UNIVERSITETI POLIS

ICCSM 2025

BOOK OF PROCEEDINGS

1ST INTERNATIONAL CONFERENCE
COMPUTER SCIENCES AND MANAGEMENT

WHERE DIGITAL & BUSINESS BECOME HUMAN

26-27 June 2025 | Tirana, Albania

ISBN 9789928347123

DOI 10.37199/c41000300

CONFERENCE CHAIR

Assoc. Prof. Merita Toska, POLIS University

PARTNER UNIVERSITIES

POLIS University, Albania
Université Lyon 2, France
Università Telematica San Raffaele, Italy
University of Western Macedonia, Greece
International University of Sarajevo, Bosnia & Herzegovina
Mother Teresa University, North Macedonia
Gebze Technical University, Turkey
Public International Business College, Kosovo
Rochester Institute of Technology – RIT Global Campus, Kosovo
Co-PLAN, Institute for Habitat Development, Albania
AI Hub Albania, Albania
Luralux, Albania

ORGANISING COMMITTEE

Dr. Blerta Mjeda
Dr. Emiliano Mankolli
Msc. Sonila Murataj
Msc. Andia Vllamasi
Msc. Klejda Hallaci
Msc. Erilda Muka
Msc. Armela Reka

SCIENTIFIC COMMITTEE

Prof. Dr. Jérôme Darmont, Université Lumière Lyon 2 (France)
Prof. Dr. Lydia Coudroy de Lille, Université Lumière Lyon 2 (France)
Prof. Dr. Jim Walker, Université Lumière Lyon 2 (France)
Prof. Dr. Besnik Aliaj, POLIS University, (Albania)
Prof. Dr. Daniela Sica, San Raffaele Roma University, (Italy)
Prof. Dr. Stefania Supino, San Raffaele Roma University, (Italy)
Prof. Dr. Arbana Kadriu, South East European University (North Macedonia)
Prof. Dr. Ing. Lejla Abazi-Bexheti, South East European University (North Macedonia)
Prof. Dr. Yusuf Sinan Akgül, Gebze Technical University (Turkey)
Assoc. Prof. Dr. Galia Marinova, Technical University of Sofia (Bulgaria)
Assoc. Prof. Dr. Vasil Guliashki, Technical University of Sofia (Bulgaria)
Assoc. Prof. Mehmet Göktürk, Gebze Technical University (Turkey)
Assoc. Prof. Yakup Genç, Gebze Technical University (Turkey)
Assoc. Prof. Habil Kalkan, Gebze Technical University (Turkey)
Assoc. Prof. Dr. Godiva Rëmbeci, POLIS University (Albania)
Assoc. Prof. Dr. Xhimi Hysa, POLIS University (Albania)
Assoc. Prof. Dr. Merita Toska, POLIS University (Albania)
Assoc. Prof. Dr. Sotir Dhamo, POLIS University (Albania)
Dr. Gennaro Maione, San Raffaele Roma University, (Italy)
Dr. Nicola Capolupo, San Raffaele Roma University, (Italy)
Dr. Benedetta Esposito, San Raffaele Roma University, (Italy)
Dr. Venera Demukaj, Rochester Institute of Technology (Kosovo)
Dr. Emil Knezović, International University of Sarajevo (BiH)
Dr. Šejma Aydin, International University of Sarajevo (BiH)
Dr. Azra Bičo, International University of Sarajevo (BiH)
Dr. Šejma Aydin, International University of Sarajevo (BiH)
Dr. Azra Bičo, International University of Sarajevo (BiH)
Dr. Hamza Smajić, International University of Sarajevo (BiH)
Dr. Panagiotis Kyratsis, University of Western Macedonia (Greece)
Dr. Delina Ibrahimaj, Minister of State for Entrepreneurship and Business Climate (Albania)
Dr. Elona Karafili, POLIS University (Albania)
Dr. Emi Hoxholli, POLIS University (Albania)
Dr. Shefqet Suparaku, POLIS University (Albania)
Dr. Manjola Hoxha, POLIS University (Albania)
Dr. Elsa Toska, POLIS University (Albania)
Dr. Emiliano Mankolli, POLIS University (Albania)

Dr. Albina Toçilla, POLIS University (Albania)
Dr. Sonia Jojic, POLIS University (Albania)
Dr. Ilda Rusi, POLIS University (Albania)
Dr. Ledian Bregasi, POLIS University (Albania)
Dr. Klodjan Xhexhi, POLIS University (Albania)
Dr. Endri Duro, POLIS University (Albania)
Dr. Remijon Pronja, POLIS University (Albania)
Dr. Vjosë Latifi, International Business Collage Mitrovica (Kosovo)
Dr. Agron Hajdari, International Business Collage Mitrovica (Kosovo)

Table of Contents

INFLUENCER MARKETING AND HUMAN CAPITAL: 8

THE STRATEGIC ROLE OF EMPLOYEES IN THE FOOD INDUSTRY 8

RECONFIGURING WORK IN THE AGRIFOOD CHAIN: PROFILING EMPLOYABILITY SKILLS VIA BIG DATA AND TRANSFORMER-BASED LANGUAGE MODELS..... 23

USER-CENTERED DIGITAL PRODUCT DESIGN: A TRANSPORTATION-RELATED CASE STUDY..... 34

REGIONAL TRANSPORT CORRIDORS: A COMPARATIVE ANALYSIS OF ALBANIA'S PERFORMANCE WITH NEIGHBOURING COUNTRIES..... 48

THE ALBANIAN INNOVATION ECOSYSTEM: POLICIES, PARTNERSHIPS, AND THE FUTURE OF ENTREPRENEURSHIP 66

THE SIX-HOUR WORKDAY: LITERATURE AND CASES ON PRODUCTIVITY, WELL-BEING, AND ECONOMIC IMPLICATIONS 78

ETHICAL ISSUES IN ARTIFICIAL INTELLIGENCE 86

INCLUSIVE PEDAGOGY AT SCALE: A MODEL FOR BUILDING CAPACITY THROUGH DIGITAL TRAINING AND POLICY IMPLEMENTATION..... 95

BLOCKCHAIN CRYPTOGRAPHY AND THE FUTURE OF DIGITAL CURRENCY SECURITY.....103

DIGITAL TWINS AS CATALYSTS FOR SUSTAINABILITY EDUCATION IN UNIVERSITY CAMPUSES: A CASE STUDY AT POLIS UNIVERSITY WITHIN THE FRAMEWORK OF EDUCATION 4.0.....115

YOUTH ENGAGEMENT AND DIGITAL CAPACITY BUILDING IN EUSAIR.....131

BRIDGING THE HUMAN-AI DIVIDE: ENHANCING TRUST AND COLLABORATION THROUGH HUMAN-TO-HUMAN TOUCHPOINTS IN ENTERPRISE AI ADOPTION.....144

THE ROLE OF AI IN PERSONALISED LEARNING.....158

BRAND INTEGRATION AND CONSUMER PERCEPTION IN POST-MERGER SCENARIOS: THE CASE OF ONE ALBANIA'S CUSTOMER-CENTRIC MARKETING STRATEGY 167

INFORMATION DIGITALISATION AS A KEY DRIVER TO ACHIEVE IMPROVEMENT OF SME PERFORMANCE 187

SAFEGUARDING DIGITAL AUTHENTICITY AND WOMEN'S IDENTITY THROUGH DEEPFAKE DETECTION 198

AUTOMATED STRATEGIES FOR DEFINING A JOB INTERVIEW 211

FROM CITIZEN VOICES TO BUSINESS VALUE: ARTIFICIAL INTELLIGENCE IN PARTICIPATORY ECOSYSTEMS.....222

AI AND IMAGE PROCESSING. SOME KEY MOMENTS IN THE IMPLEMENTATION OF THESE METHODS 233

AI IMAGE GENERATION AND ITS POSSIBLE CONTRIBUTIONS 265

IN ARCHITECTURAL LANGUAGE.....265

19

AI AND IMAGE PROCESSING. SOME KEY MOMENTS IN THE IMPLEMENTATION OF THESE METHODS

DOI: 10.37199/c41000319

Melisa TUFA

POLIS University, Tirana, Albania
melisa_tufa@universitetipolis.edu.al
ORCID 0009-0006-6376-9775

Tamara LUARASI

POLIS University, Tirana, Albania
tamara_luarasi@universitetipolis.edu.al
ORCID 0009-0002-7449-5491

Abstract

The integration of artificial intelligence (AI) into architecture and urban planning represents a significant shift in how professionals approach design, analysis, and decision-making. This article aims to underscore the relevance and applicability of various AI methodologies in the daily practices of architects and urban planners.

