



## MSS-M-2: CASE STUDY Intelligent Systems

PROJECT NAME: BUILDING AGE RECOGNITION

### FINAL REPORT

Submitted By

Haseeb Jauhar (22208669)

Ruchitaben Tajpara (12203509)

Vivek Vyas (22210450)

ARTIFICIAL INTELLIGENCE FOR SMART SENSORS/ACTUATORS

UNDER THE GUIDANCE OF PROF. JÜRGEN WITTMANN

Faculty of Applied Natural Science and Industrial Engineering

Deggendorf Institute of Technology, Cham

# Building Age Recognition using satellite Images

Ruchitaben Tajpara, Haseeb Jauhar, Vivek Vyas

## **Abstract:**

**We propose a different method for automatically predicting the age of buildings using satellite images. Our approach involves a two-step process: firstly, learning distinctive visual patterns associated with different building epochs at the patch level, and secondly, aggregating the patch-level age estimates to obtain a global prediction for the entire building. We create datasets from various sources and thoroughly evaluate our approach, examining its sensitivity to parameters and assessing the ability of deep networks to capture age-related visual patterns. The results demonstrate the remarkable accuracy of our approach. This Project represents an initial stride towards automating the assessment of building age.**

**Keywords:** *Building, Images, Data set, Model,*

## INTRODUCTION

This Case study behind this research is the automated Prediction of real estate objects. The Prediction of building age is a difficult task that takes into account many aspects (e.g., location, infrastructure, building size and condition etc.) and to date is mostly performed manually by domain experts. The most common approach for building age recognition is based on basics of line and color on the building. Age information of buildings is, however, often not available. The goal of this case study is to Predict age from satellite image. We suggest that the building age can be extracted by analyzing images showing the external view of houses.

We create ten different buildings groups from different building epochs. Once trained, the building age recognition system can process new images and apply the learned models to estimate the age of the buildings depicted in those images. The system can provide age predictions for individual buildings or classify them into predefined age categories. For a human observer, it is relatively easy to distinguish a rather old building from a newer one. The assignment of a distinct building epoch for each building is, however, a complex task. The major research question behind this work is if we can automatically capture such characteristic patterns by automated image analysis and leverage them for the automatic prediction of building age.

We present an automated method for the estimation of building age. This task is difficult because of a lack of datasets. This task is a complex task since buildings can appear in different locations of the image, from different distances and perspectives as well as under different weather and lighting conditions. Additionally, foreground objects like cars and trees often cover significant areas of the facade. We break down the

problem by first trying to learn characteristic visual primitives for each building epoch. Based on age estimations of these primitives we derive enhanced age assessments for an entire building. Also, we tried different ML models like RCNN, FAST CNN, YOLO V5 for accurate result for recognition of building age. The accuracy of the system's age predictions depends on the quality and diversity of the training data, as well as the sophistication of the machine learning models used. The system may undergo iterative refinement and improvement, incorporating feedback and new data to enhance its performance over time.

## 1. LITERATURE REVIEW

In the field of real estate image analysis (REIA), there is a growing interest in developing automated methods for estimating the age of buildings based on unconstrained photographs. The first literature we have taken was 'Automatic Prediction of Building Age from Photographs' is research where the prediction of building age is from photographs [1]. This research paper presents a novel two-stage approach utilizing deep learning techniques to accurately estimate building age. The reviewed literature consists of a research paper that introduces an automated method for estimating the age of buildings. The paper provides a detailed evaluation of the proposed approach, including its sensitivity to parameters and the ability of deep networks to learn age-related visual patterns.

The primary theme addressed in the literature is the automated estimation of the building age through visual analysis. The authors identify a gap in the existing research, as no previous methods have been developed specifically for this task. The authors aim to address this gap by introducing their two-stage approach, which serves as a baseline for the automated assessment of building parameters for price prediction. The literature begins with an abstract summarizing the key findings and contributions of the study. It concludes by discussing the results, insights gained from the analysis, and future research directions. The structure of the literature is concise, focused, and presents information in a logical sequence.

The literature introduces a pioneering method for automatically estimating the age of buildings from unconstrained photographs. The authors propose a two-stage approach that involves learning characteristic visual patterns at the patch-level for different building epochs, and then aggregating the age estimates globally over the entire building. The evaluation of the proposed approach includes the compilation of datasets from various sources and a detailed

analysis of its performance. Remarkably, the results demonstrate that the proposed method outperforms human evaluators, establishing a new performance baseline for automated building age estimation. The conclusion of the literature review emphasizes the significance of the research field of real estate image analysis and the importance of automated building age estimation. The authors discuss the challenges faced in learning age-related patterns, attributing them to the high intra-class variation of building epochs. The review suggests future research directions, such as exploring deeper network architectures, incorporating fine-grained ground truth annotations, employing building segmentation techniques, and predicting renovation dates. Overall, this literature provides valuable insights into the automated assessment of building parameters and lays the groundwork for further advancements in the field of real estate image analysis.

The second literature was ‘Estimating Building Age from Google Street View Images Using Deep Learning’ [2]. In the field of urban analysis, building databases plays a vital role in understanding urban environments. However, these databases often lack detailed information about building attributes, such as their age. With the increasing availability of building images through platforms like Google Street View and the advancements in image processing techniques, there is an opportunity to extract valuable information from images to enrich building databases. This literature review presents a novel method for estimating the building age by utilizing deep learning techniques, specifically convolutional neural networks (CNNs) for feature extraction, and support vector machines (SVMs) for construction year regression.

This study makes two significant contributions. Firstly, it is the first known attempt to estimate building age directly from images using deep learning techniques. This innovative approach provides valuable insights for urban planners and researchers, showcasing the potential of image processing and deep learning in enhancing building databases. Secondly, the study introduces an image-based framework for building age estimation that simplifies the analysis process by not requiring additional information such as building height, floor area, or construction materials. The proposed algorithm consists of three major steps: downloading Google Street View images, extracting relevant image features, and estimating building age using deeper convolutional neural networks. The results obtained from the North and West Metropolitan Region of Victoria demonstrate the effectiveness of this approach, showing accurate predictions of building age.

Overall, this literature review presents a pioneering approach for estimating building age directly from Google Street View images. By leveraging convolutional neural networks and support vector machines, the proposed method achieves accurate predictions of building age. This research opens up new possibilities for utilizing image processing and deep learning techniques to enhance building databases. Future work could involve further exploration and refinement of the proposed framework, considering different regions and datasets to evaluate its generalizability.

