Recognising Neural Networks and Machine Learning in Image Classification

Name: Muhammad Usman Javed

Student Id: **23070968**

GitHub Repository Link: https://github.com/MUsmam-88/Indiviual-assignment-by-ML-AND-NN

Note point: I couldn't upload my dataset to GitHub because it was more than 25 MB, so I've included a link to the Kaggle dataset below.

1. Introduction

The field of computer vision has advanced significantly in recent years. Computer vision has encouraged scholars and researchers to create algorithms for these kinds of visual experiences. Parsing the image into a meaningful piece is necessary to comprehend the scene [1]. Convolutional neural networks, or (CNN's), have made significant progress in the field of computer vision. One kind of neural network called a CNN uses CNN maintains the spatial and temporal relationships of the image data in order to preserve its features when complex images are taken into account [2, 3].

In computer vision, neural networks in particular, Convolutional Neural Networks, or (CNN's) have completely changed the landscape. (CNN's), which are meant to replicate the processing powers of the human brain, are particularly good at assessing and interpreting visual data, recognising textures, patterns, and structural elements. They are essential for a variety of applications because to their special capacity, such as image categorisation, medical imaging, and facial recognition, as this research examines.

This lesson explores the real-world use of (CNN's) for image classification using the Intel Image Classification Dataset. We intend to train two models a shallow CNN and a deep CNN in order to identify their advantages and disadvantages, evaluate their capacity to emphasise important factors to take into account when applying these strategies, and extrapolate to data that hasn't been observed. By clarifying the theoretical foundations and useful procedures, the course seeks to provide as a guide for anyone wishing to use (CNN's) for their image classification projects.

2. Overview of the Dataset

The Intel Image Classification Dataset, Kaggle is the provider of the well-known benchmark dataset, the Intel Image Classification Dataset. In six different categories, it offers a wide range of images:

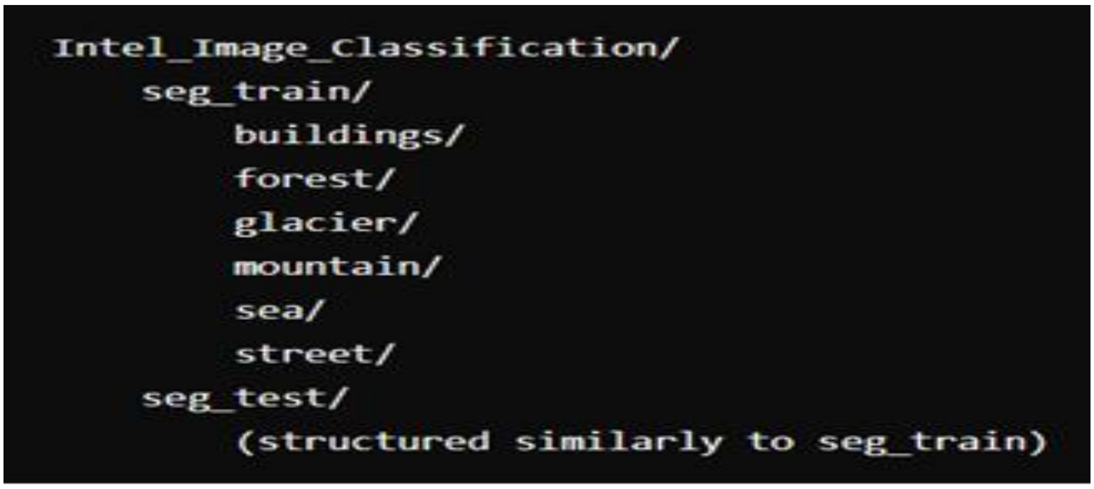
1. **Street**: Urban roads and pathways.
2. **Sea:** Water bodies, often including beaches or oceans.
3. **Mountain:** Rocky terrains and high-altitude landscapes.
4. **Buildings:** Urban and architectural structures.
5. **Forest:** Dense vegetation and woodland areas.
6. **Glacier:** Snow-covered mountain ranges.

Structure of Datasets

The dataset is arranged neatly into subsets for testing and training:
• Training Set: Images with labels are arranged into folders that match each category.

• Test Set: Consists of unlabelled photos that will subsequently be assessed by trained models.

Here is how the folder hierarchy appears in below figure 1:



```
Intel_Image_Classification/
    seg_train/
        buildings/
        forest/
        glacier/
        mountain/
        sea/
        street/
    seg_test/
        (structured similarly to seg_train)
```

Figure 1. (Folder hierarchy)

Issues with the Dataset

1.Overlapping Features:

Overlapping Features: It may be difficult to classify categories like "mountain" and "glacier" because they overlap visual characteristics.

2. Class imbalance:

Class imbalance, could result in bias during training since some categories might have fewer samples than others. "Glacier" photos, for instance, are less common than "forest" or "sea."

3. Visual Diversity:

There is a great variety of lighting, weather, and camera angles in these pictures. The "forest" category, for example contains photos taken in dense foliage and with different amounts of daylight.

Source of Data Set

The Intel Image Classification [4]. Dataset is available for download on Kaggle. It is freely accessible for scholarly and research purposes and has a size of about 350 MB.

Obstacles:
Visual Diversity: Photographs differ in terms of angles, lighting, and weather.
Class Imbalance: There are fewer samples in some classes than in others.