This article will serve as a starting point for a more detailed study of transforming existing architecture and urbanism curricula to include basic knowledge of modern technologies involving AI methods. This paper focuses on two common issues in implementing the methods mentioned above. The creation of custom datasets to solve a specific problem, and the CNN model that serves as the basis for many AI methods in image processing.

Through a complete example of the implementation of one of the AI methodologies, which is image classification, a working practice is illustrated.

Keywords: AI, Image Processing, Architecture, Urban Planning

I. INTRODUCTION

The purpose of this article is to sensitise architects and urban planners to the importance of using AI's various methodologies in their work, and to provide essential practices for these

methodologies. All this supports the idea of revising curricula for architects and urbanists, which would constitute a second phase of the study.

Only a limited number of articles, across a wide range, address the role AI has taken on in the work of architects or urbanists. The authors in Stojanovski et al. 2021 present a combination of urban visions of the future, digital tools, and AI techniques to inspire architects, engineers, programmers, and computer scientists to explore AI and improve architectural and urban practices.

In Jin et al (2024), it is mentioned that higher education has undergone major transformations with the development of artificial intelligence. According to the authors, this transformation consists not only in the use of new technologies but also in the way work is conceptualised. Undoubtedly, the introduction of these new methods into the daily work of architects and urbanists requires knowledge and skills to understand and implement them.

By possessing AI tools such as programming and data management, students would be able to handle datasets and use them in software for drawing that includes AI elements. Such software includes Rhino, Grasshopper, and Revit, which architects widely use.
There are two ways the architect can successfully use AI. Architectonic programming is a preliminary step in the design process, where architects and design professionals collect, analyse, and determine clients' desires and requirements to inform the design. Technologies such as Information and Communication Technology (ICT), Building Information Modelling (BIM), Virtual Reality (VR), and Augmented Reality (AR) are gradually changing the way architectural programming is conducted. AI in design further expands these experiences by creating highly efficient 3-dimensional interactive environments for communicating with customers (Stas, 2025)

The second direction is architectural design itself. AI algorithms, such as Generative Adversarial Networks (GANs) and other generative AI algorithms, are further expanding the boundaries of traditional architectural drawing.

A Generative Adversarial Network, or GAN, is a machine learning approach used for generative modelling designed by Ian Goodfellow and his colleagues in 2014 (Idiot Developer, 2020).

Another example is Foster + Partners (2021), where it is stated that if we train a machine with a set of hundreds of public spaces and extract those that are successful (it is easier to identify an existing successful space than to define the conditions for a space to be successful), the system in the machine can be used to generate other spaces with similar characteristics.

However, Nasir (2024) argues for a balance between traditional architectural methods and AI applications. Here, it is emphasised that, despite the use of AI to solve complex problems, hand drawing and even hand calculations remain very important for architects and cannot be replaced by AI.

Some image processing methodologies. Design is the most important element in the work of an architect or an interior design professional. An architectural design is an image for AI. In Mohit (2005), a coloured image is defined as a data holder that represents a combination of RGB values. The colour image is stored in a 3D array on the computer. The first two dimensions correspond to the image's dimensions (in pixels), and the third to the colour intensity (red, green, and blue) in each pixel.

In a program (in any programming language), such a structure can be represented as a 2-dimensional matrix of pixels, where the rows and columns represent the image's dimensions, and the value of each pixel is a composition of the colour intensities in that pixel. The image is the object of processing of many tasks that AI can do, and which can be categorised as follows:

- Image Classification;
- Object detection and localization;
- Image segmentation;
- Image retrieval;
- Image denoising/restoration;
- Image super-resolution.

In the last decade, a big progress has been made in:

- Image generation.

The field to which these methodologies belong is described in Restack.io. as: "Computer vision is a field of artificial intelligence (AI) that enables computers to interpret and extract meaningful information from digital images and videos". It involves using machine learning and neural networks to analyse visual data, allowing systems to make decisions or recommendations based on what they "see". This interdisciplinary field combines aspects of AI and image processing to help machines gain a high-level understanding of visual inputs".

To clarify the meaning of each AI methodology, a concrete example of application in architecture or urban planning is provided for each.

Image classification. A restoration researcher would be interested in classifying a building, or a part of it, or another object based on a photograph of it. This classification would relate to the time of its construction, the style used, and other factors that together create a labelling of each object. An architect would also be interested in classifying his collection of drawings into different categories and, in turn, find for each design, which classification category it would belong to. Machine Learning, part of AI, provides a solution to this problem. The idea is that the machine has learned to make this classification through a model (composed of algorithms) and a considerable number of already classified data points, and then, for each unclassified data point, it can classify it with a certain certainty or predict a classification.

Object detection and localisation. Localisation of an object refers to the identification of its position within an image. As an example of the application of this process, the Unetiq project in Munich is worth mentioning, where architects design the layout of the buildings' land and hand it over to subcontractors. Electrical engineers perform specialised planning tasks. The earth plans are designed with Computer-Aided-Design (CAD) software. They are transformed into a separate file format comprising basic geometric shapes such as lines, dots, and rectangles. The project aims to identify and locate objects in a plan.

Moreover, to enable algorithms to understand the semantic structure of a given plan, contextual information is required for different rooms and objects, for example, the position and size of a kitchen sink, as well as the use of Deep Learning (AI) techniques such as Region Convolutional Neural Networks (R-CNN) [10].

Another example of the application of object localisation is the analysis of "patterns" in pedestrian and car movements to inform city planning decisions that shape infrastructure development and meet community needs.

The implementation of this analysis requires data that, in this case, is video data from many sources, including cameras, to create datasets for training models as well as AI techniques to perform these models, which are the YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), the basis of which is the CNN algorithm.

Image segmentation. Image segmentation is the separation of objects, contours, or structures within an image for further analysis. Here, semantic segmentation assigns a class label to each pixel in an image, whereas instance segmentation goes beyond this by identifying and delineating each object. It distinguishes between separate objects of the same class. This is where deep learning algorithms are used.

Regarding the architecture, in Li et al 2023, it is shown that segmentation is the basis of so-called generational AI, and specifically related to several categories of work: 1) generating facade images based on semantic segmentation maps; 2) use of AI generating models for transferring facade image style; 3) generating semantic segmentation maps of the facade based on images. UNET is one of the algorithms the AI uses to segment an image, but its core is the CNN.

Image retrieval. A study by Condorelli (2023) presents a methodology for using photogrammetry-derived imaging data to create 3D models. A combination of AI and photogrammetry has been used specifically for cultural heritage images. Photogrammetry includes multiple photos of a building or structure to create a 3D model. This study aims to expand imaging datasets by increasing the quantity and quality of photogrammetric imagery, resulting in more accurate and detailed 3D models. AI algorithms are used to generate additional images from existing ones.

Some deep learning AI algorithms have been used here, including the visual search engine, which again uses Convolutional Neural Networks (CNNs). They are used for image classification and similarity comparison.

Image denoising/restoration. Removing noise from an image means improving its quality by eliminating noise sources, which, according to Goyal (2024), arise from various natural causes and are difficult to identify and avoid.

According to the same article, the CNN algorithm has achieved excellent results in image recognition, and Chiang and Sullivan were the first to use CNN (deep learning, a specialised subset of ML) to denoise the image.

Image super-resolution. It is stated that the abstraction of buildings from images is extremely important for numerous applications, including urban planning and management, 3D modelling of cities, and change detection. In this context, satellite and aerial images are widely used. Despite numerous successful studies in the literature, this task remains challenging and complex due to several factors that complicate imaging. The U-Net algorithm is applied to the project described in this paper in several variants. This algorithm is a variant of the CNN algorithm.

