CNN Design Assistant

Mohamed Abdirizak Ibrahim
Student number 20460384

Final Year Project – 2024
B.Sc. Single Honours in
Computer Science and Software Engineering

Department of Computer Science
Maynooth University
Maynooth, Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.
Computer Science and Software Engineering.

Supervisor: **Charles Markham**.

Table of Contents

List of Figures

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of computer science and software engineering qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Mohamed Ibrahim Date: 28/03/2024

Acknowledgements

First, I would like to thank my project supervisor, Charles Markham, for his unending assistance. He has offered me advise that I want to apply to my future profession and that has been helpful to me throughout this journey. I may have created an excellent piece of work because to his never-ending advice. Moreover, I want to express my gratitude to my mother and father for their understanding, support, and assistance, especially when I needed it most while working on this project. And they have never questioned my ability to complete this project. Finally, for the innumerable moments we have shared over the past four years at Maynooth University, I also want to thank my college friends and peers.

Abstract

This paper investigates the intricacies of Convolutional Neural Networks, how they work and the optimization of them through an innovative process called hyperparameter tuning, aiming to streamline the Convolutional Neural Network development process It introduces a tool that allows users to select from multiple hyperparameter search techniques and Convolutional Neural Networks architectures templates, simplifying model customization. The study's methods highlight the basic interface that accommodates various experience levels, from novices to experts. Empirical results demonstrate the tool's effectiveness, with models optimized using the proposed method surpassing those configured baseline parameters in accuracy and efficiency. Graphical representations offer clear evidence of the improvements achieved. They demonstrated that such an approach could significantly reduce barriers to the effective Convolutional Neural Networks development, thus yielding an approachable solution for the challenging optimization of hyperparameters.

Chapter 1: Introduction

Summary

Convolutional Neural Networks (CNNs) are leading the way, in the field of intelligence especially when it comes to analysing images and videos. Yet for those to the field getting started with designing, implementing and fine tuning CNNs can be quite daunting. It all starts with crafting the architecture of CNNs. choosing the right network structure is crucial but intricate, demanding a comprehension of how each layer functions and its influence, on the model's effectiveness.

Preparing data, for CNNs adds complexity to the process. It involves steps such, as properly prepping images precisely labelling them and expanding the dataset for better model adaptability. Moreover, adjusting hyperparameters, which play a role in enhancing the efficiency and effectiveness of model training often requires a trial-and-error approach that demands time and a thorough grasp of the algorithms involved.

1.1 Topic Addressed in This Project

Convolutional Neural Networks (CNNs) are, at the forefront in the realm of intelligence in the analysis of images and videos. However, for those to the world of creating, implementing and refining CNNs it can be quite overwhelming. The journey commences with designing the structure of CNNs – a task that requires consideration when selecting the network layout. This process is pivotal yet intricate as it relies on understanding the functions of each layer and their impact on model performance.

Data preparation for Convolutional Neural Networks (CNNs) introduces complexity. Thoroughly preprocessing images ensuring labelling and diversifying datasets for model reliability entail intricate steps. Moreover, tuning hyperparameters critical for training performance involves trial and error. This does not demand time. Also necessitates a deep comprehension of the underlying algorithms.

The vast and rapidly evolving landscape of learning can be intimidating for novices. They are confronted with choices. Recommended approaches just to kickstart their CNN projects. There is a demand for a straightforward method, in developing CNNs that newcomers can comfortably navigate.

1.2 Motivation

This initiative aims to help novices get started with coding Convolutional Neural Networks (CNNs). The initial stages of coding can feel daunting due to the learning curve in learning, which might discourage many potential learners. CNNs are intricate and using frameworks like TensorFlow or Keras to implement them only adds to the challenge. Beginners often find themselves navigating through a maze of design choices and syntax intricacies even before they dive into training and applying models.

Recognizing these obstacles the project aims to provide beginners with a path to kickstart their CNN coding journey. It simplifies the initiation process by generating code automatically based on user's

preferences and selections. This method reduces complexity at the outset. Clarifies the steps involved making deep learning more approachable and welcoming for novices.

Moreover, this project can accelerate the learning process. By offering a code structure learners can focus on understanding CNN concepts and principles of struggling with coding syntax. Emphasizing comprehension of these concepts is crucial for building a foundation, in deep learning.

1.3 Approach

The project embarked on addressing the complexity faced by novices or just the complexity in general for programming Convolutional Neural Networks (CNNs) through a methodical approach that combined firsthand experimentation with research. The initial phase involved direct engagement in creating CNN models using a dataset provided by my supervisor. This practical attempt was met with challenges but was pivotal in understanding the complexities and nuances, and potential stumbling blocks in the coding process, particularly when the model did not perform as expected. However, once the project had a successful model operation working it provided a clearer perspective on the hurdles newcomers might encounter.

With these practical insights the focus shifted towards designing a solution aimed at simplifying the CNN development process for beginners. The main idea was to develop a tool capable of generating basic CNN templates based on user inputs with a particular emphasis on hyperparameter tuning, which is difficulty and sticking point when trying to in achieve effective model performance. The project was made to not only automate but also have users make informed decisions regarding their model configurations.

The tool's generated models were tested against created ones to verify the usefulness of the automated templates and hyperparameter suggestions. Input, from users played a role in evaluating how user friendly the tool is and how it aids in simplifying the learning process. The evaluation focused on the tools ability to effectively recommend and adjust hyperparameters closely monitoring their impact on model accuracy and training time. Key factors like learning rate, batch size and number of epochs were carefully managed to understand their effects on the results while keeping model complexities constant to concentrate on hyperparameter adjustments.

This holistic approach combining hands on model development with creating a user tool through improvements showcased a promising way to make CNN programming more accessible for beginners. By emphasizing model template generation and streamlining hyperparameter tuning the project aimed to reduce barriers, for entry into learning promoting an interactive learning community.

1.4 Metrics

To be in a position to assess how the tool will simplify CNN programming for Novices, we would then engage the user to determine how friendly and useful the tools are in facilitating the learning process. This will be useful in refining the tool. Then, we will compare the propositions of hyperparameter tuning tools with the known practices to make a keen assessment of the capability these tools offer so that they impact the performance of the model. By so doing, we hope to have obtained the effectiveness of the tool in making CNN programming accessible to novices.

1.5 Project

The project accomplished the tasks:

- Created a tool that automatically generates CNN model code based on user requirements making it easier for beginners to start with CNN programming.
- Provided guidance, on tuning hyperparameters to help novices optimize their models easily.
- Successfully tested and validated the tool using external datasets demonstrating its usefulness in real life situations.
- Developed the tool with input from beginner users ensuring that it meets their learning needs and improves their overall experience.
- Made a significant impact, on deep learning education by simplifying CNN programming encouraging a wider range of people to explore AI technologies.

Chapter 2: Technical Background

Summary

This section explores the theories, methods, and advancements in the realm of Convolutional Neural Networks (CNNs) and deep learning. It aims to showcase not the extent of my research and discoveries but to highlight the challenges and opportunities that inspired this project. By delving into the foundation of CNNs our goal is to illuminate the complexity of this domain and underscore the importance of tools that facilitate beginners understanding and application of these principles.

Topic Material

2.1 Neural networks

Artificial Neural Networks (ANNs) play a role, in the field of intelligence (AI). They aim to mirror the structure of the system by creating interconnected networks of neurons across different layers. This enables these systems to possess generalization abilities and resilience [2]. Since the 1980s research and progress in ANNs have reached milestones effectively addressing challenges in areas such as pattern recognition, robotics, and automated control. These advancements span fields, like biomedicine and economics showcasing aspects of intelligence [2].