## 2. REQUIREMENTS

For this project, the computational requirements are high, making a powerful multicore CPU crucial to ensure faster processing. In addition, the utilization of a graphics processing unit (GPU) can significantly enhance the training process for ML models. GPUs excel in parallel processing, which is advantageous for complex neural networks and can greatly expedite training times. Adequate memory is essential, especially when dealing with large datasets, as it directly impacts the ability to handle and process the data efficiently. Furthermore, specialized hardware, such as high-performance computing clusters or cloud-based GPU instances, can be leveraged to enable distributed training. These options enable parallel processing across multiple machines, which can be particularly beneficial for training large-scale models or handling extensive datasets. It is important to carefully consider and allocate the appropriate hardware resources to ensure optimal performance and efficiency in this computationally intensive project.

## 3. METHODOLOGY

The method involves data collection, processing of the images for the model training and methods for the model training.

### 3.1 DATA COLLECTION

The collection of data plays a crucial role in the case study focused on the age recognition of buildings using satellite imagery. The acquisition of accurate and comprehensive data is of utmost importance to developing a robust and dependable system for estimating the age of buildings. Satellite imagery serves as a valuable source of information in this context, as it provides a broad and consistent view of the study area. By collecting high-resolution satellite imagery, it becomes possible to capture detailed visual characteristics of buildings, including roof shapes, structural features, and contextual information. The spatial and temporal coverage of the data allows for a comprehensive analysis of building ages across different regions and time periods. Moreover, the use of satellite imagery enables the study to overcome limitations associated with traditional data collection methods, such as time-consuming surveys or incomplete historical records.

The significance of data collection lies in its ability to establish a solid foundation for accurate age estimation. By collecting reliable and diverse datasets, the study can effectively train and validate the age recognition system. The integration of ground truth data, where available, enhances the accuracy of the model by providing reference points for comparison. Additionally, proper preprocessing of the collected data, such as geometric rectification and radiometric calibration, ensures that the imagery is correctly aligned and enhanced for subsequent analysis. The availability of comprehensive and well-prepared data empowers the methodology by enabling the extraction of relevant features and the development of robust algorithms for age estimation.

Ultimately, meticulous data collection forms the basis for a sound and reliable case study on the age recognition of buildings using satellite imagery.

### 3.1.1 DATA SOURCES

The major point was to acquire images of buildings based on their actual construction year. For the actual construction year, we used Google Search to locate the building first on Google Maps [3,4]. Buildings with good images are available without the visual obstruction of trees, and electric or telecom posts are used.

Some geospatial satellites like Maxar Technologies, Airbus Defence and Space, DigitalGlobe (a Maxar company), Google Earth Engine, Esri, Sentinel Hub, Planet, OpenStreetMap (OSM), and GeoIQ were used to get photos. All these satellites are paid, subscription, or limited to some regions only. Since construction here is required for the buildings, these satellites were not a viable option for data collection.

Data sources are mostly from Google Maps, as other sources could not provide the necessary images for the desired area or region. Google Street View is used for small buildings such as single-family houses, and Google Maps 3D View is used for taller buildings such as apartments and skyscrapers [5]. The construction year of buildings is available on the following websites [6,7,8]:

- a. <https://interaktiv.morgenpost.de/so-alt-wohnt-berlin/>
- b. <https://developers.arcgis.com/javascript/latest/sample-code/visualization-vv-color-animate/live/index.html>
- c. <https://googlemapsmania.blogspot.com/2018/05/the-berlin-building-age-map.html>

The limitations of using Google Maps as the primary data source have affected the dataset's quality and diversity.

### 3.1.2 COVERAGE AREA

Locations used here are from Munich, Germany and New York, USA. Different streets of these cities are used here based on the availability of images. Most of the images are from Manhattan and the Brooklyn borough of New York City. New York has a wide variety of building structures with varying architectural styles. For example, skyscrapers are more in Manhattan areas, and we could collect good pictures from here. Even small buildings like shops and small apartment buildings are also available in Manhattan. Brooklyn also has a wide variety of buildings. Single-family houses are more here compared to Manhattan.

Munich does not have skyscrapers like in New York, but small apartments and single-family houses are available here. The availability of Google Street View is not the same as in New York as some images we had to take 3D view screenshots even for small buildings, which was quite the opposite in New York as Google Street View is available on most streets of the city. Obstacles like trees are quite common in Munich. Munich

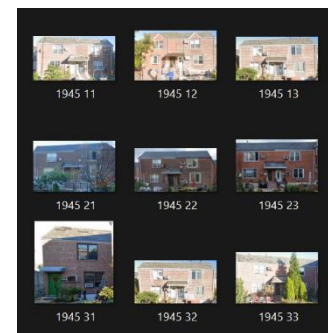
and New York have quite similar architectural styles for single-family houses and apartment buildings.

The dataset's geographical distribution might impact the results, as differences in architectural style and different time periods of images from around the world would not be able to be recognized by our model.

### 3.1.3 IMAGE ACQUISITION

First, search for Google Street View availability for single-family houses. Once good houses with good exposure are selected and construction ages are determined from the above-mentioned websites. Using the Snipping Tool, screenshots of the images are taken and saved in the folders with age categories. Google Earth Pro is also used to some extent [10], but it slows down the computer. The image-saving tool from Google Earth Pro is very useful to get good-quality images. But saving an image from this app consumes a lot of time and takes most of the system processing capacity which was not appropriate for this case study due to the time frame and unavailability of a good computer.

Each image acquired was saved in different classes with age label. The age class is on a 10-year age gap and while saving the images year of construction date is given as the name. The repeated years had to be saved as year-1, year-2 and in that sequence. Fig-01 shows how the images were saved.



**Fig-01** Images saved year wise

### 3.1.4 SPATIAL RESOLUTION

The spatial resolution of the satellite imagery refers to the level of detail captured by each pixel in the image and is typically measured in meters. Google street view images are from various sources, hence with different resolutions. Though images have different spatial resolutions depending on the area we were searching, the screenshot method resulted getting poor-quality images. Even directly saving from the tools available in Google Earth Pro could not provide good pictures due to some natural problems like obstacles and overlapping issues.

