Name: Justin Grima

Student number: 14248599

Subject: MA5852 - Data Science Master Class 2

Assessment 3: CNN for Classification

Due Date: Thursday, February 29th, 2024

Pages: 14.5 pages

ABSTRACT

The classification of soil is essential across various domains, such as agriculture, geology, and engineering. This fundamental task has broad associations, influencing construction, landscaping, ecosystem health, and agricultural practices. Accurate identification of soil types provides valuable insights into soil properties, supporting sustainable agricultural practices, land-use planning, and environmental monitoring. Traditional methods of soil classification involve extensive fieldwork and laboratory analysis, which can be time-consuming and resource-intensive (Kiran Pandiri et al., 2024). In recent years, advancements in computer vision and machine learning, particularly Convolutional Neural Networks (CNNs), have provided an efficient and automated method for soil type classification based on visual cues (images).

This study investigates various modified versions of a simple Convolutional Neural Networks (CNNs) by training and testing them on a dataset containing a limited set of images showcasing different types of soil. The goal is to identify the CNN architecture that exhibits the highest performance in accurately classifying soil images. The study revealed: i) after conducting thorough exploratory data analysis, necessary implementation of appropriate cleaning and transformations on the original data were needed, ii) techniques such as dropout, early stop, batch normalisation cause significant impacts and improvements on the performance on the CNN, and iii) the CNN, incorporating Dropout, Padding, a specified Learning Rate, and other designated parameters, emerged as the top-performing model for soil image classification, achieving a notable test accuracy of 60.87%.

In summary, using a small dataset of a variety of soil images as training and test data, we successfully developed, trained, and tested a variety of convolutional for classifying soil images. The CNN utilizing Dropout, Padding and Specified Learning rate, in addition to other specified parameter is recommended based on its superior performance. Future considerations may include exploring further implementing images to increase the dataset size to improve its performance, use alternative, more powerful hardware such as GPU for means of computation to successfully converge cross-validations with grid search to find the definite optimal hyperparameters and further implementing of other CNN methods and models to test their classification abilities to potentially uncover more effective CNN models for soil image classification.

Introduction

Beneath the seemingly unremarkable surface of the ground lies a dynamic and intricate world that profoundly influences our surroundings. Soil, often overlooked, plays a pivotal role in sustaining life, nurturing crops, and preserving the historical record of our planet. It serves as the silent cornerstone of ecosystems (Holtz, R. 2023). Gaining an understanding of soil surpass the academic pursuit as it is a crucial task with effects on agricultural methodologies, environmental regulations, the literal foundation for modernisation and development, and the trajectory of our planet. This research delves into the essential realm of soil, focusing on the classification of soil types using convolutional neural networks (CNNs). By harnessing advanced technology, we aim to enhance the understanding of soil variations and contribute to the broader field of environmental research. The ability to distinguish and classify soil types is essential for countless applications, ranging from agriculture and environmental monitoring to land-use planning and ecological studies.

As mentioned, traditional methods of soil classification involve field surveys and laboratory analyses, which, while accurate, are often time-consuming and resource-intensive (Kiran Pandiri et al., 2024). With that said, in recent years, the intersection of computer vision and machine learning has introduced innovative approaches to soil classification, providing a more efficient and automated alternative. Convolutional Neural Networks (CNNs), a class of deep learning models, is one of them; a widely implemented deep learning model with remarkable success in pattern recognition tasks (Mo, 2022). This project focuses on harnessing the potential of CNNs to automate soil type classification, capitalizing on a dataset comprising labelled images of six distinct soil types.

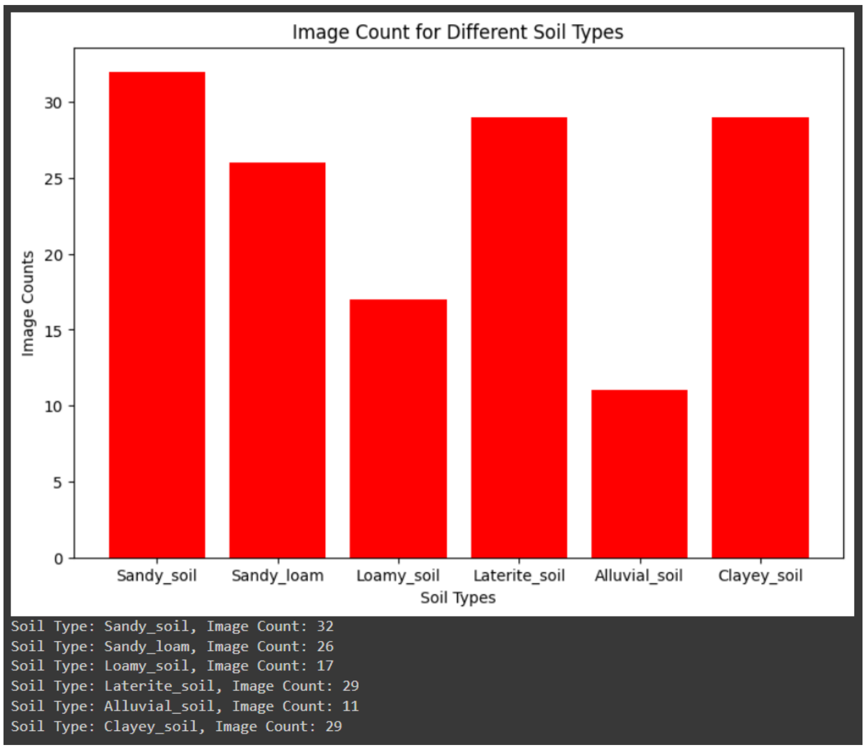
This report explores the application of CNNs in soil type classification using a dataset comprising of a small collection of images focused on different types of soil. The dataset is currently comprised of six folders, each representing a distinct soil type: 'Alluvial soil', 'Clayey soil', 'Laterite soil', 'Loamy soil', 'Sandy soil', and 'Sandy loam'. It contains a total of 144 labelled photos, highlighting a diverse set of soils in the collection. The objective is to utilise deep learning to develop a model capable of accurately identifying and classifying soils based on visual features. The study aims to contribute to soil science research, facilitate land-use planning, and offer a foundation for the development of algorithms and systems for automated soil analysis.

The significance of this study lies in its potential contributions to soil science research and practical applications. An automated soil classification model can enhance the efficiency of environmental monitoring, assist in precision agriculture, and lay the groundwork for developing systems capable of real-time soil analysis. As we delve into the methodology, results, and discussions, the aim is to establish the viability and effectiveness of CNNs in the context of soil type classification.

Data Preparation

Exploratory Data Analysis on the dataset, and implementation of cleaning or transformations

Upon the initial uploading of the Soil image dataset, we conduct the necessary Exploratory Data Analysis (EDA) to investigate the characteristics of the soil images. This EDA involved listing the contents of the soil image directory, offering a foundational understanding of the dataset the subfolders, representing each soil type, include 'Alluvial soil', 'Clayey soil', 'Laterite soil', 'Loamy soil', 'Sandy soil', and 'Sandy loam'. Furthermore, we conducted an analysis of each of the individual images to identify their key characteristics; soil types, image size and color mode, and the distribution of images in each soil type folder (See APPENDIX A for snippet of visual summaries for each soil type). What was observed is that there are 6 soil types, mentioned above, the images vary size and therefore will require the images to be resized to ensure each image has the same dimensions. All images are of RGBA colour mode. This implies that the images colour has 4 channels; Red, Green, Blue and Alpha, where Alpha refers to transparency levels that the RGB colour should have and ranges from 0 to 1 or a percentage from 0 to 100% transparency (*Defining Colors in CSS*, n.d.). Following the analysis of individual images, we observe the distribution of images for each soil type. The results in Image 1 show that 'Sandy soil' has 32 images, 'Sandy Loam' has 26, 'Loamy soil' has 15, 'Laterite soil' has 29, 'Alluvial soil' has 11, and 'Clayey soil' and 29 for a total of 144 images.



```
Soil Type: Sandy_soil, Image Count: 32
Soil Type: Sandy_loam, Image Count: 26
Soil Type: Loamy_soil, Image Count: 17
Soil Type: Laterite_soil, Image Count: 29
Soil Type: Alluvial_soil, Image Count: 11
Soil Type: Clayey_soil, Image Count: 29
```

Image 1: Distribution of images for each soil type.

To delve deeper into the folder contents, APPENDIX B showcases the initial two images from each folder. This visual exploration allows us to better grasp the visual characteristics of each soil type, assessing their similarities and differences. Such insights provide perspective on potential challenges the models may encounter in classification. For instance, closely related soil types like 'Sandy Loam' and 'Alluvial soil' exhibit resemblances, while distinct features such as soil color, texture composition, and structure set apart others like 'Laterite soil,' 'Sandy soil,' and 'Clayey soil.' Additionally, the varying image sizes within the dataset are noteworthy for contextual understanding.

After a complete analysis of the data, we can begin to implement a variety of transformations on the original data, specifically the data used for training the Convolutional Neural Networks (CNN). These transformations include resizing the image to ensure a uniform size; previously observed that the images were of different sizes. CNN architectures typically require fixed-size inputs because the weights in the convolutional layers are tied to specific filter sizes and having consistent input dimensions helps in building a well-defined and consistent architecture (Alzubaidi et al., 2021). We also identified an RGBA colour mode for all images, detailed in the discussion above. As a result, a conversion was performed to transform the RGBA colour mode to RGB because the alpha channel is not adding significant value (there is no distinguishing between soil and non-soil areas based on transparency, therefore, I made the decision to not indue it). Using RGB can further simplify the model and reduce the computational load. Additionally, we perform image normalization by scaling the image pixels down. Since pixel values are activated in the range of 0 to 255, representing the brightness of individual pixels, normalization is essential. Scaling the pixel values down to a range between 0 and 1 helps ensure that all data values fall within a consistent and manageable range. This normalization helps ease the process and analysis of the images. (NeuralNine, 2021). These preprocessing steps are essential for standardization, ensuring a consistent and well-conditioned dataset.

The dataset is composed of a total of 144 images, and as part of the train-test split, there will be a 30% reduction in the size of the training dataset. The specifics of this split will be talked about later in the report. Acknowledging the challenge of having limited number of images available for each soil type, a strategic approach is taken to increase the size and diversity of the training dataset, data augmentation. These selected augmentation strategies are specifically applied to the training data and encompass rotation, horizontal flipping, zooming, and shifting operations, including both width shift, height shift, and shear range adjustments. By introducing variations through rotation, flipping, zooming, and shifting, the augmented dataset becomes more extensive and diversified. This not only reduces the issue of limited samples for each soil type but also contributes significantly to the overall robustness of our model. Implementing data augmentation is an important step as it enriches the training dataset with a broader range of visuals and plays a crucial role in training our Convolutional Neural Network (CNN) model. For a detailed summary of the preprocessing and augmentation types, please refer to Table 1.

