**Table of Contents**

[List of Figures 3](#_Toc183878367)

[List of Tables 6](#_Toc183878368)

[**Introduction** 7](#_Toc183878369)

[**Dataset & Algorithm Selection** 7](#_Toc183878370)

[Dataset 7](#_Toc183878371)

[Algorithm Selection 7](#_Toc183878372)

[CNN 7](#_Toc183878373)

[**Literature Review** 9](#_Toc183878374)

[*1.0 CNN - Using SE-RES-CNN* (Li, 2024) 9](#_Toc183878375)

[1.1 Overview and Objectives 9](#_Toc183878376)

[1.2 Dataset Description 9](#_Toc183878377)

[1.3 SE Attention Mechanism 9](#_Toc183878378)

[1.4 ResNet 9](#_Toc183878379)

[1.5 Block Layer Design 10](#_Toc183878380)

[1.6 The SE-RES-CNN Model Architecture 10](#_Toc183878381)

[1.7 Performance Comparison Analysis 10](#_Toc183878382)

[2.0 ANN - Classification of Starling Image Using ANN (Rahman A. Y., 2021) 11](#_Toc183878383)

[2.1 Image Data 12](#_Toc183878384)

[2.2 Manual Segmentation 12](#_Toc183878385)

[2.3 Feature Extraction 12](#_Toc183878386)

[2.4 Texture Feature Extraction 12](#_Toc183878387)

[2.5 ANN Model 12](#_Toc183878388)

[2.6 Result and Discussions 13](#_Toc183878389)

[3.0 DNN - A Study by Zhou et al. (2024): Improved sports image classification using DNN and novel tuna swarm optimization 14](#_Toc183878390)

[3.1 Dataset Description and Sports Classification 14](#_Toc183878391)

[3.2 Novel Tuna Swarm Optimization (NTSO) 14](#_Toc183878392)

[3.3 Proposed DNN/NTSO Model 14](#_Toc183878393)

[3.4 Model Hyperparameters and Training Protocol 15](#_Toc183878394)

[3.5 Simulation Results and Performance Evaluation 15](#_Toc183878395)

[4.0 RNN – A Study by Gaafar et al. (2021): Comparative Analysis of Performance of Deep Learning Classification Approach based on LSTM-RNN for Textual and Image Datasets 16](#_Toc183878396)

[4.1 Overview 16](#_Toc183878397)

[4.2 Evolution of RNN Architectures 16](#_Toc183878398)

[4.3 Dataset Description 18](#_Toc183878399)

[4.4 Algorithms 19](#_Toc183878400)

[4.5 Proposed Models 19](#_Toc183878401)

[4.6 Hyperparameter Tuning 20](#_Toc183878402)

[4.7 Performance Analysis 21](#_Toc183878403)

[4.8 Comparative Results 21](#_Toc183878404)

[**Data Preparation** 23](#_Toc183878405)

[**Model Building** 30](#_Toc183878406)

[Convolutional Neural Network (CNN) 30](#_Toc183878407)

[**Model Tuning** 34](#_Toc183878408)

[CNN Hyperparameter Tuning 34](#_Toc183878409)

[**Model Evaluation & Discussion** 38](#_Toc183878410)

[CNN Base Model 38](#_Toc183878411)

[CNN Tuned Model 40](#_Toc183878412)

[**Conclusion** 44](#_Toc183878413)

[**References** 45](#_Toc183878414)

# List of Figures

[Figure 1 - Architecture of Proposed Model 9](#_Toc183878415)

[Figure 2 - ANN Layers for the Architecture 10](#_Toc183878416)

[Figure 3 - Traditional RNN Architecture (Paoletti, Haut, Plaza, & Plaza, 2020) 14](#_Toc183878417)

[Figure 4 - Simple Recurrent Unit Formula (Paoletti, Haut, Plaza, & Plaza, 2020) 15](#_Toc183878418)

[Figure 6: Import Library 21](#_Toc183878419)

[Figure 7: Location of Dataset Files 21](#_Toc183878420)

[Figure 8: Data Augmentation and Transformation 22](#_Toc183878421)

[Figure 9: Data Generators 23](#_Toc183878422)

[Figure 10: Index to Class 23](#_Toc183878423)

[Figure 11: Show sample code 24](#_Toc183878424)

[Figure 12: Sample Output 24](#_Toc183878425)

[Figure 13: Check Corrupted Images 25](#_Toc183878426)

[Figure 14: Check for Class Imbalances 25](#_Toc183878427)

[Figure 15: Plot Pie Chart for Class Counts 25](#_Toc183878428)

[Figure 16: Pie Chart Plot for: Train, Validation and Test Dataset 26](#_Toc183878429)

[Figure 17: Check Image Dimensions 26](#_Toc183878430)

[Figure 18: Check One-hot Encoding Labels 27](#_Toc183878431)

[Figure 35: CNN Base Model Architecture 28](#_Toc183878432)

[Figure 36: Compile Model 29](#_Toc183878433)

[Figure 37: Early Stopping Callback 29](#_Toc183878434)

[Figure 38: Check Model Summary Code 29](#_Toc183878435)

[Figure 39: Model Summary 30](#_Toc183878436)

[Figure 40: Total Params 30](#_Toc183878437)

[Figure 41: Fit CNN Model 31](#_Toc183878438)

[Figure 61: CNN Hyperparameter Tuning Model Architecture 32](#_Toc183878439)

[Figure 62: CNN Hyperband Tuner 33](#_Toc183878440)

[Figure 63: CNN Hyperparameter Search 34](#_Toc183878441)

[Figure 64: Best CNN Hyperparameters 35](#_Toc183878442)

[Figure 65: Save CNN Tuned Model 35](#_Toc183878443)

[Figure 87: CNN Base Model Accuracy and Loss 36](#_Toc183878444)

[Figure 88: Model Accuracy and Loss Graphs 36](#_Toc183878445)

[Figure 89: CNN Base Model Classification Report 37](#_Toc183878446)

[Figure 90: CNN Base Model Confusion Matrix 38](#_Toc183878447)

[Figure 91: CNN Tuned Model Accuracy 38](#_Toc183878448)

[Figure 92: CNN Tuned Model Classification Report 39](#_Toc183878449)

[Figure 93: CNN Tuned Model Confusion Matrix 40](#_Toc183878450)

[Figure 94: Make Predictions using CNN Tuned Model 41](#_Toc183878451)

[Figure 95: Sample CNN Tuned Model Predictions Output 41](#_Toc183878452)

# List of Tables

[Table 1 - CNN LR 1 Performance Comparison 13](#_Toc180176099)

[Table 2: CNN vs LSTM-RNN Comparison 24](#_Toc180176100)

[Table 3: Model Comparison 104](#_Toc180176101)

# **Introduction**

# **Dataset & Algorithm Selection**

## Dataset

The dataset chosen for this assignment is an image dataset from Kaggle which consists of 25,000 images, all sized at 150x150 pixels, showcasing a variety of natural scenes from around the world. These images are basically categorised into six distinct categories, buildings, forest, glacier, mountain, sea, and street. The dataset was split into three parts already by the owner of the dataset, with 14,000 images designated for training in a train folder, 3,000 for testing in test folder, and 7,000 for making predictions. Originally, this dataset was part of an image classification challenge hosted by Intel on Analytics Vidhya. This project provides a solid base for building and training a neural network to classify these natural scenes with high accuracy.

Dataset Link: <https://www.kaggle.com/datasets/puneet6060/intel-image-classification/data>

## Algorithm Selection

### CNN

Convolutional Neural Networks (CNN) are specifically designed for processing structured grid data like images (Iyer, 2020). They utilize convolutional layers that apply filters to the input data, it can automatically learn to detect local patterns like edges and textures. The architecture usually consists of convolutional layers, pooling layers, and fully connected layers, these layers work together to reduce dimensionality while preserving essential spatial relationships (Saxena, 2022). The advantages of CNNs are particularly pronounced in image classification tasks. By automatically extracting features and reducing the need for manual feature engineering, CNNs simplify the modelling process and significantly enhance accuracy . They can effectively handle high-dimensional data and recognize objects regardless of their position in the image (Alzubaidi, 2021). However, CNNs require considerable computational power and can be sensitive to hyperparameter settings, which can affect their performance (Krichen, 2023). In this project, implementing CNNs because it has exceptional capability in image recognition tasks, therefore it's the most suited for classifying natural scenes.

# **Literature Review**

## *1.0 CNN - Using SE-RES-CNN* (Li, 2024)

### 1.1 Overview and Objectives

This research has a focus on sports image classification using deep learning. Li et el propose a custom model which is SE-RES-CNN which is an incorporation of Residual Network (ResNet) and Squeeze-and-Excitation (SE) attention mechanism. The aim of this study is to classify images into 100 sports categories using Kaggle as the source of the dataset. The goal is to enhance accuracy and generalization ability. The research compares the performance of the 3 models which are VGG16, SE-RES-CNN, and ResNet50. The purpose of the SE attention mechanism is to dynamically adjusts feature map weights, to improve the model’s ability to recognize important features and improve classification performance.

### 1.2 Dataset Description

The dataset was retrieved from Kaggle’s sport image classification which consists of 100 sports categories. The training set has 13493 images, the validation set has 500 images (5 per category), and the test set has 500 images (5 images per category). The images were resized to 224 x 224 x 3 dimensions. They were also converted to JPG format and duplicates were removed using a duplicated detection algorithm for data integrity between training, testing, and validation sets.

### 1.3 SE Attention Mechanism

This is a crucial component in modern deep learning models which allow the model to focus on the most relevant parts of the input data. Squeeze-and-Excitation Network operated by assigning varying weights to different feature map channels to improve useful ones and suppressing the less information ones.

### 1.4 ResNet

Improves the deep network performance by addressing the issues of vanishing gradients and overfitting. It permits deeper networks by using residual connections which operate on residual calculations between 2 convolutional layers. Thus, the network can lean more effective features without suffering from degradation problems.

### 1.5 Block Layer Design

The SE-RES-CNN model is designed using blocks. They are a combination of several layers working together as an independent subnetwork. They improve efficiency by reusing weights and parameters to reduce computational complexity and making the architecture modular for flexibility and quicker experimentation with different configurations. Each block in the SE-RES-CNN model is designed with convolutional, pooling, and fully connected layers to handle feature extraction and classification.

### 1.6 The SE-RES-CNN Model Architecture

The architecture model’s performance is fine-tuned using evaluation metrics such as accuracy and loss functions. The model is trained using Adam Optimization Algorithm with an initial learning rate of 5e-3, 128 (batch size), and 100 epochs, EarlyStopping with a patience of 10 epochs to disallow overfitting by stopping training if performance stagnates.

The experimental environment consisted of Windows 10, Anaconda3, Python 3.7, CUDA 11.2, TensorFlow 2.9.0 as the Software side and Intel Xeon W-2245 CPU (3.91 GHz), single NVIDIA Geforce RTX 3080 (10 GB) for the Hardware side.