Image generation. The architect or urban player has a series of building sketches, photos, and photos of the real buildings. This set of images (sketch, photo) can be used to train the machine to predict the view of a building from a given sketch. This approach, called pixel-to-pixel, is explained in detail in TensorFlow. Another approach is to generate image versions from a set of images provided to the machine for training.

I.1 What is Common in Image Processing Methodologies?

DL (deep learning) is a subset of ML (machine learning), which itself is a subset of AI. It is DL that is applied in image processing. It is called deep learning because it involves neural networks with many layers. The data and images are supplied to algorithms that learn from the most informative features, enabling further predictions.

The paper focuses on two aspects that it considers essential for a beginner to start applying image processing methodologies.

- Some ways to create a customised dataset of images,
- The Convolutional Neural Network (CNN) model, used to extract the most expressive features of an image, has the same base logic as many other DL algorithms for image processing.

As a last note, for the Python implementations, Keras is used, a high-level neural network API that runs on top of TensorFlow, one of the most popular deep learning frameworks used in both academic research and commercial applications.

II. DATASETS AND DATA PREPROCESSING

II.1 Use of a dataset of images provided by Python libraries

Python is a programming language specialised in the application of algorithms and machine learning methodologies, as it includes libraries of specialised programs for various operations.

For these methodologies, Python provides ready-made datasets, and users must build and train the model. Datasets are structured data collections organised in table form, where each column represents a variable and each row a record. They also support a variety of methods that perform different actions on datasets, such as uploading data and selecting data for training, validation, and testing. Training, validation, and test data are three categories of data, usually used by ML and DL algorithms.

Among the datasets offered by Python, there are image datasets such as MNIST in K**eras.datasets** package [18] of Python. This dataset is a collection of 60,000 handwritten digit images, with dimensions 28x28, and also includes 10,000 test images. These datasets are very useful for people to learn different image processing techniques at the beginning, before using their customised datasets. The following code shows how such a dataset, provided by the Python K**eras** package, can be used.

```
#import the necessary libraries
import matplotlib.pyplot as plt
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras import utils

#data loading from mnist dataset
(training_data,train_labels), (validation_data, validation_labels)= mnist.load_data()
'''
Each image have size of 28*28 pixels; that is 28 pixels height and 28 pixels width and hence (1, 28, 28), 1 in first part is to specify color depth of the pixel. 1 is for greyscale image(black and white image).
(60000, 1, 28, 28)- is the shape of matrix the data have since we have 60000 images in which 28*28 pixels and each pixel have a value in this matrix. astype('float32') convert data in float format; this makes computation more convenient.
```

```
'''
training_data = training_data.reshape(training_data.shape[0],28,28).astype('float32')
validation_data = validation_data.reshape(validation_data.shape[0],
'''
Normalization of the data. The most common pixel format is the byte image, where this number is
stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to
be black, and 255 is taken to be white. Values in between make up the different shades of gray.
Dividing an image by 255 simply rescales the image from 0-255 to 0-1
'''
training_data = training_data /255
validation_data = validation_data /255
```

Of course, the architect or urban planner's interest would be to create a customised dataset with his images. In the following are some ways to create customised datasets.

II.2 Some methodologies to create a customised dataset

A dataset is the main component of machine learning methods. The quality of the images during their creation and a preliminary treatment of them substantially affects the outcome we receive. These steps are detailed in Kokorin (2023). One of the recommendations given here is that if the dataset is over 100K images with many classification classes, it is recommended to use a structured database where the main data could be:

- file name;
- file path;
- annotation data (coordinates of the bounding box position, and some relative coordinates);
- class data.

The case of database use is not the object of this study. It is worth noting an important element in this preliminary image treatment: "Data Augmentation," a technique for creating new images from existing ones by applying several transformations. This way, the dataset is expanded with new images, leading to more accurate results. These transformations, according to GeeksforGeeks (2023 are:

- Rotation (rotates the image by a certain specified degree).
- Shearing (transforms the image's orientation).
- Zooming (zoom in or zoom out).
- Cropping (crops the image or selects a particular area from an image).
- Flipping (flips the orientation of the image).
- Changing the brightness level (helps to combat illumination changes).

There are different ways to realise a dataset, and some of them are presented here.

Keras ImageDataGenerator with flow_from_directory(). According to StudyMachineLearning, using Keras' ImageDataGenerator class and flow_from_directory method allows loading images

from a hierarchy of folders, where the folder structure determines the image classification. The following image depicts the classification structure we will use for machine training.

```
train/
...classA/
.....a_image_1.jpg
.....a_image_2.jpg
. . . . .
...classB/
.....b_image_1.jpg
.....b_image_2.jpg
. . . . .
testing/
```

The ImageDataGenerator class provides several parameters to control scaling, the proportions of the training and validation sets, and data augmentation. In TensorFlow there are parameters that the constructor of this class can use:

flow_from_directory(): Takes the path to a directory and generates batches of augmented data. The full set of parameters of this method is given in AI Learner (2019.

The preprocessing of the data until the preparation of the training, validation and testing sets would be presented with these lines, where only a part of the possible parameters were used:

```
#imports
from keras.preprocessing.image import ImageDataGenerator
src_path_train = '…/train/'  # see the folder hierarchy above
src_path_test =' …testing/'
img_width, img_height = 100, 100 #image dimensions
batch_size=8 #number of images in each batch
train_datagen = ImageDataGenerator(
    rescale=1 / 255.0,  #normalization of the data
    # Data Augmentation parameters
    rotation=20, zoom=0.05,width_ shift=0.05, height_shift =0.05,
      shear_range=0.05, horizontal_flip=True,
    validation_split=0.20
        #proportion of data split into training and validation data
)
train_generator = train_datagen.flow_from_directory(
    directory=src_path_train, #path of the training data folder
    target_size=( img_width, img_height),  color_mode="rgb",
          #imazhe features, color_mode=3 channels
    batch_size=batch_size
    class_mode="categorical", subset='training',  shuffle=True)
     #shuffle = True, the dataset will be randomly shuffled to avoid
     #any overfitting in training. It is True only for training and validation set
validation_generator = train_datagen.flow_from_directory(
   . . . the same parameters like train_generator )
test_datagen = ImageDataGenerator(rescale=1 / 255.0)
```

```
test_generator = test_datagen.flow_from_directory(
  directory=src_path_test, #path of the test data folder
  #the same parameters for the image
  batch_size=1,   class_mode=None,  shuffle=False,)
```

flow_from_directory(directory), description: Takes the path to a directory, and generates batches (sets) of augmented/normalised data.

Labelling the images, writing them into a CSV file and using this .csv file as a dataset. Another way to create a customised dataset is to convert images to numerical data and save them in a .csv file. In Cswah (2018), there is a description of the steps necessary to read the images from a folder and to write them into a .csv file, associating them with a classification label:

- Gather images for the dataset in different folders corresponding to their classes.
- Rename the pictures according to their classes.
- Merge them into one folder.
- Resize the pictures.
- Convert all images into the same file format.
- Labelling the images.
- Convert images into a CSV file.
- A few tweaks to the CSV file.
- Load the CSV (BONUS).