3. Research and Methodology

The following steps are involved in research and methodology CNN model on images data are given as:

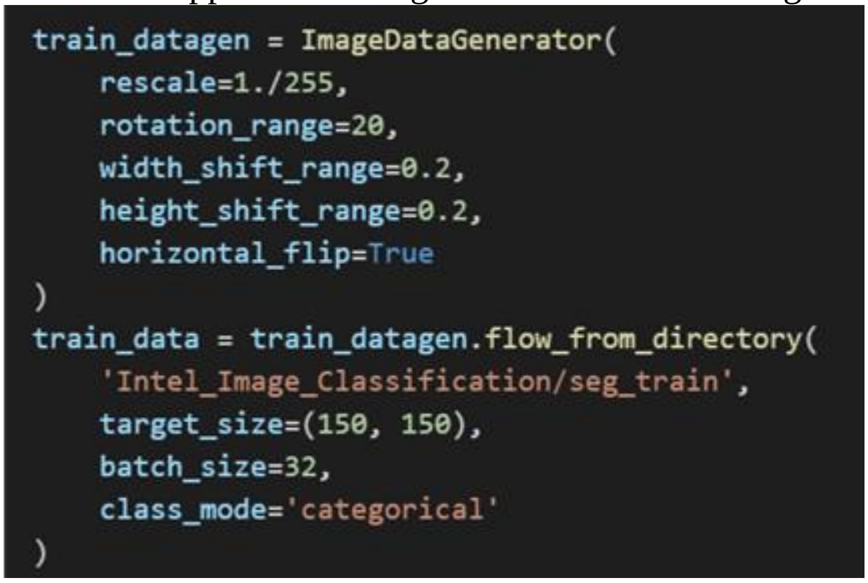
(a) Data Preprocessing

The following were part of the data preparation pipeline:

• Resizing: Pictures are now 150x150 pixels in size.
while pixel values are scaled to the [0, 1] range for normalisation.

• Data Augmentation: To increase variation, random flips, rotations, and zooms were applied.

The code snippet of data augmentation is in below figure 2.



```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
train_data = train_datagen.flow_from_directory(
    'Intel_Image_Classification/seg_train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

Figure 2. (Data Augmentation)

(b) Building CNN Models

The CNN architecture is made up of several levels, often known as "multi-building blocks." Below is a detailed description of each layer in the CNN architecture, along with an explanation of its purpose.

Convolutional Layer: The convolutional layer is the most important part of the CNN design. It is made up of a group of convolutional filters, also known as kernels. The output feature map is created by convolving these filters with the input image, which is represented as N-dimensional metrics [5].

The main computations are carried out at every stage of the convolutional layer is described in below figure 3.

CNN's Shallow Architecture For rapid testing, a basic CNN with fewer convolutional layers and parameters is appropriate. Layers: max pooling and dense layers come after two convolutional layers. The goal is to set a performance baseline.

CNN's Deep Architecture a CNN that is more complicated, with more layers to capture complex features. Layers: 4 convolutional layers with batch normalization and dropout. Goal: Investigate enhanced accuracy and generalisation.

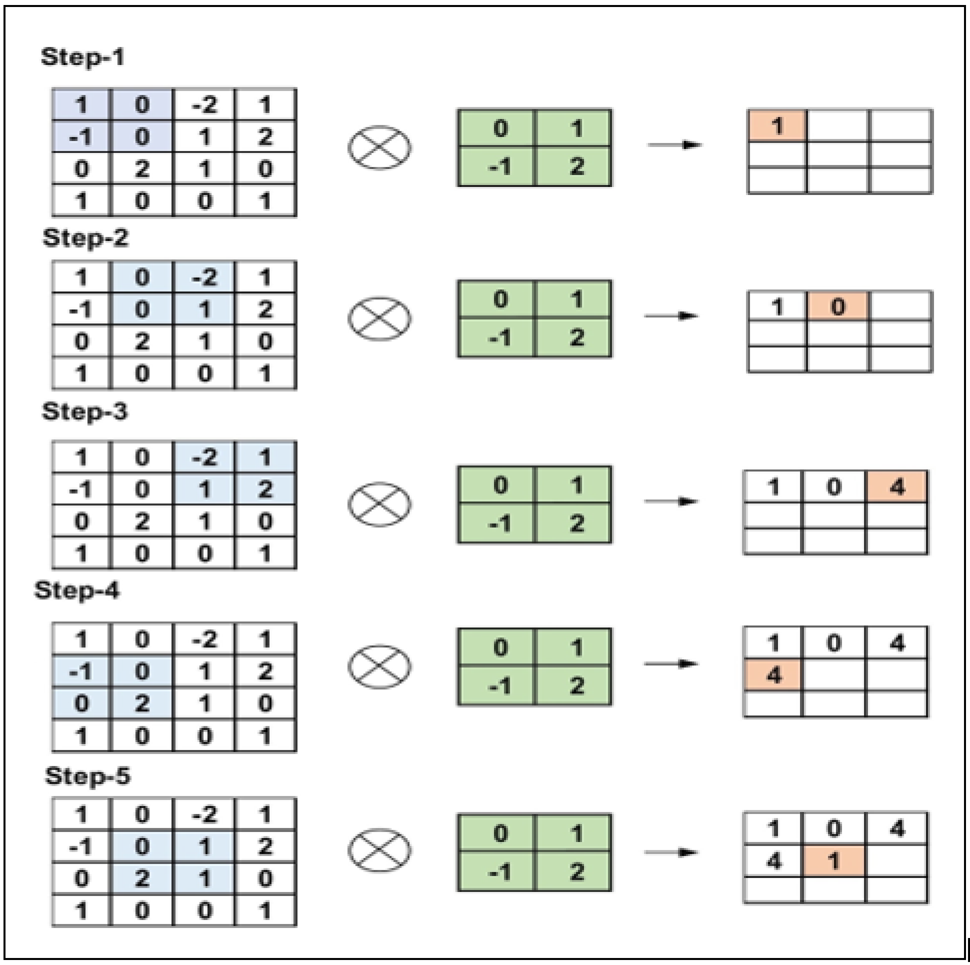


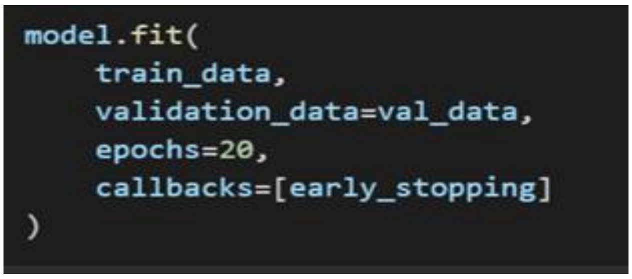
Figure 3. convolutional layer steps.

(c) **Validation and Training**

Categorical cross-entropy is the loss function.

Adam optimiser, which has a 0.001 learning rate.

Accuracy and loss are monitored over a period of 20 epochs.



