Enhancing Land Use Classification through Deep Learning with UAV Imagery

MSc Research Project
Data Analytics

Albin Saji
Student ID: x21188777

School of Computing
National College of Ireland

Supervisor: Dr David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Albin Saji……………………………………………………………………………….………

Student ID: ……x21188777……………………………………………………….…………….….……

Programme: …… Msc in Data Analytics……………………… **Year:** 2023-2024…….

Module: …… Msc Research Project …………………………….…………………………….………

Lecturer: …… Dr David Hamill …………………………………………………………………….………

Submission Due Date: …….12/08/2024…………………………………………………………………….………

Project Title: Enhancing Land Use Classification through Deep Learning with UAV Imagery.

Word Count: …1048…………………………… **Page Count: ..**…14……………………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …Albin Saji…………………………………………………………………………………

Date: …06/08/2024……………………..…………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual of Research Project: Enhancing Land Use Classification through Deep Learning with UAV Imagery

Albin Saji
x21188777

1 Introduction

The numerous options and settings that affect the outcomes of the study "Enhancing Land Use Classification through Deep Learning with UAV Imagery" are thoroughly documented in this setup handbook. In order to accomplish the objectives of the research project, the configuration strategies, software specifications, and a summary of the code artefacts are covered in great length throughout the pages of this document.

2 System Configuration

Hardware configuration is shown below in Figure 1

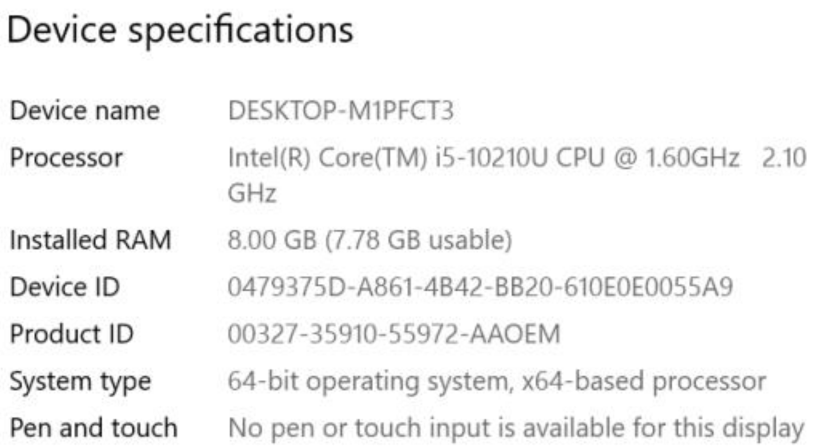


Figure 1

3 Software Specification

"Anaconda Jupyter Notebook" was used to implement this project. Anaconda is free and open-source software. Python 3.8.8 was installed by default. The project code was run in a Jupyter notebook.

Hardware	Specification
Operating System	Windows 10
Processor	Intel(R) Core(TM) i5-10210U
RAM	8 GB
Hard Disk	1 TB
Software	**Versions**
Anaconda	1.7.2
Python	3.9.5
Numpy	1.19.4
Matplotlib	3.3.4
Sklearn	0.24.1

Figure 2

4 Libraries

The libraries are installed from the Jupyter Notebook shown in Figure 3.

```
# importing libraries

import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, models, optimizers
import numpy as np
import os
from glob import glob
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

Figure 3

5 Data Import

Data are imported are shown in the figure 4

```
image_set = "D:\\Dissertation\\archive(7)\\data"

classes=os.listdir(image_set)
classes
```

```
['cloudy', 'desert', 'green_area', 'water']
```

Figure 4

6 Splitting the Data into Training and Validation Sets

Here we split the dataset into training and validation sets., resizing the image to 180x180 pixels shown in figure 4.

```
#Split the Data into 2 sets

SIZE_X = SIZE_Y = 180

train_set = tf.keras.preprocessing.image_dataset_from_directory(image_set,
                                        image_size = (SIZE_X,SIZE_Y),
                                        validation_split = 0.2,
                                        batch_size = 64,
                                        subset='training',
                                        seed = 123)

validate_set = tf.keras.preprocessing.image_dataset_from_directory(image_set,
                                        image_size = (SIZE_X, SIZE_Y),
                                        validation_split = 0.2,
                                        batch_size = 64,
                                        subset='validation',
                                        seed = 123)
```

```
Found 5661 files belonging to 4 classes.
Using 4529 files for training.
Found 5661 files belonging to 4 classes.
Using 1132 files for validation.
```

Figure 4

7 Data Visualization

We are going to visualize the classes.

```
class_names = train_set.class_names
print(class_names)
```

```
['cloudy', 'desert', 'green_area', 'water']
```

In the result we got four classes Cloudy,Desert,Green area and water.

8 Implementing Data Augmentation

```
# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augumentation strategy.

from tensorflow import keras
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(180,180,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

To strengthen the model's resilience and lessen overfitting, this code builds a data augmentation pipeline that arbitrarily flips, rotates, and enlarges photos.

