CS4287 Neural Computing

Assignment 1: Sem1 AY 25/26

Convolutional Neural Networks (CNNs)

Aron Calvert | 22370374

Leo O'Shea | 22342761

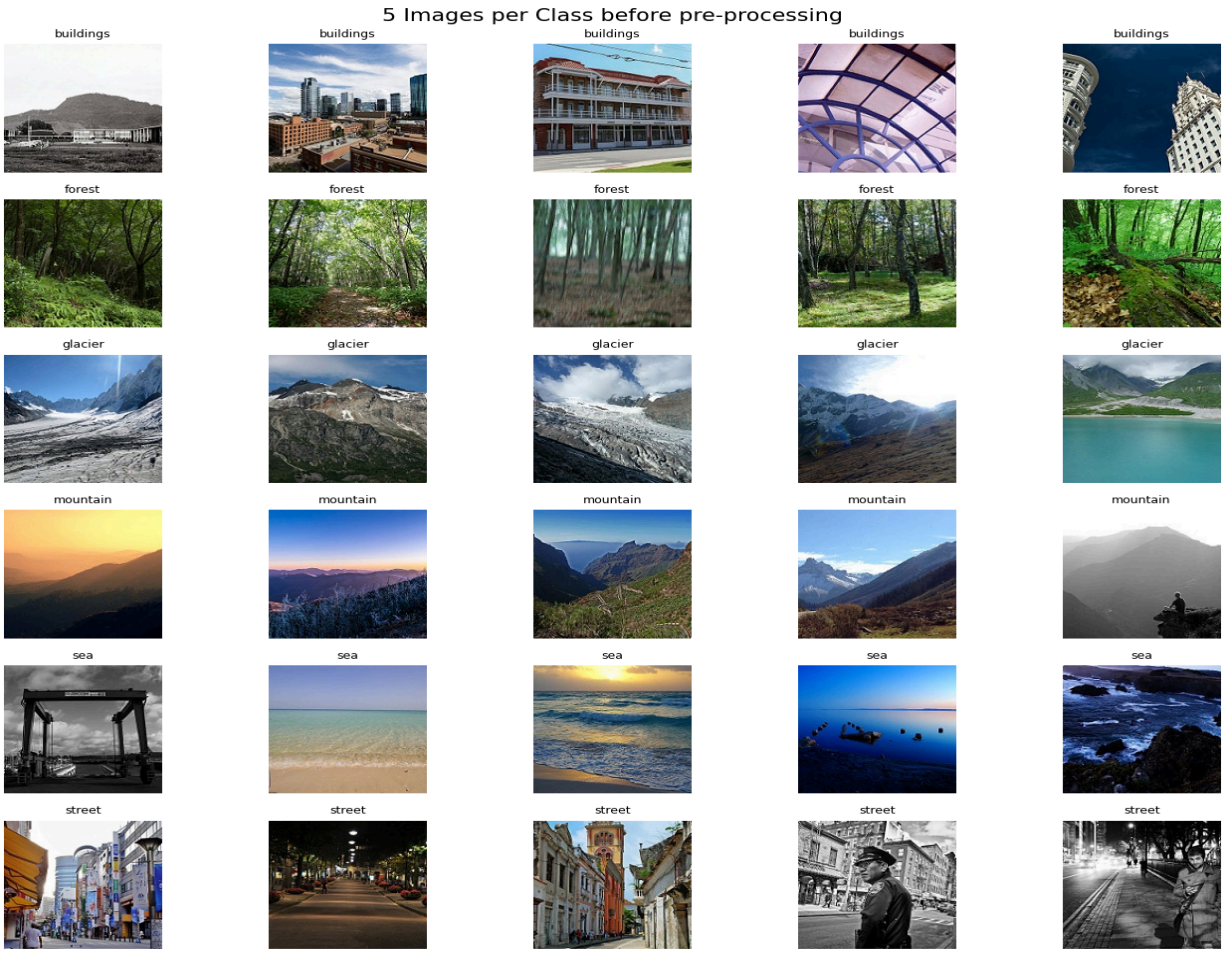
Table of Contents

1. The Data Set

For this Project we decided to utilise the Intel Image Classification[1] data-set from kaggle. We thought this data-set was a good choice as it was a manageable size for training using google colab in a reasonable time-frame and we felt it captured a good balance between neither being too simplistic or difficult. The data-set is made up of 25000 images of "Natural Scenes around the world". Each image is of size 150x150 and is divided into six categories.

'buildings' -> 0, 'forest' -> 1, 'glacier' -> 2, 'mountain' -> 3, 'sea' -> 4, 'street' -> 5

This dataset was advantageous as it is pre-divided into Train, Test and Prediction data. There are 14k images in Train, 3k in Test and 7k in Prediction. The selection of images is diverse in terms of lighting, weather and angle which should aid in generalisation.



[1]Bansal, P. (2018) *Intel Image Classification* [dataset]. Kaggle. Available at: https://www.kaggle.com/datasets/puneet6060/intel-image-classification (Accessed: 8 November 2025).

figure 1

a. Visualisation of some of the key attributes.