### 1.7 Performance Comparison Analysis

Recall, F1-score, and Training and Validation Loss are the evaluation metrics.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Accuracy | Validation Accuracy | Training Loss | Validation Loss |
| VGG16 | 0.8316 | 0.9040 | 0.6353 | 0.4912 |
| ResNet50 | 0.9270 | 0.9060 | 0.2373 | 0.3320 |
| SE-RES-CNN | 0.9761 | 0.9660 | 0.0916 | 0.1058[[1]](#endnote-2) |

Table - CNN LR 1 Performance Comparison

The SE-RES-CNN outperformed the others significantly having the highest accuracy of both the training and validation sets. Its final accuracy on the test set was 0.98 while VGG16 and ResNet50 reached an accuracy of 0.92.

The recall and F1-score for SE-RES-CNN were superior to those of the VGG16 and the ResNet50 models, reflecting its ability to retrieve more accurate and complete information from the dataset. VGG and ResNet50 achieved a recall rate and F1-score of 0.92 and 0.93 respectively while SE-RES-CNN achieved 0.98 being the most effective model. Prediction time for SE-RES-CNN was 6 seconds for 500 test images with an average prediction of 0.012 seconds/image. This model was the most suitable for large-scale applications like automated sports video classification.

## ­2.0 ANN - Classification of Starling Image Using ANN (Rahman A. Y., 2021)

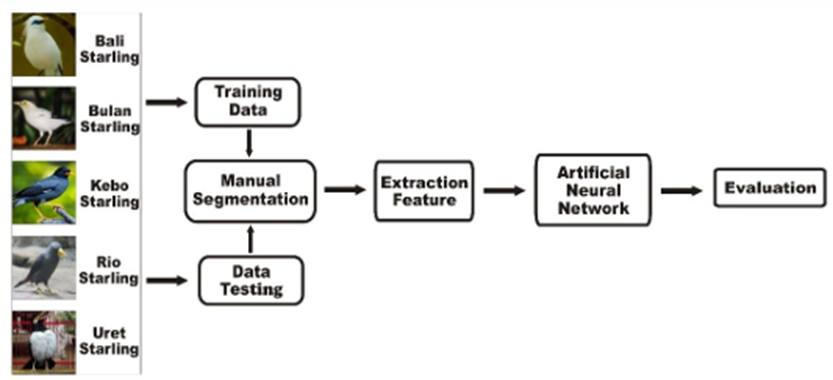


Figure - Architecture of Proposed Model

This research paper evaluates the classification of Starling birds. The architecture can be shown above. First, necessary image data must be collected. Then, the classification model should be trained and tested. The following stage involves feature extraction based on texture, shape, and colour features. The next stage applies ANN for classification. Finally, the last stage evaluates the performance the results.

### 2.1 Image Data

In this stage, 5 different types of birds were used for images to ensure accuracy. Each type of image (bird) has 3 kinds of images with different points of views. For robust testing, images were augmented by rotating from 0 Degree to 360 Degrees at an interval of multiple of 5. A total of 300 images were generated.

### 2.2 Manual Segmentation

Objects of interest from the background were segmented in this step for better focus on the features during analysis. Images were resized to maintain uniform resolution and image type. Thus, the classification is purely based on the object extract and not on irrelevant background features.

### 2.3 Feature Extraction

Key characteristics were obtained from the images in this stage, which are texture, shape, and colour. Unique properties of each image can be capture for accurate classification.

### 2.4 Texture Feature Extraction

For texture extraction, the Gray Level Co-occurrence Matrix (GLCM) was used. Image was converted to grayscale and texture features were computed from the grey values. Colour features were also extracted.

### 2.5 ANN Model

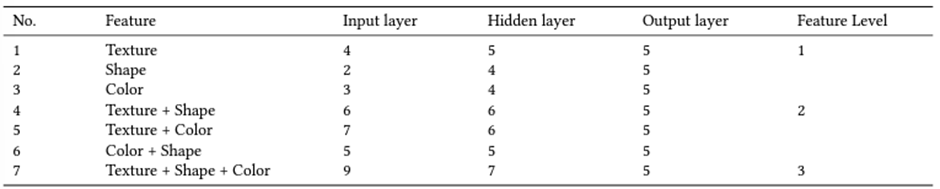


Figure - ANN Layers for the Architecture

ANN can recognize patterns withing the image data. The learning rate varied from 0.1 to 0.9 to identify the rate yielding the highest accuracy. The ANN architecture includes input layers for features, hidden layers for classed and output layers for results. Feature combinations were tested at different levels. Feature level 1 include individual texture, shape, and colour features, level 2 combination of 2 features such as texture and shapre, texture and color, and colour and shape, level 3 is the combination of all 3 features.

### 2.6 Result and Discussions

The seven different classification levels were tested based on learning rates ranging from 0.1 to 0.9.

Texture Feature Results: The highest accuracy was achieved at a learning rate of 0.5 was 68% using a 95:5 ration between training and testing data. The minimum accuracy observed was 23% with a learning rate of 0.1.

Shape Feature Results: The optimal accuracy for shape features was achieved at a learning rate of 0.3 with an accuracy of 75% at a 95:5 ratio. Lowest accuracy occurred at a learning rate of 0.2 being 31% with a ration of 5:95.

Color Feature Results: 100% accuracy was achieved with a learning rate of 0.5 with a 60:40 split ration and the lowest accuracy was 31% at a learning rate of 0.1.

Combination of all 3 features: Accuracy of 100% was achieved at a learning rate of 0.7 while the lowest accuracy was 36% with a learning rate of 0.1. There were learning rates which achieved 100% accuracy.

## 3.0 DNN - A Study by Zhou et al. (2024): Improved sports image classification using DNN and novel tuna swarm optimization

### 3.1 Dataset Description and Sports Classification

High-resolution images (240 x 320 pixels) extracted from sports videos sourced from platforms such as Olympic Channel, FIBA, ESPN. The dataset contains 7083 frames each labelled as one of 6 sports categories which are Basketball, Rugby, Badminton, Cricket, Volleyball, and Tennis. The data is evenly distributed across each category making it suitable for classification tasks such as player recognition, sport type identification, and performance analysis. Diverse lighting conditions and different image qualities are included in the dataset.

Data Partitioning: Training data is 80% while test data is 20% which is a standard practice in classification tasks.

### 3.2 Novel Tuna Swarm Optimization (NTSO)

The NTSO algorithm is a metaheuristic technique which utilized strategies such as parabolic food-seeking and parabolic food-seeking inspired from hunting behaviors of tuna fish to improve optimization performance. NTSO iteratively updated solutions to maximise the fitness value. It compares old and new solutions to select the best outcome. It boasts low computational complexity and faster convergence compared to traditional methods.

### 3.3 Proposed DNN/NTSO Model

The DNN enhanced NTSO, focuses on weight optimization across the network. The architecture includes:

1. Input Layer: Contains M nodes showing the dimensional features of the input data.
2. Hidden Layers: 2 Hidden Layers where number of nodes is evaluated by a formula involving input and output dimensions. Layers use the sigmoid activation function for non-linearity.
3. Output Layer: Contains N nodes representing the number of output classes (6 sports) and applies softmax or sigmoid function to generate class probabilities.

Optimization Process: NTSO algorithm optimized the DNN weights and biases by reducing cross-entropy loss function to enhance model’s accuracy.

### 3.4 Model Hyperparameters and Training Protocol

DNN has 4 layers with 64, 128, 256, and 512 neurons respectively. The Rectified Linear Unit is the activation function in all layers except the output layer employing a sigmoid function. Model has a dropout rate of 0.2 to not allow overfitting. Adam optimiser is used and learning rate is 0.001.

NTSO component’s hyperparameters has a population size of 50 and 100 iterations (maximum). Mutation rate is 0.05, scaling factors are between 0.8 and 1.2 and are set to adjust parameter bounds with valued bounded between -1 and +1.

The DNN model is pretrained on ImageNet for 10 epochs and fine-tuned on the SportsImg-18k dataset for an additional 20 epochs. Mentioned earlier is the 80/20 split. Mini-batched of 32 are used. Early stopping with 5 epochs, is used to prevent overfitting. A learning rate scheduler to decelerate the rate at the 10th and 15th epoch. Data augmentation techniques which include random rotations, zooming, shifting, and flipping are applied for robustness.

### 3.5 Simulation Results and Performance Evaluation

Validation of the model is done for the model through extensive simulations. Comparisons with other models (classification) showcase the superiority of the proposed DNN/NTSO model. Metrics used are accuracy, precision, recall, and F1-Score.

Confusion Matrix and Metrics: The matrix breaks down the true positives, false positives, true negatives, and false negatives for each sports category. The model has achieved an accuracy of 92%, a precision of 91%, a recall of 92%, and an F1-score of 0.91. Results obtained, demonstrate the proposed model to accurately classify sports categories with a means F1-score of 0.8787 surpassing other competing approaches.

## 4.0 RNN – A Study by Gaafar et al. (2021): Comparative Analysis of Performance of Deep Learning Classification Approach based on LSTM-RNN for Textual and Image Datasets

The study investigates the effectiveness of Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs) in classifying both textual and image data. The research highlights the significance of transfer learning, which facilitates the reuse of pre-trained models for tasks in related domains, thereby enhancing model performance across various applications. This literature review synthesizes key findings regarding the performance of LSTM-RNN architectures in handling diverse datasets.

### 4.1 Overview

RNNs are designed to process sequential data by maintaining a hidden state that is updated at each time step based on previous inputs (Gaafar, Dahr, & Hamoud). However, traditional RNNs face challenges such as vanishing gradients, which limit their ability to learn long-term dependencies. The introduction of LSTMs has addressed these shortcomings by incorporating memory cells and gating mechanisms that allow for better retention of information over extended sequences. This capability is crucial for tasks involving textual data, where context plays a significant role in understanding meaning.

### 4.2 Evolution of RNN Architectures

#### Traditional RNNs

A diagram of a complex structure

Description automatically generated

Figure - Traditional RNN Architecture (Paoletti, Haut, Plaza, & Plaza, 2020)

The foundational architecture of RNNs allows them to process sequences by maintaining hidden states that are updated at each time step based on input data and previous states (Paoletti, Haut, Plaza, & Plaza, 2020). However, traditional RNNs are limited by issues such as vanishing and exploding gradients, which impede their ability to learn long-term dependencies effectively. The study emphasizes that these limitations necessitated the development of more sophisticated architectures.

#### Long Short-Term Memory (LSTM)

LSTMs were introduced to overcome the shortcomings of traditional RNNs. By incorporating memory cells and gating mechanisms, LSTMs can selectively remember or forget information over extended periods. This architecture has been pivotal in improving performance across various NLP tasks, including language modelling and machine translation.

#### Gated Recurrent Units (GRU)

The study also discusses GRUs, which simplify the LSTM architecture by combining input and forget gates into a single update gate. GRUs have been shown to perform comparably to LSTMs while requiring fewer parameters, making them a popular choice for many applications in NLP.

#### Bidirectional RNNs

Bidirectional RNNs enhance the traditional RNN framework by processing input sequences in both forward and backward directions. This approach allows the model to capture context from both past and future inputs, significantly improving performance in tasks such as sentiment analysis and named entity recognition.