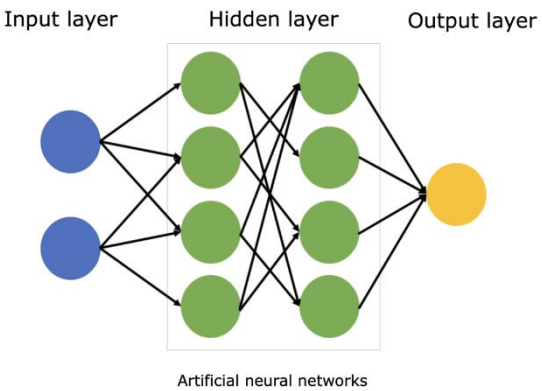


Fig.2.1 Neural Networks

2.1.1 Foundations of CNNs

Convolutional Neural Networks (CNNs) are a cornerstone in the world of modern machine learning and computer vision, especially known for their prowess in image processing [3]. These networks cleverly use convolutional layers with filters to sift through input images, picking up critical details such as edges, corners, and textures elements vital for making sense of images [3]. Following this, pooling layers step in to trim down the size of these feature maps, cutting back on the computational load while keeping the essential details intact. Additionally, activation functions like the Rectified Linear Unit, or ReLU for short, introduce non-linearity into the mix a critical aspect enabling the network to decipher and learn intricate patterns [3]. This feature is key for the network's ability to untangle and learn complex patterns. It's this strategic layering of different elements that equips CNNs with the efficiency to process and analyse image data thoroughly, paving the way for a myriad of applications from image classification to spotting objects within images [3].

Input Layer: Serves as the entry point for images, resizing them to fit subsequent layers for feature extraction [3].

Convolution Layer: Acts as filters, extracting features from images and identifying matching feature points during testing [3].

Pooling Layer: Condenses large images while preserving essential information by retaining the maximum value within each window [3].

Rectified Linear Unit (ReLU) Layer: Converts all negative numbers from the pooling layer to zero, maintaining mathematical stability by preventing learned values from stagnating near zero or escalating towards infinity [3].

Fully Connected Layer**:** Transforms high-level filtered images into categorized outputs with labels [3].

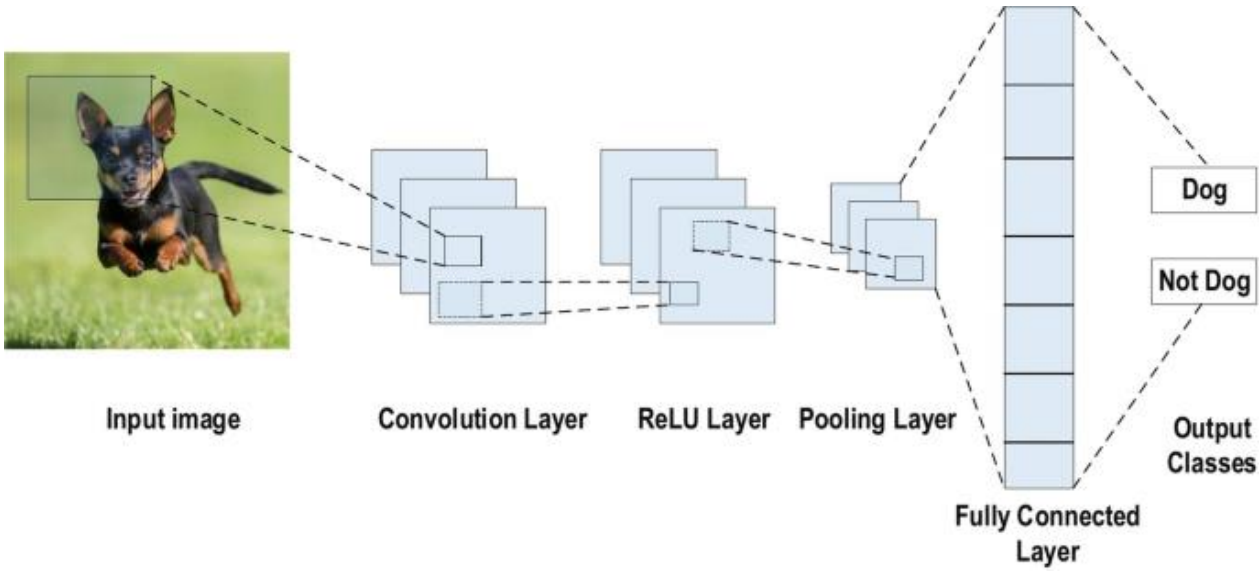


Fig 2.2 CNN layers [1]

2.1.2 Role of Data in CNNs

The success of Convolutional Neural Networks (CNNs) greatly depends on the quality of the data they are trained on. Preparing the data by normalizing and resizing it is crucial, for training these models. Moreover, enhancing the diversity of the training set by applying techniques such as rotating, flipping and scaling images helps the model better adapt to data [4]. Dropout and Data augmentation methods are commonly employed in neural networks [4]. While these initial steps are vital, they can be complex and demanding for those, to the field. This emphasizes the importance of having tools that simplify data management tasks.

2.1.3 Coding and Implementation of CNNs

Understanding the basic architecture and being able to implement that architecture in full requires not only a good understanding but also good programming skill in languages and frameworks meant for deep learning. Python has been one of the leading languages in building CNN models, including simple backbones and full support from libraries such as TensorFlow and Keras [5]. When implementing CNN, it should design the model structure, choose appropriate activation functions, and perform required data preprocessing operations. This means programming of the training phase, where settings of hyperparameters and choice of optimization algorithms to reduce the loss function are key. This, in turn, calls for the use of professional coding practices to tap the full potential out of CNNs and make sure that the expected performance is secured across different uses [5].

2.1.4 Evolution of CNN Architectures

The progression of CNN architectures marks remarkable strides in deep learning's evolution. Beginning with LeNet, the pioneer in incorporating convolutional layers, the field saw a leap in performance with AlexNet, which took advantage of deeper structures and GPU computing [1]. Following this, architectures such as VGG further deepened the networks, whereas GoogLeNet brought in inception modules to tailor learning more adaptively [1]. ResNet revolutionized the training of even deeper networks by introducing residual connections, effectively addressing the vanishing gradient issue. The latest breakthroughs, like EfficientNet, concentrate on refining network scaling to maximize both efficiency and accuracy [4]. These key developments underscore the relentless pursuit of crafting more potent and streamlined CNN models.

Technical material

2.2 Software and tools

There is the great requirement of software tools and libraries to develop and implement Convolutional Neural Networks (CNNs). These would provide necessary infrastructure for effective building, training, and testing of the CNN models that can be able to cater for a wide range of applications, from academic research to commercial development. Notable examples of these comprise TensorFlow, Keras, and MATLAB, which provides a complete deep learning support service with its Deep Learning Toolbox.

2.2.1 TensorFlow and Keras

The motivation behind using TensorFlow, with its high-level API, Keras, is due to its enormous potential in problems of deep learning. TensorFlow provides good support for building and training complex models, e.g., CNN, whereas Keras offers a very friendly interface in doing so. This combination allows for a quick development of efficient deep learning models, considering users from beginner level to advanced experts.

2.2.2 Sequential Model API from Keras

For this project, we extensively used the Sequential model API to simplify building CNN architectures. This API allows building models by stacking up layers with a simplified mapping of the CNN structure from input to output layers translated. Insofar as carried out within the purview of these projects, to make coding CNN less intimidating for beginners, these tools offer novice-friendly processes to follow in the construction of models.