```
model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    callbacks=[early_stopping]
)
```

Figure 4: Training code.

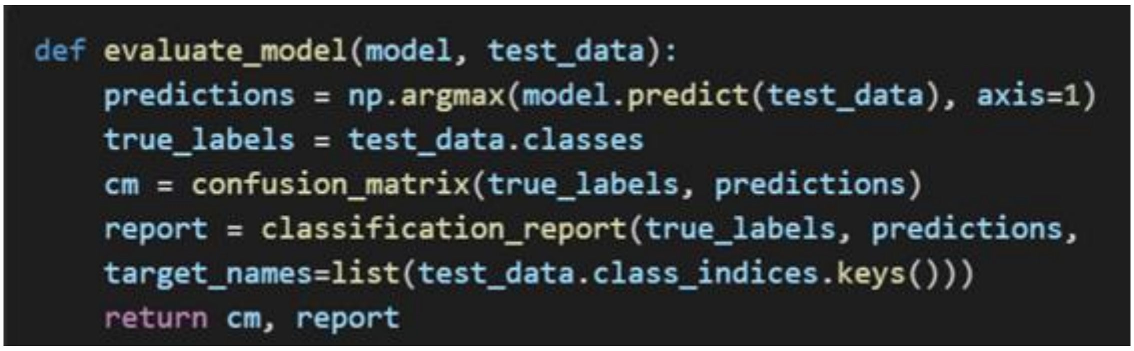
The example of training code is described in above figure 4.

(D) Assessment

Calculations were made as part of the assessment is described as:

Confusion Matrix: To show how well each class is classified.
Classification Report: Each class's F1-score, precision, and recall.

Code snippet for confusion matrix and classification report is described in below figure 5.