#### Simple Recurrent Unit - SRU

A group of math symbols

Description automatically generated

Figure - Simple Recurrent Unit Formula (Paoletti, Haut, Plaza, & Plaza, 2020)

The Simple Recurrent Unit (SRU) is a type of recurrent neural network (RNN) designed to address the limitations of traditional RNNs, particularly in handling long-term dependencies while simplifying the gating mechanisms. Unlike standard RNN architectures that rely heavily on complex gating structures, SRUs utilize a more straightforward approach that allows for faster training and improved performance on sequential tasks.

The SRU architecture is characterized by its unique design that incorporates parallelism similar to convolutional and feedforward networks. This design enables each dimension within the state to operate independently, which enhances the model's ability to capture long-term dependencies without succumbing to the vanishing gradient problem. Some features of the SRU are such that the computations for forget and reset gates in SRUs are independent, eliminating interdependence and simplifying the overall architecture. This independence allows for a more efficient training process.

### 4.3 Dataset Description

The study utilizes two distinct datasets to evaluate the performance of deep learning classification approaches based on Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs).

#### 4.3.1 Fashion MNIST Dataset

The first dataset is the Fashion MNIST, which consists of 70,000 grayscale images categorized into 10 clothing classes, including T-shirts, trousers, and sneakers. This dataset serves as a benchmark for image classification tasks and is designed to provide a more challenging alternative to the original MNIST dataset of handwritten digits. Each image in the Fashion MNIST dataset is 28x28 pixels, and the dataset is split into a training set of 60,000 images and a test set of 10,000 images. The study highlights the importance of this dataset in assessing the performance of convolutional neural networks (CNNs) and LSTM-RNN models in classifying visual data.

#### 4.3.2 IMDB Movie Reviews Dataset

The second dataset employed in the research is the IMDB movie reviews dataset, which consists of 50,000 reviews labelled as either positive or negative, making it suitable for binary sentiment analysis tasks. The dataset is divided into 25,000 training reviews and 25,000 testing reviews. The reviews are pre-processed to remove noise and ensure consistency, including tokenization and padding to handle varying lengths of text inputs. This dataset allows for a comprehensive evaluation of LSTM-RNN models in understanding contextual information and managing sequential dependencies inherent in textual data.

### 4.4 Algorithms

The study focuses on two primary algorithms: Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs).

#### 4.4.1 Long Short-Term Memory (LSTM)

LSTMs are an advanced type of RNN designed to mitigate issues related to vanishing gradients, which hinder traditional RNNs from learning long-term dependencies effectively. By incorporating memory cells along with input, output, and forget gates, LSTMs can selectively retain or discard information over extended sequences. This capability makes them particularly effective for tasks involving sequential data such as natural language processing and time series forecasting.

#### 4.4.2 Recurrent Neural Networks (RNNs)

Traditional RNNs process sequential data by maintaining a hidden state that is updated at each time step based on previous inputs. However, their performance can degrade when handling long sequences due to the vanishing gradient problem. The study emphasizes that while RNNs can capture short-term dependencies effectively, they are often outperformed by LSTMs in tasks requiring the retention of information over longer periods.

### 4.5 Proposed Models

The research proposes several models that leverage LSTM architectures for both textual and image classification tasks.

#### 4.5.1 LSTM-RNN Model for Text Classification

For textual data classification using the IMDB dataset, the proposed model utilizes an LSTM-RNN architecture that processes input sequences through multiple layers of LSTM units. This model captures contextual relationships within the text and effectively manages varying input lengths through padding techniques.

#### 4.5.2 CNN-LSTM Hybrid Model for Image Classification

In addressing image classification tasks with the Fashion MNIST dataset, the study proposes a hybrid model that combines CNN layers for feature extraction with LSTM layers for sequence modeling. This architecture allows the model to leverage spatial hierarchies captured by CNNs while utilizing LSTMs to maintain temporal dependencies across different image features.

### 4.6 Hyperparameter Tuning

Hyperparameter tuning plays a crucial role in optimizing model performance across both datasets.

#### 4.6.1 Techniques Employed

The study employs techniques such as grid search and random search to identify optimal hyperparameters for both LSTM-RNN and CNN-LSTM models. Key hyperparameters considered include learning rate. Adjusting the learning rate is critical for effective convergence during training. Batch Size, the size of mini batches used during training impacts training stability and convergence speed, and number of epochs, which determines an appropriate number of epochs helps balance between underfitting and overfitting.

#### 4.6.2 Results of Hyperparameter Tuning

Through systematic hyperparameter tuning, the study reports improved accuracy rates for both LSTM-RNN models on textual data and CNN-LSTM models on image data. The findings indicate that careful tuning can significantly enhance model performance by optimizing how well they learn from their respective datasets. This literature review provides a detailed examination of key components from the study regarding dataset descriptions, algorithms utilized, proposed models, and hyperparameter tuning strategies employed in evaluating deep learning classification approaches based on LSTM-RNN architectures across textual and image datasets.

### 4.7 Performance Analysis

#### 4.7.1 Image Classification Performance

The study utilizes the Fashion MNIST dataset to evaluate the performance of LSTM-RNN models in image classification tasks. The dataset comprises 70,000 images categorized into nine labels such as T-shirt/top, Trouser, and Sneaker. The results indicate that deep learning models leveraging CNN architectures outperform traditional methods significantly due to their ability to extract intricate features from images effectively. The study employs accuracy as a primary metric for evaluating model performance. CNN-based models achieved high accuracy rates due to their layered architecture that captures spatial features effectively.

#### 4.7.2 Textual Classification Performance

For textual data classification, the study uses the IMDB movie reviews dataset, consisting of customer sentiments labelled as positive or negative. The LSTM-RNN model demonstrated a strong capacity for understanding context and managing sequential dependencies inherent in language data. The study reports accuracy rates for LSTM-RNN models that exceed those of traditional machine learning classifiers such as Support Vector Machines (SVM) and Random Forests. This improvement can be attributed to LSTMs' ability to maintain long-term dependencies and adapt to varying input lengths.

### 4.8 Comparative Results

The comparative analysis reveals that the LSTM-RNN model outperforms conventional RNNs in both textual and image classification tasks due to its advanced architecture that mitigates issues related to vanishing gradients. While CNNs excel in image classification due to their spatial feature extraction capabilities, LSTMs demonstrate superior performance in textual data classification by effectively capturing sequential information.

Table : CNN vs LSTM-RNN Comparison

|  |  |  |
| --- | --- | --- |
| **Model** | **Dataset Type** | **Accuracy (%)** |
| CNN | Fashion MNIST | 95.3 |
| LSTM-RNN | IMDB Reviews | 90.5 |

The research highlights the effectiveness of deep learning models, particularly LSTM-RNN architectures, in classifying both textual and image datasets. While CNNs remain dominant in image classification tasks due to their hierarchical feature extraction capabilities, LSTMs prove invaluable for sequential data processing in NLP applications. The findings underscore the importance of selecting appropriate model architectures based on the nature of the data being processed, paving the way for future research into hybrid models that combine the strengths of both CNNs and RNNs. This literature review synthesizes key insights from the study while providing an overview of the comparative performance analysis conducted on various deep learning classification approaches based on LSTM-RNN architectures across different datasets.

# **Data Preparation**

In this section, it will discuss and explain the data understanding part, where exploratory data analysis (EDA) will be performed using either numerical representation, graphs or charts to help us understand the dataset structure and flaws. Then, data cleaning will be performed to check the images and transform them into proper consistent format for the deep learning models to work without issues.

**Import Library and Load Dataset**

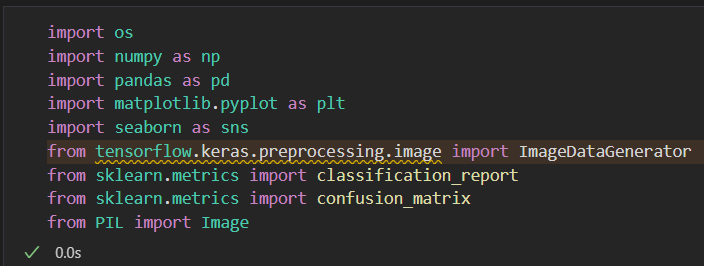


Figure : Import Library

These are the necessary libraries for the deep learning project, brief explanation for each library:

* **os**: Provides functions to interact with the operating system like file paths.
* **numpy as np**: For numerical operations and handling arrays.
* **pandas as pd**: For data manipulation and analysis, especially with dataframes.
* **matplotlib.pyplot as plt**: Enables plotting and visualization of data.
* **seaborn as sns**: A library for data visualization, built on top of Matplotlib for easier plotting.
* **tensorflow.keras.preprocessing.image.ImageDataGenerator**: Generates batches of image data for training models with real-time data augmentation.
* **sklearn.metrics.classification\_report**: Generates a text report showing main classification metrics.
* **sklearn.metrics.confusion\_matrix**: Creates a confusion matrix to evaluate classification performance.
* **PIL.Image**: A module from Pillow for opening and manipulating images.

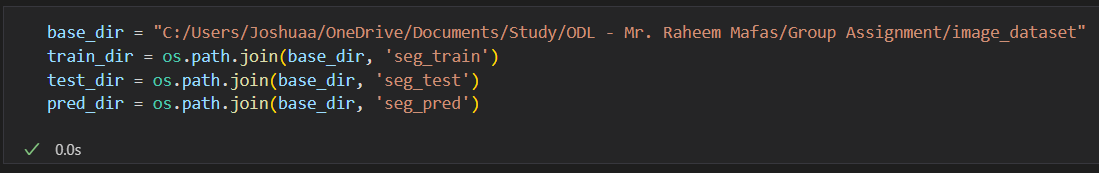


Figure : Location of Dataset Files

The code block above is to **locate the dataset files**, under the parent folder of image\_dataset, the dataset originally already split into three folders, for training data is in ‘seg\_train’, testing or evaluation data is in ‘seg\_test’, ‘seg\_pred’ consist of images for predictions using a complete model later in the project.

**Data Augmentation and Transformation**

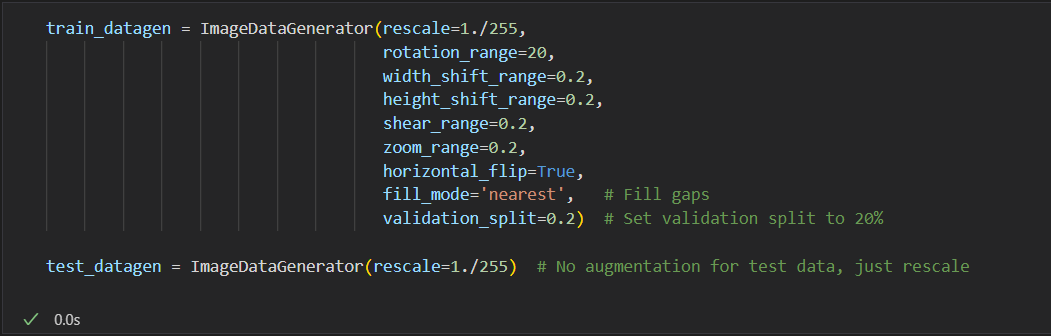


Figure : Data Augmentation and Transformation

‘ImageDataGenerator’ is a Keras utility that helps perform real-time **data augmentation**, it is used for preprocessing image data before feeding it into a neural network. It automatically modifies images in various ways to artificially increase the size and diversity of the training set to help improve model robustness. In the code, ‘train\_datagen’ is used to apply some transformations, such as **rotation, zoom, and flipping**, to help the model generalizes better to unseen data. One important note here is ‘validation\_split=0.2’, what this does is that it will further split the train dataset into **80% training data and 20% validation data** if ‘train\_datagen’ is used.