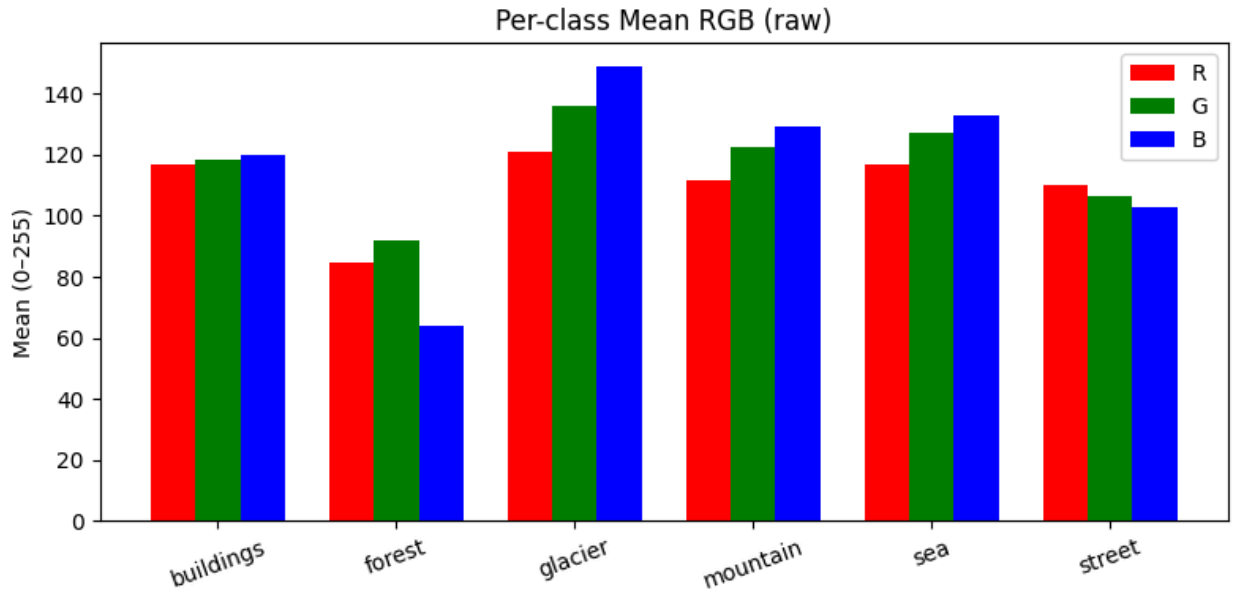


figure 2

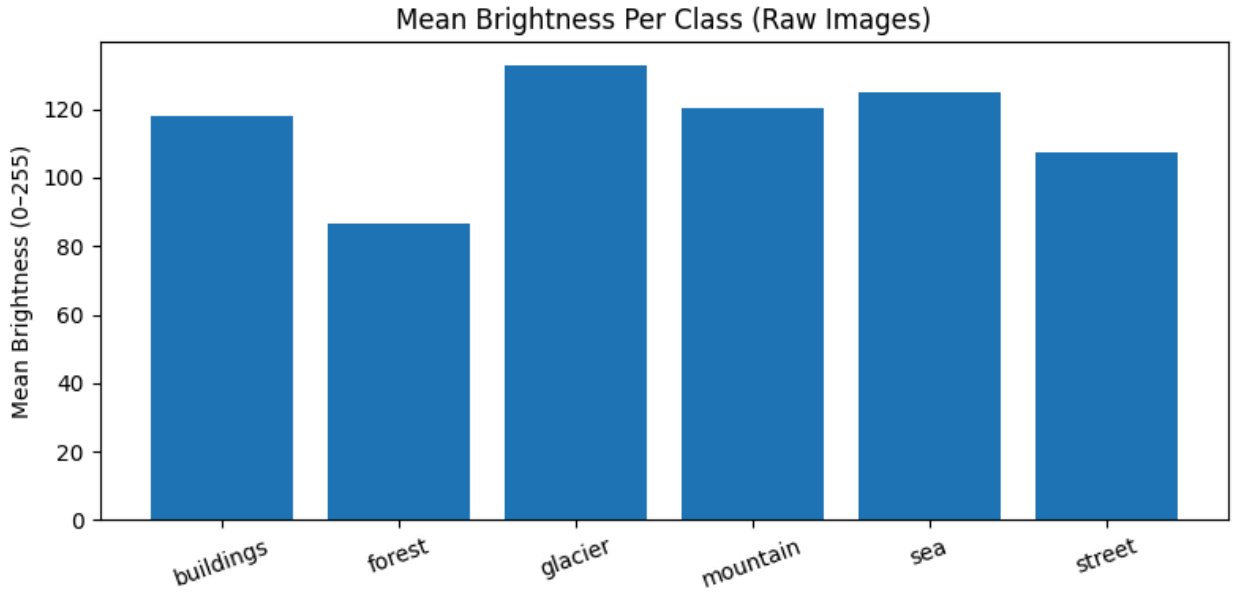


figure 3

Per-class MeanRGB and Mean Brightness analysis shows us that the classes have a relatively high difference in terms of colour intensity and luminance. This may cause a CNN to simply

learn on the basis of Brightness and Colour leading to overfitting. Preventing the learning of more meaningful features. This might be concerning for forests which have the lowest brightness and a distinct set of colours within the limited dataset used for these charts.

figure 4

The mean image per class shows us an average visual representation for a sample of each class. Fine details are lost but some visual features remain. Colours and brightness are still dominant but we can see the forest class's vertical textures representing tree trunks and that the mountain and sea classes show horizontal gradients indicating the horizon. The street image appears to reveal a bright path ahead, surrounded by buildings on each side. These mean images may potentially give us an insight into the sort of features our CNN may seek to learn. The CNN may learn low-level features such as edges and gradients aligned with these patterns in its early layers.

b. Pre-processing applied to the data

All images were preprocessed using the ImageDataGenerator pipeline from Keras.[2] We rescaled pixel values by a factor of 1./255, normalising them from the original 0–255 range to [0–1] this

[2] TensorFlow (n.d.) *ImageDataGenerator*. TensorFlow API Docs. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (Accessed: 8 November 2025).

will help prevent features with large numeric ranges from dominating the learning process. It further improves numerical stability and convergence of gradient based optimization.

The dataset was divided into training (80%) and validation (20%) subsets using the validation_split parameter within the ImageDataGenerator. A separate test set was loaded using its own dedicated generator, also rescaled to [0–1] but without augmentation, for verification at the end.

We utilised the train_datagen.flow_from_directory() from Keras in order to read our images directly from the directory structure, to ensure the images are of the standardised size (150x150), to apply the “train_gen” image preprocessing and to batch the data. We applied the categorical class mode to automatically one hot code encode class labels for multi class classification.