### 3.1.5 TEMPORAL COVERAGE

Specifying the period of time or duration in which the satellite imagery was obtained. This is important for

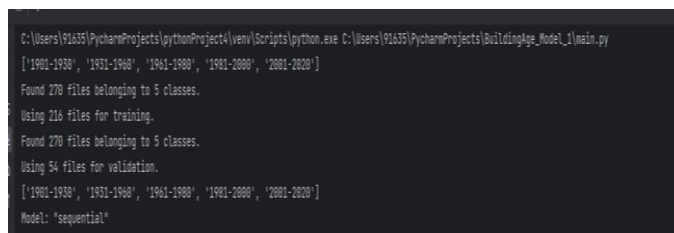
understanding the temporal context of the age recognition analysis. However, since the image selection is based on the availability of the construction year, the specific period is not considered in this case study.

### 3.1.6 GROUND TRUTH DATA

Prof. Jürgen Wittmann contacted the district office of Cham to get the resources for this case study, and the result was negative. We have tried to access some websites which have data sets with construction years, which turned out to be fewer quality images. And construction year is not mentioned on the label on most of the websites. Later with the available website links we could cross-check in Google Search the construction years and we had to limit ourselves only to this method due to the time frame to finish the case study.

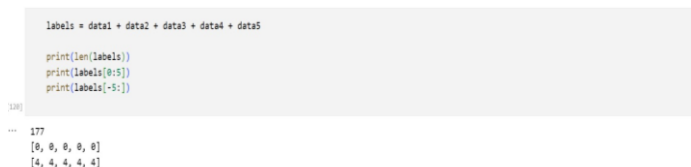
## 3.2 DATASETS

The data sets are created according to the availability of images. A total of 562 images were captured. 270 images were of single-family houses where 216 were there for training and 54 for validation, and 177 were of tall buildings like apartments and skyscrapers Fig- 02 and Fig-03 show the data sets. The rest of the images had to be scrapped due to quality issues. Five classes are made for both data sets. 1900-1930, 1931-1960, 1961-1980, 1981-2000, 2001-2020. Initially, we had classes with a 10-year gap and the number of classes was 12. But at the end of the data collection, we had to reduce the classes to 5 due to the shortage of images in certain classes.



```
C:\Users\91a35\PycharmProjects\pythonProject4\venv\Scripts\python.exe C:\Users\91a35\PycharmProjects\BuildingAge_Model_1\main.py
['1901-1930', '1931-1960', '1961-1980', '1981-2000', '2001-2020']
Found 270 files belonging to 5 classes.
Using 216 files for training.
Found 270 files belonging to 5 classes.
Using 54 files for validation.
['1901-1930', '1931-1960', '1961-1980', '1981-2000', '2001-2020']
Model: 'sequential'
```

**Fig- 02** Datasets of CNN model (Single Family Houses)



```
labels = data1 + data2 + data3 + data4 + data5
print(len(labels))
print(labels[0:5])
print(labels[-5:])
177
[0, 0, 0, 0, 0]
[4, 4, 4, 4, 4]
```

**Fig-03** Datasets of Custom CNN model (Skyscrapers and Apartments)

## 3.3 PREPROCESSING

Images with obstructions had to be cropped, and the alignment of the images had to be changed to get a better view. All images are resized to 180 x 180 pixels. Setting the image parameters, the variables `batch_size`, `img_height`, and `img_width` indicate the number of images to process at once as

well as the dimensions (height and width) to which the images will be scaled.

## 3.4 FEATURE EXTRACTION

Construction and architectural designs change over time. The lines of the buildings as a means of cracking over time and the colors of the buildings, which also change over time are the major factors determining the age of the building.

## 3.5 AGE ESTIMATION ALGORITHM

### 3.5.1 DATA PREPARATION

Images are collected and labelled accordingly. Split the dataset into training and validation sets to evaluate the performance of the age estimation algorithm. The split percentage is 80-20. For the single-family houses, training images are 216 and validation images are 54. For the second data set of apartments and skyscrapers, training images are 141 and validation images are 36.

### 3.5.2 MODEL INTEGRATION

Install the Python framework needed for building and training CNN models. TensorFlow is used here.

### 3.5.3 AGE ESTIMATION

To determine the age of buildings based on CNN detections in unseen photos, use the trained age estimation model. To evaluate its performance and accuracy, apply the age estimate model to the discovered building features in the testing dataset.

## 4. MODEL PIPELINE

We have used Convolutional Neural Network (CNN) for single-family homes age recognition and a custom Convolutional Neural Network using the Keras API from TensorFlow for the apartments and skyscrapers age recognition. CNNs are deep learning models that are specifically built for image processing applications. We also used the Keras API from TensorFlow, a famous deep-learning framework, to create a bespoke CNN. Image extraction, data preprocessing, conversion to numpy arrays, labelling, model training, validation and testing/training split, model building, model prediction, and obtaining the final output are all critical processes in the pipeline. The primary steps involved in training and assessing a CNN are mentioned below in different steps.

The code for single-family homes age recognition creates a CNN model with convolutional layers, max-pooling layers, a flatten layer, and dense layers using the Sequential API from `tensorflow.keras.models`. The second code snippet likewise makes use of the Sequential API to create a CNN model for regularization using convolutional layers, max-pooling layers, a flatten layer, dense layers, and dropout layers. Although the specific layer configurations may differ, both models adhere to the typical CNN architecture used in image classification.

#### 4.1 INPUT

The input comprises building images that must be processed by the CNN model. These photos are specifically collected for the age estimation task.

#### 4.2 IMAGE EXTRACTION

The building images are extracted from the input source, which is a dataset at this step.

#### 4.3 DATA PREPROCESSING

Building photos are pre-processed in order to be ready for training. This process involves activities including scaling the photographs to a consistent size, normalizing pixel values, and using augmentation techniques like rotation, flipping, and adding noise. These strategies aid in the enhancement of the dataset and the generalization capacity of the model.

#### 4.4 CONVERSION TO NUMPY ARRAYS

Convert the pre-processed photos into numpy arrays, the standard data format for deep learning model training. After preprocessing the photos, they must be turned into numpy arrays, which are the usual data format for training deep learning models. The photos are represented as numerical arrays for the CNN model to process, Fig-04.

```
type(images[0])
numpy.ndarray

images[0].shape
(128, 128, 3)

# converting image list and label list to numpy arrays
X = np.array(images)
Y = np.array(labels)
```

**Fig-04** Converting image to numpy arrays

#### 4.5 ASSIGNING LABELS

Assign appropriate classes to the images, indicating the age of the houses or buildings. Each building image is given an age label that indicates the age of the house or building. This stage is matching the photographs to the appropriate age classes or categories.