This sequence of steps is kept as in the original material of Baheti (2023). However, the manual steps 5, 6, 7, and 8 are converted to code, and another step is added: a preprocessing step that produces training, validation, and test data for the machine learning algorithm. The following steps represent what is mentioned:

a) Data preparation

- Gather images for your dataset; assume all images are .jpeg files. The images are stored in different folders based on their classification.
- Rename the images according to their classes:
  - Select all the images in the folder
  - Rename the images, for example, with the name class_A. The names will be the same, but associated with (1), (2).
- Merge all the files into one folder

b) Image format unification

The next steps are done by code: image resizing, image labelling, writing to a CSV file, and preprocessing that results in training data, validation data, and test data.

```
#Resize all the images with the same format, this function is used in writetocsv():
```

```
def resizeimage(imgstart): #imgstart original image
    # size of original image
    width, height = imgstart.size
    # Setting the points for cropped image
    left = 6
    top = height / 4
    right = 174
    bottom = 3 * height / 4
    # Cropped image, the original is not changed
    imgend = imgstart.crop((left, top, right, bottom))
    newsize = (48, 48)
    imgend = imgend.resize(newsize)
    return imgend
```

c) Data Labeling

```
#Labeling the images, this function is used in writetocsv():
import os
def createFileList(myDir, format='.jpg'):
  fileList = []
  labels = []
  names = []
  # classify the images by a common letter of image name
  # the names here are classA(#).jpeg, classB(#).jpeg
  keywords = {"A": "1", "B": "0", }
  # os.walk() generates the file names in the path specified
  for root, dirs, files in
      os.walk(src_path_train, topdown=True):
      for name in files:
        if name.endswith('.jpeg'):
          fullName = os.path.join(root, name)
          fileList.append(fullName)
        for keyword in keywords:
          if keyword in name:
            labels.append(keywords[keyword])
          else:
            continue
        names.append(name)
  return fileList, labels, names
```

d) Writing into .csv file

```
import csv
```

```
from PIL import Image
import pandas as pd
import numpy as np
# Write image data into a CSV file
#Create three lists with images,labels and their names
def writetocsv():
    myFileList, labels, names = createFileList(src_path_train)
     i = 0
    for file in myFileList:
      # for each image
         img_file = Image.open(file)
        # image is resized
          img_file=resizeimage(img_file)
        # the original parameters
          width, height = img_file.size
          format = img_file.format
          mode = img_file.mode
        # Make image Greyscale
          img_grey = img_file.convert('RGB')
        # convert input(pixels) to an array
          value = np.asarray(img_grey.getdata(),dtype=int).
                    reshape((width, height, 3))
        # convert array value into one dimensional array
          value = value.flatten()
        # join all pixels into one string with space in between
          s = ' '.join(str(x) for x in value)
        # create two columns: label, pixels
          value = np.append(labels[i], s)
          i += 1
        #filename=src_path_train+'classescsv.csv'
         filename="/classescsv.csv"
         with open(filename, 'a') as f:
            writer = csv.writer(f)
            writer.writerow(value)
       #add some header
         headerList = ['Labels', 'Pixels']
         data = pd.read_csv(filename)
         # converting data frame to csv  assing a header
         data.to_csv(filename, header=headerList, index=False)
```

e) Transform the Image information from a sequence of numbers into an array of float numbers

```
import numpy as np
```

```
# Image preprocessing  used in gettrainvalidationdata():
def imagesetting(data):
    image_size = (48, 48, 3)# RGB image
    width, height, color = 48, 48, 3 #RGB image
    imgspixels = []
    for pixel_sequence in data:
        # get the pixel data from pixel column in .csv
        I  mgpixels = [int(pixel) for pixel in  pixel_sequence.split(' ')]
        # create an array of pixels for the image
        imgpixels =  np.asarray(imgpixels).reshape(width, height, color)
        # Convert the pixel values of an image to uint8
        imgpixels =  np.resize(imgpixels.astype('uint8'), image_size)
        # pixel values of an image are converted this type
        # and added into an array
        imgspixels.append(imgpixels.astype('float32'))
  imgspixels = np.asarray(imgspixels)
  return imgspixels
```

Note: imgpixels.astype('uint8') is used for memory efficiency (because this format uses 1 byte per pixel), for standardisation (because image processing libraries use this format), and for performance of different operations.

f) Reading the .csv file and creating three sets of data: training data, validation data and test data

For some image processing tasks, such as image classification, three data sets are used: the training set, the evaluation set, and the test set. Training data is what the model uses to learn patterns. During this process, another set of data, the validation set, is used instead of the training set and serves to monitor performance during training. After training, the model is tested using another set, called the test set. In practice, all the data can be divided into 80% for training and 20% for validation. The test data can be data outside of the dataset.

The following Figure represents the role of each of these sets:

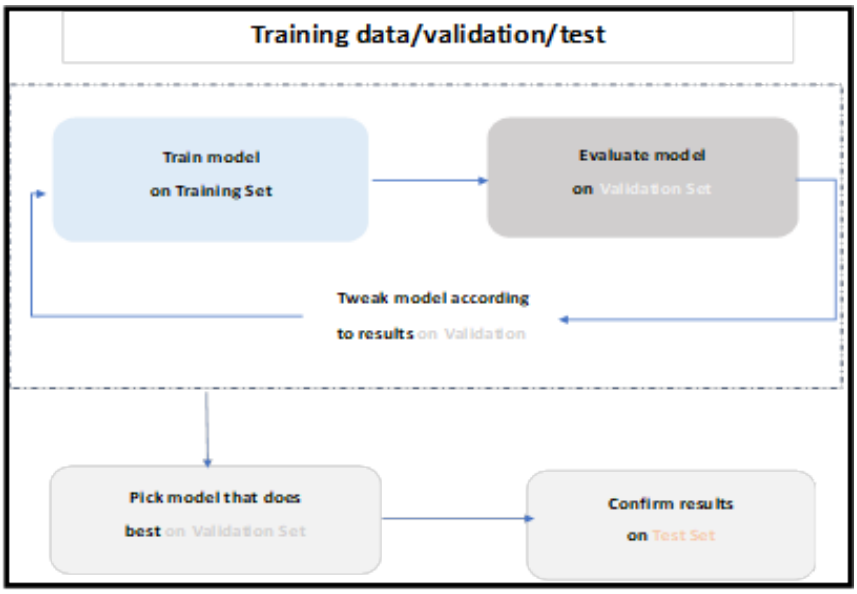


Figure 11. Training, validation, test data

Source: https://www.v7labs.com/blog/train-validation-test-set

```
# definition of the training, validation and test sets
 import pandas as pd
 from sklearn import model_selection
 def gettrainvalidationdata():
        Corpus = pd.read_csv(r"\classescsv.csv")
        X_train, rest_data,y_train, y_rest_data =
          model_selection.train_test_split(Corpus['Pixels'],
            Corpus['Labels'], train_size=0.8, shuffle=False)
        X_validation, X_test, y_validation, y_test =
          model_selection.train_test_split(rest_data,y_rest_data,
            test_size=0.5, shuffle=False)
        X_train = imagesetting(X_train)
        X_test = imagesetting(X_test)
        X_validation = imagesetting(X_validation)
        y_train = utils.to_categorical(y_train)
        y_test = utils.to_categorical(y_test)
        y_validation = utils.to_categorical(y_validation)
        print('y_train.shape[1]', y_train.shape[1])
        num_classes = y_train.shape[1]
        return X_train, X_test, y_train,
```

y_test,X_validation,y_validation, num_classes

Use images from a folder and add labels programmatically. In some cases, it is not possible to use Excel to create a dataset for problems that require images with associated labels. Excel has its own limitations; for example, the maximum number of columns is 16,384, which is not enough to store images of size 64x64. In this case, the images could be labelled programmatically. A practical way is to keep all the images in one folder, with regular filenames, and have a common part of each filename for each class. This part would be used by a program to label the image, as shown in the code below.

```
def createLabels():
    fileList = []
    labels = []
    names = []
    keywords = {"me": 1, "pa": 0, }  # keys and values to be changed as needed
    for root, dirs, files in os.walk(". . ./images", topdown=True):
        for name in files:
            if name.endswith('.jpg'):
                fullName = os.path.join(root, name)
                fileList.append(fullName)
            for keyword in keywords:
                if keyword in name:
                    labels.append(keywords[keyword])
                else:
                    continue
            names.append(name)
    return labels

def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img)
    img = tf.image.resize_with_crop_or_pad(img, img_size, img_size)
    img = tf.cast(img, tf.float32)
    img = (img - 127.5) / 127.5
    return img

def tf_dataset(images_path):
    dataset = tf.data.Dataset.from_tensor_slices(images_path)
    dataset = dataset.shuffle(buffer_size=10240)
    dataset = dataset.map(load_image)
    return dataset
```

```
images_path = glob("C:\Polis\Kenti\images\Imagemeetikete\*")
X_train = tf_dataset(images_path)
print('images_dataset',len(X_train))
y_train = createLabels()
print('y_dataset',len(y_train))
y_train = tf.data.Dataset.from_tensor_slices(y_train)
print('X_train',X_train.take(0))
dataset = tf.data.Dataset.zip(X_train, y_train)
dataset = dataset.shuffle(buffer_size=1000).batch(batch_size)
```