However, it only rescales the test data without augmentation since it’s meant for evaluation. This approach is efficient for image datasets where variety and complexity are key to improving model performance, especially when the available data size is limited.

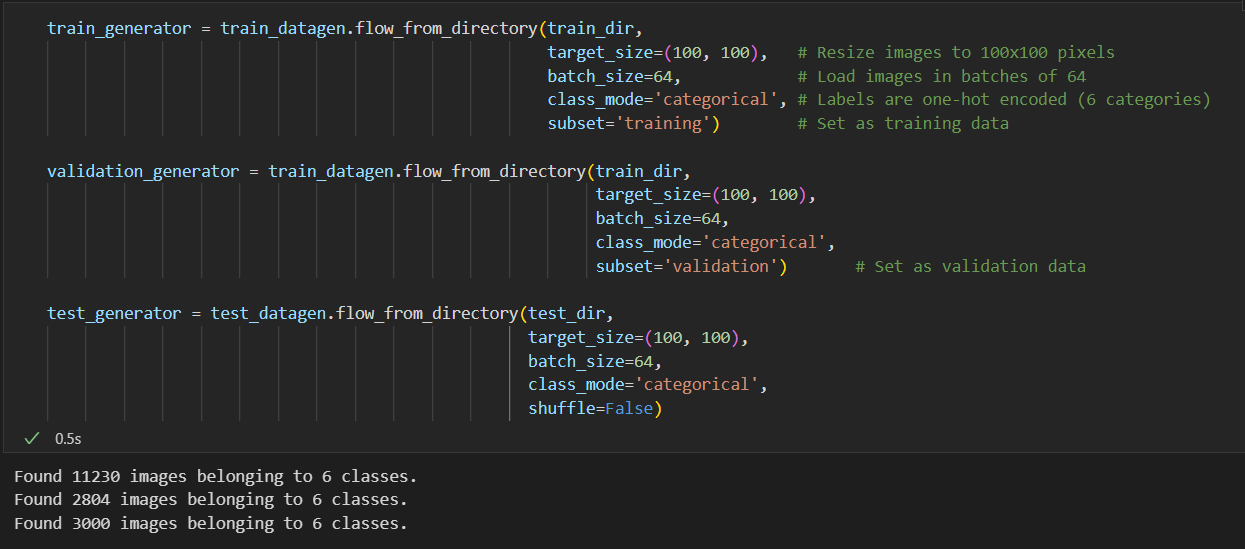


Figure : Data Generators

This code sets up **data generators** for training, validation, and testing by loading images from directories and processing them in batches. The images were resized to (100-pixels x 100-pixels), each batch size is 64 images at a time. For ‘train\_generator’, it sets the subset= ‘training’ option to ensure only the training subset (80% in this case due to the earlier validation\_split=0.2) is used for training. While ‘validation\_generator’ was set to subset=‘validation’. This approach **streamlines the training process** without needing to manually define x\_train, y\_train, x\_test, or y\_test. Instead, the data is loaded directly from the directories in batches and its more memory efficient. Additionally, the one-hot encoded labels for the classification task are generated automatically by using the code class\_mode= ‘categorical’, therefore this method is an ideal choice for the project. The ‘train\_generator’, ‘validation\_generator’, and ‘test\_generator’ seamlessly provide data for the model during training, validation, and testing phases to ensure consistency and efficiency throughout the pipeline. This allows the model to focus on learning from a diverse and well-preprocessed dataset also able to perform well on unseen data.

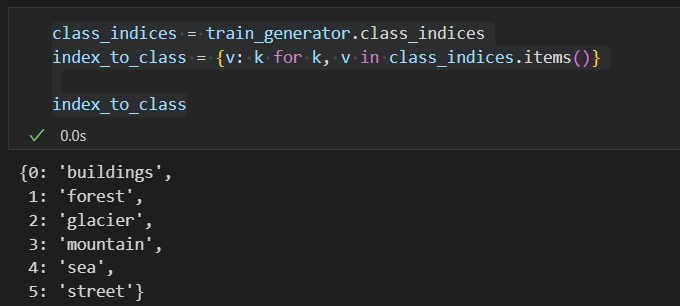


Figure : Index to Class

This step here **creates a mapping** between the numerical class indices generated by the ‘train\_generator’ and their corresponding class labels. The ‘class\_indices’ dictionary maps class names to integers, and by reversing this mapping using a dictionary comprehension (‘index\_to\_class = {v: k for k, v in class\_indices.items()}’), we can easily interpret model predictions by converting them back from numerical to human-readable class labels.

**Exploratory Data Analysis**

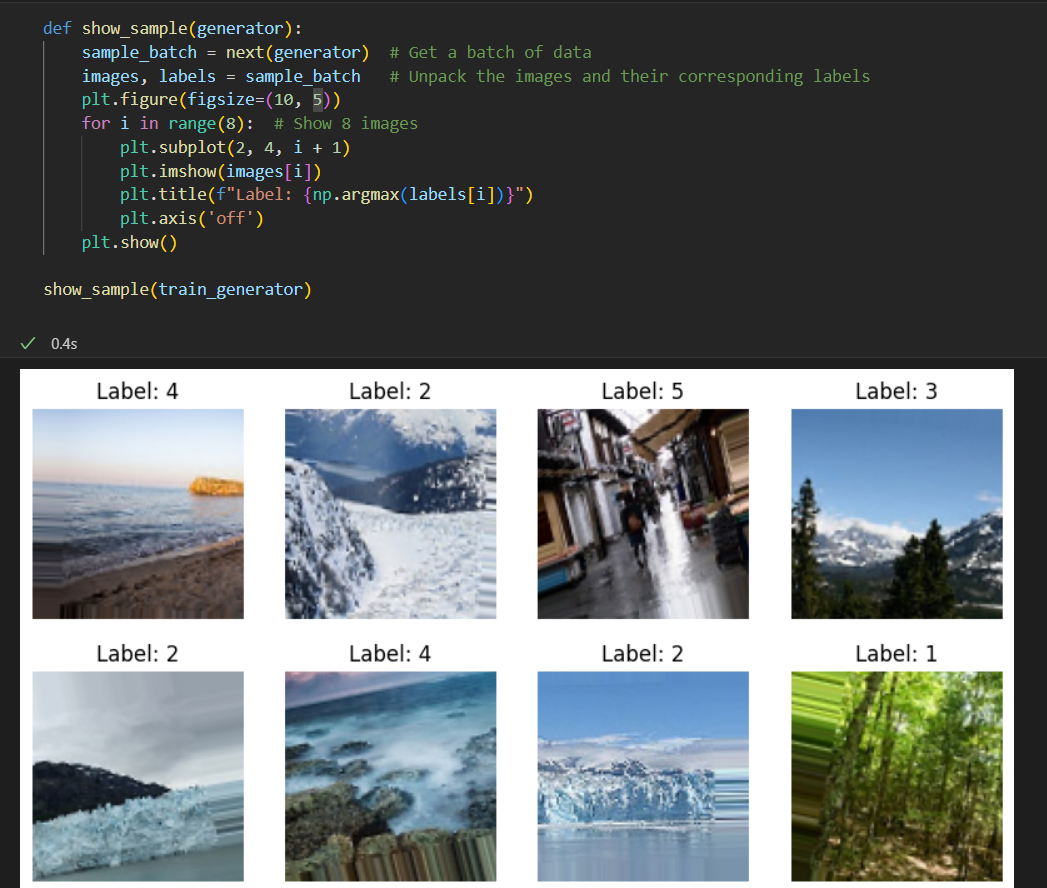


Figure : Show sample code

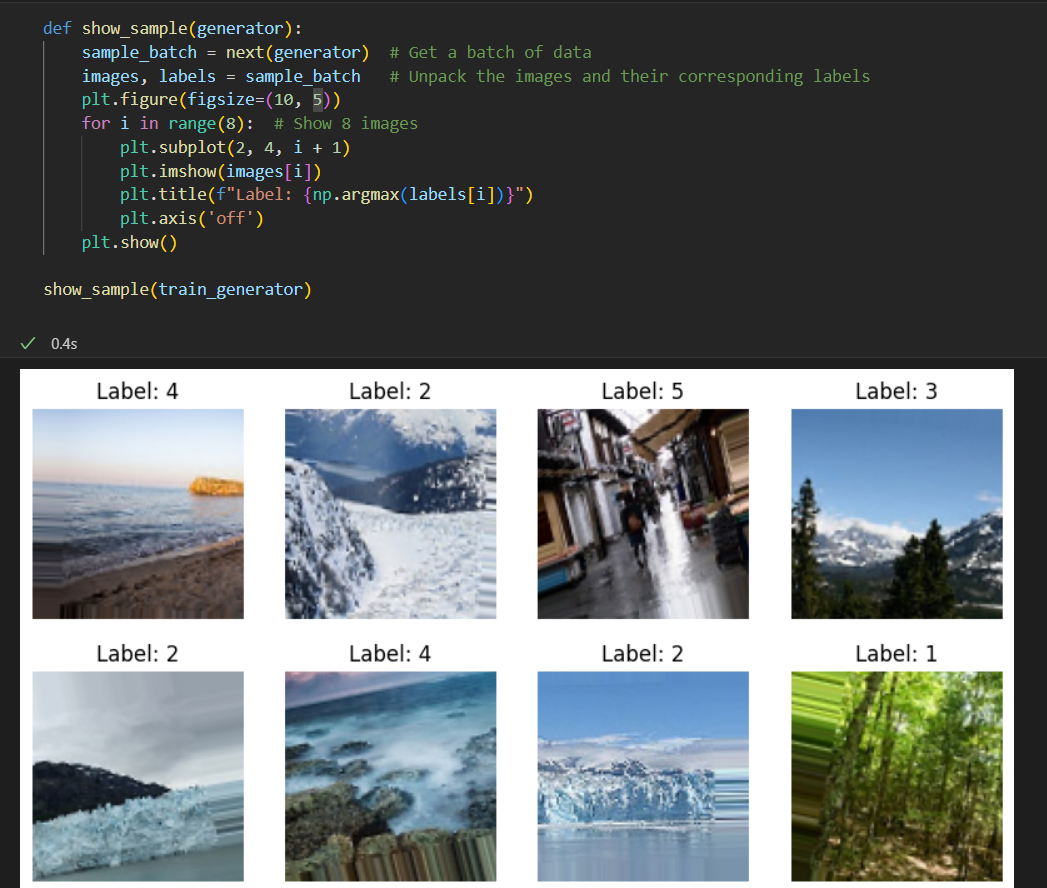


Figure : Sample Output

Now we start our EDA process, EDA is essential for gaining insights, ensuring data quality, and informing the modelling process, to let the stakeholders make more effective data-driven decision-making. Firstly, **check whether the data has properly labelled** to their corresponding categories by showing some sample, as well as identifying whether the data has been shuffled and taken from all the categories and not just one category.