#### 4.6 MODEL TRAINING

Using the labelled and preprocessed pictures, train the CNN model. The model learns to recognize patterns and features associated with various age groups during training. To minimize the loss function, model parameters such as weights and biases are iteratively updated using optimization

procedures such as gradient descent and backpropagation.

#### 4.7 VALIDATION AND TESTING/TRAINING SPLIT

Split the dataset into a training set and a validation set. In this case, the split is 80% for training and 20% for validation/testing. The validation set here helps monitor the model's performance during training and provides feedback for model optimization.

#### 4.8 MODEL COMPILING

Compile the CNN model by defining the loss function, optimization method, and any other metrics to be evaluated during training. The CNN model is built before training by setting the loss function, optimizer, and any other metrics to be evaluated during training. The loss function calculates the difference between the predicted and true age labels, and the optimizer modifies the model parameters to minimize this loss.

#### 4.9 MODEL PREDICTION

Use the trained model to make predictions on new or test images. Once trained, the CNN model can be used to make predictions on fresh or previously unseen building photos. The model takes an input image, runs it through its layers, and outputs estimated age labels.

#### 4.10 OUTPUT

Obtain the predictions of the model, which represent the predicted age labels for the test photos. The model pipeline's ultimate output is the anticipated age labels for the test photos. Based on the trained CNN model, these predictions show the estimated ages of the buildings.

### 5. CODE STRUCTURE

#### 5.1 CNN MODEL FOR SINGLE FAMILY HOUSES

##### 5.1.1 IMPORTING STATEMENTS

We imported and used many libraries and modules such as 'os', 'numpy', 'matplotlib.pyplot', 'tensorflow', 'layers from tensorflow.keras' and 'sequential from tensorflow.keras.model'.

- OS module can be helpful working the operating system, which allows us to perform main operations such as getting access to files and directories.
- We can use numpy library to import as numpy for arrays and numerical operations.
- Tensorflow library as tf is used to utilize various functionalities for deep learning, By the supporting library of tensorflow.keras we import this module to define the layers of the neural network model.
- We can create a sequential model and import this module from tensorflow.keras.models



### 5.1.2 DIRECTORY SETUP

By specifying 'data\_dir' variable we get to know the directory where image data is stored and also can print the list of files in the 'data\_dir' using 'os.listdir(data\_dir)'.

### 5.1.3 DATA LOADING AND PREPROCESSING

It is used to load and preprocess the image data 'tf.keras.preprocessing.image\_dataset\_from\_directory()'. We created two datasets one is for training 'train\_ds' and other for validation 'val\_ds'. In the end we specified the batch size, image size, validation split and seed.

### 5.1.4 CLASS NAMES

Using 'train\_ds.class\_names' we can obtain the class names from training dataset and further we can print them using 'print(class\_names)'.

### 5.1.5 MODEL DEFINITION

Creating a sequential model using 'Sequential()', then defining the model layers using 'layers' from 'tensorflow.keras.layers' layers such as 'Rescaling', 'Conv2D', 'MaxPooling2D', 'Flatten', and 'Dense'. Set the input shape and number of classes dynamically.

### 5.1.6 MODEL COMPILE

By specifying the optimizer, loss function and evaluation metrics we can compile the model using 'compile()'.

### 5.1.7 MODEL TRAINING

Providing training and validation datasets along with number of epochs to train the model using the 'fit()' method. Save the training 'history' to the history variable.

Parameters of Model training were:

Total parameters: 3,989,285

Trainable parameters: 3, 989, 285

Non-Trainable parameters: 0

```

dense_1 (Dense)          (None, 5)          645
=====
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
=====
> Users > 91635 > PycharmProjects > BuildingAge_Model_1 > main.py

```

**Fig-05** Trainable and non-trainable parameters for single family houses

### 5.1.8 IMAGE AGE PREDICTION

Using 'test\_image\_path' variable we can specify the test image. Preprocess the test image using

'tf.keras.preprocessing.image.load\_img()' method to generate the prediction of the test image and in the last we can extract the predicted class index and confidence level. We can display the test image and prediction results using 'matplotlib.pyplot'.

### 5.1.9 PRINTING PREDICTED AGE

Using 'print("Predicted age:", predicted\_class)', display the predicted age on the console.

```

7/7 [=====] - 2s 382ms/step - loss: 1.4988 - accuracy: 0.2085 - val_loss: 1.2512 - val_accuracy: 0.6111
Epoch 10/10
7/7 [=====] - 2s 359ms/step - loss: 1.4118 - accuracy: 0.5637 - val_loss: 1.3908 - val_accuracy: 0.2222
Epoch 4/10
7/7 [=====] - 2s 340ms/step - loss: 1.3716 - accuracy: 0.3981 - val_loss: 1.1849 - val_accuracy: 0.6296
Epoch 5/10
7/7 [=====] - 2s 340ms/step - loss: 1.3845 - accuracy: 0.5185 - val_loss: 1.8567 - val_accuracy: 0.7637
Epoch 6/10
7/7 [=====] - 2s 334ms/step - loss: 1.2278 - accuracy: 0.5693 - val_loss: 0.9972 - val_accuracy: 0.7637
Epoch 7/10
7/7 [=====] - 2s 325ms/step - loss: 1.0647 - accuracy: 0.6481 - val_loss: 0.9268 - val_accuracy: 0.6852
Epoch 8/10
7/7 [=====] - 2s 315ms/step - loss: 0.8842 - accuracy: 0.7637 - val_loss: 0.9268 - val_accuracy: 0.6852
Epoch 9/10
7/7 [=====] - 2s 318ms/step - loss: 0.6979 - accuracy: 0.7938 - val_loss: 0.8146 - val_accuracy: 0.7222
Epoch 10/10
7/7 [=====] - 2s 312ms/step - loss: 0.4396 - accuracy: 0.8738 - val_loss: 0.8542 - val_accuracy: 0.6667
1/1 [=====] - 8s 67ms/step
Predicted age: 1901-2000
Process finished with exit code 0

```

**Fig-06** Displaying predicted age on the console

## 5.2 CUSTOM CNN FOR SKYSCRAPERS AND APARTMENTS

### 5.2.1 INSTALLATION

In the starting block of the code, we installed packages using 'pip install'. The packages we installed were: 'tensorflow', 'tensorflow-gpu', 'opencv-python', 'matplotlib', and 'scikit-learn'.