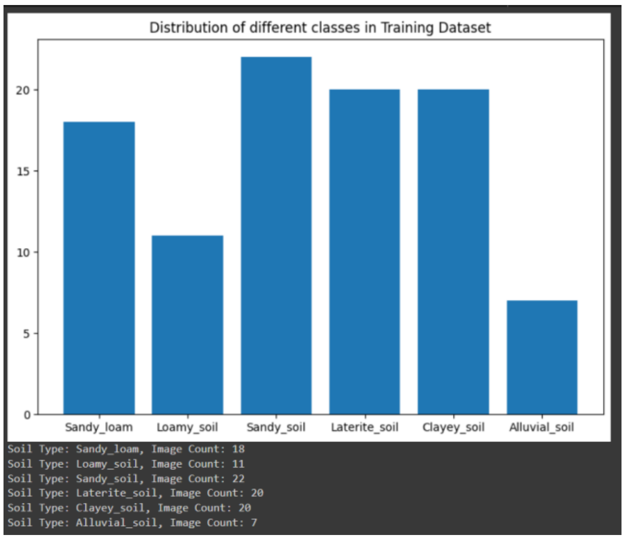
Step	Description	Purpose	Pre-processing/Augmentation
Rescaling	Pixel values are scaled from 0-255 to 0-1.	Improves training process and model convergence.	Pre-processing
Rotation	Randomly rotates images up to 40 degrees.	Helps model learn orientation-independent features.	Augmentation
Horizontal Flip	Randomly flips images horizontally.	Helps model learn features independent of left/right side.	Augmentation
Zoom	Randomly zooms images (in/out) by up to 20%.	Helps model learn features at different scales and prevents overfitting.	Augmentation
Shifts	Randomly shifts images up to 20% horizontally or vertically.	Helps model learn location-independent features.	Augmentation
Shear	Randomly sheared along x or y-axis (max angle 0.2 radians).	Introduces distortions for robustness to small perspective variations.	Augmentation
Filling	Uses nearest pixel value to fill new pixels from transformations.	Maintains consistency during augmentation.	Augmentation
RGBA to RGB conversion	Images are converted to RGB from RGBA	Remove the Alpha channel since transparency is not needed.	Pre-processing
Normalisation of image pixels	Scaling image pixel values down from 0 to 255 to a range between 0 and 1	Ensure all data values fall within a consistent and manageable range, which helps ease the process and analysis of the images.	Pre-processing

Table 1: Summary of the preprocessing and augmentation types.

Applying training and testing split

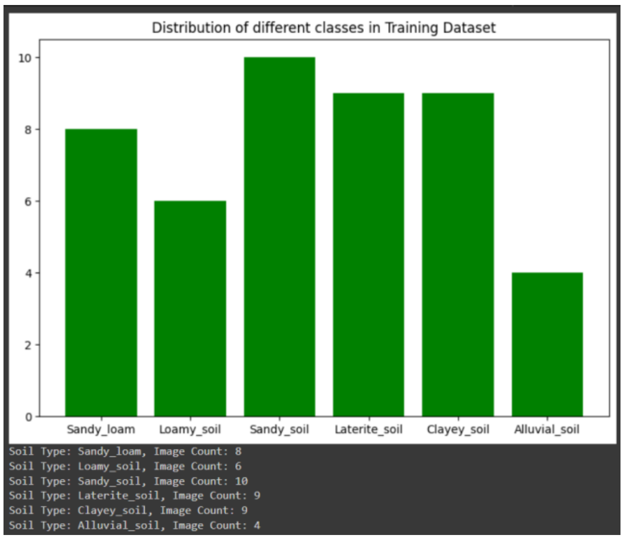
Building upon the detailed exploration and preparation in the initial phase, we now implement our strategy for splitting the dataset into training and testing using a 70-30 ratio. Best practice states that in fundamental of machine learning and data analysis, particularly during model development, it is recommended to split the dataset into training, test, and validation (Khanna, 2023). But due to the small nature of the dataset, there are not enough instances to validate this split for the model to effectively perform the task of soil image classification. The decision to split the data in a 70-30 train and test split aligns well with the dataset's relatively small size, providing a balance between training and testing data. Furthermore, to ensure data management of both the training and testing data, we organized folders for each soil type in both the training and testing datasets so that when the split occurs the images are placed in their appropriate folders, ensuring not only ease of access but also organisation, adhering to best practices for preparing the data for the convolutional neural networks. Upon completion of the split, we conduct an EDA on both the training and test folders. Specifically, we look at the distribution of soil types. In Image 2 we can see the distribution in the training dataset is consisting of 18 Sandy Loam images, 11 'Loamy soil', 22 'Sandy soil', 20 'Laterite soil', 20 'Clayey_soil', and 7 'Alluvial soil' images, resulting in a total of 96 images belonging to 6 classes. Furthermore, we have 18 Sandy Loam images, 11 'Loamy soil', 22 'Sandy soil', 20 'Laterite soil', 20 'Clayey_soil', and 7 'Alluvial soil' images, resulting in a total of 98 images. In Image 3 we can see the distribution in the training dataset is consisting of 8 Sandy Loam images, 6 'Loamy soil', 10 'Sandy soil', 9 'Laterite soil', 9 'Clayey_soil', and 4 'Alluvial soil' images, resulting in a total of 96 images belonging to 6 classes. Furthermore, we have 18 Sandy Loam images, 11 'Loamy soil', 22

‘Sandy soil’, 20 ‘Laterite soil’, 20 ‘Clayey_soil’, and 7 ‘Alluvial soil’ images, resulting in a total of 46 images belonging to 6 classes. By conduction this additional EDA we demonstrate our awareness of the importance of dataset balance, a critical factor in training models that generalize well to diverse soil types.



```
Soil Type: Sandy_loam, Image Count: 18
Soil Type: Loamy_soil, Image Count: 11
Soil Type: Sandy_soil, Image Count: 22
Soil Type: Laterite_soil, Image Count: 20
Soil Type: Clayey_soil, Image Count: 20
Soil Type: Alluvial_soil, Image Count: 7
```

Image 2: Distribution of images for each soil type in training dataset.



```
Soil Type: Sandy_loam, Image Count: 8
Soil Type: Loamy_soil, Image Count: 6
Soil Type: Sandy_soil, Image Count: 10
Soil Type: Laterite_soil, Image Count: 9
Soil Type: Clayey_soil, Image Count: 9
Soil Type: Alluvial_soil, Image Count: 4
```

Image 3: Distribution of images for each soil type in training dataset.

Build, train, and deploy a Neural Network

Propose a CNN for the classification.

A Convolutional Neural Network (CNN) is a specialized neural network designed for visual data, such as images and videos, and typically contains three layers: a convolutional layer, a pooling layer, and a fully connected layer (Mishra, 2020). The convolutional layer is the core of the CNN playing an important role in feature extraction from input data. This layer utilizes kernels (or filters), which are 2D arrays of weights, to perform convolution operations on the input data. The primary objectives are to identify essential features, such as edges or textures, and simultaneously down-sample the input size. The convolution layer output feature map is then passed through a nonlinear activation unit (e.g. Rectified Linear Unit (ReLU), Sigmoid or tanh) to convert the data into its non-linear form. The purpose of the activation function is to introduce non-linearity into the network, allowing the CNN to learn from complex patterns and capture more intricate relationships within the data.

Preceding convolution and detector operations, a pooling layer is applied to sub-sample the feature maps, which shrinks the large-size feature maps to create smaller ones. This is achieved through methods like max pooling (selecting maximum values in windows) and average pooling (computing average values in windows). This process enhances the network's robustness to minor input translations, minimizing sensitivity to object position variations for improved pattern recognition. Additionally, it reduces computation costs, conserves memory space, and contributes to the network's overall pattern recognition capabilities. After the integration of convolution and pooling layers, the final output, comprising multi-dimensional feature maps, undergoes a crucial transition with the flatten operation. This operation is vital as it converts hierarchical spatial representations into a one-dimensional array or vector, ensuring compatibility with the fully connected layers in the later phase of the CNN architecture. The flatten operation emphasizes the network's capacity to capture intricate patterns across spatial hierarchies, representing each learned feature as a distinct value in the flattened array. This transition is essential for preserving the network's understanding of complex spatial relationships and facilitating effective communication with the fully connected layers for precise classification.

Unlike convolutional layers in a CNN, where neurons are connected only to local regions in the input, fully connected layers, preceding the flattening operation, establish direct connections between every node in the output layer and all nodes in the preceding layer. This layer plays a crucial role in classifying inputs based on features extracted from previous layers and their respective filters. In the fully connected layers, we can use a SoftMax activation function to transform the raw output scores into a probability distribution over different class. This transformation generates probabilities between 0 and 1, helping the network make accurate classifications. The highest probability corresponds to the predicted class, making the SoftMax function a crucial element for the final stage of classification in a CNN.

Pertaining to the soil image classification task, a Convolutional Neural Network (CNN) architecture was chosen based on the characteristics of the dataset. Given the relatively small size of the dataset (144 images), a simpler CNN architecture was chosen as a more suitable option than larger, more complex deeper learning models like AlexNet, VGG, GoogleNet, and ResNet as a larger. Having more complex CNN models would result in a higher risk of overfitting due to the limited amount of training data, potentially leading to poor generalization performance on unseen examples (Shu, 2019). Additionally, the computational demands of training larger models might not be justified by the size of the dataset, and a simpler architecture allows for faster experimentation, tuning, and a more interpretable understanding of the model's behaviour.

In reference to our earlier discussion, at the beginning of this section, outlining the framework of a simple convolutional network, this serves as the foundational basis for crafting an efficient and successful CNN classifier tailored for soil images. The initial phase involves the implementation and testing of two variations of a simple CNN, forming the base for which the models are constructed. We enrich and diversify these base models by incorporating a variety of techniques, including dropout, padding, early stopping, and batch normalization. The objective is to find the model with the optimal performance, determined by the evaluation metric—test accuracy; measurement of the proficiency of the models in accurately classifying soil images within the test dataset. Additionally, we undertake hyperparameter tuning through cross-validation with grid search and an alternative approach of implementing the Keras Tuner package for fine-tuning key parameters such as the number of feature maps, kernel size, number of units (neurons), padding type, pooling window size, dropout rate, and learning rate, among others. A more detailed discussion of these events will be provided later in the report.

To start we begin with two variations of a simple CNN:

Input Layer:

- ➢ Both simple CNNs Input layers dimensions, for images, were resized to (256, 256) pixels.
- ➢ As discussed earlier in the report, the images will be converted from RGBA to RGB.

Convolutional Layers:

- ➢ Simple CNN 1 will have 2 convolutional layers, the default stride; the step size used when moving kernel across the input data, of 1, and default padding of 'valid'; no padding/ outer layer is added to the input (image).
- ➢ Simple CNN 2 will have 3 convolutional layers with the same parameter settings describe in CNN1. By adding an additional layer, the network to learn complex patterns which in turn should increase the model's performance.

Activation Functions:

- ➢ The Rectified Linear Unit (ReLU) activation functions is after each convolutional layer and fully connected layer to introduce non-linearity; a piecewise linear function that outputs the input directly if positive and zero otherwise (Brownlee, 2019).