Figure : Check Corrupted Images

The ‘check\_corrupted\_images’ function helps to **check any corrupted images** in both training and testing directory to ensure the data are reliable and usable. Training a model on corrupted image can affect the model performance and have misleading results, so its important to prevent that from happening.

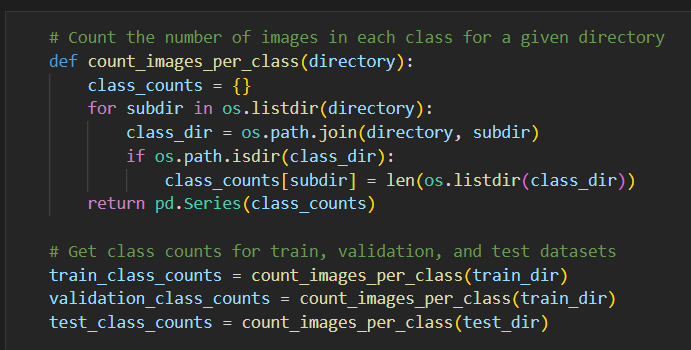


Figure : Check for Class Imbalances

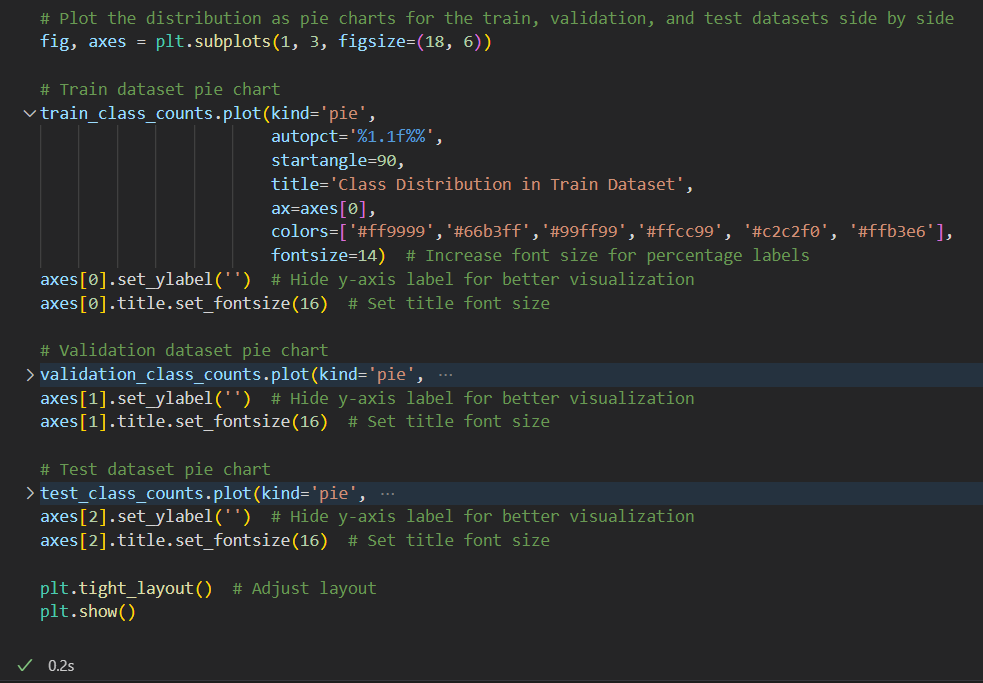


Figure : Plot Pie Chart for Class Counts

The code serves the purpose of **analysing and visualizing the class distribution** of images across three datasets, training, validation, and test. By counting the number of images in each class within these directories, this code provides valuable insights into the dataset's composition to ensure the classes are balanced. Imbalances in class distribution can lead to biased models that perform poorly on underrepresented classes. This information was visualised using pie charts to easier **to identify potential class imbalance.**

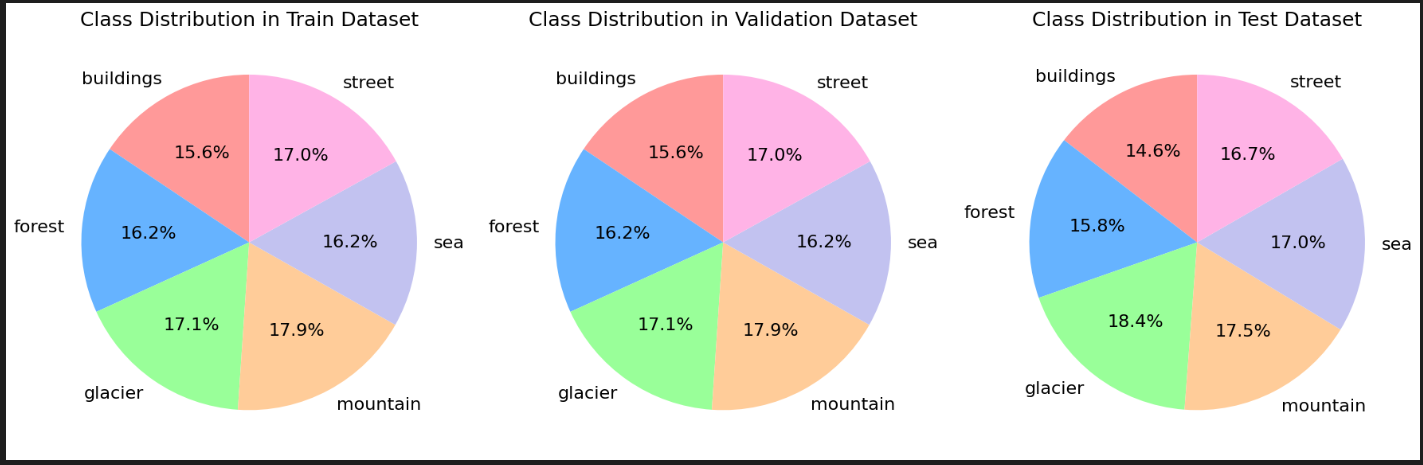


Figure : Pie Chart Plot for: Train, Validation and Test Dataset

The result from the pie charts show that all three categories have **balanced class data**, therefore, no further preprocessing needed to address class imbalance issue.

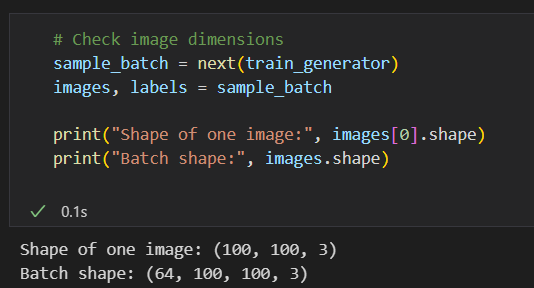


Figure : Check Image Dimensions

Before modelling, let’s **understand our image dimensions**, the results of the shape of one image shows (100,100,3) meaning that each image has a **height of 100 pixels and width of 100 pixels** which are correctly resized previously, with **3 colour channels, the RGB.** We also verified the batch shape which then shows (64,100,100,3) meaning **64 images are loaded at once as one batch** for training. Checking the shapes helps to ensure that the data being fed into the neural network is consistent with the model's expected input shape. It also helps to verify that the data augmentation and preprocessing steps are done correctly.

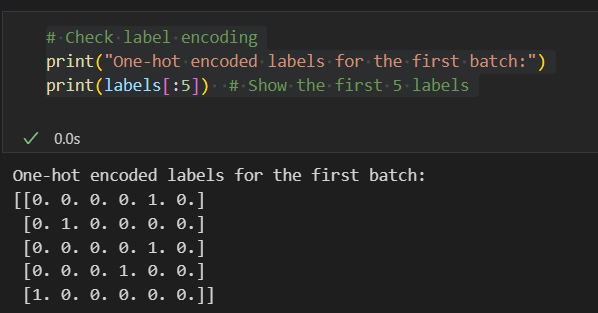


Figure : Check One-hot Encoding Labels

Lastly, we should also **check whether the labels are being processed or one-hot encoded correctly** and that they match the expected format for the model's output layer. Each label is represented as a binary vector corresponding to the class index. For example, [0,0,0,0,1,0] indicates that the label belongs to the fifth class. This encoding method is commonly used in multi-class classification problems to ensure that the model can interpret the categorical labels correctly.

# **Model Building**

## Convolutional Neural Network (CNN)

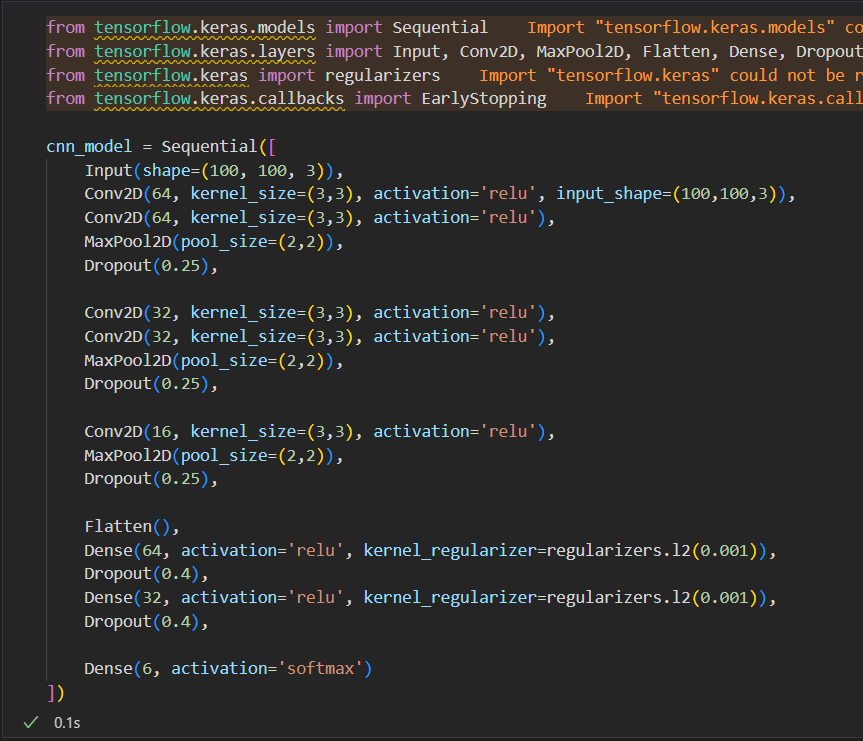


Figure : CNN Base Model Architecture

For Convolutional Neural Network (CNN), is it also built using the Keras Sequential API, CNN is particularly effective for image classification tasks due to its ability to automatically extract hierarchical features from images. The model started off by defining an **input layer** designed for images of size 100x100 pixels with three color channels (RGB).Then, it is made up of **multiple convolutional layers**, each of which is succeeded by a **max pooling layer.** This appropriately downsamples and preserves key characteristics by reducing the spatial dimensions of the feature maps. The use of convolutional layers (Conv2D) with **ReLU activation functions** enables the model to learn complex patterns and features from the training images. To prevent the model from overfitting, **dropout layers** are utilized after convolutional and dense layers; to increase robustness in feature learning, it randomly sets a fraction of input units to 0 during training. The final output layer employs **softmax activation** for multiclass classification, so that the model to predict one of six classes. Additionally, **L2 regularization** is applied to the dense layers to penalize large weights and to further enhancing the model's generalization ability. Overall, this base CNN model architecture is able to balance complexity and performance.