### 5.2.2 DATA LOADING AND PREPROCESSING

To initialize the directory of the image dataset the code is: 'folder\_path'. It retrieves the list of files in 'folder\_path' using 'os.listdir()' and filters out the '.ipynb\_checkpoints' folder. All the other files we printed using a for loop. The code in the next step proceeds to load and preprocess images for every individual range of dataset (1901-1930, 1931-1960, 1961-1980, 1981-2000, and 2001-2020) using 'Image.open()', 'image\_resize', 'image\_convert()', and 'np.array()'. The images are added to the images list.

- Initialization: The code sets the directory path for the picture dataset as the value for the variable "folder\_path."
- File List Retrieval: The 'os.listdir()' function is used by the code to retrieve a list of files in the 'folder\_path'. Within the given directory, a list of all files and directories is available.
- Filtering Files: The '.ipynb\_checkpoints' folder is excluded from the list of files by the code. This folder is often created by Jupyter Notebook, but it has no bearing on the picture dataset.

- d) **Printing Files:** Using a for loop, the code prints the directory's remaining files. This procedure is probably provided for informational reasons to confirm the existence of the files in the dataset.
- e) The code loads and preprocesses the photos for each dataset according to an age range. For each image, it carries out the following operations:
- f) The Python Imaging Library (PIL) function "Image.open()" opens the image file.
- g) Using the 'resize()' method, the code resizes the image to the desired size. It is customary to ensure that every image has the precise dimensions, which is frequently required for deep-learning models.
- h) Using the 'convert()' method, the code transforms the image into the specified color space or image mode. To guarantee consistency in image representations, this step may be taken.
- i) 'np.array()': The code uses 'np.array()' to turn the image data into a NumPy array. Using array-based operations, this phase enables simpler manipulation and processing of the picture data.
- j) **Images:** After preprocessing, the code adds the finished image to a list called "images," probably for later use or analysis.

### 5.2.3 TRAIN-TEST SPLIT AND SCALING

The dataset was divided using the 'train\_test\_split()' function from 'sklearn.model\_selection'. In order to scale the data, divide it by 255.

### 5.2.4 MODEL CREATION AND COMPILATION

- 'keras.Sequential()' is used to generate a sequential model.
- 'model.add()' is used to add convolutional and pooling layers to the model. The model is 'flattened' and more fully connected layers are added. The Adam optimizer, "SparseCategoricalCrossentropy loss," and "accuracy metric" are used to build the model.

### 5.2.5 MODEL TRAINING

Using the model.fit() function, the code trains the model. A crucial technique in TensorFlow/Keras for training a model using labeled data is this function. It accepts the relevant goal labels (Y\_train) and input features (X\_train) as inputs. The history variable contains the training history.

Parameters used for training were:  
 Total parameters: 7,400,901  
 Trainable parameters: 7,400,901  
 Non-trainable parameters: 0

```
dropout_1 (Dropout)      (None, 64)      0
dense_11 (Dense)         (None, 5)       325
...
Total params: 7,400,901
Trainable params: 7,400,901
Non-trainable params: 0
```

**Fig-07** Trainable and non-trainable parameters for Skyscrapers and Apartments

### 5.2.6 TRAINING HISTORY

A variable called history in the code is used to store the training history. The loss and metrics values recorded at each epoch, as well as other data regarding the model's performance during training, are frequently included in the training history. If validation data is provided during training, extra information such as the validation loss and metrics may also be included in the history object.

### 5.2.7 EVALUATION AND VISUALIZATION

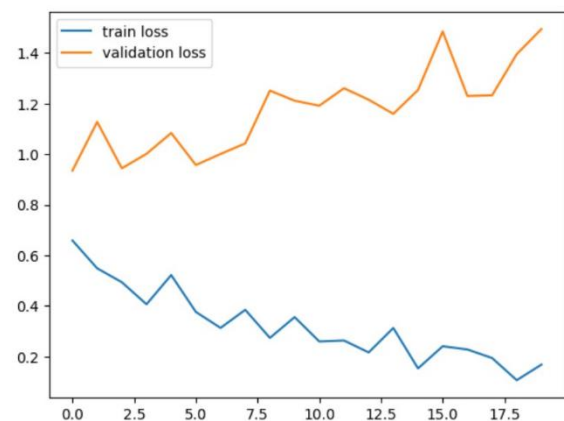
- Using the model.evaluate() function, the code assesses the trained model on a test set. On the given test set, this function computes the loss and any metrics (like accuracy) supplied during the model building. The function most likely receives the test set as input.
- 'matplotlib.pyplot.plot()' is used to plot the training and validation loss.'

### 5.2.8 LOSS AND ACCURACY PRINTING

The code outputs the calculated values for loss and accuracy to the console. The test set performance of the model is disclosed in this stage.

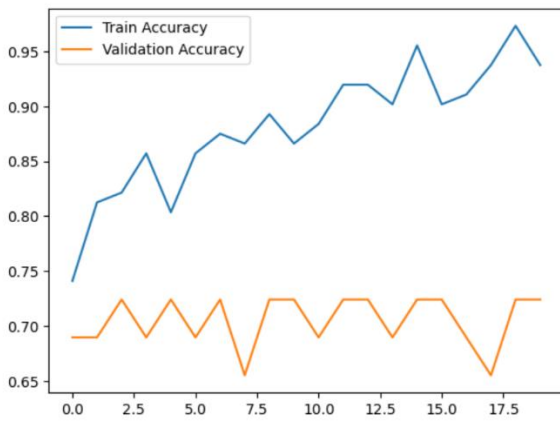
### 5.2.9 PLOTTING TRAINING AND VALIDATION LOSS

The training and validation of loss values are plotted by the code. The plot is made using the matplotlib.pyplot.plot() function. Values for training and validation losses are probably acquired during model training and saved in variables. This plot gives us a visual picture of how the model is being trained and allows for analysis of convergence and potential overfitting.