2.2.3 Conv2D, MaxPooling2D, Flatten, Dense, and Dropout Layers

The layers chosen, therefore, for that reason should exactly perform these roles in CNN architecture. Conv2D layers are meant for feature extraction, while MaxPooling2D helps reduce spatial dimensions. Flatten layers help to reshape matrices into vectors for further processing by Dense (fully connected) layers. On the other hand, Dropout layers help in preventing overfitting. These elements of this tool lay down the ground for effective CNN models and integrate in the tool to provide hands-on learning to the user with reference to how CNNs function.

2.2.4 Adam Optimizer

The optimizer chosen is an Adam optimizer and possesses an effectuality that is needed during the optimization for the neural network. An Adam optimizer was chosen because of its ability to adjust the learning rate and because it helps to converge fast and stably. The latter is very important for novices who wish to get a good result yet do not own a deep understanding of optimization algorithms.

2.3 Web Development

2.3.1 Flask Framework

To ensure the tool was both accessible and easy to use the project is using the Flask web framework to develop a web-based interface. Flask's adaptability is ideal for crafting dynamic web applications, allowing users to engage with the tool directly via a web browser. This choice is mainly for to making the development of CNNs more user-friendly, by hiding the technical complexities behind a straightforward web interface. it is also a great option for the scope of the project.

Chapter 3: Design and Implementation (Solution)

Summary

This chapter is illustrating the technical challenges and user requirements our project is designed to address. The development and application of Convolutional Neural Networks present a complex series of technical challenges, particularly for those new to the deep learning domain. We aim to unpack these challenges here, providing a solid groundwork for the solutions we propose to implement as the project

progresses. Understanding user expectations is equally crucial, as it informs our development process, ensuring that the tool we create meets the demands and requirements of its intended users.

3.1 Problem analysis

The challenges of Convolutional Neural Network (CNN) that a CNN Assistant project tries to solve include the complexity behind the CNN architecture, hyperparameter tuning, and data preprocessing. All these complexities will pose a great challenge for their complexity and the intricate understanding to take care of them effectively.

The initiative will introduce automation in the generation of model templates and hyperparameter tuning to ease the development process. This aims at making it easier to get started on CNNs and make the development flow easy. And this will make simple data preprocessing, as it contains the tools and guidelines for preparing the data for training efficiently in the project.

As a result of getting over these difficulties, CNN Assistant becomes even more accessible and democratic toward the user communities in their desire to engage in CNN development and technology, much like it is toward deep learning technologies.

3.1.1 Approach to Understanding the Problem/ Prototyping

Further, to understand the complexities in the implementation of a Convolutional Neural Network (CNN), I began with a project that had initially the problem of data set compatibility between the nut and bolt images. I then decided to switch over to a pre-made Kaggle dataset for fruits and vegetables classification and, to my satisfaction, I was able to develop a working model of the pipeline on it. In this journey, the insights into CNN architectures, where the convolutional layers play a critical role in feature extraction, and the data augmentation techniques like rotation and flipping help the model in generalization, have been very valuable.

The project involved development of a sequential CNN model involving multiple convolution and pooling layers, followed by fully connected layers for classification. It was a series of almost trial-and-error practices starting from the optimization of the model's architecture to hyperparameters, and then to the complexity of data preprocessing for training the model.

Building the model from scratch proved to be a very educational exercise. It put me in a learning curve over the complexities, and not just with CNN development, but it put highlights on where the subtleties are with respect to handling data and configuring the model. This experience was quite instrumental to the design and working methodology of the tool under development, as it ensured that the tool was designed in such a way that the major issues it aimed to solve during the CNN development process were taken care of.

3.2 Project UML Documentation

UML diagrams are diagrams graphically representing the structure of the software system and the components of the software system. It helps in communication and visualization of the structure of the system, the behaviour, and interactions of the system. In the enhancement of the designing and development of this tool, many diagrams have been developed.

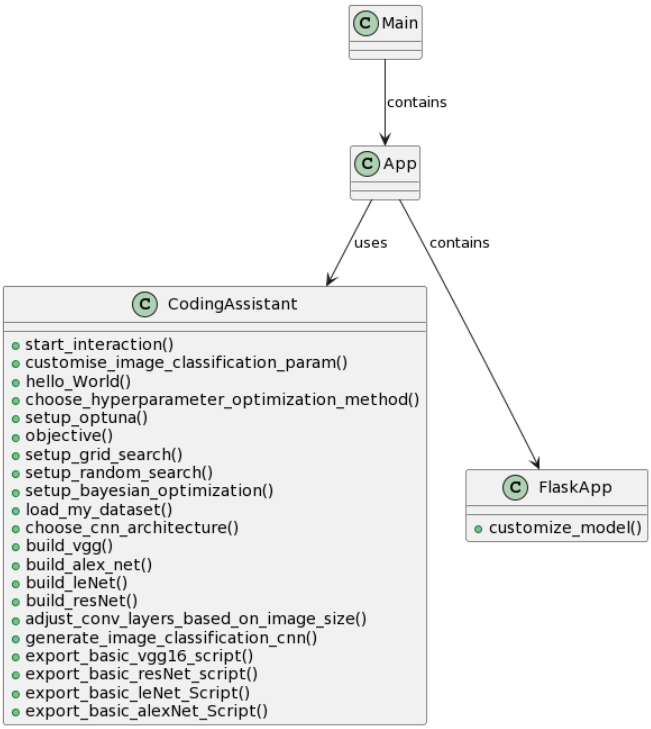


Fig 3.1 UML Class Diagram Overview.

3.2.1 Use Case Diagram

Figure Fig 3.2 below details the high-level features and domain of application presented in the use case diagram. The use case diagram was instrumental in making one visualize the possible interactions of actors/users with a program throughout the development of my mobile app.

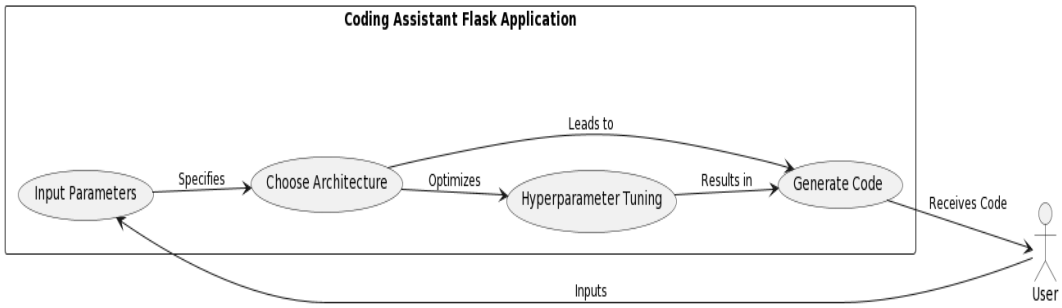


Fig 3.2 Use Case Diagram

3.2.2 User Story

A user story is a brief, nontechnical description of the needs and goals of the user regarding some feature of software. It represents what a user needs and what value a particular feature provides to a user. A user story seeks to understand the goal of the user; it is formulated as scenarios that guide development in the realization of such set goals. This user story is distilled from the user feedback and seeks to capture the essence of what a user would need in a tool that would help him create and tune CNN models.