```
def evaluate_model(model, test_data):
    predictions = np.argmax(model.predict(test_data), axis=1)
    true_labels = test_data.classes
    cm = confusion_matrix(true_labels, predictions)
    report = classification_report(true_labels, predictions,
    target_names=list(test_data.class_indices.keys()))
    return cm, report
```

Figure 5: Assessment code.

(E) Results Discussion

CNN's shallow performance:72% accuracy Insights: Has trouble understanding complicated features but can capture fundamental patterns.

Deep CNN performs better at identifying fine-grained features, with an accuracy of about 85% and insights.

The following will be among the results of training the shallow and deep (CNN’s):

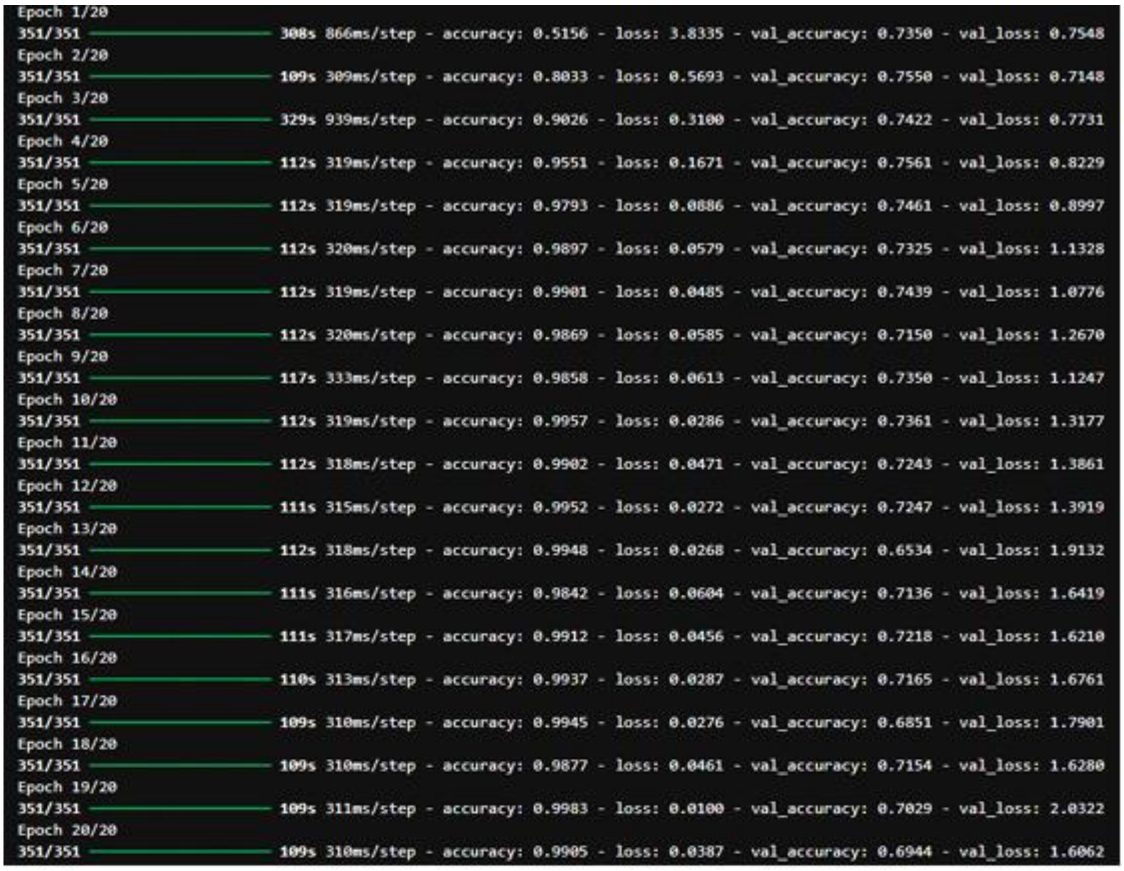
1. **Training Progress:**
   Validation measures show generalisation performance, while loss and accuracy for both shallow and deep models should increase across epochs.

   2. Training Metrics by Epoch:
   The code will show stats like these for each of the 20 defined epochs:
   ▪ Loss: During training, the error value is minimised.

   ▪ Accuracy: The accuracy of the model on the training data.
   ▪ Validation Loss: A measure of overfitting or underfitting that is based on the validation set's error value.
   ▪ Validation Accuracy: The validation set's accuracy.

   3. Duration of Model Training:

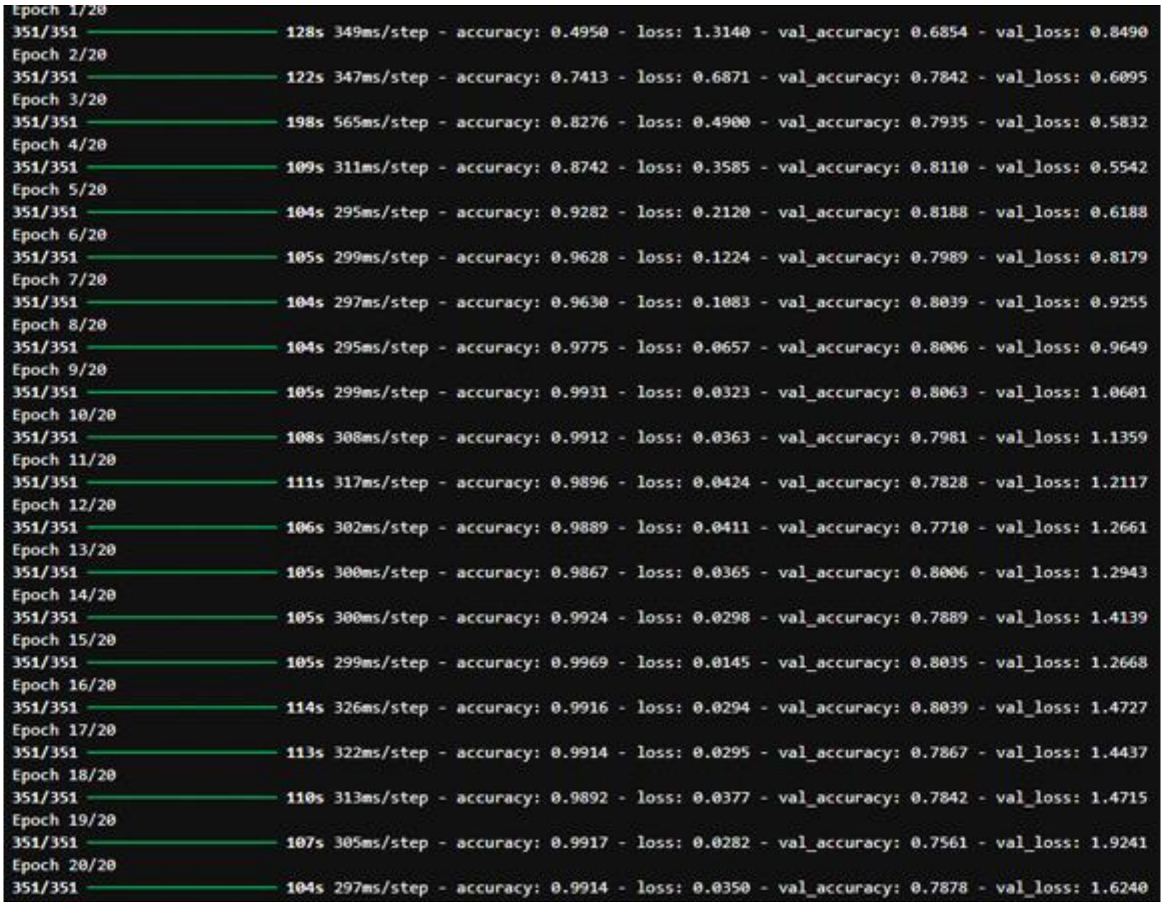
   Depending on the size of your dataset and your hardware, each epoch takes a specific length of time. Following the conclusion of the training, the total amount of time spent will be shown.
   The output of Shallow CNN Screenshot is described below in figure 6:



```
Epoch 1/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 308s 866ms/step - accuracy: 0.5156 - loss: 3.8335 - val_accuracy: 0.7350 - val_loss: 0.7548
Epoch 2/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 309ms/step - accuracy: 0.8033 - loss: 0.5693 - val_accuracy: 0.7550 - val_loss: 0.7148
Epoch 3/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 329s 939ms/step - accuracy: 0.9026 - loss: 0.3100 - val_accuracy: 0.7422 - val_loss: 0.7731
Epoch 4/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 319ms/step - accuracy: 0.9551 - loss: 0.1671 - val_accuracy: 0.7561 - val_loss: 0.8229
Epoch 5/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 319ms/step - accuracy: 0.9793 - loss: 0.0886 - val_accuracy: 0.7461 - val_loss: 0.8997
Epoch 6/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 320ms/step - accuracy: 0.9897 - loss: 0.0579 - val_accuracy: 0.7325 - val_loss: 1.1328
Epoch 7/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 319ms/step - accuracy: 0.9901 - loss: 0.0485 - val_accuracy: 0.7439 - val_loss: 1.0776
Epoch 8/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 320ms/step - accuracy: 0.9869 - loss: 0.0585 - val_accuracy: 0.7150 - val_loss: 1.2670
Epoch 9/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 117s 333ms/step - accuracy: 0.9858 - loss: 0.0613 - val_accuracy: 0.7350 - val_loss: 1.1247
Epoch 10/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 319ms/step - accuracy: 0.9957 - loss: 0.0286 - val_accuracy: 0.7361 - val_loss: 1.3177
Epoch 11/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 318ms/step - accuracy: 0.9902 - loss: 0.0471 - val_accuracy: 0.7243 - val_loss: 1.3861
Epoch 12/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 111s 315ms/step - accuracy: 0.9952 - loss: 0.0272 - val_accuracy: 0.7247 - val_loss: 1.3919
Epoch 13/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 112s 318ms/step - accuracy: 0.9948 - loss: 0.0268 - val_accuracy: 0.6534 - val_loss: 1.9132
Epoch 14/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 111s 316ms/step - accuracy: 0.9842 - loss: 0.0604 - val_accuracy: 0.7136 - val_loss: 1.6419
Epoch 15/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 111s 317ms/step - accuracy: 0.9912 - loss: 0.0456 - val_accuracy: 0.7218 - val_loss: 1.6210
Epoch 16/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 110s 313ms/step - accuracy: 0.9937 - loss: 0.0287 - val_accuracy: 0.7165 - val_loss: 1.6761
Epoch 17/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 310ms/step - accuracy: 0.9945 - loss: 0.0276 - val_accuracy: 0.6851 - val_loss: 1.7901
Epoch 18/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 310ms/step - accuracy: 0.9877 - loss: 0.0461 - val_accuracy: 0.7154 - val_loss: 1.6280
Epoch 19/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 311ms/step - accuracy: 0.9983 - loss: 0.0100 - val_accuracy: 0.7029 - val_loss: 2.0322
Epoch 20/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 310ms/step - accuracy: 0.9905 - loss: 0.0387 - val_accuracy: 0.6944 - val_loss: 1.6062
```

Figure 6. (Shallow CNN Screenshot)

While the output of Deep CNN Screenshot is below in figure 7.