**Fig-09** Plotting of Loss Graph for Skyscrapers and Apartments.





**Fig-09** Plotting of Accuracy Graph for Skyscrapers and Apartments.

### 5.2.10 IMAGE PREDICTION

- Using "input()," the user is asked to enter the path of the image to be predicted. 'cv2.imread()' is used to load the input image. Using 'cv2.cvtColor()', 'matplotlib.pyplot.imshow()', and 'matplotlib.pyplot.show()', the image is converted from BGR to RGB color space and provided.

To fit the input shape predicted by the model, the image is reduced in size, scaled, and reshaped. To get the prediction probabilities, the model's 'predict()' method is used with the input image. Using 'np.mean()', the class probabilities are averaged over the prediction axis. 'np.argmax()' is used to get the anticipated class index. Based on the projected class index, the predicted age range is retrieved from the 'class\_labels' array. The console displays the predicted age range.



```
[[0.88780737 0.8526299 0.10233852 0.13519236 0.53211015]]
[0.88780737 0.8526299 0.10233852 0.13519236 0.53211015]
0
The predicted age range of the building is: 1901-1930
```

**Fig-10** Predicted age using custom CNN model

## 6. MODEL BUILDING

```
model = Sequential()
```

```
model.add(keras.layers.Conv2D(32, kernel_size=(3,3),
```

```
activation='relu', input_shape=(128,128,3)))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)
))
```

```
model.add(keras.layers.Conv2D(64, kernel_size=(3,3),
activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)
))
```

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(num_of_classes,
activation='sigmoid'))
```

The model is built using the Keras Sequential API. The Sequential class allows us to create a linear stack of layers.

### 6.1 FOLLOWING IS THE MODEL ARCHITECTURE

- A 2D convolutional layer (Conv2D) with 32 filters, a kernel size of 3, and the ReLU activation function makes up the top layer.
- For RGB images, the input\_shape parameter defines the shape of the input images (128,128,3).
- MaxPooling2D, a 2D max pooling layer with a pool size of (2,2), is then added.
- A max pooling layer is added after a second 2D convolutional layer with 64 filters and a kernel size of (3,3).
- To make way for the fully connected levels, the Flatten layer converts the output of the preceding layer into a 1D array.
- The addition of two fully interconnected (Dense) layers with ReLU activation.
- There are 128 units in the first dense layer and 64 units in the second dense layer.
- To avoid overfitting, the Dropout layer is applied after each dense layer.
- The last dense layer uses the sigmoid activation function and has num\_of\_classes units (5 in this case). The final output probabilities for each class are generated by this layer.

### 6.2 MODEL COMPILING

The model is compiled using the following parameters after it has been built:

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCatCrossentropy
(from_logits=True), metrics=['accuracy'])
```

- This line indicates the optimizer used to train the model, which is 'adam'. The Adam optimizer is selected in this case. A popular optimization algorithm frequently applied in deep learning is the Adam

optimizer. It adjusts the learning rate while you're training, which makes it useful and efficient for a variety of problems.

- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`: The loss function, which is measured as the difference between the model's predicted output and the actual labels, is defined in this line. When the labels are given as integers, the `SparseCategoricalCrossentropy` loss is frequently used for multiclass classification tasks. The `from_logits=True` argument indicates that the loss function's inputs are the model's output logits (raw, unnormalized predictions).
- This line specifies the evaluation metric(s) to be used during training and testing, `metrics=['accuracy']`. The accuracy metric, which evaluates the percentage of accurate predictions made by the model, is selected in this instance.
- The model is set up for training by specifying the optimizer, loss function, and metrics in the `compile()` function. In order to evaluate the performance of our model, we used this specified optimizer to train and `loss` a function while keeping a focus on the accuracy metric.
- The function `model.summary()` shows a summary of the model architecture along with the number of parameters in each layer.
- Finally, using the fit function and the training sets "X\_train\_scaled" and "Y\_train," we trained the model. In order to distribute some training data for validation during training, the validation split 0.1 is used.
- There will be 20 epochs total.

## 7. MODEL TRAINING

The following is the model training section:

```
X_train_scaled, Y_train, validation_split=0.1,
epochs=20, history = model.fit
```

- The fit function is used to train the model, which uses the training data `X_train_scaled` and labels `Y_train`.
- The following variables are used in the fit function:
- `X_train_scaled`: A numpy array of scaled images serves as the input training data.
  - `Y_train`: The labels that proceed with the training data.
  - The percentage of the training data to be used for validation during training is determined by the `validation_split` parameter. 0.1 (10%) of the training data are used in this instance for validation.
  - `epochs`: How many epochs the model uses to iterate through the entire training dataset determines how often it performs so.
- During training, the model modifies its weights in accordance with the specified loss function and optimizer. On the training data, it tries to reduce loss and increase accuracy. The history object that the fit function returns contains the training metrics and

progress. For the training and validation sets at each epoch, it details the loss and accuracy values.

## 7.1 MODEL EVALUATION

After training, the model can be evaluated using the `evaluate` function on the test data:

```
loss, accuracy = (X_test_scaled, Y_test)
model.evaluate
print(accuracy, "Test Accuracy =", accuracy)
```

- `model.evaluate(Y_test, X_test_scaled)`: The effectiveness of the model on the test dataset is assessed in this line. The loss and metrics specified during model compilation are calculated by the `evaluate()` function; in this case, the loss is probably the `SparseCategoricalCrossentropy` and the metric is accuracy. The test dataset is made up of the `X_test_scaled` input features and the `Y_test` accurate labels that correspond to those features.
- `Loss, Accuracy = Loss and Accuracy of the variable` are set to the values returned by the `evaluate()` function. Both the loss and accuracy variables will contain the calculated values for their respective variables.
- `print (accuracy, "Test Accuracy =")`: The test accuracy is printed to the console by this line. It shows the accuracy variable's value, revealing information about how well the model did on the test dataset.

## 8. OUTLOOK

In this case study, we explored the application of image classification algorithms to successfully classify different age ranges of buildings based on their architectural styles. We analysed two code snippets that employed convolutional neural network (CNN) models for the image classification task. The objective was to gain insights into the workflow, techniques, and challenges involved in building an age recognition of buildings using satellite images.