2. The Network Structure and Hyperparameters

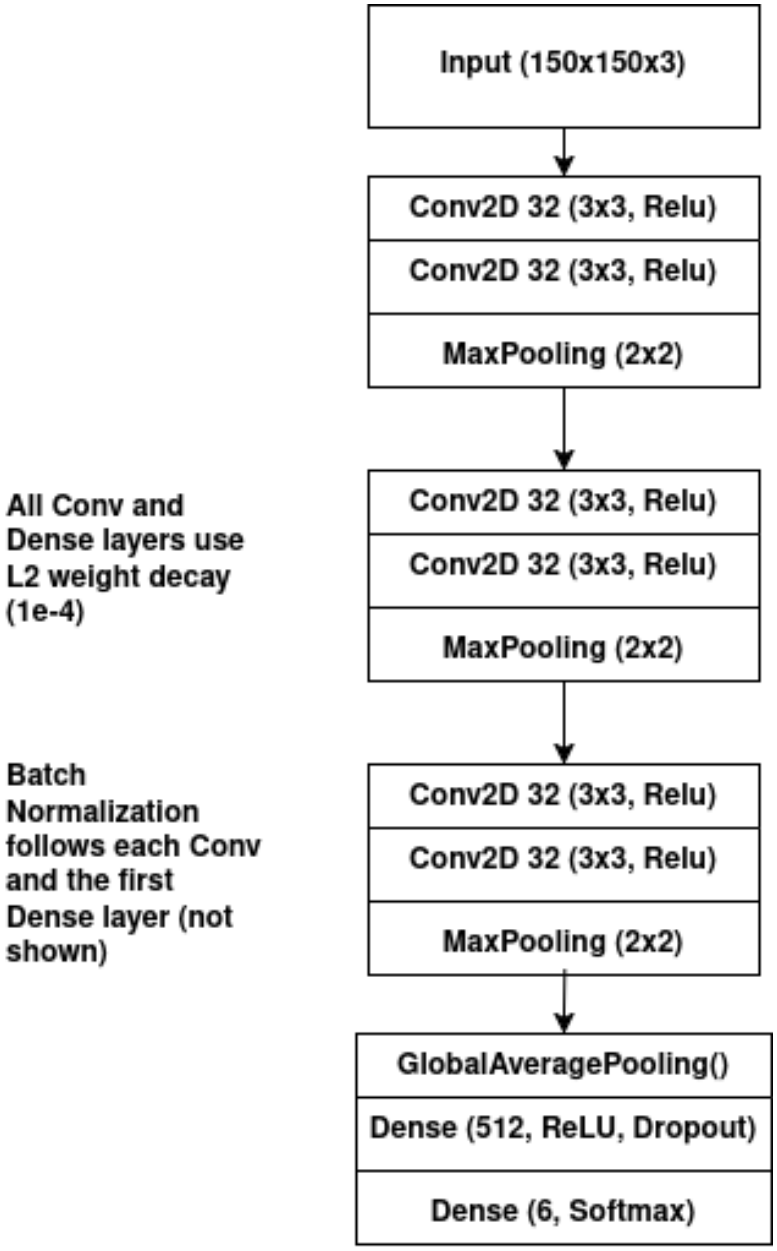


figure 5: A diagram of the network structure

The CNN we implemented is a VGG-like model that follows the usual VGG pattern of stacked convolutional blocks. Each block consists of two convolutional layers followed by a MaxPooling layer. Just like the VGG architecture, it employs 3×3 kernels and the ReLU activation function, the number of filters doubled in each block.

The design is different from the standard VGG design in a few important ways. The VGG-16 is a lot deeper, it has five convolutional blocks followed by three fully connected layers.[3] Our CNN model is shallower consisting of just 3 convolutional blocks before the connected layers,

Instead of the three fully connected layers and the use of the Flatten() function seen in VGG's, this model incorporates a GlobalAveragePooling2D() layer followed by two fully connected layers. Flatten() works by transforming a multidimensional tensor into a vector. GlobalAveragePooling2d() on the other hand "performs an operation on the data. It calculates the average value of each feature map in the input tensor and outputs a tensor that is smaller in size." [4] GlobalAveragePooling2D() is commonly used in ResNet and more modern CNN's. Flatten creates a much larger number of parameters and may potentially lead to overfitting. On our smaller dataset GlobalAveragePooling2D() was thought to be a better choice to reduce computational overhead and to improve generalisation.

We used the ReLU activation function, the standard choice in VGG-like architectures. ReLU gives us non-linearity and is computationally efficient, this is because it outputs zero for negative inputs and has a linear response for positive inputs. This avoids the problems that sigmoid or tanh functions have and will hopefully give us faster convergence during training.

He_Normal is the weight initializer used in VGG-style models. We trained the model using both He_Normal and the Keras default Glorot_Uniform initializers in separate training runs and there were negligible performance differences. The He_Normal initializer was chosen because it is designed for use with ReLU activations to help prevent vanishing or exploding gradients during backpropagation.[5]

The original VGG-19 and VGG-16 were developed before batch normalisation was introduced. However since then it has been seen that VGG's developed utilising batch normalisation "show a drastic improvement, both in terms of optimization and generalization performance"[6]. As such we integrated batch normalisation after each convolutional and dense layer.

---

[3] iMind Labs (n.d.) *VGG16*. iMind Labs Wiki. Available at: https://wiki.imindlabs.com.au/ds/dl/1_models/2-cnn/4_vgg16/ (Accessed: 8 November 2025).

[4] Saturn Cloud (2023) 'Understanding the Difference Between Flatten() and GlobalAveragePooling2D() in Keras'. *Saturn Cloud Blog*, 10 July. Available at: https://saturncloud.io/blog/understanding-the-difference-between-flatten-and-globalaveragepooling2d-in-keras/ (Accessed: 8 November 2025).

[5] GeeksforGeeks (2025) 'Kaiming Initialization in Deep Learning'. Available at: https://www.geeksforgeeks.org/deep-learning/kaiming-initialization-in-deep-learning (Accessed: 8 November 2025).

[6] Santurkar, S., Tsipras, D., Ilyas, A. and Madry, A. (2018) 'How Does Batch Normalization Help Optimization?', *arXiv*. Available at: https://arxiv.org/abs/1805.11604 (Accessed: 8 November 2025).

We used Dropout as utilised in conventional VGG models, it being integrated into our first fully connected layer. Our additional regularisation method was the application of L2-regularisation. L2 regularisation helps prevent overfitting by penalising large weights. Due to the application of drop-out we must note that comparing the training and validation loss may be misleading.

We used the SoftMax activation function for our output layer. This was chosen as each image should be a member of only one class. SoftMax outputs a vector that represents the probability of it being each class.

3. The Loss Objective function

A loss function is a type of objective function used to measure a model's performance by tracking its degree of error. It does this by calculating the deviation of a model's predictions from the correct predictions. Their main purpose is to be a measure of a model's success. By looking at the loss function, one can judge when a model is sufficiently trained. During training, the model will adjust its parameters with the goal of reducing loss to get as close as possible to being below the loss's predetermined value.[7]

The type of loss function we chose to use is Categorical Cross-Entropy (CCE), with label smoothing to reduce overconfidence. A cross-entropy loss function calculates the difference between the true probability distribution and the predicted probability distribution of a classification model.

Categorical Cross-Entropy (CCE) is regarded as one of the best choices for multi-class classification models, but there are other loss functions we considered, such as Sparse Categorical Cross-Entropy (SCCE), Mean Squared Error (MSE), and Focal Loss. As our model is classification, we would not choose any regression loss functions such as MSE or square loss. Similarly, we did not consider binary cross-entropy, as it is used for classification models where there are only two possible outcomes (e.g., cat or dog).

Focal loss is a modified cross-entropy function that tackles cross-entropy's failure to tell apart easy and difficult examples. Focal loss uses "down weightening" to make the model focus on learning harder examples, putting less emphasis on the easy examples. This can be useful in imbalanced scenarios, but our dataset is balanced, meaning this would add unnecessary complexity.

Sparse categorical cross-entropy (SCCE) is said to be more memory efficient and faster due to its use of integer labels rather than one-hot encoded labels. We did not want to use SCCE as we did not want to rank the classes. Still, due to the similarities between SCCE and CCE, we chose to use both of them and compare the results. Surprisingly, CCE had an accuracy of 89.4%, while SCCE had an accuracy of 88.8%. We believe this was due to SCCE being unable to use label smoothing, causing its accuracy to be worse, if even slightly.

[7]IBM (n.d.) 'Loss function'. *IBM Think*. Available at: https://www.ibm.com/think/topics/loss-function (Accessed: 8 November 2025).