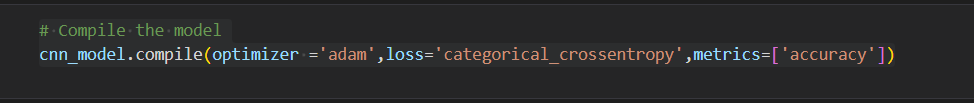


Figure 36: Compile Model

Then, **compile the CNN model** to configures the learning process. The **Adam optimizer** is chosen for its adaptive learning rate capabilitie because it is suitable for a variety of deep learning tasks. The loss function selected is **categorical\_crossentropy**, which is **ideal for multiclass classification problems** as it measures the performance of the model in predicting the probability distribution over multiple classes. The metric set for evaluation is **accuracy,** its more straightforward to assess how well the model is performing by comparing the predicted class labels to the true labels during training and validation.

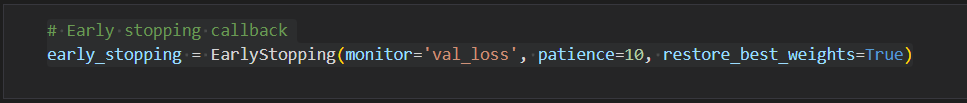


Figure 37: Early Stopping Callback

Next, the ‘early\_stopping’ callback is implemented to monitor the validation loss during training. This mechanism is crucial for **preventing overfitting**, as it allows the training process to halt if the validation loss does not improve for a specified number of epochs (in this case is 10 epochs). By restoring the best weights from the epoch with the lowest validation loss, the model is ensured to have the best possible performance on unseen data.

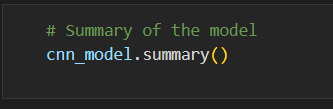


Figure 38: Check Model Summary Code

Finally, the ‘summary()’ method is called on the model to provide a better overview of its architecture. This summary includes details such as the types and order of layers, the output shape of each layer, the number of parameters in each layer, and the total number of trainable parameters in the model. This information able to help in the understanding of the model's complexity and the computational resources it may require, so that the researcher have better-informed decisions regarding training and deployment.

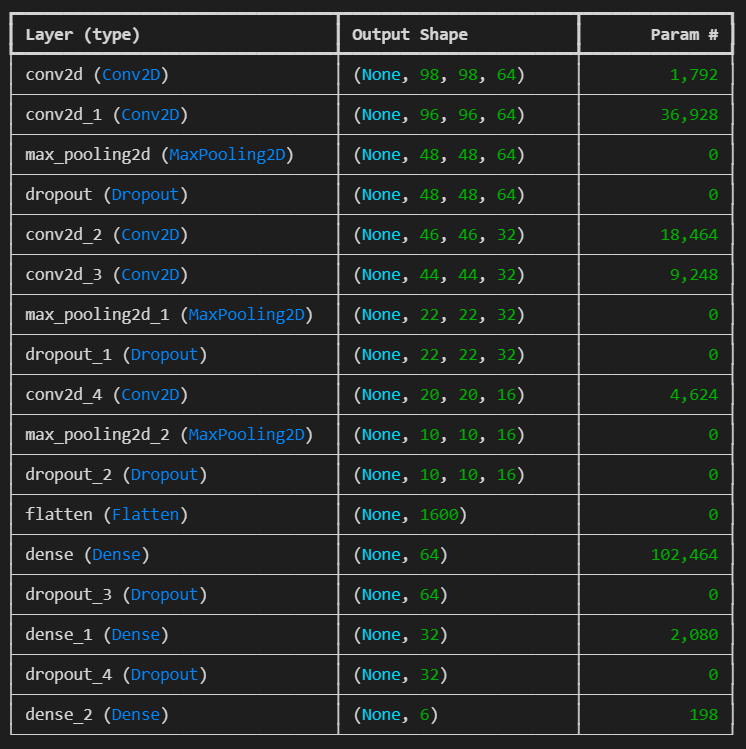


Figure 39: Model Summary

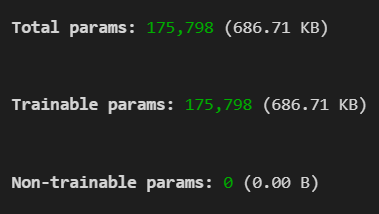


Figure 40: Total Params

The model summary provides a detailed overview of the architecture of the CNN model. Sequential meaning it consists of a linear stack of layers. The layers were explained in the CNN model building code, it’s completely the same in this summary. Overall, the model contains a total of 175,798 parameters, all of which are trainable, indicating that the network is capable of learning from the data during the training process.

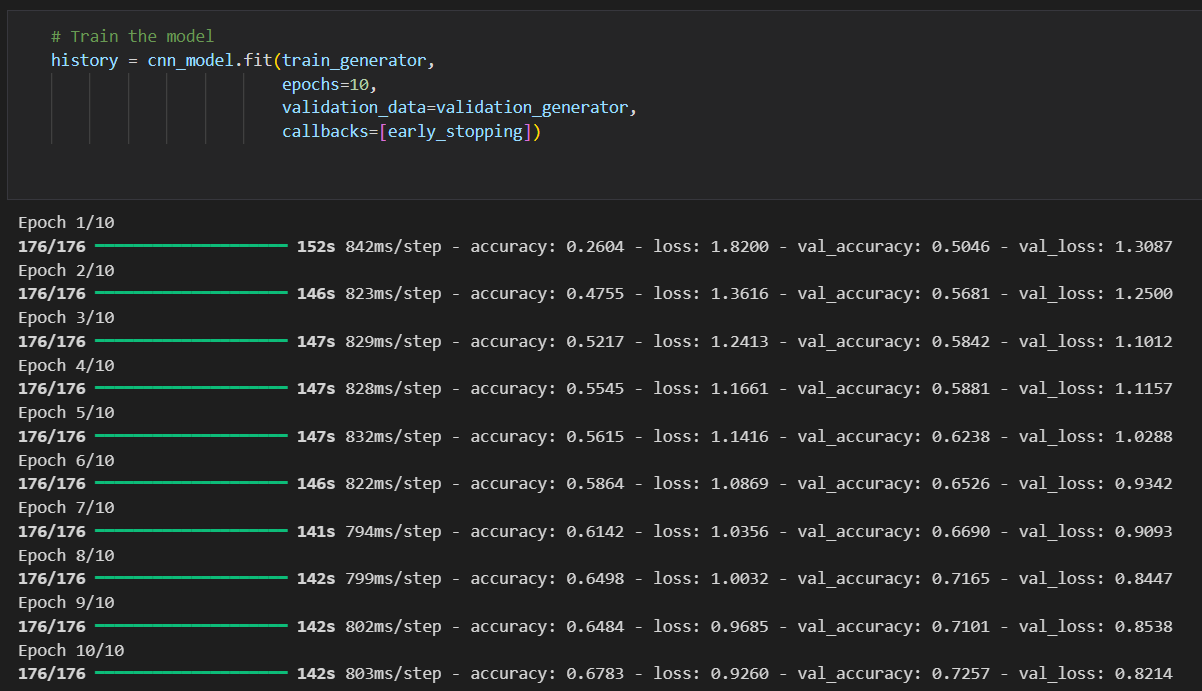


Figure : Fit CNN Model

Lastly, to train the model, the ‘fit()’ method from Keras is used. The training is performed on the ‘train\_generator’ that provided the batches of training data and labels, while the model is validated on the ‘validation\_generator’ during each epoch to monitor its performance on unseen data. The training process is set to run for 20 epochs so that the model can iteratively learn from the training data. Additionally, the ‘EarlyStopping’ callback is included, which halts the training if the validation loss does not improve for 10 epochs, thus preventing overfitting and ensuring that the best model weights are restored at the end of training. The ‘history’ will contain information about the training and validation loss and accuracy for each epoch, this is useful for analyzing the model's performance using a graph to visualise and making further improvements.

# **Model Tuning**

## CNN Hyperparameter Tuning



Figure : CNN Hyperparameter Tuning Model Architecture

This is the code for hyperparameter tuning of CNN base model using **Keras Tuner** to optimize key parameters. The model architecture is flexible, with tuneable components such as the number of filters in the convolutional layers, the dropout rates, the number of units in the dense layers, and the learning rate for the Adam optimizer. The researcher used ‘hp.Int’ to dynamically adjust the number of filters in each convolutional layer that’s ranging from 32 to 128 to identify the best filter size for feature extraction. Similarly, ‘hp.Float’ was applied to tune the dropout rates between 0.2 and 0.5 to prevent overfitting by disabling a fraction of neurons during training. The number of units in the dense layers is also optimized, with a range set between 32 and 128, to strike the right balance between model complexity and performance. Additionally, the learning rate of the Adam optimizer is tuned using ‘hp.Float’ over a logarithmic scale (1e-4 to 1e-2), ensuring efficient training without overshooting or slow convergence. By leveraging Keras Tuner, the search for the best combination of these hyperparameters can be automated and significantly improve the model performance without heavy manual trial and error.

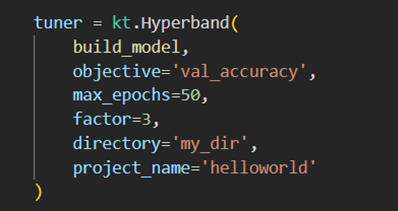


Figure : CNN Hyperband Tuner

Keras Tuner's Hyperband algorithm was used here to optimize the hyperparameters of a CNN model. Hyperband is an efficient method for hyperparameter tuning as it dynamically allocates resources, starting with many models trained for a few epochs and then progressively focusing on the best-performing ones. The objective is to maximize validation accuracy and to ensure the model generalizes well to unseen data. In this Hyperband setup, the number of epochs run per trial is determined dynamically, with ‘max\_epochs=50’, but early stopping may halt training sooner based on performance. Hyperband begins by testing many configurations with a few epochs and then allocates more resources to better-performing models, reducing configurations by a factor of ‘3’ after each round. While multiple trials are evaluated, only the best-performing ones are saved. The number of saved trials is less than the total explored, as weaker models are eliminated early on. Ultimately, only the top models achieving the highest validation accuracy are retained for further analysis. The results and logs are stored in ‘my\_dir’ under the project name ‘helloworld’, this way its more organized and efficient.