Pooling Layers:

- ➢ Simple CNN 1 will have 2 max pooling layers, with a pooling window of 2x2; combining the spatial information by selecting the maximum value within each window, down sampling the input and retaining essential features. The default stride of 1 is also implemented.
- ➢ Simple CNN 2 will have 3 max pooling layers with the same parameter settings describe in CNN1. Fully Connected Layers:

Flatten layer:

- ➢ Both Simple CNNs will have a flatten layer that flattens the output of the convolutional layers into a 1D vector to feed into the fully connected layers.

Fully Connected layer:

- Simple CNN 1 will have 1 fully connected layer.
- Simple CNN 2 will have 2 fully connected layers.
- Again, both use ReLU activation functions to introduce non-linearity.

Output Layer:

- Both Simple CNNs will have an output layer with 6 nodes, each class representing a soil type that is to be classified.
- Both Simple CNNs use Softmax activation function; converts a vector of numbers into a probability distribution, with each probability proportional to the relative scale of the corresponding value in the vector (Brownlee, 2020), is used in the output layer for multi-class classification.

Refer to APPENDIX C; CNN 1, and APPENDIX D; CNN 2, for model summaries. Additionally, the Adaptive Moment Estimation (Adam) optimizer - minimizes the loss function during the training of neural networks (Vishwakarma, 2023), is employed, which is well-suited for optimizing neural networks due to its adaptive learning rates and helps the model converge more efficiently and handle variations in the learning rates across different parameters. For the loss function, the categorical crossentropy is implemented since it is a multi-class classification task. This choice aligns with the softmax activation function in the output layer and is suitable for scenarios where each input image belongs to only one class. In terms of evaluation metrics, accuracy is chosen as a primary metric. This metric provides a straightforward measure of the model's overall correctness in classifying images.

The foundational choices for the CNN, outlined above, will be refined through the application of various techniques, and hyperparameter tuning will be conducted on these models. Ultimately, the final selected model, chosen based on its superior performance, will embody the characteristics described in the proposed architecture. The next sections will provide a detailed breakdown of the proposed model's evolution and performance improvements. The results of the multiple CNN created using the Simple CNN 1 and 2 as their foundation, will be further discussed in the upcoming sections of the report.

Applied techniques and hyperparameter tuning for a variety of CNNs

As mentioned, these standalone CNN models are susceptible to challenges such as overfitting, where the model memorizes the training data and performs poorly on new data, and convergence issues that hinder efficient training. Additionally, issues like information loss at edges, sensitivity to input variations, and limited model expressiveness can impact the model's ability to capture complex patterns effectively. To boost the performance of these models, we recognize the importance of incorporating key techniques, including regularization methods and normalization techniques.

To overcome these challenges, we employ techniques such as Batch Normalization, which normalizes the activations of each layer in a neural network, promoting stable and efficient training by mitigating issues such as internal covariate shift (Doshi, 2021). We also utilize dropout, randomly deactivating input units during training updates, temporarily disregarding specific neurons and their connections based on a defined probability (Martins, 2023). The use of padding involves adding extra layers of zeros or values around the input matrix ('same') to preserve its spatial size, ensuring that the output dimensions, post-filter application, stay the same or are adjusted as needed (Padding (Machine Learning), 2019). Furthermore, we incorporate techniques like early stoppage, which ends training when parameter updates no longer produce improvements on a validation set (Papers with Code - Early Stopping Explained, n.d.), and we control the learning rate, which dictates how much adjustments in the weights of the network will occur with respect to the loss gradient (Hafidz Zulkifli, 2018).

Additionally, hyperparameter tuning is conducted to fine-tune critical parameters such as the number of feature maps, kernel size, number of units (neurons), padding type, pooling size window, dropout rate, and learning rate, among others. These collective strategies are implemented separately and in combination to enhance the CNN's robustness, accelerate convergence speed, prevent overfitting, and strengthen its capacity to generalize effectively to new and diverse data and ultimately determine which variation in the simple CNNs will produce the best performing CNN to classify soil images.

In this next section, the discussion focuses on the presentation of various CNNs implemented using a range of techniques, including the regularization methods and normalization techniques previously discussed. Additionally, cross-validation with grid search and the utilization of the Keras Tuner for hyperparameter tuning are vital parts of this exploration. The discussion unfolds by detailing the process of implementing these techniques during the coding phase, involving both random and systematic utilization. Hyperparameter tuning is applied at different stages in the creation of each presented CNN. See Table 2 for summary of techniques, including the regularization methods and normalization techniques, and hyperparameter tuning methods.

Technique	Type	Description/Effect on CNN
Batch Normalization	Normalization	Normalizes activations of each layer, stabilizing and enhancing training by reducing internal covariate shift.
Dropout	Regularization	Randomly deactivates input units during training, preventing reliance on specific neurons and enhancing model generalization.
Padding ('same')	Regularization	Adds extra layers of zeros/values around the input matrix, preserving spatial size and preventing information loss at edges.
Padding ('valid')	Regularization	No padding is added, and the convolution operation is applied directly to the input data, potentially leading to a reduction in spatial dimensions.
Early Stoppage	Regularization	Ends training when parameter updates no longer improve validation set performance, preventing overfitting.
Learning Rate Control	Regularization	Controls how much adjustments in the weights of the network will occur with respect to the loss gradient.
Hyperparameter Tuning	Hyperparameter Tuning	Fine-tunes critical parameters like the number of feature maps, kernel size, dropout rate, etc., optimizing model performance.

Table 2: Summary of techniques to implement on CNN.

The following CNNs are showcased in the sequence of their development during the coding process. The previous discussion explaining the suggested CNNs and outlining the structure of Simple CNN 1 and 2—including the input layer image resizing, RGBA to RGB transformation, number of convolutions, pooling, flatten, dense, output layers, activation functions, optimizer, loss function, and accuracy—will not be reiterated in the following discussion. It has been established that these features remain constant. Furthermore, in the upcoming sections of the report, the term 'validation' refers to the test data. An attempt to change the name resulted in a code error. Therefore, the output shows 'validation' which is 'test'. Please refer to the mentioned. A detailed discussion regarding the performance of each created CNN will be provided later in the report.

Model 1: Simple CNN 1 with Batch Normalisation, Early Stoppage and Hyperparameter Tuning; K-fold cross validation with grid search.

- Implementation of Batch Normalisation and Early stoppage.
- Early stoppage parameters are set to monitor 'validation (test) loss', patience is set to 5, and the 'restore best weights' is set to true. The early stoppage parameters also remain the same throughout the models and therefore future mention of early stoppage infers these parameter settings.
- Hyperparameter tuning is implemented using cross-validation with grid search.
- Grid parameters include; feature maps for convolution layer 1 = 32 and 64, kernel 1 sizes =(3, 3) and (5, 5), feature maps for convolution layer 2 = 64 and 128, kernel 2 sizes = (3, 3) and (5, 5), dense layer 1 units = 64 and 128.
- Parameters for K-fold cross validation include; number of splits = 2, shuffle = True and random state = 42.
- Number of classes for the output layer is set to 6, which does not change in any of the created CNN and therefore can be assumed in the remaining presented CNNs.
- Epoch is set to 30 for this and all created CNNs and therefore can be assumed in the remaining presented CNNs.

Model 2: Simple CNN 2 with Batch Normalisation, Early Stoppage and Hyperparameter Tuning; K-fold cross validation with grid search.

- Implementation of Batch Normalisation and Early stoppage.
- Hyperparameter tuning is implemented using cross-validation with grid search.
- Grid parameters include; feature maps for convolution layer 1 = 32 and 64, kernel 1 sizes =(3, 3) and (5, 5), feature maps for convolution layer 2 = 64 and 128, kernel 2 sizes = (3, 3) and (5, 5), feature maps for convolution layer 3 = 64 and 128, kernel 3 sizes = (3, 3) and (5, 5), dense layer 1 units = 128 and 256, dense layer 2 units = 64 and 128.
- Parameters for K-fold cross validation include number of splits = 2, shuffle = True and random state = 42.

Model 3: Simple CNN 1 with Batch Normalisation and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX E for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Batch Normalisation.

Model 4: CNN 1 with Dropout, Early Stoppage and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX F for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Drop out and Early Stoppage.
- Dropout parameter set 0.5.

Model 5: CNN 1 with Dropout, Padding and Early Stoppage and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX G for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Dropout, Padding and Early Stoppage.
- Dropout parameter set 0.5 and Padding parameter set to 'same' adding zeros around the perimeter to maintain the spatial dimensions of the input and output.

Model 6: CNN 1 with Dropout, Padding and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX H for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Dropout and Padding.
- Dropout parameter set 0.5 and Padding parameter set to 'same' adding zeros around the perimeter to maintain the spatial dimensions of the input and output.

Model 7: CNN 1 with Dropout, Batch Normalisation, Early Stoppage and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX I for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Dropout, Batch Normalisation and Early stoppage.
- Dropout parameter set 0.5.

Model 8: CNN 1 with Dropout, Batch Normalisation, and optimal hyperparameters from cross validation with grid search (Refer to APPENDIX J for model summary).

- Implementation of optimal parameters from hyperparameter tuning: convolution layer 1 = 64, kernel 1 sizes = (3, 3), feature maps for convolution layer 2 = 64, kernel 2 sizes = (3, 3), dense layer 1 units = 64.
- Implementation of Dropout and Batch Normalisation.
- Dropout parameter set 0.5.

Model 9: Hyper parameter tuning using Keras Tuner for Model 4 (best performing model) with Dropout, Padding, Learning Rate and Early Stoppage (Refer to APPENDIX K for model summary and best hyperparameter results).

- Implementing Kera's Tuner to perform hyperparameter tuning on the best performing model; model 4 to try and improve further.
- Implementation of Dropout, Padding, Learning Rate and Early Stoppage.
- Parameters tested include feature maps for convolution layer 1 and 2, and dense layer 1 = minimum value = 32, maximum value = 128, step = 32, dropouts = minimum value = 0.3, maximum value = 0.7, step = 0.1, and learning rate = minimum value = 0.001, maximum value = 0.1, sampling = 'log'.

Model 10: Hyper parameter tuning using Keras Tuner for Model 4 (best performing model) with Dropout, Padding, and Learning Rate (Refer to APPENDIX L for model summary and best hyperparameter results).

- Implementing Kera's Tuner to perform hyperparameter tuning on the best performing model; model 4 to try and improve further.
- Implementation of Dropout, Padding, and Learning Rate.
- Parameters tested include feature maps for convolution layer 1 and 2, and dense layer 1 = minimum value = 32, maximum value = 128, step = 32, dropouts = minimum value = 0.3, maximum value = 0.7, step = 0.1, and learning rate = minimum value = 0.001, maximum value = 0.1, sampling = 'log'.

During the process of hyperparameter tuning, utilizing K-fold cross-validation and grid search for models 1 and 2, certain limitations were encountered, particularly in relation to the available RAM in Google Colab, the platform where the code was executed (See APPENDIX M for RAM error). The predefined RAM usage limits led to challenges for Model 1, which could only complete 19 iterations through the grid combinations before facing a full RAM usage issue, necessitating a disconnection from Google Colab and the followed by a rerun of all cells. Model 2 encountered a similar obstacle, reaching 104 iterations before hitting the RAM usage threshold. The results obtained from the 19 and 104 iterations of the hyperparameter tuning process , showcasing the top 10 models for both Model 1 and Model 2, are presented in Table 3. The parameters from the performing model, using the evaluation metric test accuracy, were then implemented, and applied to Models 3 to 8, as discussed above.