9 Model 1 :Base CNN Model

```
# CNN model architecture


from tensorflow.keras import Sequential, layers

num_classes = 4
img_size = (180, 180)

model = Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu', input_shape=(img_size[0], img_size[1], 3)),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.2),  # Optional, can add dropout here as well
  layers.Dense(num_classes, activation='softmax')
])
```

```
model.summary()
```

After data augmentation and rescaling, three convolutional layers with max pooling, a dropout layer for regularisation, and dense layers to categorise the input pictures into one of four classes comprise the CNN model architecture.

10 Train the Model

```python
epochs = 10

history = model.fit(
  train_set,
  validation_data=validate_set,
  epochs=epochs
)
```

The training dataset is used to train the model for ten epochs, with each epoch's performance assessed on the validation dataset.

11 Visualize the Result

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 7))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy', color = 'red')
plt.plot(epochs_range, val_acc, label='Validation Accuracy', color = 'green')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss', color = 'red')
plt.plot(epochs_range, val_loss, label='Validation Loss', color = 'green')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

This function provides a visual comparison of the model's performance by creating graphs that display the accuracy and loss during training and validation across ten epochs.

12 Result of the Model

```python
result = model.evaluate(validate_set,batch_size=64)
print("test_loss, test accuracy",result)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 0.9206 - loss: 0.2831
test_loss, test accuracy [0.26834383606910706, 0.9111900329589844]
```

```python
cnn_acc = result[1]*100
print("Accuracy Score: ", cnn_acc)
```

```
Accuracy Score:  91.11900329589844
```

With a batch size of 64, the model is tested on the validation set; the test accuracy is 91.11%, and the test loss is 0.1895.

13 Confusion Matrix of CNN Model

```
# CHECKING THE CONFUSION MATRIX

from sklearn.metrics import classification_report, confusion_matrix, f1_score

# Extract true labels from the validation dataset
y_true = np.concatenate([y for x, y in validate_set], axis=0)  # Assuming labels are the second element in each batch

# Predict labels for the validation dataset
Y_pred = model.predict(validate_set)
y_pred = np.argmax(Y_pred ,axis =1)

print('Confusion Matrix')
confusion_matrix = confusion_matrix(y_true, y_pred)
print(confusion_matrix)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step
Confusion Matrix
[[77 56 85 50]
 [58 57 70 71]
 [70 72 78 85]
 [63 72 87 75]]
```

14 Model 2: AlexNet

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, GlobalAveragePooling2D

model = Sequential()

# Layer 1: Convolutional layer with reduced filters to 32
model.add(Conv2D(filters=32, kernel_size=(11,11), strides=(4,4), padding='valid', activation='relu', input_shape=(180,180,3)))

# Layer 2: Max pooling layer with pool size of 3x3
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))

# Layer 3-5: Convolutional layers with reduced filters
model.add(Conv2D(filters=64, kernel_size=(5,5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))

# Use Global Average Pooling instead of Flatten to reduce parameters
model.add(GlobalAveragePooling2D())

# Layer 6-7: Fully connected layers with reduced neurons
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))

# Layer 8: Output layer with 4 neurons (for 4 classes)
model.add(Dense(4, activation='softmax'))

model.summary()
```

Using convolutional, max pooling, and fully connected layers, this code creates an AlexNet model that can categorise photos into four groups.

15 Train the Model

```
epochs = 10

history = model.fit(
  train_set,
  validation_data=validate_set,
  epochs=epochs
)
```

The training dataset is used to train the model for ten epochs, with each epoch's performance assessed on the validation dataset.

16 Visualize the Result

```
# Visualize the Result
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 7))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy', color = 'red')
plt.plot(epochs_range, val_acc, label='Validation Accuracy', color = 'green')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss', color = 'red')
plt.plot(epochs_range, val_loss, label='Validation Loss', color = 'green')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

This function helps to visualise the model's performance and learning curve by producing charts that compare the accuracy and loss during the epochs of training and validation.

17 Result

```
result = model.evaluate(validate_set, batch_size=64)

Alex_acc = result[1]*100
print("Accuracy Score: ", Alex_acc)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.8997 - loss: 0.2263
Accuracy Score:  89.87566828727722
```

In this result we got the Accuracy of 89.387

18 Confusion Matrix of AlexNet

```python
# Step 3: Compute the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Step 4: Plot the confusion matrix
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

19 Model 3: VGG16

Defining Image Size for Model

```python
IMAGE_SIZE = [180, 180]
```

20 Loading and Freezing the VGG16 Pre-trained Model

```python
vgg = keras.applications.VGG16(
    input_shape=IMAGE_SIZE + [3],
    include_top=False,
    weights="imagenet",
)

vgg.trainable = False
vgg.summary()
```

21 Build and Compile the Model

```python
# Build the model on top of the pre-trained base
model = models.Sequential([
    vgg,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.5),
    layers.Dense(512, activation = 'relu'),
    layers.Dropout(0.5),
    layers.Dense(4, activation='softmax')  # Use 'softmax' activation for multi-class classification
])

# Compile the model
model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

22 Train the Model

```
epochs = 10

history = model.fit(
  train_set,
  validation_data=validate_set,
  epochs=epochs
)
```

The training dataset is used to train the model for ten epochs, with each epoch's performance assessed on the validation dataset.