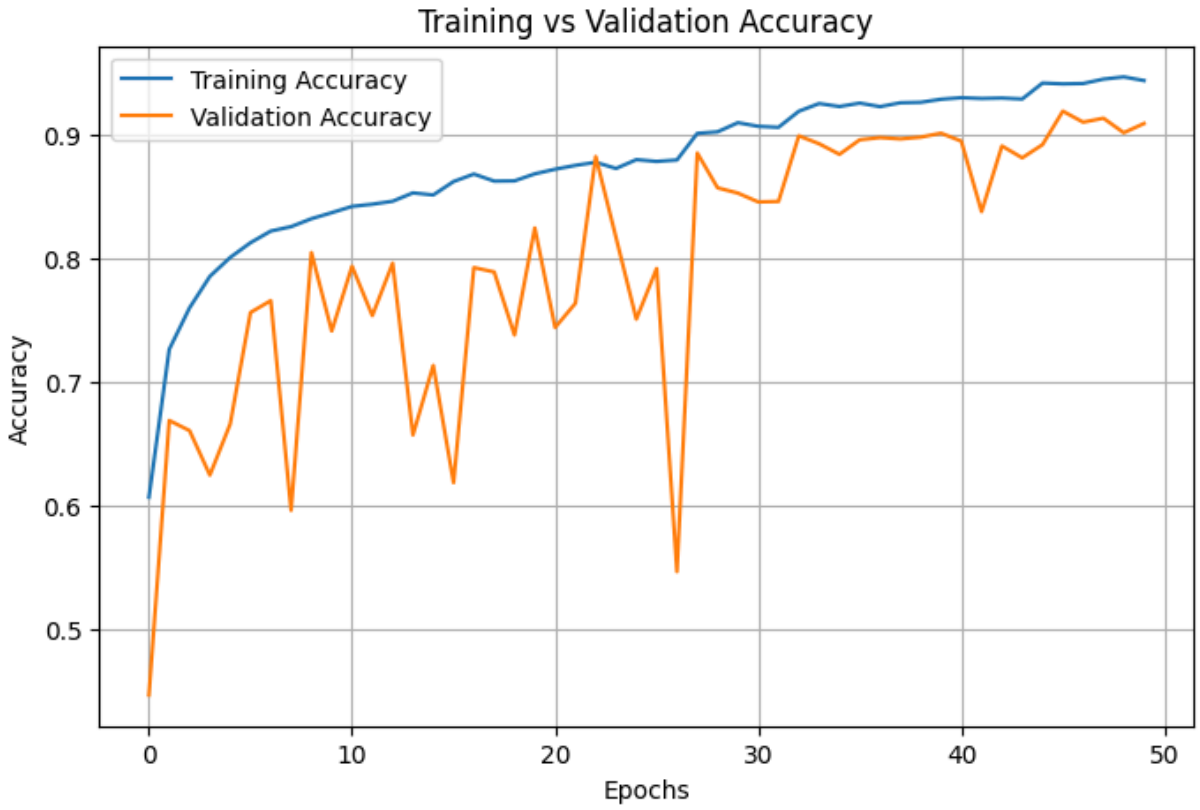


figure 6: Training vs Validation Accuracy for the model when using sparse categorical cross-entropy.

4. The Optimiser

The main optimisers we have considered are Adam, Stochastic Gradient Descent (SGD), and Adaptive Gradient Descent (Adagrad). In most cases, each of these optimisers can be the best choice depending on the problem.

SGD is an efficient optimisation algorithm suited for dealing with large datasets and models due to its memory efficiency. Furthermore, SGD's stochastic nature leads to noisy updates that can help the model avoid local minima, potentially leading to better solutions. Despite all this, SGD still runs into some detrimental issues. This is mainly its tendency to have a slower convergence. The cost function fluctuates due to its noisy updates, causing convergence to be slower. Most importantly, it is greatly affected by the choice of learning rate. If the learning rate is too high, the algorithm will diverge, but if it is too small, it will be slow to converge.[8]

Meanwhile, Adagrad is an optimisation algorithm suited for dealing with small datasets, meaning it is already not suitable for our model. Still, some features of Adagrad are used in Adam. Compared to SGD, it vastly improves on SGD's fixed learning rate by having an adaptive learning rate for each parameter. Still, Adagrad has its drawbacks. It is still affected by the initial choice of the learning rate, and the learning rate is prone to shrink continuously, causing slow convergence and early stopping.[9]

Adam combines the advantages of Adagrad and RMSprop (both extensions of SGD). It is suited for large datasets and models, is memory efficient, and has an adaptive learning rate. Furthermore, it builds upon momentum, an update rule used to accelerate gradient descent, resulting in faster convergence. Adam performs vastly better than other optimisers while still giving superior results and requiring less tuning, meaning it was not a very hard decision to choose Adam over our considered choices.[10]

[8]GeeksforGeeks (30 Sep, 2025) 'Stochastic Gradient Descent (SGD)'. Available at: https://www.geeksforgeeks.org/machine-learning/ml-stochastic-gradient-descent-sgd/ (Accessed: 7 Nov 2025).
[9]GeeksforGeeks (30 Sep, 2025) 'Intuition behind AdaGrad Optimizer'. Available at: https://www.geeksforgeeks.org/machine-learning/intuition-behind-adagrad-optimizer/ (Accessed: 7 Nov 2025).
[10]GeeksforGeeks (4 Oct, 2025) 'Adam Optimizer'. Available at: https://www.geeksforgeeks.org/deep-learning/adam-optimizer/ (Accessed: 7 Nov 2025).

5. Cross Fold Validation

Cross validation is used to see how well a model performs with unseen data, all the while helping to prevent overfitting. It splits the data into multiple parts, using some parts to train the model on, and the rest to test, and repeats this. Then, it averages the result from each validation step until it gets the final result.

Unfortunately, we did not use any form of cross validation. It is not commonly used in VGG models, but we would have used K-Fold Cross Validation as, compared to other cross validation methods such as the holdout method, it has a lower bias and reduces risk of overfitting. For our case, K-Fold Cross validation is more suited to smaller datasets because the model is trained 'k' times, and would have slowed down the execution time massively due to our large dataset. With our equipment, we could not afford this massive increase in demand from the model. Training would have taken 'k' times longer.

Furthermore, K-Fold Cross validation seeks to solve an issue with unevenly split data. We don't have this problem, our data is split evenly. We have strong data augmentation implemented already which reduces decorrelated batches and reduces overfitting which is one of the main reasons for cross fold validation. Given these, we have very little reason to implement K-Fold Cross validation, so we chose to exclude it from our model.[11]

[11] Baeldung (28 Feb, 2025) 'How to Use K-Fold Cross-Validation in a Neural Network?'. Available at: https://www.baeldung.com/cs/k-fold-cross-validation (Accessed: 7 Nov 2025).