2D conv 1	Kernel 1	2D conv 2	Kernel 2	2D conv 3	Kernel 3	Max Pooling	Dense layer 1 neurons	Dense layer 2 neurons	Output Number of classes	Test and Train (if test are the same) Accuracy (%)
64	(3, 3)	64	(3, 3)	NA	NA	(2,2)	64	NA	6	34.78
64	(3, 3)	64	(5, 5)	64	(3, 3)	(2,2)	128	64	6	32.61
64	(5, 5)	64	(5, 5)	NA	NA	(2,2)	128	NA	6	28.26 / 87.76
64	(5, 5)	64	(3, 3)	NA	NA	(2,2)	64	NA	6	28.26 /83.67
32	(3, 3)	128	(5, 5)	NA	NA	(2,2)	64	NA	6	26.09/ 86.73
32	(3, 3)	64	(3, 3)	64	(5, 5)	(2,2)	128	64	6	26.07/ 78.57
64	(5, 5)	64	(3, 3)	128	(5, 5)	(2,2)	128	64	6	26.07/ 76.53
64	(3, 3)	64	(3, 3)	NA	NA	(2,2)	128	NA	6	21.74/ 85.71
32	(3, 3)	128	(3, 3)	NA	NA	(2,2)	128	NA	6	21.74/ 81.63

Table 3: Top model results for Cross- validation with grid search hyperparameter optimisation tuning results for Model 1 and 2.

Performance of the proposed Convolution Neural Network models

Having completed a variety of models, we can evaluate their performances using the test accuracy; the ability for the models to effectively classify unseen images to the best of their ability based on their model's technique implementations and parameters used and its training on the training dataset, based on a percentage. Table 4 provides a full breakdown of the models performances based on their test accuracy which are presented in the following order from most successful to least; **Model 10:** 60.87 % test accuracy, **Model 9:** 52.17 % test accuracy, **Model 4:** 50.00 % test accuracy, **Model 5:** 47.83 % test accuracy, **Model 2**: 34.78 % , **Model 1:** 32.61% test accuracy, **Model 6:** 28.26 % test accuracy, **Model 7:** 19.57 % test accuracy, **Model 3**: 17.39 % test accuracy, test accuracy, and **Model 8:** 8.70 % test accuracy. Overall, based on the evaluation metric, 'test accuracy', **Model 10**; Convolutional layer 1 and 2 = 32 and 128, Activation Function = ReLU and Softmax (Output layer), Kernel size 1 and 2 = 3x3, Dropout rate 1, 2 and 3 = 0.3, 0.4 and 0.5, Max pooling window 1 and 2 = 2x2, Dense units = 128, Optimizer = Adma ,Learning Rate = 0.001 and Loss Function = categorical crossentropy, is the chosen model, out of all implemented and tested, for best classifying soil images based on the having the highest test accuracy percentage score of 60.87 %.

Models	Test Accuracy	Parameters
10	60.87%	Feature Map 1: 32, Dropout Rate 1: 0.3, Feature Map 2: 128, Dropout Rate 2: 0.4, Dense Units: 128, Learning Rate: 0.001, Activation: ReLU, Dropout 3: 0.5, Activation: Softmax.
9	52.17%	Feature Map 1: 32, Dropout Rate 1: 0.3, Feature Map 2: 128, Dropout Rate 2: 0.3, Dense Units: 128, Learning Rate: 0.003, Early Stoppage, Activation: ReLU, Dropout 3: 0.5, Activation: Softmax.
4	50.00%	Filter 1: 64, Activation: ReLU, Input shape: (256,256,3), Kernel Size 1: (3, 3), Dropout 1: 0.5, Max Pooling: (2,2) Filter 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Dropout 3: 0.5, Activation: Softmax.
5	47.83%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Padding: same, Dropout 1: 0.5, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Padding: same, Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Dropout 3: 0.5, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy, Early Stoppage.
1	34.78%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Batch Normalisation, Dropout 1: 0.5, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Dropout 3: 0.5, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy, Early Stoppage.
2	32.61%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Batch Normalisation, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Feature Map 3: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Dense Neurons 1: 128, Dense Neurons 2: 64, Activation: ReLU, Batch Normalisation, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy, Early Stoppage.
6	28.26%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Padding: same, Dropout 1: 0.5, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Padding: same, Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Dropout 3: 0.5, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy.
7	19.57%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Batch Normalisation, Dropout 1: 0.5, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Dropout 3: 0.5, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy, Early Stoppage.
3	17.39%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Batch Normalisation, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy.
8	8.70%	Feature Map 1: 64, Activation: ReLU, Kernel Size 1: (3, 3), Input shape: (256,256,3), Batch Normalisation, Dropout 1: 0.5, Max Pooling: (2,2) Feature Map 2: 64, Activation: ReLU, Kernel Size 2: (3, 3), Batch Normalisation, Dropout 2: 0.5, Dense Neurons: 64, Activation: ReLU, Batch Normalisation, Dropout 3: 0.5, Activation: Softmax, Optimizer = Adam, Loss: Categorical Crossentoropy, Metric: Accuracy.

Visualizations in Images 4 to 7 offer insights into various crucial aspects related to Model 10s training and evaluation. Image 4 presents a timeline of training and test accuracy throughout the model iteration with training and test data. The graph indicates higher accuracy in the training data compared to testing data, showing consistent fluctuations, and concluding with a training accuracy of 72.45% and a test accuracy of 60.87%. Moving on to Image 5, the loss values, representing the difference between predicted and true classes, are depicted. Test loss exhibits higher values than training loss, fluctuating as the data iterates through the model. After 30 epochs, the training loss stands at 0.5983, while the test loss reaches 1.154. Image 6 showcases a confusion matrix for Model 10, providing a visual representation of the CNN model's performance in classifying different soil types. Each box shows the number of samples: diagonally, the model correctly classified that soil type (e.g., 9 for "Clayey_soil"), while off-diagonally, it incorrectly classified that type as another (e.g., 2 "Alluvial_soil" predicted as "Clayey_soil"). Despite an overall accuracy of 60.87%, the model performed better for some types ("Clayey_soil") compared to others ("Alluvial_soil", "Laterite_soil"), suggesting potential for improvement in distinguishing specific soil types. Moreover, an additional perspective on the model's classification capabilities is presented in Image 7, illustrating the model's classifications compared to the true classes of the images. Considering the test accuracy score, Model 10 has demonstrated the most favourable performance and is therfore selected as the preferred model for soil image classification.

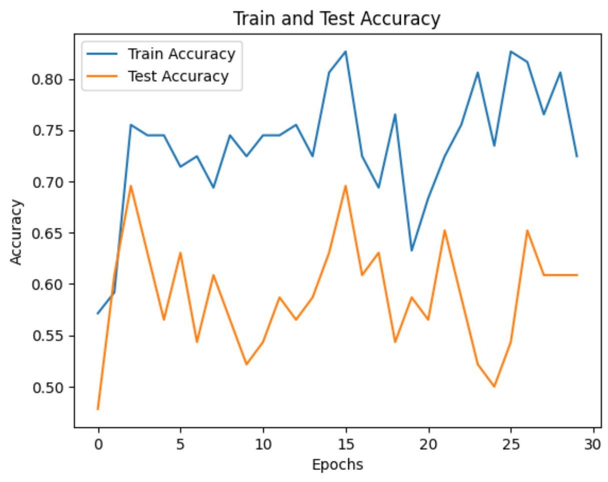


Image 4: Training and Test accuracy timeline for Model 10.

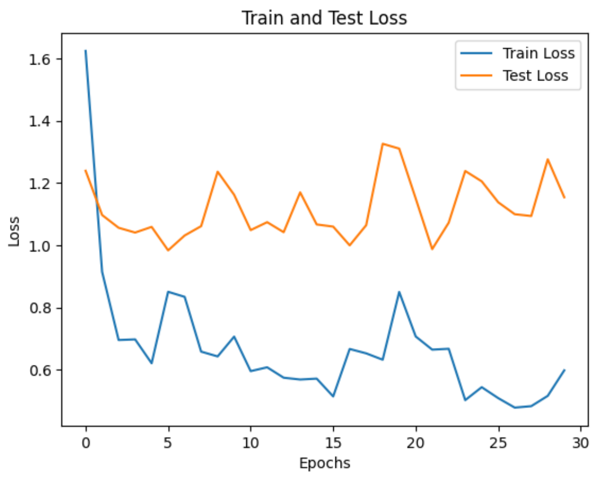


Image 5: Training and Test loss timeline for Model 10

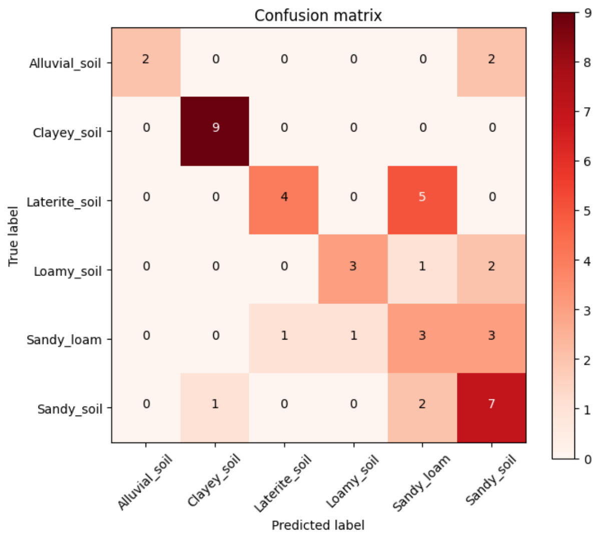
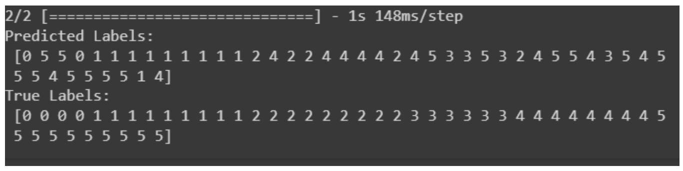


Image 6: Confusion matrix for Model 10 classification performance.



```
2/2 [==============================] - 1s 148ms/step
Predicted Labels:
 [0 5 5 0 1 1 1 1 1 1 1 1 1 2 4 2 2 4 4 4 4 2 4 5 3 3 5 3 2 4 5 5 4 3 5 4 5
 5 5 4 5 5 5 5 1 4]
True Labels:
 [0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4 5
 5 5 5 5 5 5 5 5 5]
```

Image 7: Predicted vs True classification performance for Model 10

Limitations of the proposed Convolution Neural Network models

The primary limitation of the models stems from the use of a relatively small and imbalanced dataset for constructing the CNN. With only 144 images, the model may not capture the full diversity and complexity of the underlying patterns in the soil images, can potentially lead to overfitting, and the imbalance in the number of images for each soil type can lead to biased model predictions. To overcome these limitations, data augmentation techniques were applied, to introducing variations in the dataset through operations like rotation, flipping, and zooming. But further obtaining additional soil images to build the dataset even more would provide significant improvements to the model. Additionally, addressing the class imbalance by oversampling the minority classes or applying techniques such as (Synthetic Minority Over-sampling Technique) SMOTE can improve the model's ability to generalize across different soil types. Another significant limitation is imposed by the computational constraints of Google Colab, particularly the restricted RAM usage. This restriction became evident during hyperparameter tuning, where some models could not complete the iterations due to memory limitations. A potential solution involves changing the computational tasks to platforms or hardware with more extensive computing resources or utilizing cloud services that offer higher RAM capacity. Alternatively, optimizing the model architecture to minimize memory requirements or implementing model pruning techniques may help with the constraints of the chosen platform.

