred)

    print("Accuracy:", accuracy)

    break

# Calculate confusion matrix

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

# Print confusion matrix

print("Confusion Matrix:")

print(conf\_matrix)

# Display confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix)

disp.plot()

show()

# Calculate accuracy from confusion matrix

accuracy = np.trace(conf\_matrix) / np.sum(conf\_matrix)

print("Accuracy:", accuracy)