6. Results

The model completed a full 50 epochs of training, early stopping to restore the best weights based on validation loss was not utilised as our best result was achieved at our final Epoch. This achieved a training accuracy of 94.4% and a training loss of 0.4844. The validation accuracy was 91.6% with validation loss of 0.5401. By this point learning rate had been reduced to a size of 1.5625e-05

Following training this final saved model was tested on the test set of 3000 un-augmented images. The model achieved an 89.4% test accuracy and 0.5458 loss. This shows consistent performance across validation and test sets.

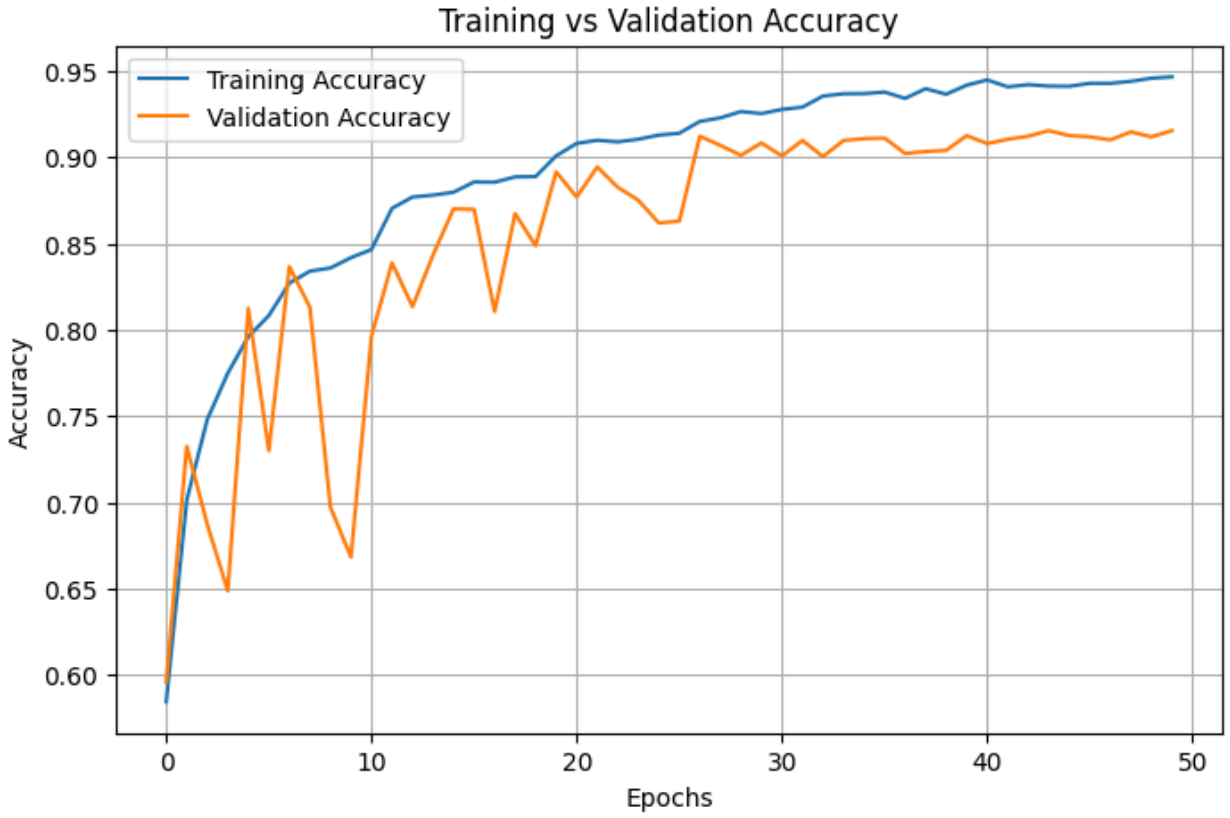


figure 7

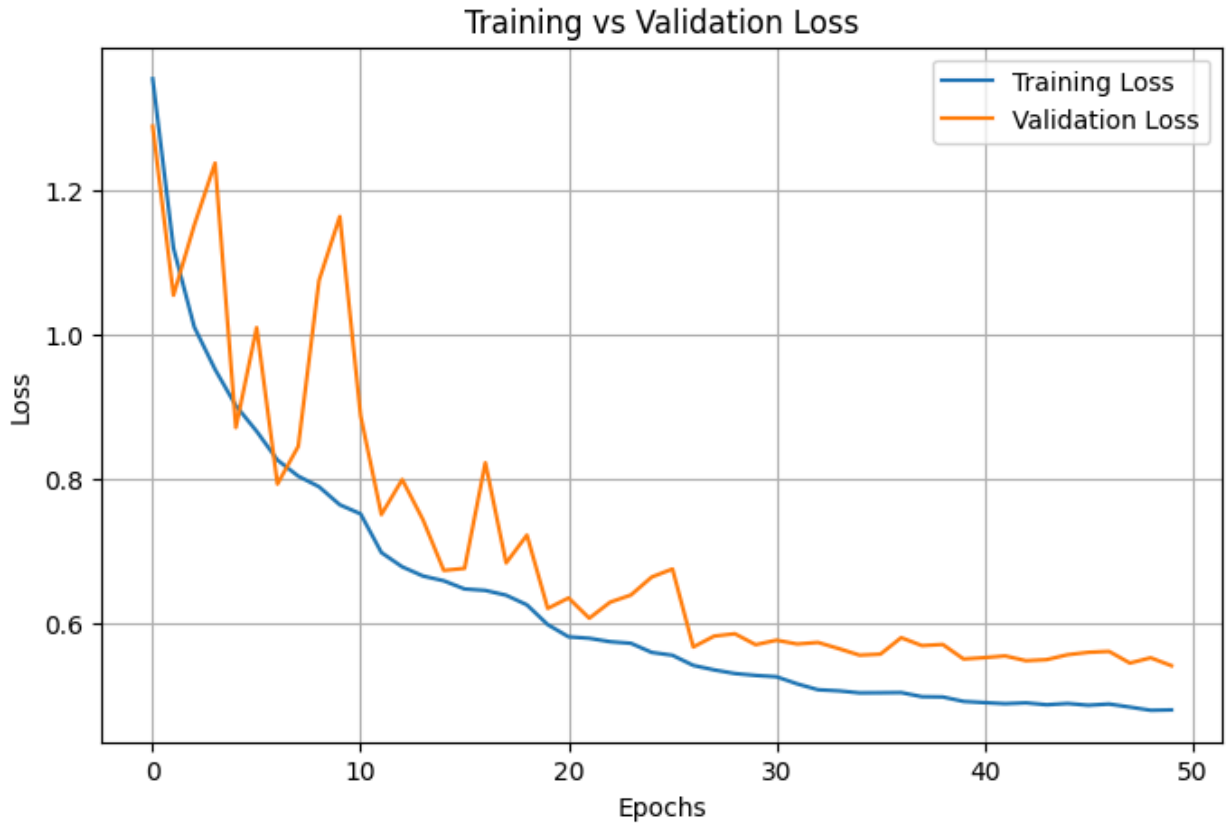


figure 8

Figures 7 and 8 show us the training and validation accuracy and loss across the 50 epochs Training accuracy increased from 0.58 in the first epoch to 0.94 by the final epoch, while validation accuracy followed a similar trajectory, stabilising around 0.91 after roughly 25 epochs. Correspondingly, both training and validation loss values decreased throughout the training process, with the training loss falling from around 1.3 to 0.48, and validation loss converging to approximately 0.54.