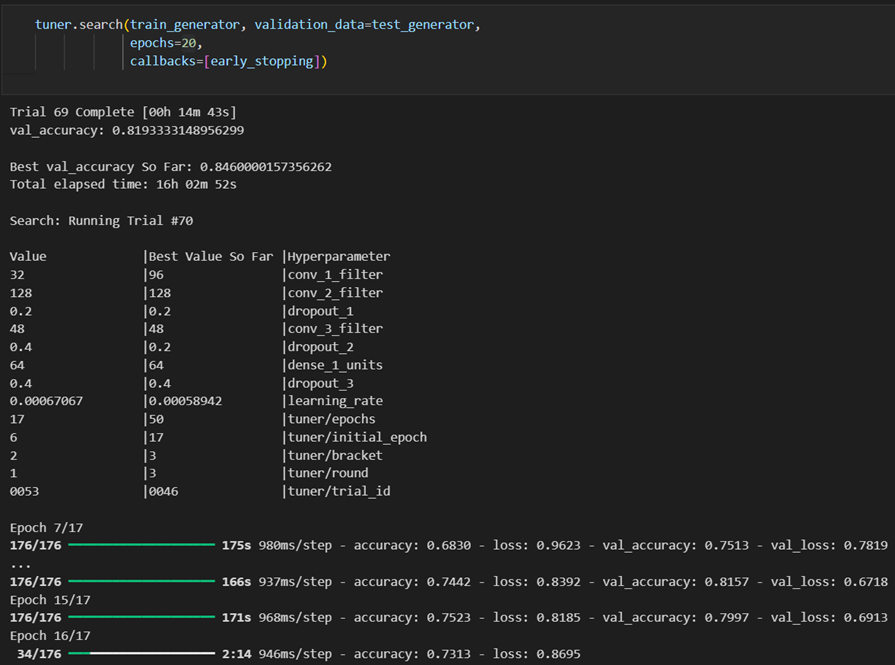


Figure : CNN Hyperparameter Search

Now, let’s **search for the best model** using the ‘tuner.search()’ function, the search was running for 70 trials over a span of 16 hours, but the process was **manually interrupted** ultimately due to diminishing returns, as only **marginal improvements** were observed in validation accuracy **after a significant time.** At the time of interruption, the best validation accuracy achieved was **0.8460**, and subsequent trials were showing smaller improvements , for example, Trial 69 yielded 0.8193 val\_accuracy. The tuning process explored variations in the model’s architecture, such as different dropout rates and learning rates, but ultimately the improvements plateaued, leading to the decision to halt the search prematurely to save time.

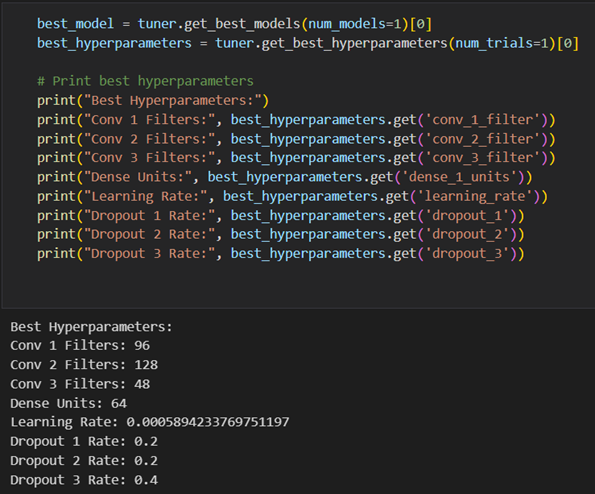


Figure : Best CNN Hyperparameters

The code above displays the best hyperparameters identified by the Keras Tuner during the hyperparameter optimization process. After executing the tuning, the best model and its corresponding hyperparameters are obtained using the‘get\_best\_models’ and ‘get\_best\_hyperparameters’ methods respectively. The output showcases key hyperparameters that were tuned, including the number of filters for the **first three convolutional layers (96, 128, and 48, respectively)**, the number of units in the **dense layer (64),** and **the learning rate (approximately 0.00059**). Additionally, the **dropout rates for the first, second, and third layers** are specified as **0.2, 0.2, and 0.4** respectively. These hyperparameters are crucial for optimizing the model's performance and generalization on unseen data, also reflects that these configurations are those that yielded the best validation accuracy during the tuning process.



Figure 65: Save CNN Tuned Model

Lastly, the best model was saved into ‘tuned\_cnn\_model.keras’. It’s easier to restore and deploy the trained model using .keras format, so that all configurations are saved for future use or further fine-tuning.

# **Model Evaluation & Discussion**

## CNN Base Model

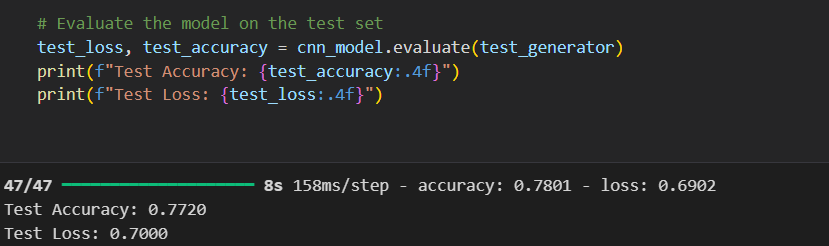


Figure : CNN Base Model Accuracy and Loss

To evaluate the performance the CNN base model, the **test accuracy of 0.7720** indicates that approximately **77.20%** of the images in the test dataset **were classified correctly**, which is a **above average performance** depending on the complexity of the task and the diversity of the dataset. The **test loss of 0.7000** represents the model's **average error** when making predictions on the test data. A lower loss value generally indicates a better fit to the data, although the interpretation of the loss value depends on the specific loss function used.

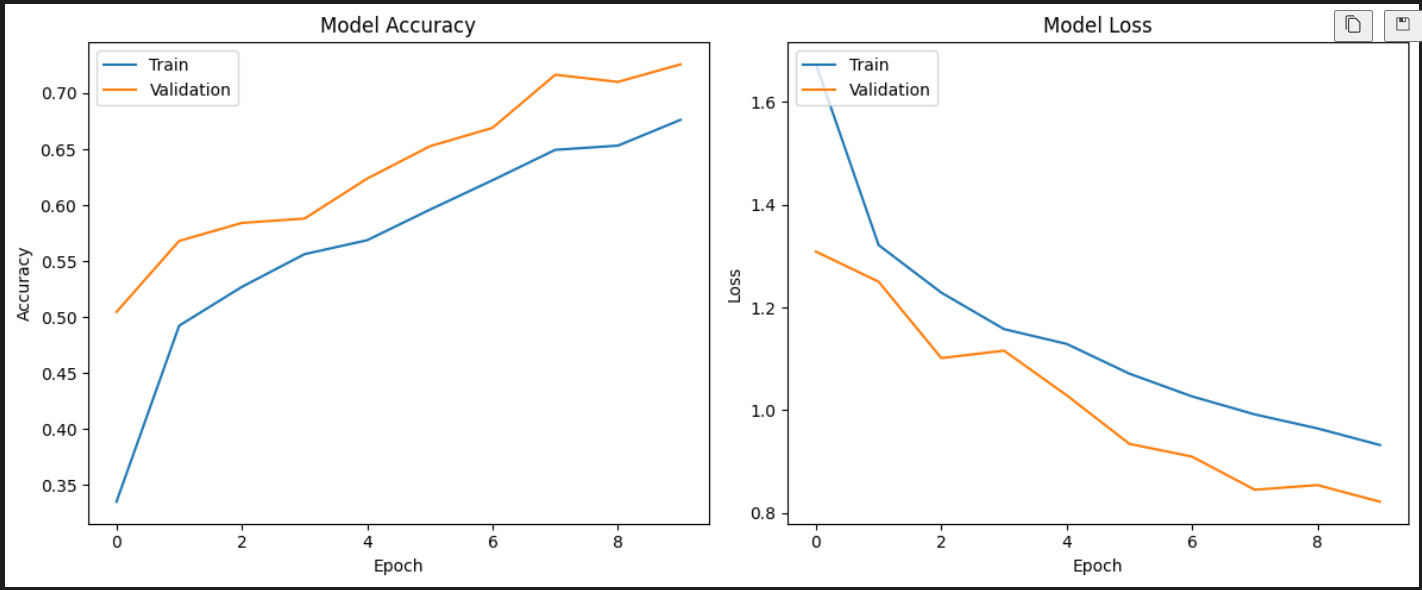


Figure 88: Model Accuracy and Loss Graphs

The model accuracy and loss graphs show that the model is **experiencing a bit of overfitting.** While the training accuracy shows a consistent increase, reaching about 0.75, and the training loss declines, indicating effective learning, the validation accuracy only rises to approximately 0.60 and the validation loss remains higher than the training loss. This inconsistency could tell that the model performs well on the training data but struggles to generalize to the validation set. Further steps might need to address this issue.

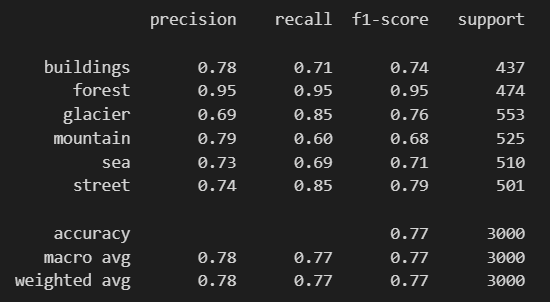


Figure 89: CNN Base Model Classification Report

The classification report looked into the precision, recall, and F1-score metrics of the CNN base model's which are critical for classification problems. The model shows moderate performance in the "buildings" class, with a precision of 0.78, recall of 0.71, and F1-score of 0.74, indicating it correctly identifies a significant number of buildings but also misses several. In contrast, the "forest" class excels with perfect scores of 0.95 across all metrics, reflecting the model's reliability in this category. The "glacier" class shows a decent precision of 0.69 and high recall of 0.85, resulting in a solid F1-score of 0.76. For the "mountain" class, the model exhibits high precision (0.79) but low recall (0.60), yielding an F1-score of 0.68, which points to its difficulty in identifying actual mountains. The "sea" class shows moderate performance with precision (0.73) and recall (0.69), leading to an F1-score of 0.71. The "street" class performs well, achieving an F1-score of 0.79. Overall, the model's accuracy stands at 0.77, consistent with the reported test accuracy of 0.7720, indicating balanced performance across classes. While the weighted average F1-score of 0.77 indicates consistent performance among classes, even with uneven support, the macro average F1-score of 0.77 emphasises a largely balanced performance without taking class imbalance into consideration. Although the model performs well in the "forest" class, it displays overfitting, notably in the "buildings" and "mountains" categories, where recall is poor. This might restrict its practicality in real-world situations, particularly when differentiating between comparable classes.

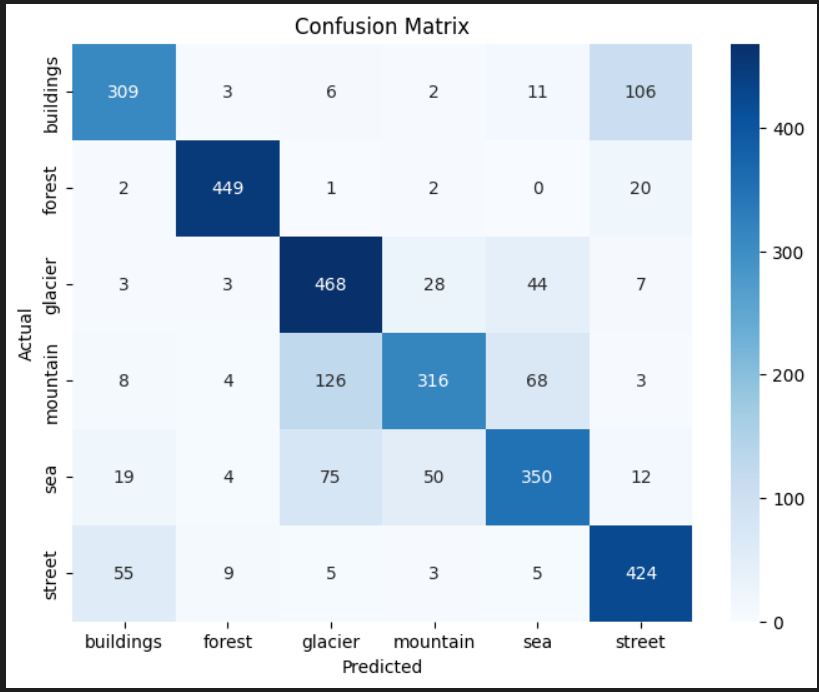


Figure 90: CNN Base Model Confusion Matrix

From the confusion matrix, observation that can be made is that the model performs relatively well in classifying the "forest" class, with a high count of correctly predicted instances (449) and only a few misclassifications. Conversely, the "buildings" class shows a significant number of misclassifications, particularly towards the "street" class, where it confused 106 instances. The "glacier" and "mountain" classes also exhibit some misclassifications, with the "mountain" class being confused with both "glacier" and "sea." Notably, the "sea" class also shows a moderate number of misclassifications. The diagonal values represent correct classifications, the great concentration in these values indicated a good performance from the model. Overall, the CNN base model shows strong performance with some classes, it struggles with some, with these insights, further model tuning and data augmentation strategies can be applied to enhance classification accuracy across all categories.