The first step in the project was to collect the images and preprocess the image dataset. The images were organized into different folders representing distinct time periods or age ranges of buildings. The images were loaded, resized, and converted to RGB format. They were then converted into numpy arrays to facilitate further processing. Additionally, the dataset was split into training and testing sets to evaluate the model's performance.

Both code snippets utilized CNN models for image classification. The models were defined using the Sequential API from the TensorFlow Keras library. They consisted of convolutional layers to extract features from the input images, followed by max pooling layers to downsample the feature maps. The models also incorporated flattening layers to convert the multidimensional feature maps into a one-dimensional vector, which was then passed through dense layers for

classification. Dropout layers were used in the second code snippet to mitigate overfitting.

After defining the model architecture, the next step was to train the models on the training dataset. The models were compiled with suitable optimizers, loss functions, and evaluation metrics. The training process involved iterating over the training dataset for multiple epochs. The models learned to recognize patterns and classify images based on the architectural features associated with different age ranges of buildings. The training progress was monitored, and the model's performance was evaluated on the validation data during the training process.

To assess the models' performance, they were evaluated on the testing dataset. The evaluation metrics, such as accuracy and loss, were calculated to measure how well the models classified unseen images. The training and validation loss and accuracy were visualized using plots to understand the models' learning progress and potential overfitting. The visualization provided insights into the models' performance and guided further improvements or adjustments.

Once the models were trained, they were used for predictions on new, unseen images. The input images were pre-processed by resizing, scaling, and reshaping to match the model's input requirements. Predictions were made using the trained models, and the class probabilities were calculated. The predicted class index with the highest probability was determined, and the corresponding age range label was assigned. The results were displayed alongside the input images, providing an overview of the model's predictions.

The project successfully demonstrated the use of CNN models and image classification algorithms to classify the age ranges of buildings based on their architectural styles. By following the outlined steps and fine-tuning the models, the accuracy of the predictions was improved. This technology has various real-world applications and can help in understanding and appreciating different architectural styles in a new way. The completed project provides valuable insights into the workflow, techniques, and challenges involved in building an effective image classification system for architectural analysis.

Factors contributing to a positive outlook for this case study report on age recognition of buildings using satellite imagery include:

- Relevance and importance in the field of geospatial analysis
- Novelty and innovation in leveraging remote sensing technology and machine learning
- Practical applications in urban planning, restoration projects, and heritage preservation
- Methodological contribution to the field
- Potential for collaboration with other researchers and organizations
- Academic and professional development opportunities

Though the drawbacks of our method could not distinguish the reconstructed or renovated buildings. More data sets and deep dive into machine learning are required to recognise the

age of these buildings. Furthermore, seasonal, and regional changes will affect the methodology as more data sets from different regions of the world and different seasons are required to further extend the age recognition process.

## 9. CONCLUSION

In summary, it is now possible to automate the process of calculating the age of buildings thanks to the application of cutting-edge technology like machine learning and image analysis. The technology can extract distinctive visual patterns and attributes connected to various building epochs by utilizing image data and applying sophisticated algorithms. This makes it possible to determine a building's age accurately even in the lack of clear age data. The age recognition system in buildings has several advantages. It does away with the requirement for manual expert evaluation, saving time and effort while enhancing accuracy and efficiency. It creates opportunities for extensive building age study, advancing plans for urban development, historical investigation, and architectural studies. The age recognition system in the building marks a substantial improvement in automating the assessment of building parameters. Overall, Continued research and refinement of the system will undoubtedly lead to even more accurate and reliable building age predictions.

## 10. REFERENCES

- [1] Zeppelzauer, Matthias & Despotovic, Miroslav & Sakeena, Muntaha & Koch, David & Doeller, Mario. (2018). *Automatic Prediction of Building Age from Photographs*. 126-134. 10.1145/3206025.3206060.
- [2] Li, Yan & Chen, Yiqun & Rajabifard, Abbas & Khoshelham, Kourosh. (2018). *Estimating building age from Google Street view images using deep learning*.
- [3] <https://www.google.com/>, 2023
- [4] <https://www.google.com/maps>, 2023
- [5] <https://www.google.com/streetview/>, 2023
- [6] <https://interaktiv.morgenpost.de/so-alt-wohnt-berlin/> Marie-Louise Timcke, André Pätzold, David Wendler and Christopher Möller, *Berliner Morgenpost*, May 23, 2018
- [7] <https://developers.arcgis.com/javascript/latest/sample-code/visualization-vv-color-animate/live/index.html> *New York Construction*, NYC OpenData, New Jersey Office of GIS, Esri, HERE, Garmin, SafeGraph, GeoTechnologies, Inc, METI/NASA, USGS, EPA, NPS, USDA, Powered by Esri
- [8] <https://googlemapsmania.blogspot.com/2018/05/the-berlin-building-age-map.html>. *The Berlin Building Age Map, Maps Mania*. Powered by Blogger.
- [9] <https://www.google.com/earth/versions/#earth-pro>, 2023

Authors:

	Names	Contribution	Page Number
1	Ruchitaben Tajpara	Abstract	1
		Introduction	1
		Literature Review	1
		Requirements	2
		Conclusion	10
2	Haseeb Jauhar	Methodology	2
		Model Pipeline	4
		Outlook	9
3	Vivek Vyas	Code Structure	5
		Model Building	8
		Model Training	9

➤ CODES:1. CODE OF 1<sup>ST</sup> CNN MODEL FOR SMALL HOUSES:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

data_dir = "Houses"

print(os.listdir(data_dir))

batch_size = 32
img_height = 180
img_width = 180

train_ds =
tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds =
tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

class_names = train_ds.class_names
print(class_names)

num_classes = len(class_names)

model = Sequential([
    layers.experimental.preprocessing.Rescaling(1./255,
input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