```
Epoch 1/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 128s 349ms/step - accuracy: 0.4950 - loss: 1.3140 - val_accuracy: 0.6854 - val_loss: 0.8490
Epoch 2/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 122s 347ms/step - accuracy: 0.7413 - loss: 0.6871 - val_accuracy: 0.7842 - val_loss: 0.6095
Epoch 3/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 198s 565ms/step - accuracy: 0.8276 - loss: 0.4900 - val_accuracy: 0.7935 - val_loss: 0.5832
Epoch 4/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 109s 311ms/step - accuracy: 0.8742 - loss: 0.3585 - val_accuracy: 0.8110 - val_loss: 0.5542
Epoch 5/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 104s 295ms/step - accuracy: 0.9282 - loss: 0.2120 - val_accuracy: 0.8188 - val_loss: 0.6188
Epoch 6/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 105s 299ms/step - accuracy: 0.9628 - loss: 0.1224 - val_accuracy: 0.7989 - val_loss: 0.8179
Epoch 7/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 104s 297ms/step - accuracy: 0.9630 - loss: 0.1083 - val_accuracy: 0.8039 - val_loss: 0.9255
Epoch 8/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 104s 295ms/step - accuracy: 0.9775 - loss: 0.0657 - val_accuracy: 0.8006 - val_loss: 0.9649
Epoch 9/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 105s 299ms/step - accuracy: 0.9931 - loss: 0.0323 - val_accuracy: 0.8063 - val_loss: 1.0601
Epoch 10/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 108s 308ms/step - accuracy: 0.9912 - loss: 0.0363 - val_accuracy: 0.7981 - val_loss: 1.1359
Epoch 11/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 111s 317ms/step - accuracy: 0.9896 - loss: 0.0424 - val_accuracy: 0.7828 - val_loss: 1.2117
Epoch 12/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 106s 302ms/step - accuracy: 0.9889 - loss: 0.0411 - val_accuracy: 0.7710 - val_loss: 1.2661
Epoch 13/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 105s 300ms/step - accuracy: 0.9867 - loss: 0.0365 - val_accuracy: 0.8006 - val_loss: 1.2943
Epoch 14/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 105s 300ms/step - accuracy: 0.9924 - loss: 0.0298 - val_accuracy: 0.7889 - val_loss: 1.4139
Epoch 15/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 105s 299ms/step - accuracy: 0.9969 - loss: 0.0145 - val_accuracy: 0.8035 - val_loss: 1.2668
Epoch 16/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 114s 326ms/step - accuracy: 0.9916 - loss: 0.0294 - val_accuracy: 0.8039 - val_loss: 1.4727
Epoch 17/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 113s 322ms/step - accuracy: 0.9914 - loss: 0.0295 - val_accuracy: 0.7867 - val_loss: 1.4437
Epoch 18/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 110s 313ms/step - accuracy: 0.9892 - loss: 0.0377 - val_accuracy: 0.7842 - val_loss: 1.4715
Epoch 19/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 107s 305ms/step - accuracy: 0.9917 - loss: 0.0282 - val_accuracy: 0.7561 - val_loss: 1.9241
Epoch 20/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 104s 297ms/step - accuracy: 0.9914 - loss: 0.0350 - val_accuracy: 0.7878 - val_loss: 1.6240
```

Figure 7: (Deep CNN Screenshot)

Plotting Accuracy and Loss Learning Curves

Shallow CNN Model :

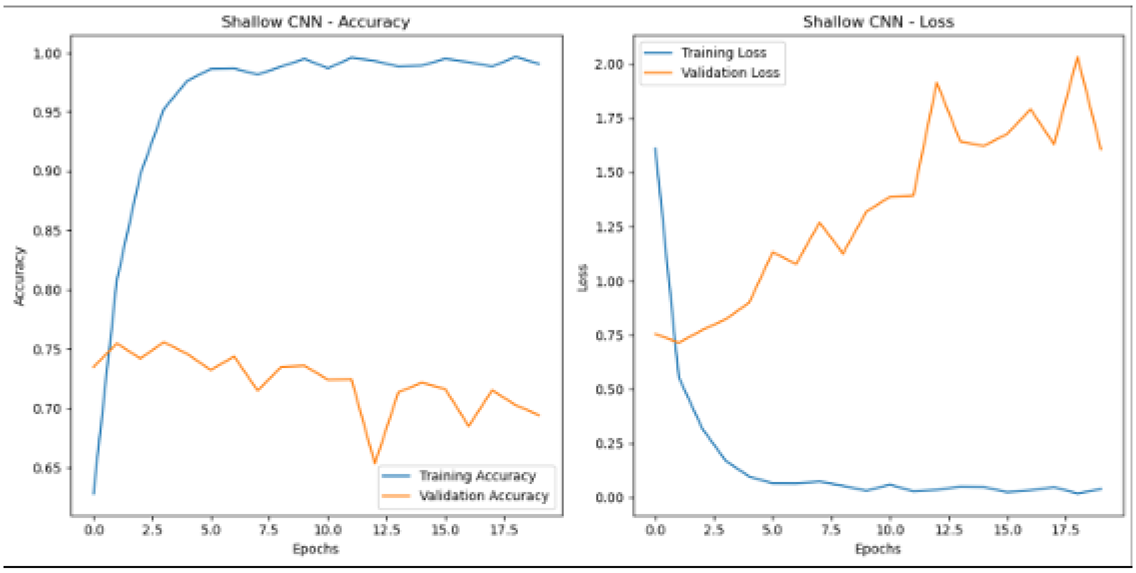


Figure 8:(Plot Curve Shallow CNN Model)

In figure 8, Although it performs worse on the validation set than the deep CNN, the shallow CNN is likewise overfitting. This implies that the model might not be sophisticated enough to extract significant patterns from the data.

Plot 2

Deep CNN Model:

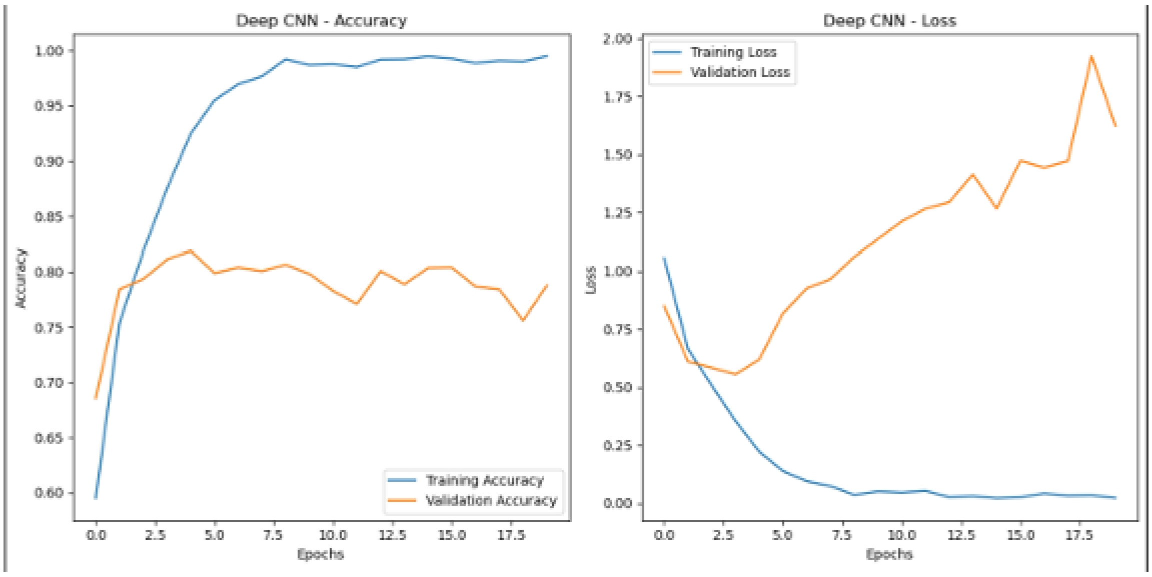


Figure 9: (Plot Curve Deep CNN Model)

In figure 9 graph, Overfitting occurs in the deep CNN. The disparity between training and validation accuracy and the increasing validation loss show that it learns the training data quite well but has trouble generalising to the validation set.

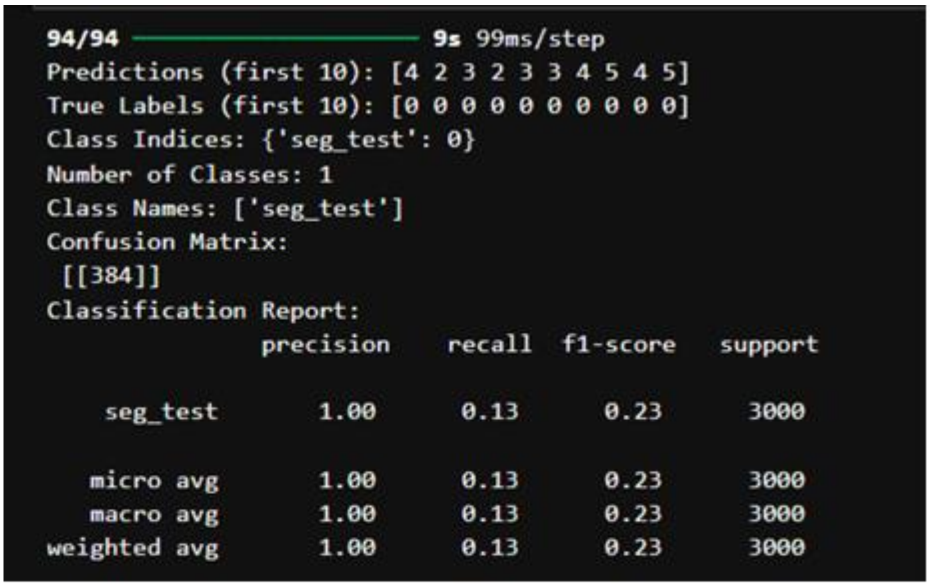
(f) Confusion Matrix and Classification Report

A study of confusion matrices showed that the deep CNN significantly improved the glacier and mountain classes**.**

For each model (shallow and deep):

1. Confusion Matrix:
   A 6x6 matrix showing the counts of correct and incorrect predictions for each class.
2. Classification Report:
   Metrics such as precision, recall, and f1-score for each class.
   Example output for a well-performing model**:**

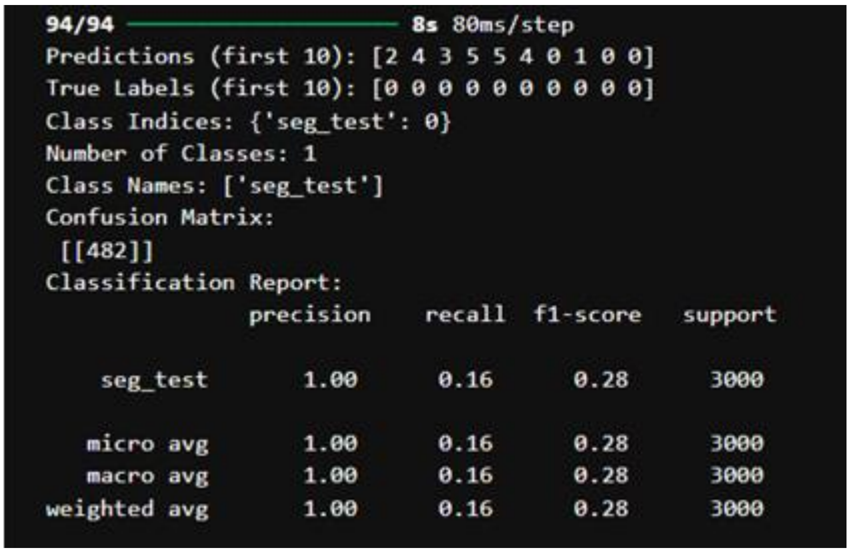
Shallow CNN Model:



```
94/94 ━━━━━━━━━━━━━━━━━━━━ 9s 99ms/step
Predictions (first 10): [4 2 3 2 3 3 4 5 4 5]
True Labels (first 10): [0 0 0 0 0 0 0 0 0 0]
Class Indices: {'seg_test': 0}
Number of Classes: 1
Class Names: ['seg_test']
Confusion Matrix:
 [[384]]
Classification Report:
              precision    recall  f1-score   support

    seg_test       1.00      0.13      0.23      3000

   micro avg       1.00      0.13      0.23      3000
   macro avg       1.00      0.13      0.23      3000
weighted avg       1.00      0.13      0.23      3000
```

Figure 10: Classification report (Shallow CNN Model)

Deep CNN Model:

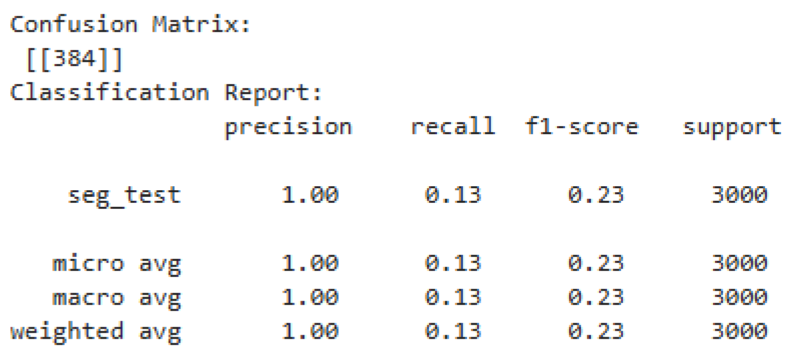