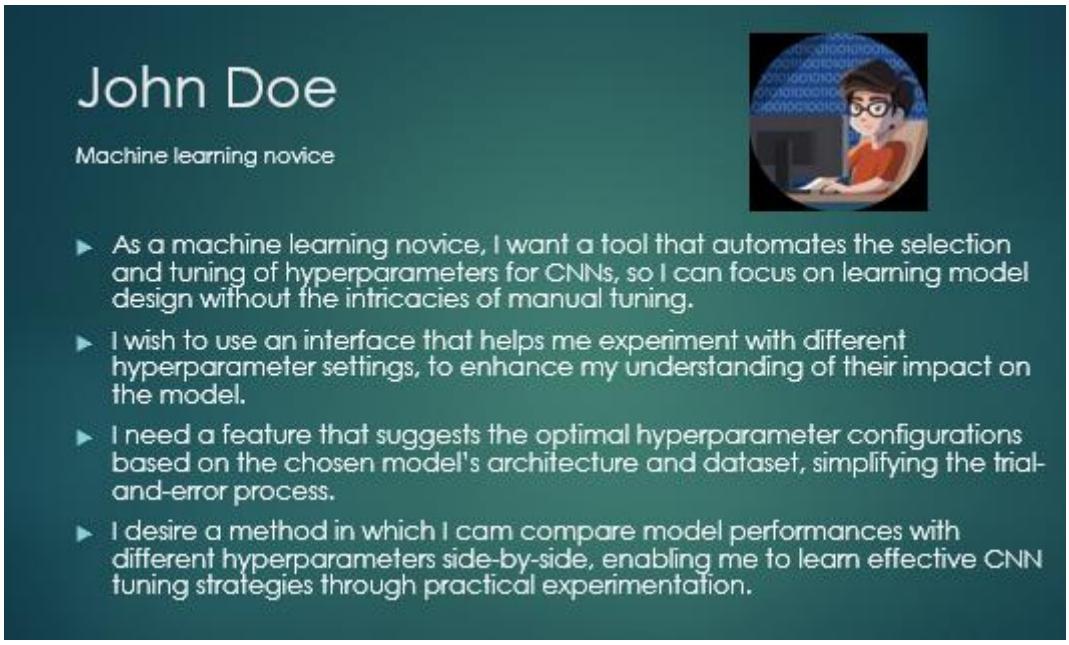


Fig 3.3 User Story

3.3 Specifications Overview

In the development of the CNN Assistant project, software and tools were selected to support the varying needs of a machine learning practitioner in robustness and ease of use. Python is the main programming language and is admired for its clarity and the number of libraries it has in the domain of machine learning. This is the language on top of which TensorFlow and Keras reside, being two key libraries used in building CNNs. TensorFlow comes with a highly flexible toolset suitable for efficient model construction, training, and serving. It is the most powerful library among the best machine learning libraries that one can use for the development of the most complex neural networks. For further simplification when building the neural networks, Keras is used with TensorFlow, which is an API for higher abstraction. Users can define the model’s layer by layer, that's it, with clear and easy syntax. In its class, it is absolute best. This is a great help for beginners, as they will probably concentrate on the structure without hustling with other settings of a lower level. In this project, the Sequential API from Keras is specifically used to create a linear stack of layers that can be used in building CNNs. Keras provides the Sequential model so that people can very easily build a model and work on real-world datasets, tuning several architectures and hyperparameters.

3.4 Design Overview

From an architectural point of view, CNN Assistant is designed around a web application based on Flask that serves as an interactive sandbox where users are free to conduct experiments with, for example, model configurations, input data, and hyperparameters. The Flask application, written in Python, controls the flow of requests and responses from the users.

In this project, the Flask framework provides functionality to handle incoming HTTP requests. Flask is implemented with a web interface that has fields for data entry and parameters for the customization of CNN, which is handled through routes and views.

The core logic of the CNN Assistant uses Python at the heart of it, with the TensorFlow and Keras libraries providing a very flexible and powerful backend. The backend is going to be responsible for setting up, training, and evaluating the CNN models that have been configured by the user.

The system will have users interact through the front end with HTML forms where they will define the parameters of the model: the size of the image, the number of classes of the image, and the colour channels of the image. If the user wants to do hyperparameter tuning for the model, then he can add the parameters in the text area provided. This is facilitated via the index.html form.

Upon form submission, custom data is to be taken along the Flask route (/customize), whereupon backend Python logic is invoked for the generation of CNN based on selected parameters and architecture type (e.g., ResNet, VGG16) picked from the frontend.

After the model is trained, the Python code for the model, optionally with the parameter tuning part, can be shown to the user through the result.html template, equipped with the clipboard copying ability next a developed module, export_templates.py, exports a Python script for different CNN architectures based on the optimum hyperparameters found.

Code templates and hyperparameter tuning. The module codingAssistant.py has incorporated the user interface functionality in dynamically creating the CNN model code with respect to user input for different types of layers and other hyperparameters, like optimizer, loss, activation functions, among other key parameters, including input and output dimensions.

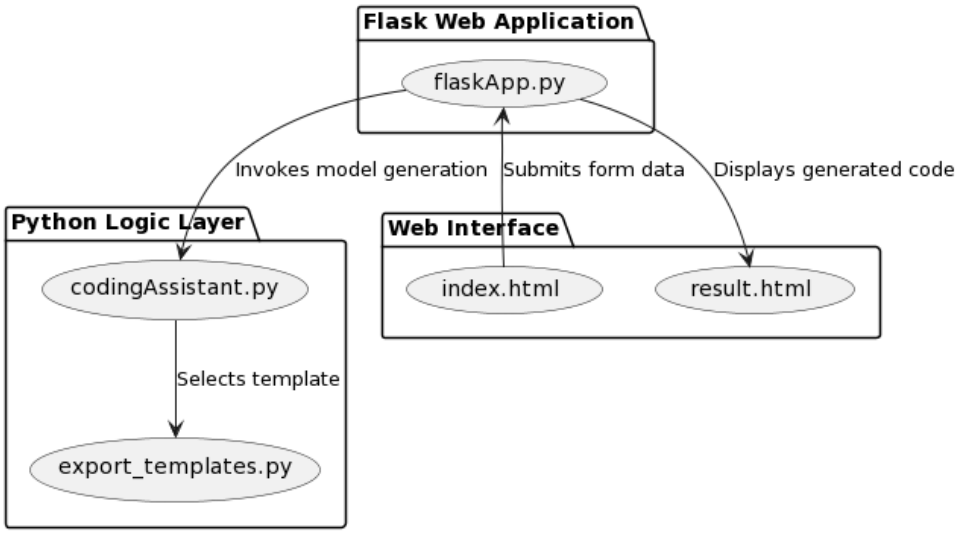


Fig 3.4 Design Architecture

3.5 Approach to Software Development

My software development strategy, for the CNN Assistant project was based on methodologies emphasizing progress, adaptability, and continual enhancement. The implementation of practices influenced every stage of the project spanning from planning and coding to delivery and user feedback.

The fundamental principles of Agile were at the heart of how the project unfolded. By organizing development into sprints, we could swiftly. Adjust in response to feedback. This method ensured that the project stayed attuned to user requirements and could easily accommodate changes and fresh perspectives. Regular team meetings and sprint reviews fostered a work atmosphere to responsive development.

Git played a role in managing version control for the project. It offered a platform for monitoring modifications experimenting with functionalities and collaborating on code while safeguarding against data loss. Utilizing Git also meant that the project was accessible from anywhere making it indispensable, for maintaining a current codebase.

For this endeavour PyCharm served as our Integrated Development Environment (IDE). Recognized for its features facilitating code writing, testing, debugging, integrated testing capabilities and version control management.

All the tasks, for development were completed within the PyCharm environment from writing the code to troubleshooting logic and executing the entire application. This significantly boosted productivity and efficiency throughout the project.

By utilizing methodologies Git for version control and PyCharm as the development tool we were able to maintain a cohesive and well managed project environment both in terms of coding practices and project documentation. This approach did not optimize the software development process. Also provided a structured framework necessary to navigate the complexities inherent in creating an application, like CNN Assistant.