Furthermore, the limited exploration of CNN architectures presents a constraint. Only 10 CNN models were tested, building on a simple architecture. To overcome this limitation, a more extensive exploration of architectural variations can be undertaken. This includes experimenting with different layer depths, kernel sizes, and network structures to identify configurations that better capture complex patterns in the soil images. Advanced techniques such as transfer learning, where pre-trained models are leveraged as a starting point, or automated architecture search methods can be employed to discover more effective and efficient models. Lastly, the choice of TensorFlow and Keras as the primary packages for building CNNs introduces a limitation. While these frameworks are widely used and well-supported, exploring alternative deep learning libraries such as PyTorch or MXNet may reveal additional optimizations or efficiencies. Different libraries may have distinct advantages in terms of ease of use, flexibility, or specialized functionalities. Exploring these alternatives can lead to insights and improvements in the performance of the CNNs for soil image classification.

References

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). Accessed on February 26th, 2024, from, https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8

author. (2023, February 28). *Soil classification and why it matters*. NSW Environment and Heritage. Accessed on February 26th, 2024, from, https://www.environment.nsw.gov.au/news/soil-classification-and-why-it-matters

Brownlee, J. (2019, April 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU) for Deep Learning Neural Networks*. Machine Learning Mastery. Accessed on February 25th, 2024, from, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, J. (2020, October 18). *Softmax Activation Function with Python*. Machine Learning Mastery. Accessed on February 25th, 2024, from https://machinelearningmastery.com/softmax-activation-function-with-python/

Defining Colors in CSS. (n.d.). Web.simmons.edu. Accessed on February 26th, 2024, from, http://web.simmons.edu/~grovesd/comm244/notes/week3/css-colors

Doshi, K. (2021, May 29). *Batch Norm Explained Visually — How it works, and why neural networks need it*. Medium. Accessed on February 27th, 2024, from, https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Hafidz Zulkifli. (2018, January 21). *Understanding Learning Rates and How It Improves Performance in Deep Learning*. Medium; Towards Data Science. Accessed on February 27th, 2024, from, https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

Holtz, R. D., Kovacs, W. D., & Sheahan, T. C. (2023). *An Introduction to Geotechnical Engineering (3rd ed.).* Pearson. Accessed on February 26th, 2024, from https://www.researchgate.net/publication/376260208_EARTH'S_PALETTE_A_COMPREHENSIVE_GUIDE_TO_SOIL_CLASSIFICATION

Khanna, S. (2023, November 16). *A Comprehensive Guide to Train-Test-Validation Split in 2023*. Analytics Vidhya. Accessed on February 28th, 2024, from, https://www.analyticsvidhya.com/blog/2023/11/train-test-validation-split/

Kiran Pandiri, D. N., Murugan, R., & Goel, T. (2024). *Smart soil image classification system using lightweight convolutional neural network.* Expert Systems with Applications, 238, 122185. Accessed on February 27th, 2024, from, https://doi.org/10.1016/j.eswa.2023.122185

Martins, C. (2023, September 20). *Dropout Layer Explained in the Context of CNN*. Medium. Accessed on February 26th, 2024, from, https://cdanielaam.medium.com/dropout-layer-explained-in-the-context-of-cnn-7401114c2c#:~:text=The%20Dropout%20Layer%20is%20a

Matshidiso. (May, 2023). *Soil types*. Www.kaggle.com. Accessed on February 24th, 2024, from,

https://www.kaggle.com/datasets/matshidiso/soil-types

Mishra, M. (2020, September 2). *Convolutional Neural Networks, Explained*. Medium. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20Convolutional%20Neural%20Network%2C%20also

NeuralNine. (2021) *Image Classification with Neural Networks in Python*. Www.youtube.com. Accessed on February 24th, 2024, from, https://www.youtube.com/watch?v=t0EzVCvQjGE

Mo, X. (2022). *Convolutional Neural Network in Pattern Recognition.* Kuscholarworks.ku.edu. Accessed on February 27th, 2024, from, https://kuscholarworks.ku.edu/handle/1808/34204

Padding (Machine Learning). (2019, May 17). DeepAI. Accessed on February 25th, 2024, from,

https://deepai.org/machine-learning-glossary-and-terms/padding

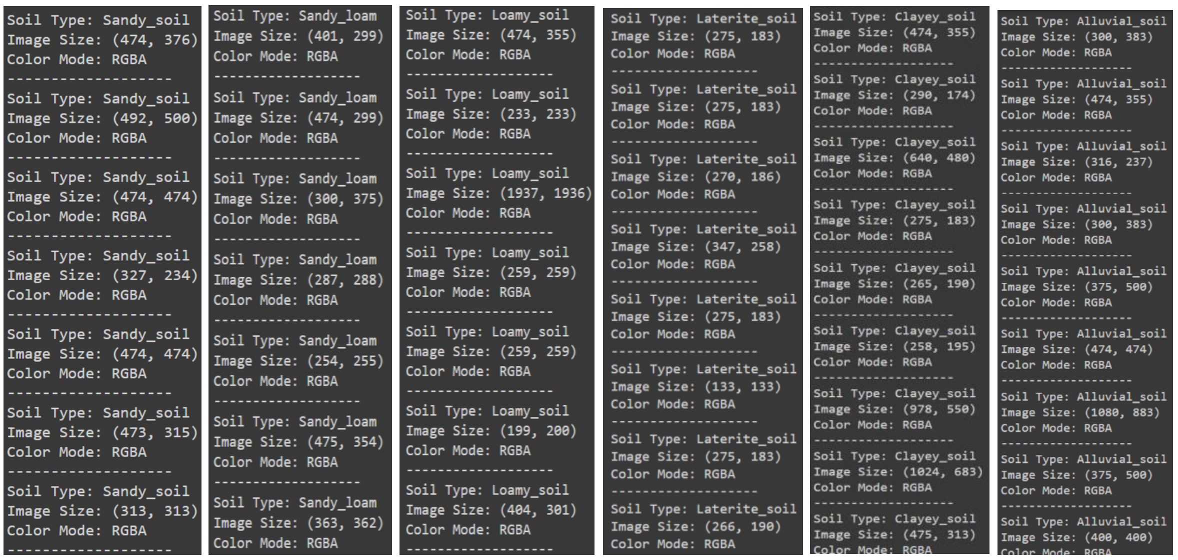
Papers with Code - Early Stopping Explained. (n.d.). Paperswithcode.com. Accessed on February 27th, 2024, from, https://paperswithcode.com/method/early-stopping#:~:text=Early%20Stopping%20is%20a%20regularization

Shu, Mengying (2019). *Deep Learning for Image Classification on Very Small Datasets Using Transfer Learning*. Iowa State University. Accessed on February 27th, 2024, from, https://dr.lib.iastate.edu/entities/publication/a0584fb3-c466-44f2-b4a8-1f7bf40d1805

Vishwakarma, N. (2023, September 29). *What is Adam Optimizer?* Analytics Vidhya. Accessed on February 27th, 2024, from, https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/#:~:text=The%20Adam%20optimizer%2C%20short%20for

Appendix

APPENDIX A: Snippet of visual summaries for each soil type.



```
Soil Type: Sandy_soil
Image Size: (474, 376)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (492, 500)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (474, 474)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (327, 234)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (474, 474)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (473, 315)
Color Mode: RGBA
-------------------
Soil Type: Sandy_soil
Image Size: (313, 313)
Color Mode: RGBA
-------------------
```

```
Soil Type: Sandy_loam
Image Size: (401, 299)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (474, 299)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (300, 375)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (287, 288)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (254, 255)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (475, 354)
Color Mode: RGBA
-------------------
Soil Type: Sandy_loam
Image Size: (363, 362)
Color Mode: RGBA
```

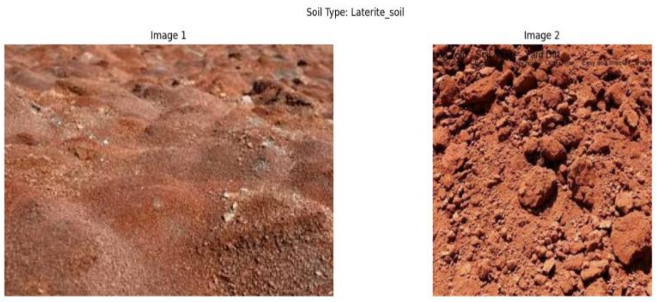
```
Soil Type: Loamy_soil
Image Size: (474, 355)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (233, 233)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (1937, 1936)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (259, 259)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (259, 259)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (199, 200)
Color Mode: RGBA
-------------------
Soil Type: Loamy_soil
Image Size: (404, 301)
Color Mode: RGBA
-------------------
```

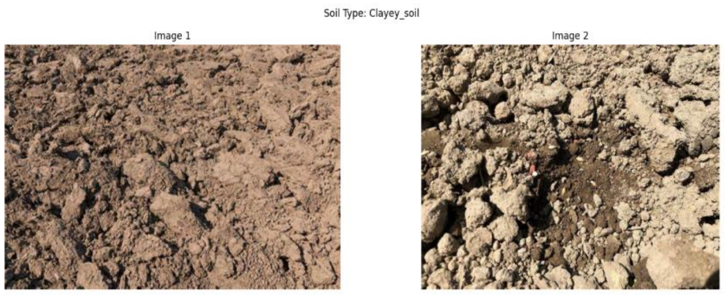
```
Soil Type: Laterite_soil
Image Size: (275, 183)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (275, 183)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (270, 186)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (347, 258)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (275, 183)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (133, 133)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (275, 183)
Color Mode: RGBA
-------------------
Soil Type: Laterite_soil
Image Size: (266, 190)
Color Mode: RGBA
```

```
Soil Type: Clayey_soil
Image Size: (474, 355)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (290, 174)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (640, 480)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (275, 183)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (265, 190)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (258, 195)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (978, 550)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (1024, 683)
Color Mode: RGBA
-------------------
Soil Type: Clayey_soil
Image Size: (475, 313)
Color Mode: RGBA
```

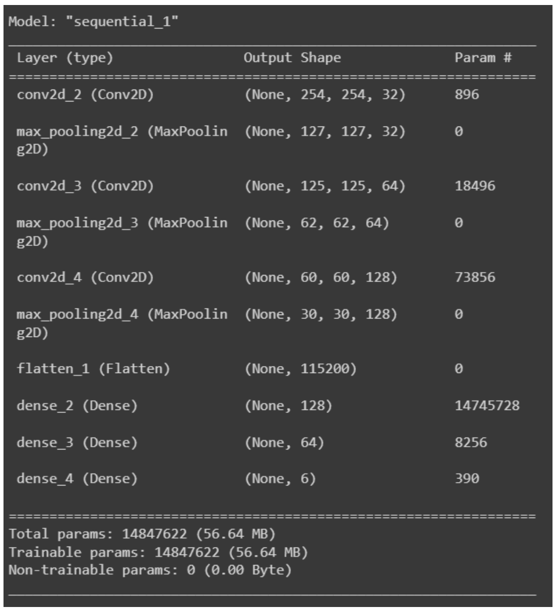
```
Soil Type: Alluvial_soil
Image Size: (300, 383)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (474, 355)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (316, 237)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (300, 383)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (375, 500)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (474, 474)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (1080, 883)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (375, 500)
Color Mode: RGBA
-------------------
Soil Type: Alluvial_soil
Image Size: (400, 400)
Color Mode: RGBA
```

APPENDIX B: Preview of soil images.