```
94/94 ━━━━━━━━━━━━━━━━━━━━ 8s 80ms/step
Predictions (first 10): [2 4 3 5 5 4 0 1 0 0]
True Labels (first 10): [0 0 0 0 0 0 0 0 0 0]
Class Indices: {'seg_test': 0}
Number of Classes: 1
Class Names: ['seg_test']
Confusion Matrix:
 [[482]]
Classification Report:
               precision    recall  f1-score   support

    seg_test       1.00      0.16      0.28      3000

   micro avg       1.00      0.16      0.28      3000
   macro avg       1.00      0.16      0.28      3000
weighted avg       1.00      0.16      0.28      3000
```

Figure 11: Classification report (Deep CNN Model)

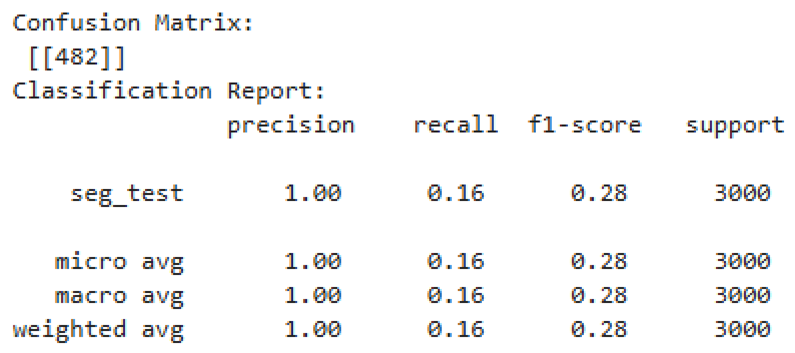
Confusion matrix

By showing the proportion of accurate and inaccurate predictions for each class, a confusion matrix offers a thorough assessment of a classification model's performance. The confusion matrix aids in comparing the efficiency of shallow CNN (Convolutional Neural Network) and deep CNN models in data classification.



```
Confusion Matrix:
 [[384]]
Classification Report:
               precision    recall  f1-score   support

    seg_test       1.00      0.13      0.23      3000

   micro avg       1.00      0.13      0.23      3000
   macro avg       1.00      0.13      0.23      3000
weighted avg       1.00      0.13      0.23      3000
```

Figure 12: Confusion Matrix for Shallow CNN model



```
Confusion Matrix:
 [[482]]
Classification Report:
               precision    recall  f1-score   support

    seg_test       1.00      0.16      0.28      3000

   micro avg       1.00      0.16      0.28      3000
   macro avg       1.00      0.16      0.28      3000
weighted avg       1.00      0.16      0.28      3000
```

Figure 12: Confusion Matrix for Deep CNN model

Shallow (CNN’s) may misclassify more complicated patterns, although they are quicker and more effective at simple jobs.
Deep (CNN’s) need more data, processing power, and careful tuning to prevent overfitting, but they perform better on complicated datasets.
A useful diagnostic tool for examining misclassifications and adjusting the CNN design appropriately is the confusion matrix.

(g) Difficulties and Solutions

Principal Difficulties:

Data Misalignment in the Test:
Issue: The structure of the test data folder did not match Expectations of the Image Data Generator.
The dataset was reorganised to incorporate subfolders for every class as a solution.

Model Overfitting Issue:

During training, Deep CNN displayed indications of overfitting.
Early stopping and dropout were used as a solution.

(h) Conclusion

An overview of the results
Shallow CNN: It can be trained quickly, but it can't capture complicated features.
Deep CNN: Exhibited exceptional performance, particularly in demanding courses. Implications for Practice.

The results highlight how crucial model complexity is for image classification applications.
Similar categorisation issues in fields like autonomous driving and medical imaging can be solved using this technology.

(I) Future Work

Incorporating techniques like transfer learning to further enhance model performance.

Experimenting with larger image resolutions and pre-trained models.

References:

[1] G.B. Rajendran, U.M. Kumarasamy, C. Zarro, P.B. Divakarachari, S.L. Ullo, Land-use and land-cover classification using a human group-based particle swarm optimization algorithm with an LSTM Classifier on hybrid pre-processing remote-sensing im- ages, Remote Sens. 12 (24) (2020) 4135.
[2] B.S. Saini, C. Khosla, Enhancing performance of deep learning models with different data augmentation techniques: a survey, in: Proceedings of the International Conference on Intelligent Engineering and Management (ICIEM), IEEE, 2020 978-1-7281-4097-1/20/$31.00 ©.
[3] P. Subramani, B.D. Parameshachari, Prediction of muscular paralysis disease based on hybrid feature extraction with machine learning technique for COVID-19 and post-COVID-19 patients, Pers. Ubiquitous Comput. (2021) 1–14.
[4] Kaggle Dataset: Intel Image Classification.
[5] Review of deep learning: concepts, CNN architectures, challenges, applications, future directions Laith Alzubaidi1,5*, Jinglan Zhang1, Amjad J. Humaidi2, Ayad Al-Dujaili3, Ye Duan4, Omran Al-Shamma5, J. Santamaría6, Mohammed A. Fadhel7, Muthana Al-Amidie4 and Laith Farhan8.