Use images from a folder when labels are not needed (for image generation). As we mentioned above, unsupervised approaches, such as image generation, do not require associating images with labels. The following code provides a dataset of images for further processing.

```
#Import TensorFlow and Other Libraries
from glob import glob
import tensorflow as tf
from tensorflow.keras.layers import *
from matplotlib import pyplot as plt
IMG_H = 64
IMG_W = 64
IMG_C = 3
batch_size = 128
latent_dim = 128
 images_path = glob(". . ./train/*")

def load_image(image_path):
   img = tf.io.read_file(image_path)
   img = tf.io.decode_jpeg(img)
   img = tf.image.resize_with_crop_or_pad(img, IMG_H, IMG_W)
   img = tf.cast(img, tf.float32)
   img = (img - 127.5) / 127.5
   return img

def tf_dataset(images_path, batch_size):
   dataset = tf.data.Dataset.from_tensor_slices(images_path)
   dataset = dataset.shuffle(buffer_size=10240)
   dataset = dataset.map(load_image,
            num_parallel_calls=tf.data.experimental.AUTOTUNE)
   dataset = dataset.batch(batch_size)
   dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
   return dataset
```

III. CONVOLUTIONAL NEURAL NETWORK (CNN) MODEL AND ITS IMPLEMENTATION.

There are many papers and materials on the Convolutional Neural Network, which is used specifically in image processing and is known for its high efficiency. In recent years, supervised and unsupervised learning with convolutional networks (CNNs) has seen widespread adoption in computer vision applications.

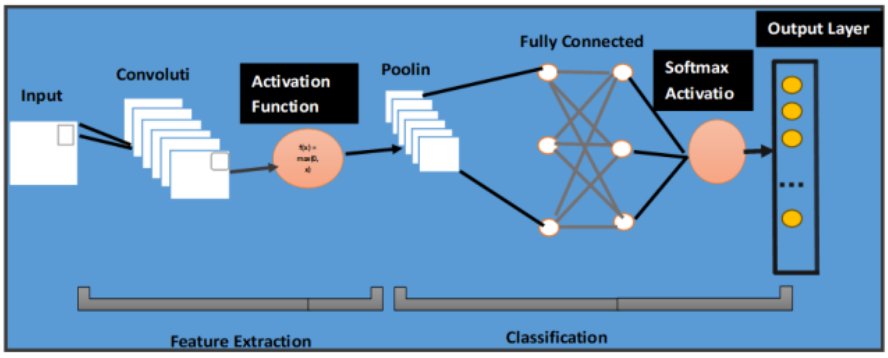


Figure 2. Block diagram of the CNN model

Source. https://blog.talent500.co/wp-content/uploads/2024/03/images-19-1.png

The idea of this model is to extract image features from images that will serve to build a law that defines image patterns when this model (features and law) is trained on a large number of images, so-called training images.

To clarify how this model works, we take as an example one of the machine learning methodologies that uses it: image classification into several classes or categories. Machine learning classification methods aim to create a model or function that, using a large dataset of images, learns a pattern for each image category. Then, each new image is compared with these patterns, and the closest one determines its category. This function or model is not derived from a prior assumption; rather, it is built on the training data and some operations that define its laws. Therefore, such a model belongs to the deep learning family of methods.

In CodezUp, several papers present this model by building it step by step, creating a so-called layer at each step, where each layer has an input and produces an output as a result of an algorithm. The basic layers are:

1. Data Preprocessing
2. Convolutional Layers
3. Pooling Layers
4. Flatten Layers

5. Fully Connected Layers

Below are the steps of this model, illustrated in Fig. 1, implemented in Python.

Input and Preprocessing. For images, the input data are pixels. Therefore, an image is presented as a matrix of numbers representing the colour intensity at each coordinate. However, the number of these values is extremely large as the input variable. The CNN model, using several algorithms, extracts the most determining qualities of an image by reducing the large number of qualities.

Convolutional Layer and Function: **ReLU (Rectified Linear Unit).** The defining qualities in relation to images are the image's most significant colour intensities. The methodology for determining these defining qualities consists of applying a matrix, called a kernel or filter, to the square of the image's pixels, with the same dimensions, element-by-element. This kernel matrix is initialised to random values, but over time, its values are learned. This matrix operates by sliding in steps across the entire pixel space of the image.

The result will again be a matrix with smaller dimensions than the initial pixel matrix, and the change in size depends on the sliding window step. The result is called a feature map, and its components are neurons. For simplicity of calculations, we are considering Figure 2 which shows the pixels of a three-color image, with dimension (5x5x3) and a kernel with dimension (3,3,3), where the third dimension is presented into three 3x3 planes, where each of them, operates with one of the colors of the image (red, green, blue), which in the language of the algorithm are called channels. So, there are 3 of them. The results are summarised , and the final result presents, in a summarised way, the representative quality of all the qualities of the squares in the three channels of the image pixels.

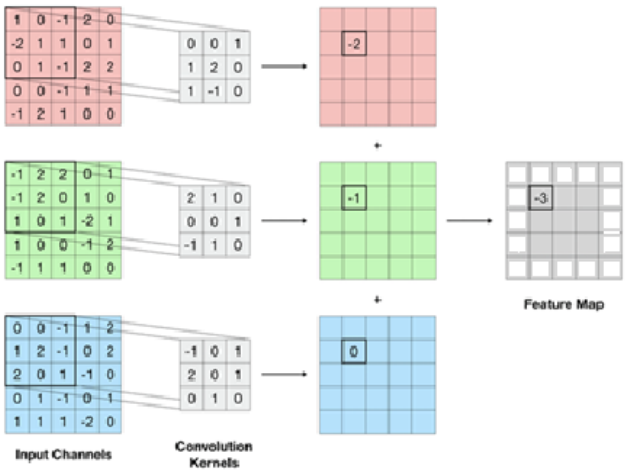


Figure 3. Convolution operation demonstration

Source: https://setosa.io/ev/image-kernels/

This is how the value -3 is calculated

The result from first kernel is:

$$1 * 0 + 0 * 0 + (-1) * 1 + (-2) * 1 + 1 * 2 + 1 * 0 + 0 * 1 + 1 * (-1) + (-1) * 0 = -2$$

Form the second one the calculation gives the result:

$$(-1) * 2 + 2 * 1 + 2 * 0 + (-1) * 0 + 2 * 0 + 0 * 1 + 1 * (-1) + 0 * 1 + 1 * 0 = -1$$

And from the third one the result is:

$$0 * (-1) + 0 * 0 + (-1) * 1 + 1 * 2 + 2 * 0 + (-1) * 1 + 2 * 0 + 0 * 1 + 1 * 0 = 0$$

Summarising the above results it is taken: $-2 + (-1) + 0 = -3$

Then a (3x3x3) kernel or filter defines a feature map with dimension (3x3) for a 5x5x3 image.

In CNN commands, a number of filters is defined as 16, 32, 64, etc. If 32 such filters were to operate as above, they would generate 32 feature maps with dimensions 3x3 in this case, which would be considered the input for further operations, called channels. So initially, we had 3 colour channels, and then the components of the created feature map were used as channels.

The dimension of the feature map, as we see, is smaller than that of the input matrix. Since we want to keep the feature map at the same dimension as the initial image, we complete the pixel matrix to the same dimension by padding with zeros.

Using an input matrix with a contour consisting mostly of zeros allows the example above to maintain the dimension of the input image matrix. In algorithm terminology, this is called padding, and considering this element, the CNN uses the following formula to determine the dimension of the feature map:

The output feature map size is calculated as follows:

$$O = \lfloor \frac{n - f + 2 \cdot p}{s} \rfloor + 1$$

were,

- O: Output size
- n: Input size (height or width)

- f: kernel size
- p: Padding
- s: Stride

Padding based on is calculated by the formula

$$if\ n\%s = 0:$$

$$p = max(f - s, 0)p$$

$$else:$$

$$p = max(Fh - (n\%s),0)$$

In this case, p=3-1=2

Applied to Fig 2, it would be

$$O = \lfloor \frac{5 - 3 + 2 \cdot 1}{1} \rfloor + 1 = \lfloor 4 \rfloor + 1 = 5$$

A so-called activation function is applied to this result. Its goal is to provide a nonlinear relationship between the input and output that better reflects the reality of complexity. There are different activation functions, such as Sigmoid, Tanh, Softmax, ReLU, and Softplus.