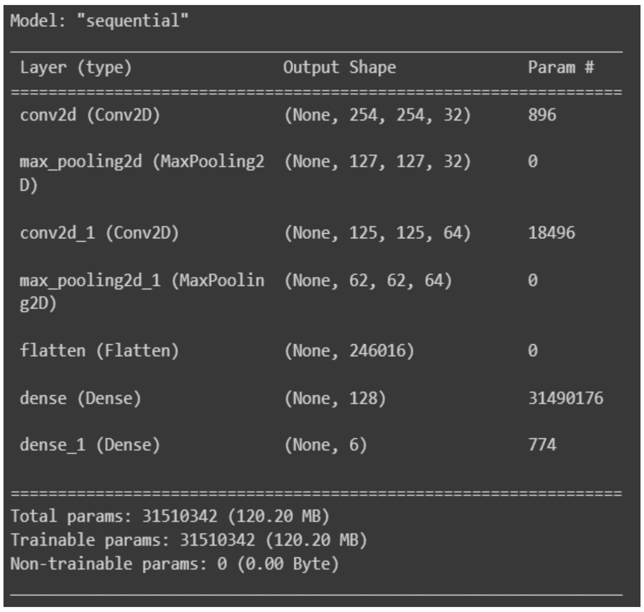


APPENDIX C: Simple CNN 1 model summaries.



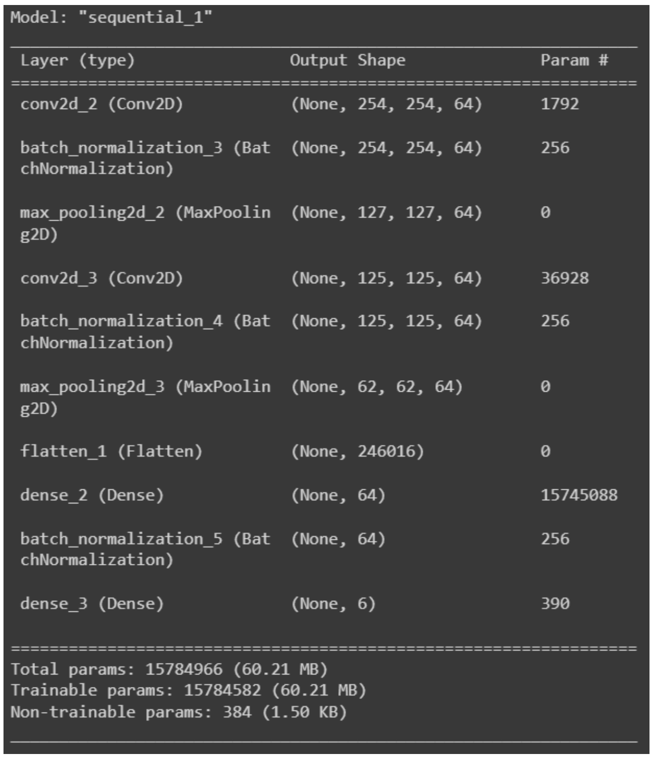
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_2 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_3 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_3 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_4 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 115200)            0

 dense_2 (Dense)             (None, 128)               14745728

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 6)                 390

=================================================================
Total params: 14847622 (56.64 MB)
Trainable params: 14847622 (56.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX D: Simple CNN 2 model summaries.



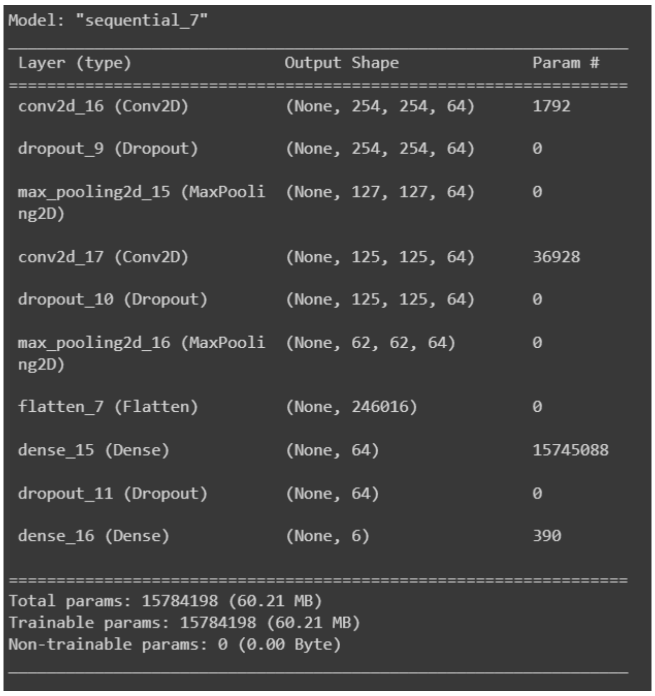
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 flatten (Flatten)           (None, 246016)            0

 dense (Dense)               (None, 128)               31490176

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 31510342 (120.20 MB)
Trainable params: 31510342 (120.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX E: Model 3 - Simple CNN 1 with Batch Normalisation and optimal hyperparameters from cross validation with grid search.



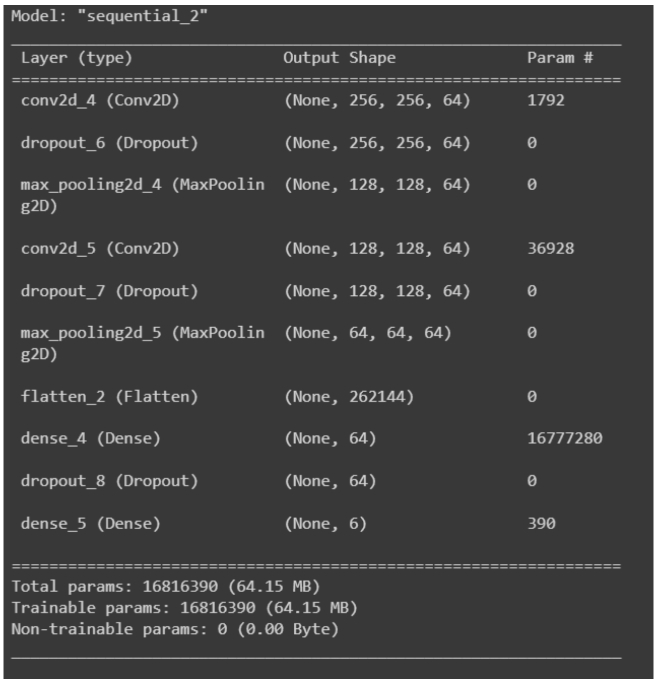
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 254, 254, 64)      1792

 batch_normalization_3 (Bat  (None, 254, 254, 64)      256
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 127, 127, 64)      0
 g2D)

 conv2d_3 (Conv2D)           (None, 125, 125, 64)      36928

 batch_normalization_4 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 flatten_1 (Flatten)         (None, 246016)            0

 dense_2 (Dense)             (None, 64)                15745088

 batch_normalization_5 (Bat  (None, 64)                256
 chNormalization)

 dense_3 (Dense)             (None, 6)                 390

=================================================================
Total params: 15784966 (60.21 MB)
Trainable params: 15784582 (60.21 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

APPENDIX F: Model 4 - CNN 1 with Dropout, Early Stoppage and optimal hyperparameters from cross validation with grid search.



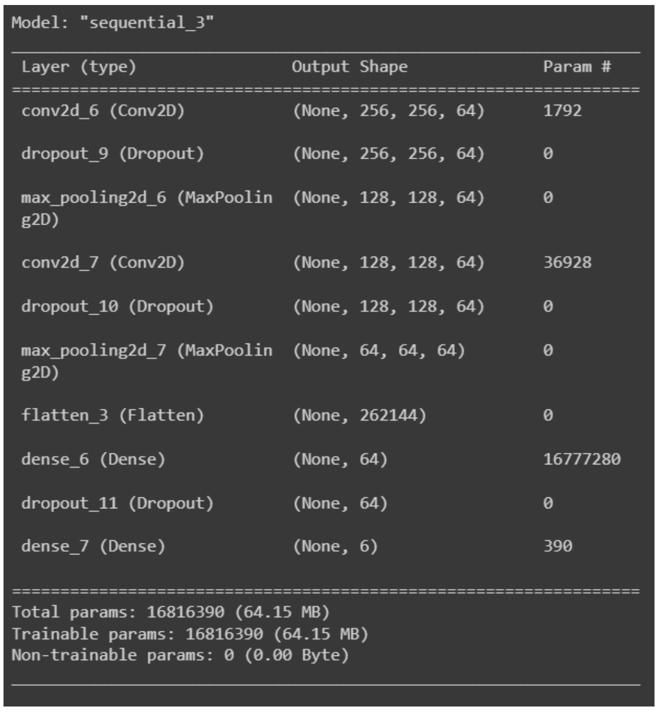
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 254, 254, 64)      1792

 dropout_9 (Dropout)         (None, 254, 254, 64)      0

 max_pooling2d_15 (MaxPooli  (None, 127, 127, 64)      0
 ng2D)

 conv2d_17 (Conv2D)          (None, 125, 125, 64)      36928

 dropout_10 (Dropout)        (None, 125, 125, 64)      0

 max_pooling2d_16 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 flatten_7 (Flatten)         (None, 246016)            0

 dense_15 (Dense)            (None, 64)                15745088

 dropout_11 (Dropout)        (None, 64)                0

 dense_16 (Dense)            (None, 6)                 390

=================================================================
Total params: 15784198 (60.21 MB)
Trainable params: 15784198 (60.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX G: Model 5 - CNN 1 with Dropout, Padding and Early Stoppage and optimal hyperparameters from cross validation with grid search.