Classification Report:

	precision	recall	f1-score	support
Buildings	0.84	0.92	0.88	437
Forest	0.97	0.98	0.98	474
Glacier	0.88	0.84	0.86	553
Mountain	0.86	0.86	0.86	525
Sea	0.91	0.93	0.92	510
Street	0.92	0.85	0.89	501
Accuracy			0.89	3000
Macro avg	0.90	0.90	0.90	3000
Weighted avg	0.90	0.89	0.89	3000

figure 9

The classification report (figure 9) shows us that the model achieved an overall accuracy of 89% for all classes. Performance varied between classes with Forest achieving the highest scores. Mountain and Glacier achieved the lowest scores. Despite this the overall performance was mostly balanced with an macro-averaged f1-score of 0.90 and a weighted average of 0.89.

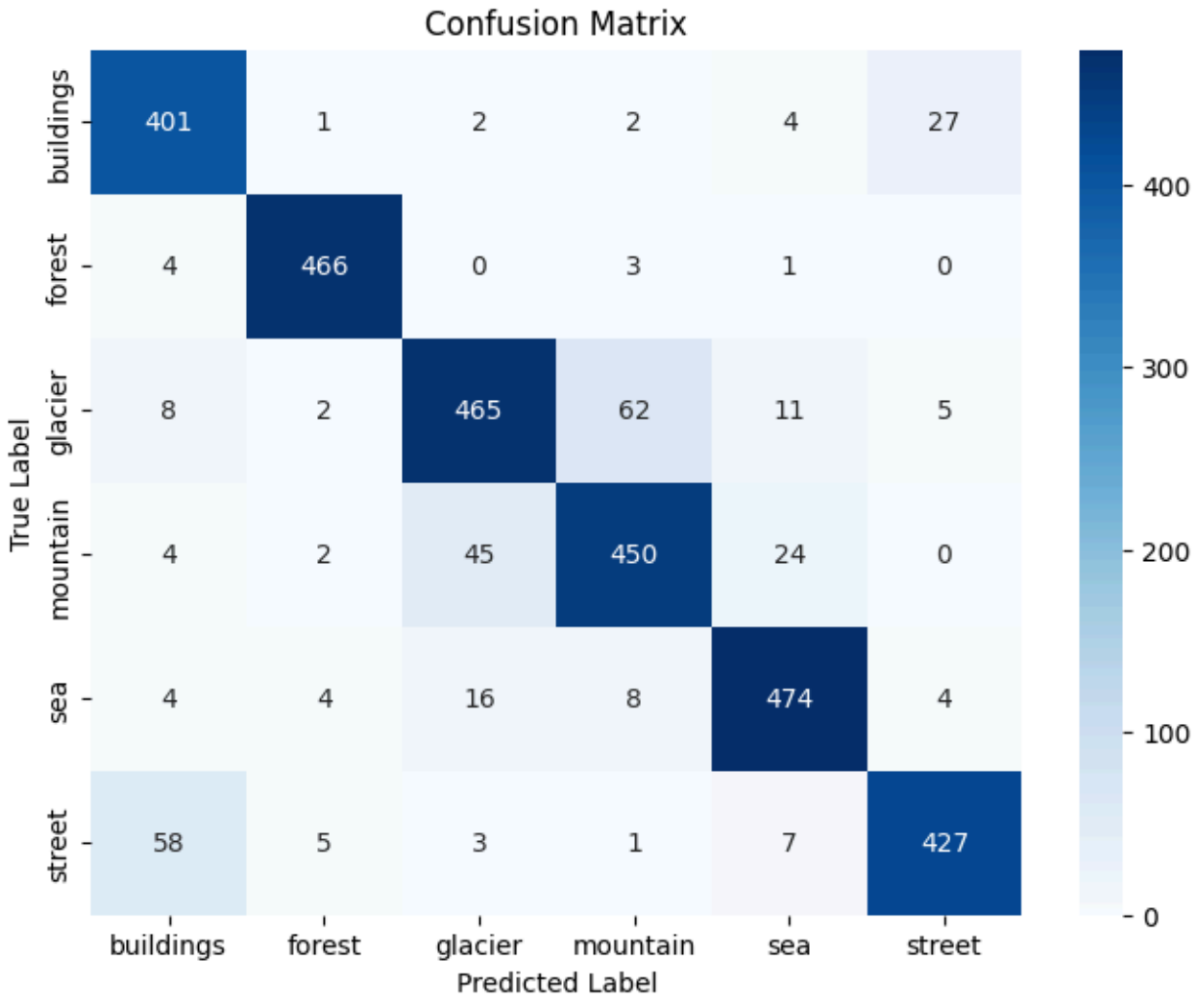


figure 10

Figure 10 is the confusion matrix for the model's predictions on the test set. The majority of samples lie along the diagonal that shows strong overall classification performance across all six classes. The forest and sea classes have the highest correct prediction counts, with very little confusion. Some confusions can be seen between mountains and glaciers, as well as between streets and buildings. The confusion matrix shows that correct classifications are the most common and that confusions are relatively minor.

7. Evaluation of the results

The model demonstrated a strong performance, with a final test accuracy of 89.4% and a macro-averaged F1-score of 0.90, proving that it generalised effectively to unseen data. The convergence of training and validation accuracy and the reduction of loss, indicates stable training behaviour with no massive signs of overfitting. This tells us that the use of batch normalisation, dropout, and L2 regularisation successfully helped to stabilise learning and improved generalisation.

Per-class analysis and the confusion matrix show that visually distinct categories such as forest and sea achieved the highest precision and recall likely over their clear colour and texture characteristics. Mountain and glacier performed worse with a lower F1-scores (0.86), which is probably due to their shared visual features, dominated by blue and grey tones and a similar gradient, this likely lead to occasional misclassification between them which can be seen in the confusion matrix. Similarly, streets and buildings overlapped slightly due to common features.

The model was very good at preventing overfitting, as the validation metrics remained close to the training results for most of training. We think that batch normalisation contributed to faster and more stable convergence by normalising activations, while L2 regularisation penalised large weight magnitudes, reducing sensitivity to noise.