In this implementation, the ReLU function is chosen, and the graph is shown in Figure 3.

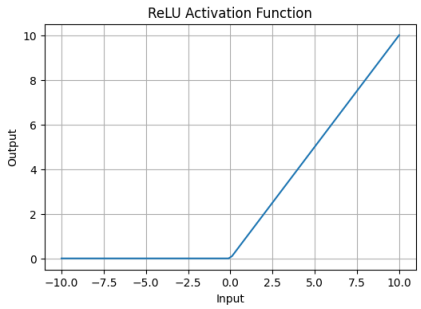


Figure 4: Visualisation of the Rectified Linear Unit (ReLU) Activation Function

Source: Author processing

Expressed by a formula, this functionality is $f(x) = max(0, x)$. So, for the value of the feature map in Figure 2, the result is f(-3)=0.

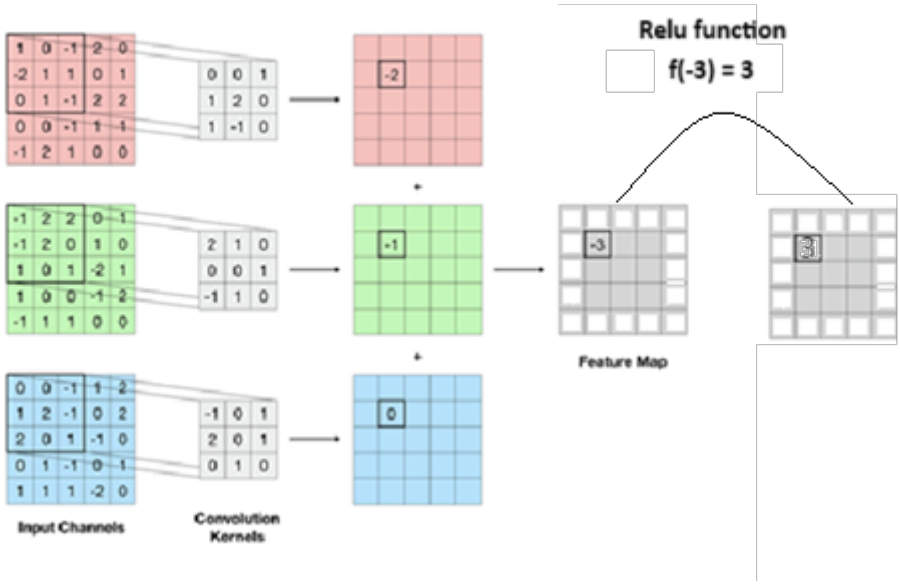


Figure 5. Convolution Process Across Input Channels Followed by ReLU Activation in a CNN

Source: Author's processing

Convolutional Layer Implementation. What is mentioned is implemented in Python by the following lines.

```
model=Sequential()
model.add(Conv2D(32,3,1, padding='same', input_shape=(48,48,3),

activation='relu'))
```

Here, the feature map is created using 32 3x3x3 kernels, with padding to maintain the same output dimension as the input. Conv2D is used for the 3-dimensional images (RGB). The output dimension is = $\lfloor\frac{48-3+2*1}{1}\rfloor + 1 = 48$ and there are 32 channels. A formula in GeeksforGeeks calculates the number of parameters.

Parameters=(kernel_w x kernel_h x channels+1)× filters=(3 x 3 x 3+1) x 32= 896

Where:

- kernel_w = kernel width
- kernel_h = kernel height
- channels = number of input channels
- filters= number of filters (output channels)

This is the result that model.summary() gives as well.

Model: "sequential"

Layer (type)	*Output Shape*	*Param #*
conv2d (Conv2D)	*(None, 48, 48, 32)*	*896* . . .

Pooling Layer: **Max Pooling/Average Pooling.** After creating a convolutional layer, an operation is applied to the feature maps to reduce their dimensions. The following Figure shows how it works. The square of values in feature maps is replaced with the maximum or average of their values. An example is shown in Figure 6.

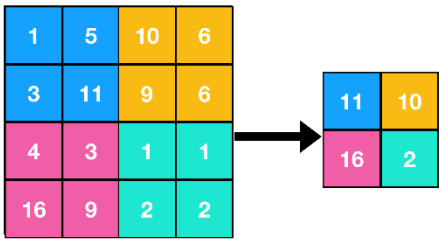


Figure 6. Fully connected layers & SoftMax function output

Source: Authors processing

Then the result's dimensions are reduced. The formula in[34] calculates the output feature map size:

$$O = \left(\frac{h-f}{s}+1\right) X \left(\frac{w-f}{s}+1\right) x\ c$$

where,

- O: Output size
- h: height of feature map
- w: width of feature map
- f: size of pooling filter
- c: channels, or the depth of the output
- s: Stride length

Python implementation

model.add(MaxPooling2D(pool_size=(2,2), padding='same'))

Note:

In the case of odd dimensions, padding can be used by adding a column on the right and a row on the bottom.

This case would be implemented by

model.add(MaxPooling2D(pool_size=(2,2), padding='same', ceil_mode=true))

In this implementation, the formula gives $\frac{h-f}{s}+1=\frac{48-2}{2}+1=24$, which can be shown by model.summary()

Model: "sequential"

Layer (type)	*Output Shape*	*Param #*
conv2d (Conv2D)	*(None, 48, 48, 32)*	*896*
max_pooling2d (MaxPooling2D)	*(None, 24, 24, 32)*	*0*

Flatten Layer

In this step, the feature maps are converted into a 1D array, and the implementation in Python is:
model.add(Flatten())
The result shown by the model summary()

Model: "sequential"

Layer (type)	*Output Shape*	*Param #*
conv2d (Conv2D)	*(None, 48, 48, 32)*	*896*
max_pooling2d (MaxPooling2D)	*(None, 24, 24, 32)*	*0*
flatten (Flatten)	*(None, 18432)*	*0* . . .

Fully Connected Layer, or Dense Layer. The 1D array that comes from the Flatten function step represents the most important features that define the image and is used as input for functionalities that predict each image's class label. The following Figure shows the operations in this step:

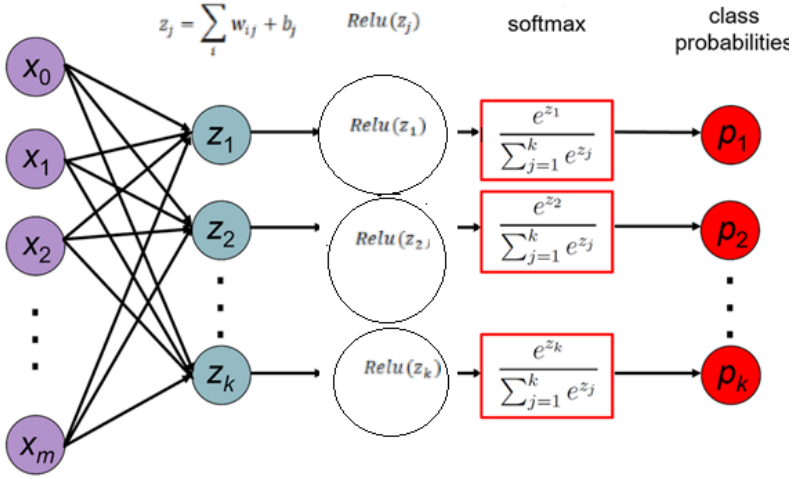


Figure 7. Operations in steps

Source: Adapted from

https://purwadhika.com/blog/neural-network-implementation-for-image-classification-using-cnn

Figure 7 shows a very defining element in the neural network. Here, there are two layers of neurons, where each element of the first layer from the previous step is connected to each element of the second layer, the number of which equals the number of classes that will categorise the images. The equation expresses this connection.

$$z_j = \sum_i \quad w_i \cdot x_i + b_j$$

where the input neurons $x_1, x_2, \ldots$ the output neurons $z_1, z_2, \ldots$ are determined by means of the so-called weights w_i and biases b_j. The weights show the influence of each input neuron on that output and the bias defines a type of regulator regarding the dependence $z_j = \sum_i \quad w_i \cdot x_i$.