3.6.1 The Solution

The solution to the project will be that it can create, optimize, and automatically generate a convolutional neural network (CNN) model that aligns with the specifications given by the user. First, it interacts with the user to give the input options such as image size, number of classes, colour mode, and many more. It also takes the location of the dataset and the hyperparameters for tuning. The user shall also specify the mostly used CNN architectures, including VGG16, ResNet, LeNet, and AlexNet.

The second option that comes for the user, who will be interested in hyperparameter tuning, will present three choices for optimization strategies: Random Search, Grid Search, and Optuna. The program would then proceed through the hyperparameter space in a very systematic way, depending on the selection, up to the specified bounds for a number of trials by the user. Searching, in this case, would involve finding a hyperparameter set that has the highest validation accuracy model performance reporting.

The next step was to choose the hyperparameters based on the selected architecture and then select the best set of hyperparameters. These hyperparameters, which would be the basis of our model, were stored

in a template built by the framework. Finally, the customized template is exported as a Python file for direct model training and evaluation. On top of that, the program is built in such a way that it allows explicit interaction with users through a simple front-end interface for seamless selection processes and an optional facility to copy code directly to the clipboard, which would ensure easiness of workflow from specification until model deployment.

3.6.1 Hyperparameter Tuning.

Hyperparameter tuning is a critical step in the process of optimizing machine learning models. They are settings or parameter settings, set model for the learning process but are not learned from the data itself, hence "hyperparameters." For example, the learning rate, batch size for training, the neural network architecture (e.g., how many layers), and regularization parameters. The final objective with hyperparameter tuning is zeroing in on a particular combination of hyperparameters that, when implemented, will give the model the best possible performance for a given task: performance measured by metrics such as accuracy for classification tasks or mean squared error for regression tasks. Hyperparameter tuning techniques can range from very simple, such as manual search and grid search, to some more sophisticated approaches like random search, Bayesian optimization, and evolutionary algorithms, each with its strengths and the ideal use case. Well, performance of the model tuning may improve the effectiveness and efficiency of the model to a greater extent; therefore, it represents an important aspect in the machine learning pipeline [7].

3.6.1 Random Search Implementation

Random search selects hyperparameters randomly from a given search space. This approach would be most useful for any kind of problem for which the dimension of the search space is high, and the cost to evaluate the performance of the model is high (computational resources + time). In other words, the underlying hypothesis is that good hyperparameters are not well isolated in the search space, and that the random sample would be able to efficiently find regions that show good performance.

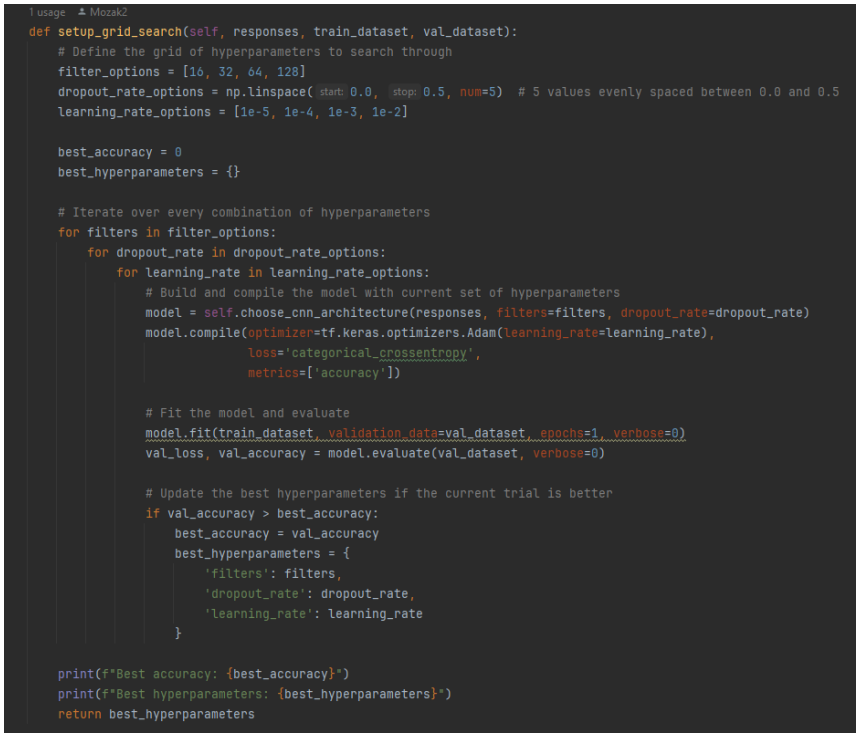
```
1 usage   Mozak2 *
def setup_random_search(self, responses, n_trials, train_dataset, val_dataset):
    # Define the search space for hyperparameters
    filter_options = [16, 32, 64, 128]
    dropout_rate_options = np.linspace( start: 0.0,  stop: 0.5, num=10)  # 10 linearly spaced values
    learning_rate_options = [1e-5, 1e-4, 1e-3, 1e-2]

    best_accuracy = 0
    best_hyperparameters = {}

    for _ in range(n_trials):
        # Randomly select hyperparameters
        filters = random.choice(filter_options)
        dropout_rate = random.choice(dropout_rate_options)
        learning_rate = random.choice(learning_rate_options)

        # Here, choose_cnn_architecture needs to accept filters, dropout_rate as arguments
        model = self.choose_cnn_architecture(responses, filters, dropout_rate, learning_rate)

        # Fit the model and evaluate
        model.fit(train_dataset, validation_data=val_dataset, epochs=1, verbose=0)
        val_loss, val_accuracy = model.evaluate(val_dataset, verbose=0)

        # Update the best hyperparameters if current trial is better
        if val_accuracy > best_accuracy:
            best_accuracy = val_accuracy
            best_hyperparameters = {'filters': filters, 'dropout_rate': dropout_rate,
                                    'learning_rate': learning_rate}

    print(f"Best accuracy: {best_accuracy}")
    print(f"Best hyperparameters: {best_hyperparameters}")
    return best_hyperparameters
```

Fig 3.5 setup_random_search Method

The code shown in fign 3.5 performs hyperparameter optimization for a CNN using a random search strategy. It essentially defines a search space for three critical hyperparameters: the number of filters, the dropout rate, and the learning rate. Random samples from this space are taken iteratively, while the random combination is trained and tested over a given training and validation dataset. It finally identifies the combination that from the tried resulted in the highest validation accuracy, accordingly updates the best hyperparameter record, and finally returns this optimal set. This is able to effectively search the hyperparameter space for better model performance without systematic demands during the searches that are computationally exhaustive.

3.6.2 Grid Search

Grid search is a hyperparameter optimization using a brute-force approach. It can be either done in a systematic way, by iterating through a set of the hyperparameters, training based on each combination of network parameters to choose the best parameters which yield the best accuracy or performance metric, or randomly. Although it's comprehensive, grid search can be very time-consuming, especially with a large number of hyperparameters.



```
1 usage   Mozak2
def setup_grid_search(self, responses, train_dataset, val_dataset):
    # Define the grid of hyperparameters to search through
    filter_options = [16, 32, 64, 128]
    dropout_rate_options = np.linspace( start: 0.0,  stop: 0.5, num=5)  # 5 values evenly spaced between 0.0 and 0.5
    learning_rate_options = [1e-5, 1e-4, 1e-3, 1e-2]

    best_accuracy = 0
    best_hyperparameters = {}

    # Iterate over every combination of hyperparameters
    for filters in filter_options:
        for dropout_rate in dropout_rate_options:
            for learning_rate in learning_rate_options:
                # Build and compile the model with current set of hyperparameters
                model = self.choose_cnn_architecture(responses, filters=filters, dropout_rate=dropout_rate)
                model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                              loss='categorical_crossentropy',
                              metrics=['accuracy'])

                # Fit the model and evaluate
                model.fit(train_dataset, validation_data=val_dataset, epochs=1, verbose=0)
                val_loss, val_accuracy = model.evaluate(val_dataset, verbose=0)

                # Update the best hyperparameters if the current trial is better
                if val_accuracy > best_accuracy:
                    best_accuracy = val_accuracy
                    best_hyperparameters = {
                        'filters': filters,
                        'dropout_rate': dropout_rate,
                        'learning_rate': learning_rate
                    }

    print(f"Best accuracy: {best_accuracy}")
    print(f"Best hyperparameters: {best_hyperparameters}")
    return best_hyperparameters
```