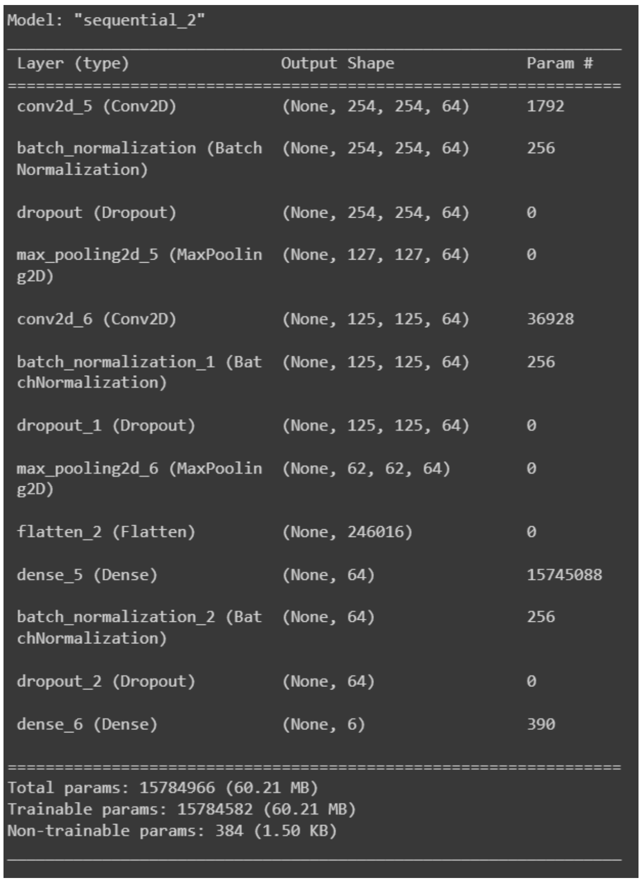
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 256, 256, 64)      1792

 dropout_6 (Dropout)         (None, 256, 256, 64)      0

 max_pooling2d_4 (MaxPoolin  (None, 128, 128, 64)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 128, 128, 64)      36928

 dropout_7 (Dropout)         (None, 128, 128, 64)      0

 max_pooling2d_5 (MaxPoolin  (None, 64, 64, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 262144)            0

 dense_4 (Dense)             (None, 64)                16777280

 dropout_8 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 6)                 390

=================================================================
Total params: 16816390 (64.15 MB)
Trainable params: 16816390 (64.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX H: Model 6: CNN 1 with Dropout, Padding and optimal hyperparameters from cross validation with grid search.



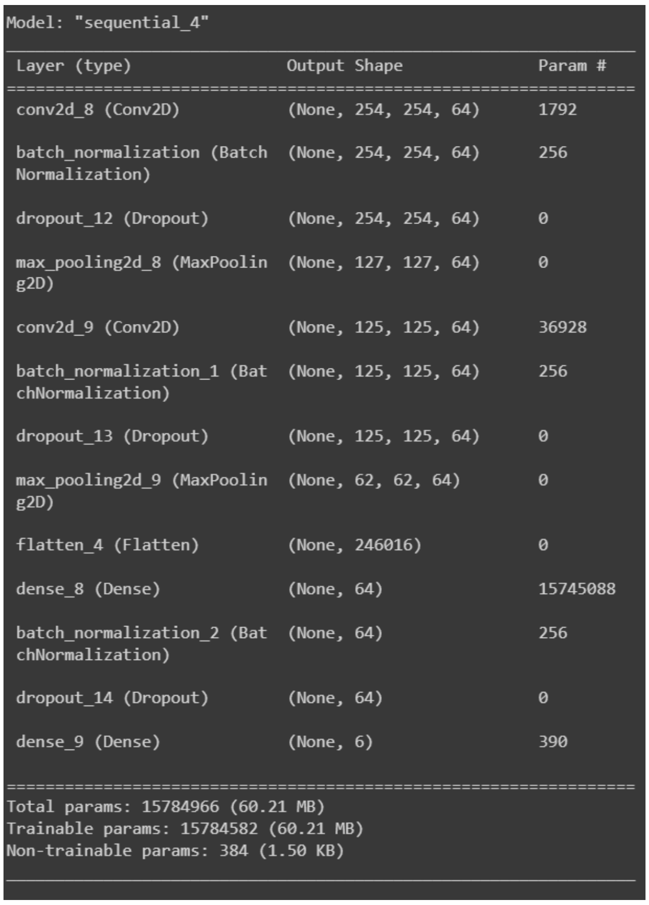
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 256, 256, 64)      1792

 dropout_9 (Dropout)         (None, 256, 256, 64)      0

 max_pooling2d_6 (MaxPoolin  (None, 128, 128, 64)      0
 g2D)

 conv2d_7 (Conv2D)           (None, 128, 128, 64)      36928

 dropout_10 (Dropout)        (None, 128, 128, 64)      0

 max_pooling2d_7 (MaxPoolin  (None, 64, 64, 64)        0
 g2D)

 flatten_3 (Flatten)         (None, 262144)            0

 dense_6 (Dense)             (None, 64)                16777280

 dropout_11 (Dropout)        (None, 64)                0

 dense_7 (Dense)             (None, 6)                 390

=================================================================
Total params: 16816390 (64.15 MB)
Trainable params: 16816390 (64.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX I: Model 7: CNN 1 with Dropout, Batch Normalisation, Early Stoppage and optimal hyperparameters from cross validation with grid search.



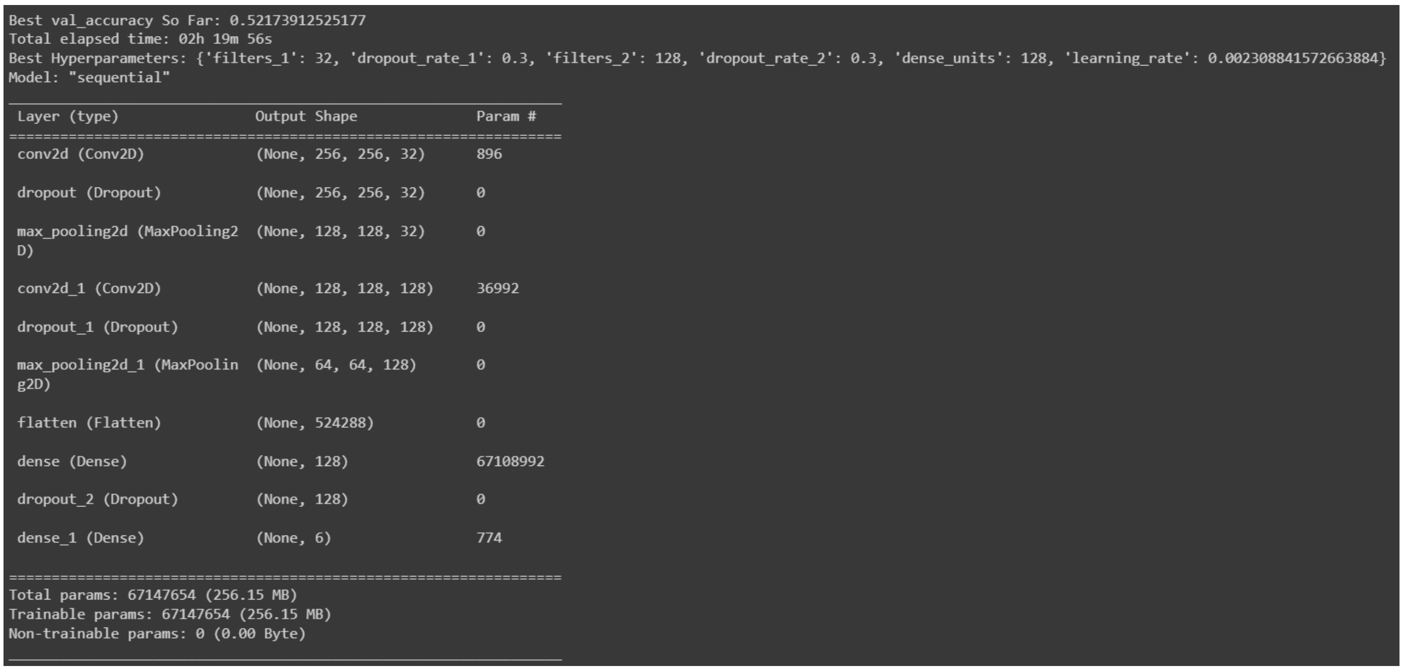
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 254, 254, 64)      1792

 batch_normalization (Batch  (None, 254, 254, 64)      256
 Normalization)

 dropout (Dropout)           (None, 254, 254, 64)      0

 max_pooling2d_5 (MaxPoolin  (None, 127, 127, 64)      0
 g2D)

 conv2d_6 (Conv2D)           (None, 125, 125, 64)      36928

 batch_normalization_1 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 dropout_1 (Dropout)         (None, 125, 125, 64)      0

 max_pooling2d_6 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 246016)            0

 dense_5 (Dense)             (None, 64)                15745088

 batch_normalization_2 (Bat  (None, 64)                256
 chNormalization)

 dropout_2 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 6)                 390

=================================================================
Total params: 15784966 (60.21 MB)
Trainable params: 15784582 (60.21 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

APPENDIX J: Model 8: CNN 1 with Dropout, Batch Normalisation, and optimal hyperparameters from cross validation with grid search.



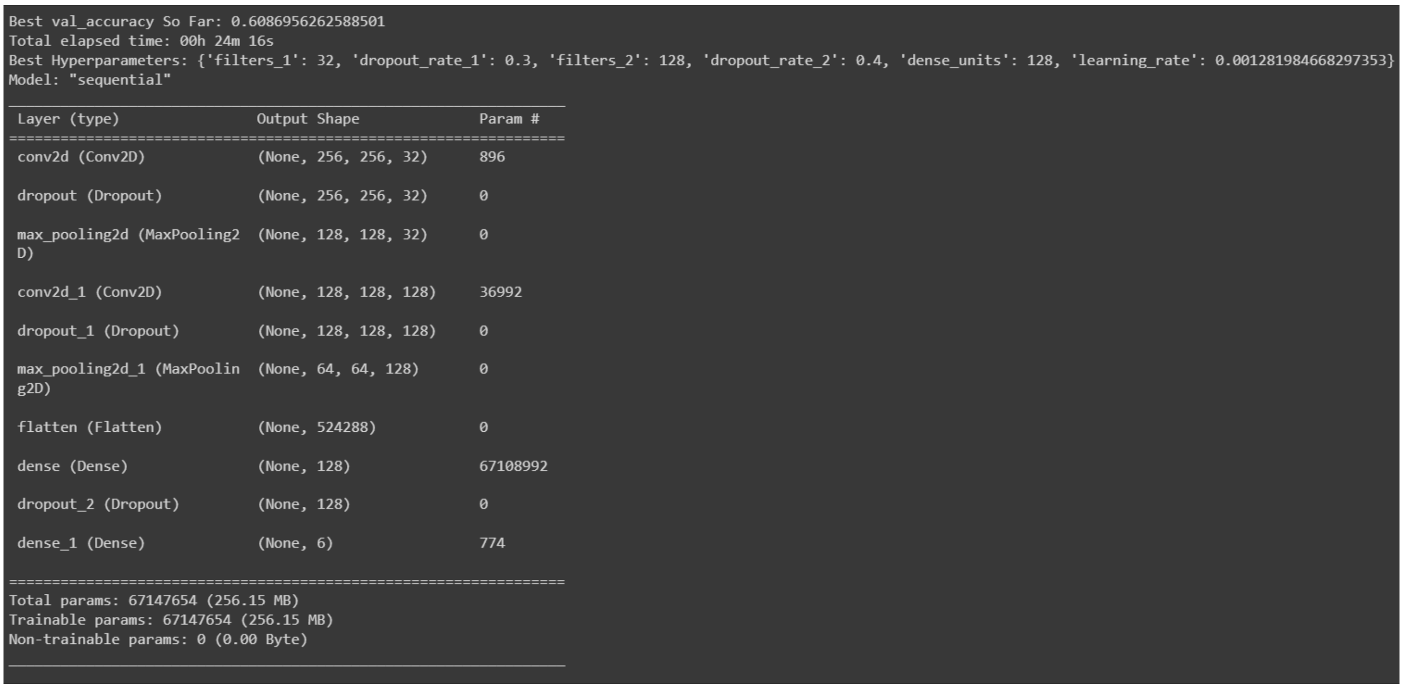
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 254, 254, 64)      1792