Then it operates the Relu function that returns all z_j values to positive and the softmax function that determines the probability for each output neuron in the form of an expression

$$Softmax(z_j) = \frac{e^{z_j}}{\sum_{l=1}^{k} \quad 1e^{z_l}}$$

The following two lines represent the implementation of the above operations:

```
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

A very clear explanation of the Dense layer is given in [37]. The weights and biases are learnable parameters that the neural network adjusts during training to minimise the loss function.

In this case, the dimensionality reduction happens because the weight matrix in the Dense layer has a shape of (*18432*, 128). Therefore, there are *18432* neurons (input_dim), reduced to 128 neurons (output_dim). The weight matrix would have a shape of (*18432*, 128). The result would be a 128-dimensional vector, effectively reducing the input dimensionality.

After the linear operation, in this layer an activation function is typically applied to introduce non-linearity into the model, allowing the network to learn complex patterns. Here are two Dense layers. The first reduces the output dimension to 128, and the second gives an output dimension equal to the number of classes. For each class, the Softmax activation function computes the probability.

During the process described above, the concept of learning is mentioned twice: once regarding the values of weights and biases related to the filter values in the convolutional layer, and once regarding the values of weights and biases related to the dense layer, which are called model trainable parameters. The process of learning is based on a function, called a loss function, that measures the error between the true distribution of class values in the training set and the distribution of class values the model outputs.

Based on the values returned by the loss function, the above process is repeated a number of times, and before each repetition, the trainable parameters are updated. One way to define this loss function and some other parameters for the model is the following

```
model.compile(loss='categorical_crossentropy', optimizer='adam',  metrics=['accuracy'])
```

Shortly:

loss='categorical_crossentropy' measures the difference between the predicted probability distribution and actual (true) class distribution. Categorical Cross-Entropy (CCE), also known as softmax loss or log loss, is one of the most commonly used loss functions in machine learning, particularly for classification problems.

optimizer='adam' activates a method that adjusts the learning rates of parameters during training.

metrics=['accuracy'] allows the use of several accuracy metrics, in this case the Keras Accuracy metric, to evaluate the performance of machine learning models.

The completed information given by model.summary() would be

Model: "sequential"

Layer (type)	*Output Shape*	*Param #*
conv2d (Conv2D)	*(None, 48, 48, 32)*	*896*
max_pooling2d (MaxPooling2D)	*(None, 24, 24, 32)*	*0*
flatten (Flatten)	*(None, 18432)*	*0*
dense (Dense)	*(None, 128)*	*2359424*
dense_1 (Dense)	*(None, 2)*	*258*

Total params: 2360578 (9.00 MB)
Trainable params: 2360578 (9.00 MB)
Non-trainable params: 0 (0.00 Byte)

All that is mentioned above defines the construction of the CNN model. The model creates patterns using the training and validation data, and it can be tested on test data. The lines implement these two moments:

```
cnnmodel.fit(X_train, y_train, validation_data=(X_validation,y_validation),epochs=16, batch_size=10, verbose=2)

score= cnnmodel.evaluate(X_test, y_test, verbose=0)
```

It is the function fit() that performs the training, and to do so, the data are divided into "batches" of size 10. This process is repeated 16 times, defined by the number of epochs. It is just these iterations that help learn better which parameters work inside the model, improving them at each iteration. The model is trained on the training set, and evaluation is performed on the validation set after every epoch.

Some notes about improving the model. The CNN model's presented layers are the main ones it uses to extract images and create patterns. Nevertheless, the studies and practical experiences show that the same ways can improve this model:

- Using some convolutional layers (implemented by conv2d())
- Adding the dropout layer (implemented by dropout())
- Using some fully connected layers (implemented by dense())

Dropout is a technique that helps avoid overfitting, making the model more general and better adapted to reality. Moreover, this is achieved by randomly dropping some outputs during training. The figure below shows this trick.

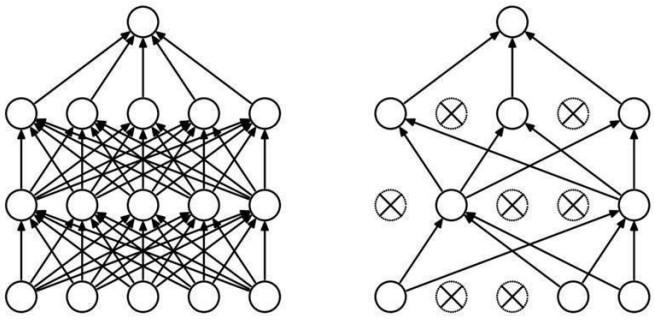


Figure 8. Effect of Dropout on Neural Network Architecture

Source: Authors processing based on LearnDataSc

In this paper, several models and evaluation metrics are used, and the following results are shown: accuracy, classification report, and confusion matrix.

The models:

```
#Model1

model = Sequential()
model.add(Conv2D(32, 3,  padding='same', input_shape=(48, 48, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

```
#Model 2

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(48, 48, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu')),
model.add(Dense(num_classes, activation='softmax'))
```

```
#Model 3

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(48, 48, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu')),
model.add(Dense(num_classes, activation='softmax'))
```

III. SOME RESULTS AND INTERPRETATIONS

These models are trained on a dataset of road images. The goal is to classify the data into irregular and regular roads. The labelling is done with 1 for irregular roads and 0 for regular roads. There are

different metrics to evaluate classification results, such as accuracy, classification report, and confusion matrix. Accuracy is included in the classification report.

Three indicators are elements of a classification report: precision, recall, and F1 score for each class in a classification problem. Each of these indicators helps provide a complete understanding of classification performance.

The classification report for the database of road images and for the labelling done represents the following indicators:

$$precision_{class0} = \frac{correctly\ predicted\ broken\ roads}{total\ \ broken\ roads}$$

$$recall_{class0} = \frac{correctly\ predicted\ broken\ roads}{predicted\ as\ broken\ (broken+not\ broken)}$$

$$f1 - score_{class0} = 2 * \frac{precision*recall}{precision+recall}$$

The confusion matrix shows four outcomes. For the dataset used in this application, the confusion matrix gives this information:

$$r_{00=}regular\ road_{classified\ as\ not\ regular}$$

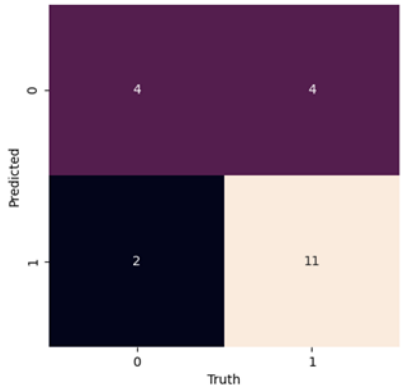
$$r_{01=}iregular\ roads_{classified\ as\ regular}$$

$$r_{10=}regular\ roads_{classified\ as\ iregular}$$

$$r_{11} = iregular\ road\ _{classified\ as\ iregular}$$

Model 1

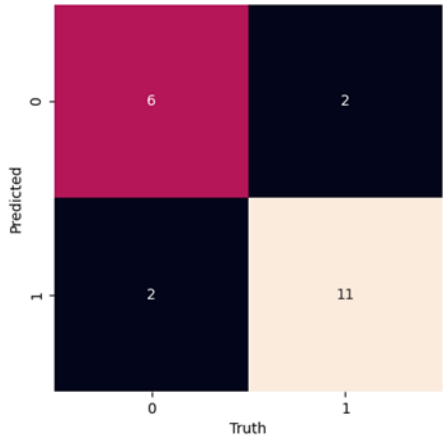
Classification	Report			
	precision	recall	f1 score	support
0	0.67	0.5	0.57	8
1	0.73	0.85	0.79	13
accuracy			0.71	21
macro avg	0.7	0.67	0.68	21
weighted avg	0.71	0.71	0.7	21