Fig 3.6 setup_grid_search Method

The above function, fig 3.6, is a systematically designed hyperparameters tuner for a convolutional neural network using the grid search method. It basically searches from a pre-specified grid of hyperparameters' values for dropout rates, learning rates, and the number of filters over an exhaustive search grid. Here the function runs over all possible combinations of these hyperparameters and checks by building the model for each set, with specified training dataset overfitting and validation dataset overperformance evaluation. The search was able to find the combination of dropout and learning rate that would give us the highest validation accuracy. The combination that does this will then be considered the best set of

hyperparameters. After all the combinations are checked, the function reports and returns the combination of hyperparameters it has encountered that gave the best result, and therefore enables one to pick an optimally tuned model and use it for the further training or testing process. By its very nature, it is an exhaustive approach but can become computationally very expensive, especially with large sets of hyperparameters.

3.6.3 Architectures

Some of the greatest architectures for deep learning in general, and convolutional neural networks (CNNs), are VGG16, ResNet, and AlexNet. In the present work, we include the following architectures, since they have marked the development of CNNs and since then have become widely adopted for many by different people: image recognition tasks and network designs, in terms of diversity. Each architecture brings unique features and benefits to the table:

3.6.4 Optuna

Optuna was integrated into the project to facilitate hyperparameter optimization, utilizing Bayesian optimization techniques. This advanced framework efficiently searches for optimal hyperparameters by intelligently predicting promising configurations based on previous results. Optuna's approach significantly enhances model performance, requiring fewer trials compared to traditional methods like grid or random search, making it a powerful tool for tuning complex models on large datasets with minimal manual effort.

```
1 usage   Mozak2 *
def setup_optuna(self, n_trials, responses, train_dataset, val_dataset):
    print("_________________ inside setup_optuna ____________________")
    Mozak2 *
    def objective_wrapper(trial):
        # Ensure this method accepts a trial and operates correctly with it
        return self.objective(trial, responses, train_dataset, val_dataset)
    study = optuna.create_study(direction='maximize')
    study.optimize(objective_wrapper, n_trials=n_trials)  # Adjust n_trials as needed

    # Now, you can access the best trial results
    best_params = study.best_trial.params
    print("Best hyperparameters found:", best_params)
    return best_params

# Your existing Optuna setup code
1 usage   Mozak2 *
def objective(self, trial, responses, train_dataset, val_dataset):

    # Example: Define hyperparameters using trial suggestions
    filters = trial.suggest_categorical('filters', [16, 32, 64, 128])
    dropout_rate = trial.suggest_uniform('dropout_rate', 0.0, 0.5)
    learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-1)
    model = self.choose_cnn_architecture(responses, filters,dropout_rate, learning_rate)
    # Fit the model
    history = model.fit(train_dataset, validation_data=val_dataset, epochs=1, verbose=0)
    # Assuming X_val and y_val are your validation dataset and labels
    val_loss, val_accuracy = model.evaluate(val_dataset, verbose=0)
    return val_accuracy
```

Fig 3.7 setup_optuna and objective Method

The format in Fig 3.7 is similar to how the other hyperparameter methods are trained however this method utilizes Optunas library for hyperparameter tuning in training convolutional neural networks (CNNs). It sets up an Optuna study to maximize an objective function over a specified number of trials. The objective function, defined in objective, proposes hyperparameters like the number of filters, dropout rate, and learning rate for each trial based on Optuna's suggestion mechanisms. For each set of hyperparameters, it constructs a CNN model using choose_cnn_architecture, trains it on a dataset, and evaluates its performance on a validation dataset.

3.6.5 Implementation

After deciding on the best hyperparameters to be used, A template will be programmatically exported to the selected configuration of the CNN model to a Python file with settings in it. This will be saved on the server, with the content of the file available through basic HTML on the frontend. The server reads and displays the model code when a user navigates to this page. This is how generated hyperparameters and the model architecture are seen by the user. This is where the "copy to clipboard" feature would come in to ease the copying of the code model for use in the project. This creates a direct interconnectedness between hyperparameter optimization and practical use and therefore enables the user to experience an improved outcome by using the optimized model configurations.

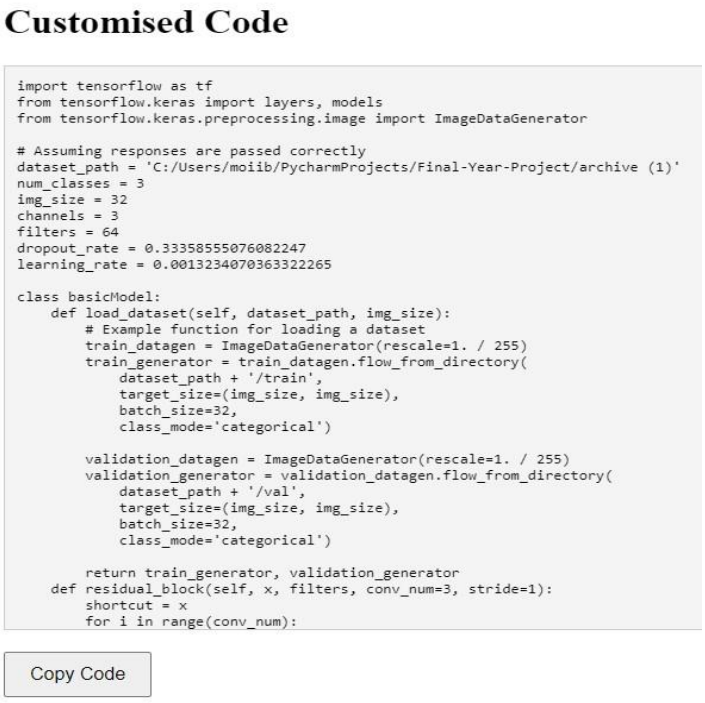


Fig 3.8 Screenshot

Chapter 4: Evaluation

Summary

This chapter shows the evaluating methods that were used to estimate the performance of the application. Solution verification, solution design verification, Unit testing and results.

4.1 Solution Verification

This is where the pivotal role of the Agile Board was used in this project as a dynamic tool for Solution Verification. It was in charge of the organization, management, and record of varied tasks that were done in looking into the functionality and stability of the software. An Agile board would follow up with each stage of development and testing eye-wise, hence a continuous loop of feedback and iterations. This way, it led to finding the problems and implementing their solutions immediately, hence making the project more adaptive to changes and, in general, improving the attainable quality of the product. Moving from the to-do, in progress, to verification stages systematically, ensuring each feature was subjected to the rigors of testing and validation before deemed complete. The agile board enhanced team collaboration, in which team members could always communicate any stage and blocking issues or gaps through proper communication, hence effective Solution Verification and high software integrity with user satisfaction.