 batch_normalization (Batch  (None, 254, 254, 64)      256
 Normalization)

 dropout_12 (Dropout)        (None, 254, 254, 64)      0

 max_pooling2d_8 (MaxPoolin  (None, 127, 127, 64)      0
 g2D)

 conv2d_9 (Conv2D)           (None, 125, 125, 64)      36928

 batch_normalization_1 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 dropout_13 (Dropout)        (None, 125, 125, 64)      0

 max_pooling2d_9 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 flatten_4 (Flatten)         (None, 246016)            0

 dense_8 (Dense)             (None, 64)                15745088

 batch_normalization_2 (Bat  (None, 64)                256
 chNormalization)

 dropout_14 (Dropout)        (None, 64)                0

 dense_9 (Dense)             (None, 6)                 390

=================================================================
Total params: 15784966 (60.21 MB)
Trainable params: 15784582 (60.21 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

APPENDIX K: Model 9: Hyper parameter tuning using Keras Tuner for Model 4 (best performing model) with Dropout, Padding, Learning Rate and Early Stoppage, with hyperparameter results..



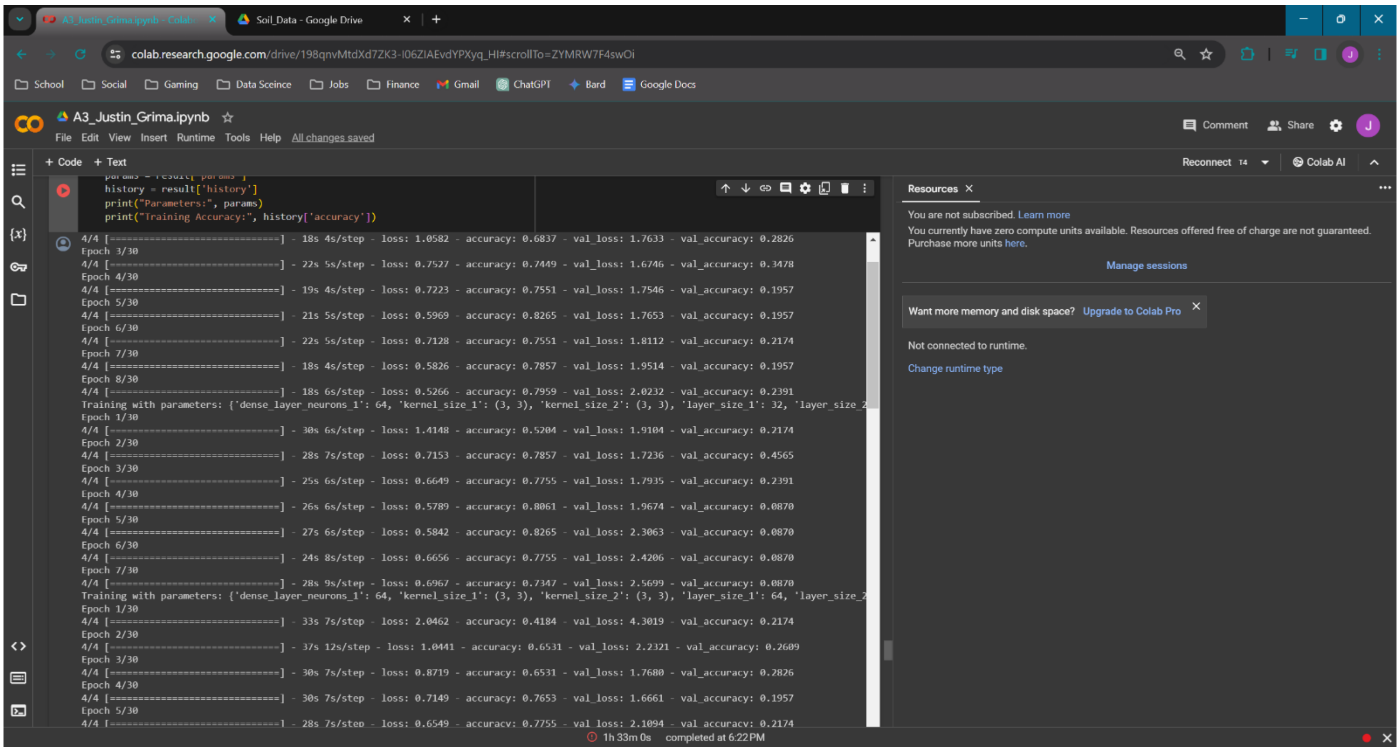
```
Best val_accuracy So Far: 0.52173912525177
Total elapsed time: 02h 19m 56s
Best Hyperparameters: {'filters_1': 32, 'dropout_rate_1': 0.3, 'filters_2': 128, 'dropout_rate_2': 0.3, 'dense_units': 128, 'learning_rate': 0.002308841572663884}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 dropout (Dropout)           (None, 256, 256, 32)      0

 max_pooling2d (MaxPooling2  (None, 128, 128, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 128)     36992

 dropout_1 (Dropout)         (None, 128, 128, 128)     0

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 128)       0
 g2D)

 flatten (Flatten)           (None, 524288)            0

 dense (Dense)               (None, 128)               67108992

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 67147654 (256.15 MB)
Trainable params: 67147654 (256.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX L: Model 10 - Hyper parameter tuning using Keras Tuner for Model 4 (best performing model) with Dropout, Padding, and Learning Rate, with hyperparameter results.



```
Best val_accuracy So Far: 0.6086956262588501
Total elapsed time: 00h 24m 16s
Best Hyperparameters: {'filters_1': 32, 'dropout_rate_1': 0.3, 'filters_2': 128, 'dropout_rate_2': 0.4, 'dense_units': 128, 'learning_rate': 0.001281984668297353}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 dropout (Dropout)           (None, 256, 256, 32)      0

 max_pooling2d (MaxPooling2  (None, 128, 128, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 128)     36992

 dropout_1 (Dropout)         (None, 128, 128, 128)     0

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 128)       0
 g2D)

 flatten (Flatten)           (None, 524288)            0

 dense (Dense)               (None, 128)               67108992

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 67147654 (256.15 MB)
Trainable params: 67147654 (256.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

APPENDIX M: Screenshot of no available RAM memory error.



```
history = result['history']
print("Parameters:", params)
print("Training Accuracy:", history['accuracy'])
```

```
4/4 [==============================] - 18s 4s/step - loss: 1.0582 - accuracy: 0.6837 - val_loss: 1.7633 - val_accuracy: 0.2826
Epoch 3/30
4/4 [==============================] - 22s 5s/step - loss: 0.7527 - accuracy: 0.7449 - val_loss: 1.6746 - val_accuracy: 0.3478
Epoch 4/30
4/4 [==============================] - 19s 4s/step - loss: 0.7223 - accuracy: 0.7551 - val_loss: 1.7546 - val_accuracy: 0.1957
Epoch 5/30
4/4 [==============================] - 21s 5s/step - loss: 0.5969 - accuracy: 0.8265 - val_loss: 1.7653 - val_accuracy: 0.1957
Epoch 6/30
4/4 [==============================] - 22s 5s/step - loss: 0.7128 - accuracy: 0.7551 - val_loss: 1.8112 - val_accuracy: 0.2174
Epoch 7/30
4/4 [==============================] - 18s 4s/step - loss: 0.5826 - accuracy: 0.7857 - val_loss: 1.9514 - val_accuracy: 0.1957
Epoch 8/30
4/4 [==============================] - 18s 6s/step - loss: 0.5266 - accuracy: 0.7959 - val_loss: 2.0232 - val_accuracy: 0.2391
Training with parameters: {'dense_layer_neurons_1': 64, 'kernel_size_1': (3, 3), 'kernel_size_2': (3, 3), 'layer_size_1': 32, 'layer_size_2
Epoch 1/30
4/4 [==============================] - 30s 6s/step - loss: 1.4148 - accuracy: 0.5204 - val_loss: 1.9104 - val_accuracy: 0.2174
Epoch 2/30
4/4 [==============================] - 28s 7s/step - loss: 0.7153 - accuracy: 0.7857 - val_loss: 1.7236 - val_accuracy: 0.4565
Epoch 3/30
4/4 [==============================] - 25s 6s/step - loss: 0.6649 - accuracy: 0.7755 - val_loss: 1.7935 - val_accuracy: 0.2391
Epoch 4/30
4/4 [==============================] - 26s 6s/step - loss: 0.5789 - accuracy: 0.8061 - val_loss: 1.9674 - val_accuracy: 0.0870
Epoch 5/30
4/4 [==============================] - 27s 6s/step - loss: 0.5842 - accuracy: 0.8265 - val_loss: 2.3063 - val_accuracy: 0.0870
Epoch 6/30
4/4 [==============================] - 24s 8s/step - loss: 0.6656 - accuracy: 0.7755 - val_loss: 2.4206 - val_accuracy: 0.0870
Epoch 7/30
4/4 [==============================] - 28s 9s/step - loss: 0.6967 - accuracy: 0.7347 - val_loss: 2.5699 - val_accuracy: 0.0870
Training with parameters: {'dense_layer_neurons_1': 64, 'kernel_size_1': (3, 3), 'kernel_size_2': (3, 3), 'layer_size_1': 64, 'layer_size_2
Epoch 1/30
4/4 [==============================] - 33s 7s/step - loss: 2.0462 - accuracy: 0.4184 - val_loss: 4.3019 - val_accuracy: 0.2174
Epoch 2/30
4/4 [==============================] - 37s 12s/step - loss: 1.0441 - accuracy: 0.6531 - val_loss: 2.2321 - val_accuracy: 0.2609
Epoch 3/30
4/4 [==============================] - 30s 7s/step - loss: 0.8719 - accuracy: 0.6531 - val_loss: 1.7680 - val_accuracy: 0.2826
Epoch 4/30
4/4 [==============================] - 30s 7s/step - loss: 0.7149 - accuracy: 0.7653 - val_loss: 1.6661 - val_accuracy: 0.1957
Epoch 5/30
4/4 [==============================] - 28s 7s/step - loss: 0.6549 - accuracy: 0.7755 - val_loss: 2.1094 - val_accuracy: 0.2174
```

Resources

You are not subscribed. Learn more

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

Manage sessions

Want more memory and disk space? Upgrade to Colab Pro

Not connected to runtime.

Change runtime type

1h 33m 0s completed at 6:22 PM