## CNN Tuned Model



Figure 91: CNN Tuned Model Accuracy

The hyperparameter-tuned CNN model achieves a training accuracy of 0.8519 and a validation accuracy of around 0.846, showing a notable increase in performance over the base model. The accuracy of the base model was 0.7720, which indicates a significant improvement. This indicates that the tuning procedure was successful in optimising the model's parameters and improving generalisation on the validation set. The model's improved capacity to reduce prediction errors is further demonstrated by the decreased validation loss of 0.5526, which implies that the modifications made during hyperparameter tuning have improved the model's overall performance in categorising the image dataset.

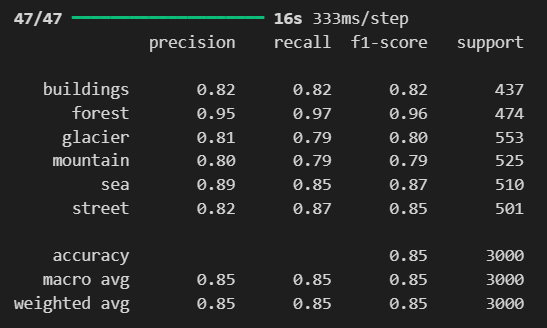


Figure 92: CNN Tuned Model Classification Report

Comparing the hyperparameter-tuned CNN model to the base model, the classification report shows a noticeable improvement in performance across all classes. Especially for the buildings class, which attained a precision of 0.82 and a recall of 0.82, indicating a more balanced performance, the accuracy, recall, and F1-scores have significantly increased. With an accuracy of 0.95 and a recall of 0.97, the forest class continues to perform quite well, showing very few misclassifications. While the mountain class continued to perform steadily at 0.80 accuracy and 0.79 recall, the glacier class also improved, with precision increasing to 0.81 and recall to 0.79. Furthermore, improvements were seen in the street and sea classes, with accuracy values of 0.82 and 0.89, respectively. A strong improvement in the model's ability to correctly identify images is proven by the overall accuracy, which rose to 0.85 and was in line with the macro and weighted averages of 0.85 for precision, recall, and F1-score. Overall, the tuned model shows notable gains in classification performance, indicating that the hyperparameter tuning modifications have improved the model's performance.

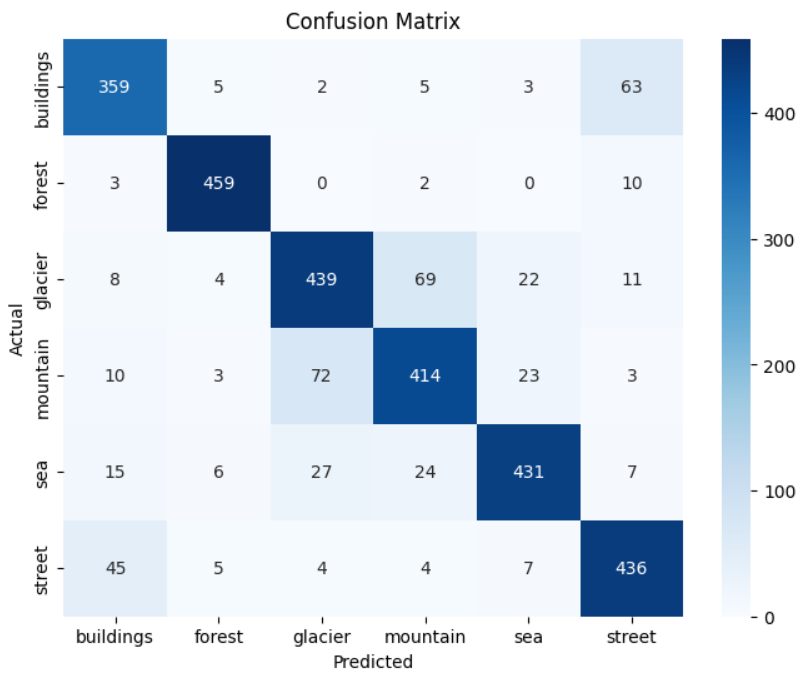


Figure 93: CNN Tuned Model Confusion Matrix

The confusion matrix for the hyperparameter-tuned CNN model reveals improved classification performance compared to the base model as well, as evidenced by the higher true positive counts across most classes. For the buildings class, the model accurately identified 359 instances, with a relatively low misclassification of only 5 to forests and 2 to glaciers, although there are notable misclassifications to streets (63 instances). The forest class shows a strong performance with 459 correctly classified samples and minimal confusion with other classes. The glacier class, however, experienced some confusion, misclassifying 69 instances as mountains and 22 as seas, indicating a challenge in distinguishing between these classes. The mountain class also displayed some misclassifications, particularly with 72 instances confused with glaciers, but overall maintained a majority of correct identifications (414). The sea class performed well, with 431 correct classifications but a few misclassifications primarily to mountains. The street class showed a solid performance with 436 correct identifications, although it did misclassify 45 instances as buildings. Overall, the confusion matrix highlights the model's improved ability to classify images accurately, with a reduction in overall misclassifications, although certain classes still face challenges that could be addressed with further tuning or additional data.

A screen shot of a computer program

Description automatically generated

Figure 94: Make Predictions using CNN Tuned Model

Now that we have the final best CNN tuned model, let’s make some prediction on new unseen data in the seg\_pred folder. This code shows that the model has predicted the dataset and the prediction shape is 7301 images and 6 classes.



Figure 95: Sample CNN Tuned Model Predictions Output

As shown in the plot above, the CNN tuned model got all 10 predictions correct based on human judgement because there are no validation for it, although the sample size is small, but the accuracy, precision and f1-score is high from the looks

# **Conclusion**

This report comprehensively explored and evaluated the performance of Convolutional Neural Networks (CNN) across multiple datasets and tasks. Through both base and tuned models, the study provided insights into the effectiveness, limitations, and applications of these architectures in image classification, with direct comparisons to prior literature.

The findings demonstrate that CNNs are the most suitable architecture for image-related tasks due to their ability to effectively extract spatial hierarchies from high-dimensional data, with an accuracy of 85% in the project implementation, closely aligning with results reported in prior literature, such as Li et al. (2024), where advanced CNN architectures like SE-RES-CNN achieved 98% accuracy on similar image datasets.

The hyperparameter tuning processes across models confirmed that careful adjustments improve model performance, with CNNs benefiting the most. Furthermore, the comparisons with literature highlight the importance of model selection based on the nature of the task. CNNs remain dominant in image classification due to their spatial feature extraction capabilities.

In summary, the report emphasizes that CNNs are the most effective model for image classification, providing significant performance improvements. The insights gathered from this project underscore the importance of leveraging the right neural architecture for a given task and support future research into advanced models, including hybrid approaches that combine the strengths of multiple architectures. This study lays a foundation for further exploration into optimizing these architectures and applying them to more complex, real-world datasets.

# **References**

Li, Q., Lei, J., Ren, C., Peng, Z., & Hong, J. (2024). Research on sports image classification

method based on SE-RES-CNN model. *Scientific Reports*, *14*(1). <https://doi.org/10.1038/s41598-024-69965-5>

Rahman, A. Y. (2021). Classification of Starling Image Using Artificial Neural Networks.

*Department of Informatics Engineering, Faculty of Engineering, Universitas Widyagama Malang, Indonesia*. <https://doi.org/10.1145/3479645.3479690>

Paoletti, M. E., Haut, J. M., Plaza, J., & Plaza, A. (2020). Scalable recurrent neural network for hyperspectral image classifcation. *The Journal of Supercomputing (2020) 76:8866–8882*, 17.

M. Thorat, S. P. (2022). Artificial Neural Network: A brief study. *Asian journal of convergence in technology*, https://doi.org/10.33130/ajct.2022v08i03.003.

Bilgiç, G. B. (2023). Recent advances in artificial neural network research for modeling hydrogen production processes. *International Journal of Hydrogen Energy*, 48(50), 18947–18977. https://doi.org/10.1016/j.ijhydene.2023.02.002.

DiPietro, R. &. (2020). Deep learning: RNNs and LSTM. *In Elsevier eBooks* , (pp. 503–519). https://doi.org/10.1016/b978-0-12-816176-0.00026-0.

Gaafar, A. S., Dahr, J. M., & Hamoud, A. K. (n.d.). Comparative Analysis of Performance of Deep Learning Classification Approach based on LSTM-RNN for Textual and Image Datasets.

Hiremani, V. A. (2021). Quantifying apt of RNN and CNN in Image Classification. *K. K.In Lecture notes in electrical engineering* , (pp. 721–733). https://doi.org/10.1007/978-981-16-0275-7\_59.

Iyer, N. S. (2020). Classification of Natural Scene using Convolution Neural Network. *International Journal of Engineering Research And*, V9(07). https://doi.org/10.17577/ijertv9is070440.

Saxena, A. (2022). An Introduction to Convolutional Neural Networks. . *International Journal for Research in Applied Science and Engineering Technology*, 10(12), 943–947. https://doi.org/10.22214/ijraset.2022.47789.

Krichen, M. (2023). Convolutional Neural Networks: A Survey. Computers. *De Computis*, 12(8), 151. https://doi.org/10.3390/computers12080151.

Alzubaidi, L. Z.-D.-S.-A. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1). https://doi.org/10.1186/s40537-021-00444-8.

Rahman, A. Y. (2021). Classification of Starling Image Using Artificial Neural Networks.

Rahman, A. Y. (2021). Classification of Starling Image Using Artificial Neural Networks. Rahman, A. Y. (2021). Classification Classification of Starling Image Using Artificial Neural Networks. . *Department of Informatics Engineering, Faculty of Engineering, Universitas Widyagama Malang, Indonesia*, https://doi.org/10.1145/3479645.3479690.

Zhou, Z., Zhang, H., & Effatparvar, M. (2024). Improved sports image classification using

deep neural network and novel tuna swarm optimization. *Scientific Reports*, *14*(1). <https://doi.org/10.1038/s41598-024-64826-7>

1. [↑](#endnote-ref-2)