Our inclusion of data augmentation (random rotations, flips, and zooms) was also likely important to model performance. It effectively increased the difference in data, allowing the CNN to generalise better to unseen test samples. Without these transformations, the network would maybe have overfit to colour and brightness distributions that we showed in the unaugmented dataset.

Our implementation of ReduceLROnPlateau() was potentially the hyperparameter with the most visible effect on training. By monitoring validation loss and automatically reducing the learning rate by a factor of 0.5 after four stagnant epochs, it gave us smoother convergence and helped us work across stagnation in training.

Our training log showed, throughout the first ten epochs the validation accuracy stagnated and the validation loss increased again (e.g., val_loss at Epoch 3: 0.9975, then at Epoch 4: 1.6240). After implementing ReduceLROnPlateau at Epoch 12 (learning rate reduced from 0.001 to 0.0005) the validation accuracy improved (Epoch 13 val_accuracy: 0.8584) and the loss dropped to 0.6623.

This behaviour is also shown with research that tells us that dynamically reducing the learning rate when the loss plateaus enables the optimiser to take finer steps and avoid getting stuck in

local minima. It has been found that ReduceLROnPlateau "converges faster while maintaining a similar or even better loss on the validation set when compared to OneCycleLR".[12]

We also experimented with the implementation of an additional block (2 Convolutional, 1 maxpooling layers) into the Model. We thought that additional depth in our CNN would allow for better feature extraction. In reality the difference was minimal apart from a large increase in training time. The model with an additional block performed at a test accuracy of 90% vs the 89.5% of the chosen model. Figure 11 shows the training vs validation accuracy for this model.

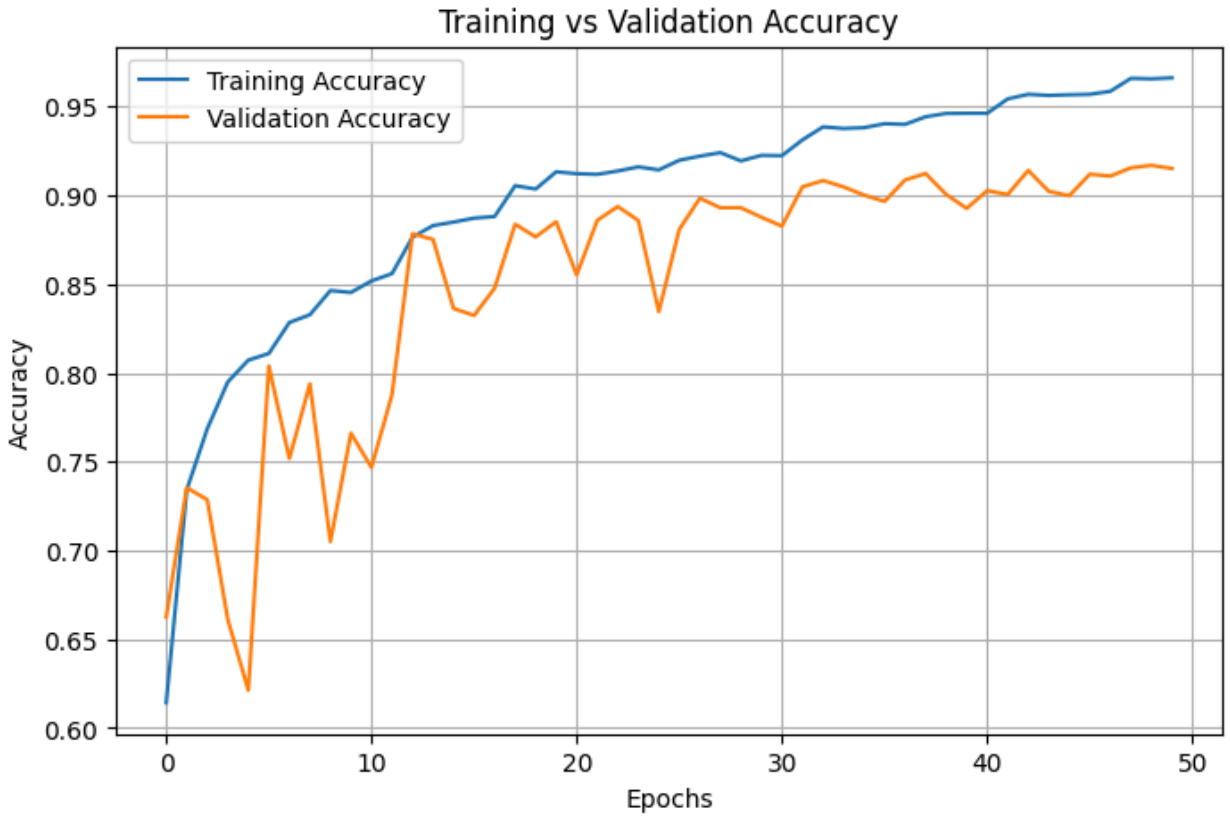


figure 11

Overall, our chosen models performance was high, minor misclassifications highlight where we might improve. These could include the integration of transfer learning from a pretrained model (e.g. VGG16 or ResNet50), which could enhance feature extraction, or fine-tuning of the learning rate schedule and regularisation strength to further balance generalisation and convergence.

[12]Al-Kababji, A., Bensaali, F. and Dakua, S.P. (2022) 'Scheduling Techniques for Liver Segmentation: ReduceLROnPlateau Vs OneCycleLR', *arXiv*. Available at: https://arxiv.org/abs/2202.06373 (Accessed: 8 November 2025).

8. Impact of varying hyperparameters and data augmentation in addressing overfitting

The results shown above came as a result of the fine tuning of hyperparameters and the implementation of data augmentation following early training runs that showed clear examples of overfitting. Figure 12 shows the gap between training accuracy and validation accuracy in one of our earlier iterations that more closely followed a conventional VGG architecture.