4.2 Software Design Verification

The UML (Unified Modelling Language) diagrams were massively important to ensure that the software worked fine. This means the UML diagrams of software architecture gave an opportunity for the development team to look over the structure and interactions of the system in detail. Class diagrams will be used to explain relationships between objects and will aid in the assessment of whether the system is logically consistent. The sequence diagrams further explained the course of operations across the different components, thereby ensuring integration and data flow through the different components. Use case diagrams, on the other hand, assured that all user interactions were part of the diagram and that they all aligned to the intended functionality of the software.

The use of UML diagrams in the development stages was very important. It sought to find and debug the established design errors or inconsistencies. With this kind of prediction of potential problems in the design, correction of the problem would be easy before it goes far with the development. This automatically increased the reliability and performance of the software. UML diagrams, therefore, were not designs by themselves but played a crucial role in the verification of the cohesive operation of the software components toward the production of a robust final product meeting all specified requirements in addition to user expectations.

4.3 Software Verification

In fact, the software was largely verified using selected unit tests, which had been taken as a method because of its efficient isolation of individual units for testing. The development team would have been able to ensure that each one of them works perfectly in the system before the final integration process is

complete through unit tests, such as the ones for individual components developed to build specific CNN models, for example, AlexNet—those that verify dataset loading functionality. This made sure that errors are realized and rectified right in time, hence the software became very reliable, and its performance was boosted very high. The unit tests would be a systematic, automated way of validating that the code was correct, allowing fast iterations to occur, resulting in an effective development process while enabling regression testing. This turned out to be very valuable for keeping high-quality code, making sure that every part of the software met its design, and, therefore, functioned according to it, hence added to the strength and stability of the application in totality.

4.3.1 Unit Testing

The project implemented unit tests to prove the effectiveness and reliability of the code. The tests were developed with an emphasis on basic program logics, such as building the CNN models, loading the datasets, and others. For example, buildalexNetTest.py: it contains some tests that check, first, the instance of AlexNet, its input, and output shapes if some layers are there. The model is properly built according to some prescribed parameters. It will test if the AlexNet model follows the expected architecture and functionality requirements, like input and output dimensions, required layers in the right order. A similar process was applied to all the other model architectures.

```
class TestvggloModel:
    6 usages  new *
    @pytest.fixture(scope="class")
    def setup(model_config):
        # This setup method will be run once for all tests in this class
        assistant = CodingAssistant()
        responses = [32, 10, None, 3]  # Example: image size, num_classes, dataset_path, channels
        filters = 64
        dropout_rate = 0.5
        learning_rate = 1e-3
        model = assistant.build_vgg(responses, filters=filters, dropout_rate=dropout_rate, learning_rate=learning_rate)
        return model

    new *
    def test_instance(self, setup):
        model = setup
        assert isinstance(model, tf.keras.models.Model), "The method did not return a Keras model instance."

    new *
    def test_output_shape(self, setup):
        model = setup
        assert model.output_shape == (None, 10), "The output shape does not match the expected number of classes."

    new *
    def test_dropout_layers(self, setup, dropout_rate=0.5):
        model = setup
        dropout_layers = [layer for layer in model.layers if isinstance(layer, tf.keras.layers.Dropout)]
        assert len(dropout_layers) > 0, "No dropout layers found."
        assert all(layer.rate == dropout_rate for layer in dropout_layers), "Dropout rates do not match the expected value."
```

Fig 4.1 Unit Tests

```
6 usages  new *
@pytest.fixture
def alex_net_setup():
    # Example setup with arbitrary values
    responses = [224, 10, None, 3]  # [img_size, num_classes, dataset_path, channels]
    dropout_rate = 0.5
    learning_rate = 1e-3
    assistant = CodingAssistant()
    model = assistant.build_alex_net(responses, dropout_rate, learning_rate)
    return model, responses

new *
def test_alex_net_instance(alex_net_setup):
    model, _ = alex_net_setup
    assert isinstance(model, tf.keras.Model), "The method did not return a Keras Model instance."

new *
def test_alex_net_input_shape(alex_net_setup):
    model, responses = alex_net_setup
    expected_shape = (None, responses[0], responses[0], responses[3])
    assert model.input_shape == expected_shape, f"Input shape {model.input_shape} does not match expected {expected_shape}."

new *
def test_alex_net_output_shape(alex_net_setup):
    model, responses = alex_net_setup
    assert model.output_shape == (None, responses[1]), f"Output shape {model.output_shape} does not match expected number of classes {responses[1]}."

new *
def test_alex_net_layers(alex_net_setup):
    model, _ = alex_net_setup
    layer_types = [type(layer) for layer in model.layers]
    expected_layers = [tf.keras.layers.Conv2D, tf.keras.layers.BatchNormalization, tf.keras.layers.MaxPooling2D,
                       tf.keras.layers.Flatten, tf.keras.layers.Dense, tf.keras.layers.Dropout]
    for expected_layer in expected_layers:
        assert expected_layer in layer_types, f"Expected layer {expected_layer} not found in model."
```

Fig 4.2 Unit Tests

4.3.2 Test Data

The CIFAR-10 dataset was selected to test the application; it has been well-documented and contains a total of ten classes of images that were used for the test. This choice is strategic, since due to its simplicity and manageability, CIFAR-10 is a very attractive candidate for the first set of tests to allow quick verification possibilities of functionality at stages like data loading, preprocessing, model training, and evaluation without the complexities presented by a bigger and more varied dataset. CIFAR-10 was used just to be able to have a timely spotting of any issue between the data pipeline and model architecture, for proper implementation of basic functionalities that allow obtaining correct performances.

The program could only start testing on a custom dataset once it demonstrated fluent work with CIFAR-10 and gave grounds to be sure of its stable functionality. The transition was central for several reasons. This allowed for a fair assessment of the system under real operating conditions, where the data can be inadequately preprocessed, unlike CIFAR-10. Besides, this testing also availed insights into the scalability and adaptivity of the application to diversified data types and sizes, which are critical attributes of any solid deep learning application.