model.compile(optimizer='adam',
```

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

model.summary()

epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# ... Rest of your code for predicting image age ...
# Code for predicting image age
test_image_path = "1991-2000-test2.jpg"
# Preprocess the test image
img =
tf.keras.preprocessing.image.load_img(test_image_path,
target_size=(img_height, img_width))
x = tf.keras.preprocessing.image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x / 255.0 # Normalize the image
# Predict the age
predictions = model.predict(x)
predicted_class_index = np.argmax(predictions[0])
predicted_class = class_names[predicted_class_index]
confidence = predictions[0][predicted_class_index] * 100
# Display the result
plt.imshow(img)
plt.title(f"Predicted class: {predicted_class}
({confidence:.2f}% confidence)")
plt.axis("off")
plt.show()
print("Predicted age:", predicted_class)

2. CODE OF 2ND CUSTOM CNN MODEL FOR
SKYSCRAPPERS AND APARTMENTS:

!pip install tensorflow tensorflow-gpu==2.8.0 opencv-
python matplotlib
!pip list
import tensorflow as tf
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
from PIL import Image
#-----
folder_path = "SkyScrappers&Apartments"
file_list = os.listdir(folder_path)

# Filter out the '.ipynb_checkpoints' folder
file_list = [file for file in file_list if file !=
'.ipynb_checkpoints']

# Print the remaining files
for file in file_list:
```



```

    print(file)
    data1 = os.listdir('SkyScrappers&Apartments/1901-1930')
    print(data1[0:5])
    print(data1[-5:])

    data2 = os.listdir('SkyScrappers&Apartments/1931-1960')
    print(data2[0:5])
    print(data2[-5:])

    data3 = os.listdir('SkyScrappers&Apartments/1961-1980')
    print(data3[0:5])
    print(data3[-5:])

    data4 = os.listdir('SkyScrappers&Apartments/1981-2000')
    print(data4[0:5])
    print(data4[-5:])

    data5 = os.listdir('SkyScrappers&Apartments/2001-2020')
    print(data5[0:5])
    print(data5[-5:])
    print('Number of images in data1:', len(data1))
    print('Number of images in data2:', len(data2))
    print('Number of images in data3:', len(data3))
    print('Number of images in data4:', len(data4))
    print('Number of images in data5:', len(data5))
    labels = data1 + data2 + data3 + data4 + data5

    print(len(labels))
    print(labels[0:5])
    print(labels[-5:])

    # displaying the image of 1st dataset
    # Read the image file
    img = mpimg.imread('SkyScrappers&Apartments/1901-1930/1900.jpg')

    # Display the image
    imgplot = plt.imshow(img)
    plt.show()

    # displaying the image of 2nd dataset
    img = mpimg.imread('SkyScrappers&Apartments/1931-1960/1958.jpg')
    imgplot = plt.imshow(img)
    plt.show()

    # convert images to numpy arrays
    data1_path = 'SkyScrappers&Apartments/1901-1930/'

    data1 = os.listdir(data1_path)
    images = []

    for img_file in data1:

        image = Image.open(data1_path + img_file)
        image = image.resize((128,128))
        image = image.convert('RGB')
        image = np.array(image)
        images.append(image)

        #-----

    data2_path = 'SkyScrappers&Apartments/1931-1960/'

    data2 = os.listdir(data2_path)

    for img_file in data2:
        if not img_file.endswith('.jpg'): # Skip non-image files
            continue
        image_path = os.path.join(data2_path, img_file)
        image = Image.open(image_path)
        image = image.resize((128,128))
        image = image.convert('RGB')
        image = np.array(image)
        images.append(image)

        #-----

    data3_path = 'SkyScrappers&Apartments/1961-1980/'

    data3 = os.listdir(data3_path)

    for img_file in data3:
        if not img_file.endswith('.jpg'): # Skip non-image files
            continue
        image = Image.open(data3_path + img_file)
        image = image.resize((128,128))
        image = image.convert('RGB')
        image = np.array(image)
        images.append(image)

        #-----

    data4_path = 'SkyScrappers&Apartments/1981-2000/'

    data4 = os.listdir(data4_path)

    for img_file in data4:
        if not img_file.endswith('.jpg'): # Skip non-image files
            continue
        image = Image.open(data4_path + img_file)
        image = image.resize((128,128))
        image = image.convert('RGB')
        image = np.array(image)
        images.append(image)

        #-----

    data5_path = 'SkyScrappers&Apartments/2001-2020/'

    data5 = os.listdir(data5_path)

    for img_file in data5:
        if not img_file.endswith('.jpg'): # Skip non-image files
            continue
        image = Image.open(data5_path + img_file)
        image = image.resize((128,128))

```

```

image = image.convert('RGB')
image = np.array(image)
images.append(image)
type(images[0])
# converting image list and label list to numpy arrays

X = np.array(images)
Y = np.array(labels)
!pip install scikit-learn
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
# scaling the data

X_train_scaled = X_train/255

X_test_scaled = X_test/255
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,
MaxPooling2D, Dense, Flatten, Dropout
model = Sequential()
num_of_classes = 5

model = keras.Sequential()

model.add(keras.layers.Conv2D(32, kernel_size=(3,3),
activation='relu', input_shape=(128,128,3)))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

model.add(keras.layers.Conv2D(64, kernel_size=(3,3),
activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(num_of_classes,
activation='sigmoid'))
# compile the neural network
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

model.summary()

!pip install pandas
# training the neural network
history = model.fit(X_train_scaled, Y_train,
validation_split=0.1, epochs=20)

loss, accuracy = model.evaluate(X_test_scaled, Y_test)
print('Test Accuracy =', accuracy)

h = history

# plot the loss value
plt.plot(h.history['loss'], label='train loss')
plt.plot(h.history['val_loss'], label='validation loss')
plt.legend()
plt.show()

# plot the accuracy value
plt.plot(h.history['accuracy'], label='Train Accuracy')
plt.plot(h.history['val_accuracy'], label='Validation
Accuracy')
plt.legend()
plt.show()

input_image_path = input('Path of the image to be
predicted: ')

input_image = cv2.imread(input_image_path)

input_image_rgb = cv2.cvtColor(input_image,
cv2.COLOR_BGR2RGB)

plt.imshow(input_image_rgb)
plt.axis('off')
plt.show()

input_image_resized = cv2.resize(input_image, (128, 128))

input_image_scaled = input_image_resized / 255

input_image_resized = np.reshape(input_image_scaled,
[1, 128, 128, 3])

input_prediction = model.predict(input_image_resized)

print(input_prediction)

class_probabilities = np.mean(input_prediction, axis=0)
print(class_probabilities)

predicted_class = np.argmax(class_probabilities)
print(predicted_class)

class_labels = ['1901-1930', '1931-1960', '1961-1980',
'1981-2000', '2001-2020']

predicted_age_range = class_labels[predicted_class]
print('The predicted age range of the building is:',
predicted_age_range)

```