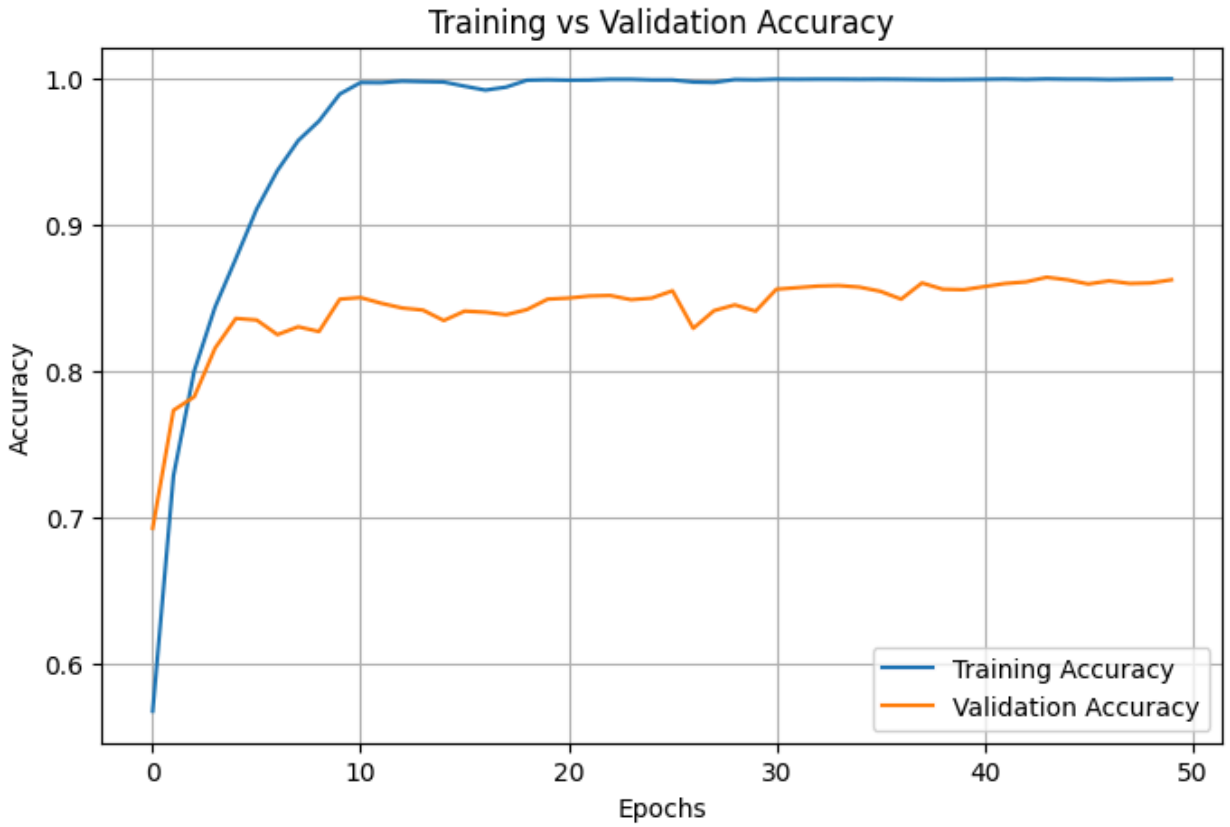


figure 12

To begin to address this overfitting, we introduced additional data augmentation and modified several hyperparameters. We applied data augmentation techniques to aid the model with generalisation. We applied a rotation of 20% and a zoom range of 20% along with applying a horizontal flip randomly to each image. These alterations are minor enough to preserve the image "realism". This approach ensures that the model has a new perspective on the data being fed to it each epoch. This reduces the chances of overfitting and hopefully leaves it being able to learn more features.

As an architectural adjustment to stop overfitting, we removed the Flatten() and replaced it with GlobalAveragePooling2d() as well as removing one fully connected later, this has been justified fully above. The effect was to reduce the number of parameters and prevent memorisation of the data.

L2 regularisation was also added to all convolutional and dense layers to prevent overfitting by penalising large weight values. By reducing the magnitude of weights, L2 regularisation promotes smoother decision boundaries and lowers the model's variance, improving generalisation performance on unseen data.

We further fine-tuned other regularisation hyperparameters. Choosing a dropout rate of 0.3 after the dense layer, reducing it from 0.5 we used in our initial runs following the removal of the Flatten() and one dense layer. We experimented with both the he_normal and the Keras standard glorot_uniform weight initialisers to minimal differences in performance.

In conclusion the alterations we made with hyperparameters, architectural adjustments and Data Augmentation greatly reduced overfitting as can be seen from the results section above.

9. Statement of Work:

Aron Calvert (22370374)

Me and Leo worked closely throughout the project. We co-planned the overall approach in regular meet-ups from start to finish. Together we selected the Intel Natural Scenes dataset and agreed on a VGG-like CNN with stacked 3×3 conv layers and MaxPooling in each block. I implemented the data exploration and visuals (Mean RGB/brightness charts, per-class mean images) and set up the Keras ImageDataGenerator pipeline with rescaling and augmentation (rotations, zoom, horizontal flips). I built the original model blocks with ReLU, and Dropout that suffered from overfitting, I took a lead on introducing the data augmentation and testing its effects. I ran and logged the 50-epoch training with Adam, added ReduceLROnPlateau, and produced the accuracy/loss plots, confusion matrix and classification report used in Results and Evaluation. I drafted large parts of Sections 1, 2(b), 6–8 and helped ensure the reported metrics and graphs align with the final model we present.

Leo O’Shea (22342761)

We planned together throughout, and together we agreed on the VGG-style design. I documented why we did not use k-fold CV for this dataset. I built the training setup (Categorical Cross-Entropy with label smoothing, Adam with an initial 1e-3 LR) and ran comparative experiments, including a deeper variant with an extra conv block (observing minimal gains vs much longer training). I took a lead on introducing new elements to reduce overfitting. We agreed together to implement GlobalAveragePooling() and reduce the number of dense layers. I led on introducing and justifying batch normalisation and L2 regularisation and updated the DropOut rate. I experimented with weight initialization. I created the architecture diagram, wrote Sections 2(a), 3–5 and 7, and edited Sections 6 and 8 so the figures (accuracy/loss curves, confusion matrix, class report) accurately reflect the final 50-epoch run. I also checked the final report and aided with citations and finalising its formatting.

10. Use of Generative AI: a listing of ALL prompts, one line summarising the subsequent response, and another line of text to explain how it was used in the submission.

Prompt: Give methods for visualising an image based data set beyond simple RGB approach.
Response: Suggested, brightness distributions, the mean image as used and also other later options such as confusion matrix etc
Usage: We implemented the mean image per class visualisation

Prompt: Can you find a list of sources on the application of Batch Normalisation in VGG's
Response: Provided us with a list of sources that showed batch norm in VGG's
Usage: Read sources and used the above referenced source that shows Batch normalisation improves VGG's.

Prompt: Explain the differences between Flatten() and GlobalAveragePooling in a CNN.
Response: Explained the differences between the two
Usage: Used it as one information point while researching the usage of GlobalAveragePooling

Prompt: Why isn't this working

```
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
   monitor="val_acc", factor=0.5, patience=3, verbose=1
)
```

Response: Updated code showing the need to replace val_acc with val_accuracy
Usage: Debugged the code

11. Level of Difficulty: short paragraph outlining your opinion of the challenge posed using the following guide.

This project involved building a custom VGG-like convolutional neural network from the ground up rather than using a pre-trained model. The network follows the basic VGG designed of stacked convolutional layers but is made better for the specific task by batch normalisation, L2 regularisation, dropout and the specific choices of optimiser, loss function and the adaptive learning rate. Getting the model to train well took a lot of time and experimentation with hyperparameters, depth and data augmentation. We managed to do well to effectively reduce overfitting and build an accurate model. We believe this was a moderate to high level of difficulty and worth 2 marks.