4.4.1 Results

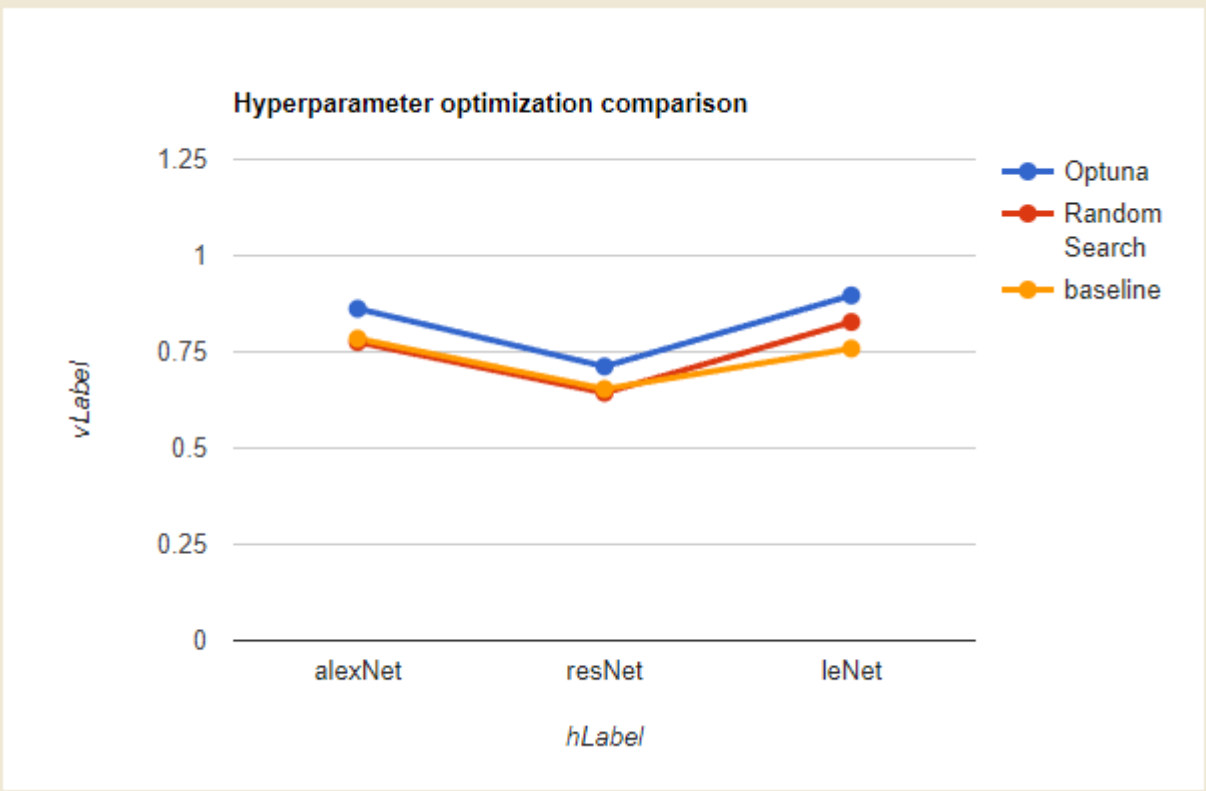


Fig 4.3 Line Chart

4.4.2 Explanation of Results

The graph presented in Figure 4.3 illustrates the comparative performance of different hyperparameter optimization strategies applied to various CNN architectures. This showed that, with the evolutionary algorithm method, Optuna turned out to have given a better performance than the baseline hyperparameter settings. The much bigger caveat was that the model tuned at the baseline configuration gave better results than the models tuned using random search. It reinforced the point that the Optuna strategy toward exploring the hyperparameter space is effective in pointing out improvements in model performance from standard settings, and at the same time underscores the possible limitations or difficulty proper of the random search in this context.

4.4.2 Analysis of Results

The chart, in Figure 4.3 shows how model accuracy changes with methods of tuning hyperparameters over 10 epochs. This measure is crucial as it directly impacts the model's capability to make predictions on data. Optuna, utilizing an algorithm and represented by the line performs better than the standard when applied to alexNet and leNet designs suggesting its effectiveness in exploring hyperparameter options efficiently. However, when used with the resNet design Optuna matches the performance level indicating that certain designs may not benefit much from tuning. The standard method, shown in orange uses hyperparameter values like a dropout rate of 0.5 and a learning rate of 0.01. Values often chosen for their suitability across different models and scenarios.

Random Search, represented by the red line falls slightly behind both the baseline and Optuna in all instances signalling a search strategy within the same hyperparameter space. It's important to note that both Random Search and Optuna were tested using search spaces for comparison.

Grid Search is not included in this graph due to its nature that requires computational resources. Making it impractical for this analysis due to time constraints and budget limitations. When we limit the comparison, to methods that are doable within a computational budget and timeframe the graph shows a straightforward view of the pros and cons of each method.

Chapter 5: Conclusion

Summary

The paper introduces a tool that eases CNN development through hyperparameters tuning and, therefore, makes it beginner-friendly and suitable for all levels of skills. Empirical evidence prove that this approach is effective and, in some cases, exhibits superior performance in comparison with the baseline parameters. This further proves that the tool can make the development of CNN and the optimization of hyperparameters even easier, not only for the experienced researcher but also for beginners.

5.1 Project Approach

The first step in this project will be to create a custom Convolutional Neural Network (CNN) model from scratch This initial phase served as a practical learning curve for the author, who possessed limited prior knowledge in the field of machine learning. Through the development and testing of this custom CNN model, it became evident that one of the many challenges was the selection of appropriate hyperparameters. This realization highlighted a knowledge gap and underscored the complexity of optimizing CNNs, particularly for individuals new to machine learning. This now brought out the problem that this project would be geared to address, pointing its objective towards investigating approaches that would simplify the tuning process. making it much more available and user-friendly for beginners interested in the development of CNNs. This has given rise to the development of a tool designed with intent and deliberation to make CNN technology more accessible by surmounting these barriers through automation and guidance of the hyperparameter tuning.

5.2 Results Discussion

This report highlights the benefit of using algorithms to improve the accuracy of CNN models compared to standard hyperparameters. These sophisticated techniques have proven to be highly effective, in architectural contexts illustrating their potential in optimizing hyperparameters. Interestingly the outcomes from approaches for the resNet design suggest that certain models may not adapt well to changes in hyperparameters. This study underscores the importance of selecting suitable and efficient hyperparameter tuning strategies for individuals with computing resources as seen by the intentional exclusion of exhaustive methods like Grid Search. The research outcomes contribute to enhancing the understanding of CNN development and provide insights, into utilizing hyperparameter tuning methods in machine learning simplifying and streamlining the process.

5.3 Future Work

Intended future Work for this project would mean giving further enhancements to the tool's power and improving user interaction, A significant planned update is enabling the user to provide their CNN code for the hyperparameter-tuning function of the tool. It is meant for easy optimization to eliminate complexities from manual changes and make the tool efficient.

A tool that automatically decides the number of layers in the CNN architectures. Using algorithms able to estimate model performance with respect to its depth, therefore, the tool would suggest layer configurations that would optimize the model architecture. This will simplify the model design to the user. Furthermore, ease of use will substantially increase by updating the frontend interface. This shall work to provide the main objective of making the platform both intuitive and appealing to the effect that it is much operational in meeting the needs of the user through improved navigability and functionality. These future directions point to our firm commitment to making development in CNN more accessible and user-friendly, in larger alignment with demystifying machine learning for a diverse audience.

References

[1] L. Alzubaidi, J. Zhang, A.J. Humaidi, et al., "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," J Big Data, vol. 8, 53, 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00444-8. [Accessed 20 March 2024].

[2] A. Goel, A.K. Goel, A. Kumar, "The role of artificial neural network and machine learning in utilizing spatial information," Spat. Inf. Res., vol. 31, no. 3, pp. 275–285, 2023. [Online]. Available: https://doi.org/10.1007/s41324-022-00494-x. [Epub 18 November 2022]. [Accessed 22 March 2024].

[3] N. Sharma, V. Jain, A. Mishra, "An Analysis of Convolutional Neural Networks for Image Classification," Procedia Computer Science, vol. 132, pp. 377-384, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.05.198. [Accessed 24 March 2024].

[4] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang and Y. Miao, "Review of Image Classification Algorithms Based on Convolutional Neural Networks," Remote Sensing, vol. 13, no. 22, p. 4712, 2021. [Online]. Available: https://doi.org/10.3390/rs13224712. [Accessed 24 March 2024].

[5] TensorFlow, "Keras guide," TensorFlow, [Online]. Available: https://www.tensorflow.org/guide/keras. [Accessed 25 March 2024].

[6] A.M. Vincent, P. Jidesh, "An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms," Sci Rep, vol. 13, 4737, 2023. [Online]. Available: https://doi.org/10.1038/s41598-023-32027-3. [Accessed 25 March 2024].

[7] Elgeldawi, E. et al. (2021) Hyperparameter tuning for machine learning algorithms used for Arabic sentiment analysis, MDPI. Available at: https://doi.org/10.3390/informatics8040079# (Accessed: 25 March 2024).

[8] Science Learning Hub, "Neural network diagram," 2023. [Online]. Available: https://www.sciencelearn.org.nz/images/5156-neural-network-diagram. [Accessed 26 March 2024].

Appendices

Code developed for this project.

The project has too much code to display inside my thesis, so the code can be found on my GitHub on a public repository: https://github.com/Mozak2/Final-Year-Project