Confusion Matrix

Model 2

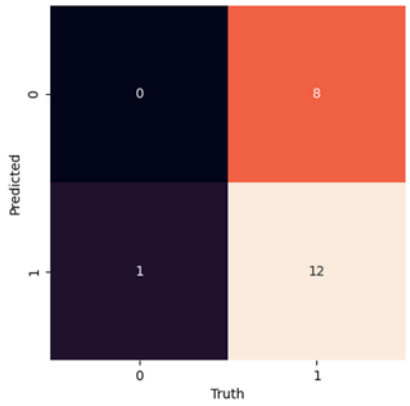
Classification	Report			
	precision	recall	f1-score	support
0	0.75	0.75	0.75	8
1	0.85	0.85	0.85	13
accuracy			0.81	21
macro avg	0.8	0.8	0.8	21
weighted avg	0.81	0.81	0.81	21



Confusion Matrix

Model 3

Classification	Report			
	precision	recall	f1-score	support
0	0	0	0	8
1	0.6	0.92	0.73	13
accuracy			0.57	21
macro avg	0.3	0.46	0.36	21
weighted avg	0.37	0.57	0.45	21



Confusion Matrix

Where the calculations (for instance, for class 1, for Naive Bays, in the first model) are as follows:

$$precision_{class0} = \frac{correctly\ predicted\ broken\ roads}{total\ \ broken\ roads} = \frac{11}{15} = 0.73$$

$$recall_{class0} = \frac{correctly\ predicted\ broken\ roads}{predicted\ as\ broken\ (broken+not\ broken)} = \frac{11}{13} = 0.85$$

The f1-score is the harmonic mean of precision and recall:

$$f1 - score_{class0} = 2 * \frac{precision*recall}{precision+recall} = 2 * \frac{0.73*0.85}{0.73+0.85} = 0.79$$

Support is the number of actual occurrences, which is 21.

The macro average is the average of the precision, recall, and F1-scores:

macro avg: $\frac{0.67+0.73}{2} = 0.7$

The weighted average considers the number of samples in each class:

$$weighted\ avg = \frac{0.67*8+0.73*13}{21} = 0.71$$

The idea of using three models here is to understand and compare their performances on this dataset and with the number of epochs used for the three models, based on the information provided by the classification report and confusion matrix.

In the table below, the averages and weighted averages of the indicators:

	Model 1	Model 2	Model 3
Accuracy	0.71	0.81	0.57
Macro precision	0.7	0.8	0.3

Weighted precision	0.71	0.81	0.37
Macro recall	0.67	0.8	0.46
Weighted recall	0.71	0.81	0.57
Macro f1-score	0.68	0.8	0.36
Weighted f1-score	0.7	0.81	0.45
performance		The best model	

- Accuracy: The number of correctly classified roads (the broken ones and the regular ones) divided by the total number of roads. It seems the second model has a better result for this indicator, but this is not enough information; it is a general one.
- Precision: the formula $\frac{correctly\ predicted\ broken\ roads}{total\ \ broken\ roads}$ consider only the broken roads and count how many are correctly predicted among them. Ideally, it should be 1; therefore, the closer to 1, the better the performance. In that aspect, the second model is the best.
- F1-score: the formula $2 * \frac{precision*recall}{precision+recall}$ Becomes when both precision and recall are high, and we see that the second model has the highest value of this indicator.

Therefore, the second model is the best for classifying roads.

IV. CONCLUSIONS

This paper serves as a sensitisation for architects and urban planners on the use of AI methodologies, specifically DL methodologies, in their work.

For a beginner in this field, it is helpful to know how to start using them. Therefore, this paper focuses on two aspects common to the implementation of image processing methodologies, specifically important for architects and urban planners.

The first aspect is the organisation of the data into a dataset. Therefore, this article presents a summary of alternatives for creating image datasets to address various problems. Given that the convolutional neural network (CNN) model is a key component of many other Deep models for image processing, the CNN is presented in detail, and some alternatives are applied to a dataset of images of regular and irregular roads.

An interpretation and comparison of the results are provided, giving a beginner in this field practical experience. After all, this paper will, for a start, help review the teaching programs related to the inclusion of AI elements.

REFERENCES

Arat, M. M. (2019, January 17). *Implementing "SAME" and "VALID" padding of TensorFlow in Python*. Mustafa Murat ARAT. https://mmuratarat.github.io/2019-01-17/implementing-padding-schemes-of-tensorflow-in-python

Baheti, P. (2021, September 13). *Train test validation split: How to & best practices*. V7 Labs. https://www.v7labs.com/blog/train-validation-test-set

Condorelli, F. (2023). Image retrieval for 3D modelling of architecture using AI and photogrammetry. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLVIII-M-2*, 441–446. https://doi.org/10.5194/isprs-archives-XLVIII-M-2-2023-441-2023

GeeksforGeeks. (2023, August 3). *Python | Data augmentation*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/python-data-augmentation

GeeksforGeeks. (2024). *Categorical cross-entropy in multi-class classification*. GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/categorical-cross-entropy-in-multi-class-classification

GeeksforGeeks. (2025, June 20). *How to calculate the number of parameters in CNN?* GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/how-to-calculate-the-number-of-parameters-in-cnn/

GeeksforGeeks. (2025, July 15). *What is Adam optimizer?* GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/adam-optimizer/

Goyal, C. (2024, November 12). *Image denoising using autoencoders – A beginner's guide to deep learning project.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/image-denoising-using-autoencoders-a-beginners-guide-to-deep-learning-project

Idiot Developer. (2020, July 24). *DCGAN — Implementing deep convolutional generative adversarial network in TensorFlow*. Idiot Developer. https://idiotdeveloper.com/dcgan-implementing-deep-convolutional-generative-adversarial-network-in-tensorflow/

Jin, S., Tu, H., Li, J., Fang, Y., Qu, Z., Xu, F., Liu, K., & Lin, Y. (2024). Enhancing architectural education through artificial intelligence: A case study of an AI-assisted architectural programming and design course. *Buildings, 14*(6), Article 1613. https://doi.org/10.3390/buildings14061613

Kohir, V. (2020, June 25). *Calculating output dimensions in a CNN for convolution and pooling layers with Keras*. Medium. https://kvirajdatt.medium.com/calculating-output-dimensions-in-a-cnn-for-convolution-and-pooling-layers-with-keras-682960c73870

Kokorin, O. (2023, September 5). *7 steps to prepare a dataset for an image-based AI project*. HackerNoon. https://hackernoon.com/7-steps-to-prepare-a-dataset-for-an-image-based-ai-project

LearnOpenCV. (2023, January 18). *Convolutional neural networks (CNN): A complete guide*. LearnOpenCV. https://learnopencv.com/understanding-convolutional-neural-networks-cnn

Li, C., Zhang, T., Du, X., Zhang, Y., & Xie, H. (2025). Generative AI models for different steps in architectural design: A literature review. *Frontiers of Architectural Research*. https://doi.org/10.1016/j.foar.2024.10.001

Mohit. (2025, May 1). *Image processing using CNN: A beginner's guide.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/

Nasir, O. (2024, October 31). *AI education in architecture: Revolutionising learning*. Parametric Architecture. https://parametric-architecture.com/ai-education-in-architecture

Prasanth, A., Vadakkan, D. J., Surendran, P., & Thomas, B. (2023). Role of artificial intelligence and business decision making. *International Journal of Advanced Computer Science and Applications, 14*(6). https://doi.org/10.14569/IJACSA.2023.01406103

Purwadhika. (2024, March 21). *Neural network implementation for image classification using CNN.* Purwadhika. https://purwadhika.com/blog/neural-network-implementation-for-image-classification-using-cnn

Sai, U. (2020, August 24). *How to create a new custom dataset from images.* Towards AI. https://pub.towardsai.net/how-to-create-a-new-custom-dataset-from-images-9b95977964ab

Stas. (2025, January 14). *AI in architecture rendering: Benefits and examples.* Pixready. https://www.pixready.com/blog/ai-in-architecture-rendering-benefits-and-examples

Stojanovski, T., Zhang, H., Peters, C., Frid, E., Lefosse, D., & Chattre, K. (2021, April). *Architecture, urban design and artificial intelligence (AI): Intersection of practices and approaches.* Society for Modeling & Simulation International (SCS). SimAUD 2021. https://www.diva-portal.org/smash/record.jsf?pid=diva2:1901303

TUM-DI-LAB. (2019). *Object localisation on building plans using machine learning techniques*. Munich Data Science Institute. https://www.mdsi.tum.de/en/di-lab/past-projects/unetiq-object-localization-on-building-plans-using-machine-learning-techniques