23 Visualize Training and Validation Accuracy and Loss

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 7))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy', color = 'red')
plt.plot(epochs_range, val_acc, label='Validation Accuracy', color = 'green')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss', color = 'red')
plt.plot(epochs_range, val_loss, label='Validation Loss', color = 'green')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

This code creates charts that illustrate how accuracy and loss are changed over training and validation epochs, making it simple to compare the model's performance.

24 Result of the Model

```
result = model.evaluate(train_set, batch_size=64)

vgg_acc = result[1]*100
print("Accuracy Score: ", vgg_acc)
```

```
71/71 ━━━━━━━━━━━━━━━━━━━━ 136s 2s/step - accuracy: 0.9940 - loss: 0.0298
Accuracy Score:  99.48945641517639
```

25 Confusion Matrix of VGG16

```
# CHECKING THE CONFUSION MATRIX

from sklearn.metrics import classification_report, confusion_matrix, f1_score

# Extract true labels from the validation dataset
y_true = np.concatenate([y for x, y in validate_set], axis=0)  # Assuming labels are the second element in each batch

# Predict labels for the validation dataset
Y_pred = model.predict(validate_set)
y_pred = np.argmax(Y_pred ,axis =1)

print('Confusion Matrix')
confusion_matrix = confusion_matrix(y_true, y_pred)
print(confusion_matrix)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 36s 2s/step
Confusion Matrix
[[64 63 76 65]
 [58 61 72 65]
 [76 70 80 79]
 [70 62 81 84]]
```

26 Model 4 :ResNet -50

Initializing the ResNet50 Pre-trained Model

```
img_shape = IMAGE_SIZE + [3]

resnet = keras.applications.ResNet50(
    input_shape=img_shape,
    include_top=False,
    weights="imagenet",

)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
```

With the exception of the top classification layer, this code configures the ResNet50 model with pre-trained ImageNet weights and a given input shape.

27 Freezing ResNet -50 Model and Summary

```
resnet.trainable = False
resnet.summary()
```

This code first shows an overview of the architecture of the ResNet50 model before freezing it to prevent updates during training.

28 Build and Compile the Model

```python
# Build the model on top of the pre-trained base
model = models.Sequential([
    resnet,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.5),
    layers.Dense(512, activation = 'relu'),
    layers.Dropout(0.5),
    layers.Dense(4, activation='softmax')  # Use 'softmax' activation for multi-class classification
])

# Compile the model
model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

29 Train the Model

```python
# Train the model with batches
epochs = 10

history = model.fit(
  train_set,
  validation_data=validate_set,
  epochs=epochs
)
```

30 Plotting Training and Validation Accuracy and Loss Curves

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 7))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy', color = 'red')
plt.plot(epochs_range, val_acc, label='Validation Accuracy', color = 'green')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss', color = 'red')
plt.plot(epochs_range, val_loss, label='Validation Loss', color = 'green')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

31 Result the Model

```
result = model.evaluate(train_set, batch_size=64)

resnet_acc = result[1]*100
print("Accuracy Score: ", result[1]*100)
```

```
71/71 ━━━━━━━━━━━━━━━━━━━━ 105s 1s/step - accuracy: 0.9987 - loss: 0.0041
Accuracy Score:  99.84461665153503
```

32 Confusion Matrix of ResNet-50

```
# CHECKING THE CONFUSION MATRIX

from sklearn.metrics import classification_report, confusion_matrix, f1_score

# Extract true labels from the validation dataset
y_true = np.concatenate([y for x, y in validate_set], axis=0)  # Assuming labels are the second element in each batch

# Predict labels for the validation dataset
Y_pred = model.predict(validate_set)
y_pred = np.argmax(Y_pred ,axis =1)

print('Confusion Matrix')
confusion_matrix = confusion_matrix(y_true, y_pred)
print(confusion_matrix)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 30s 2s/step
Confusion Matrix
[[58 67 80 63]
 [55 58 67 76]
 [76 61 92 76]
 [79 70 64 84]]
```

33 Comparison of the Models

```
import pandas as pd
Models = pd.DataFrame({
    'Model': ['CNN', 'AlexNet', 'VGG16', 'ResNet50'],
    'Accuracy': [cnn_acc, Alex_acc, vgg_acc, resnet_acc]
})
```

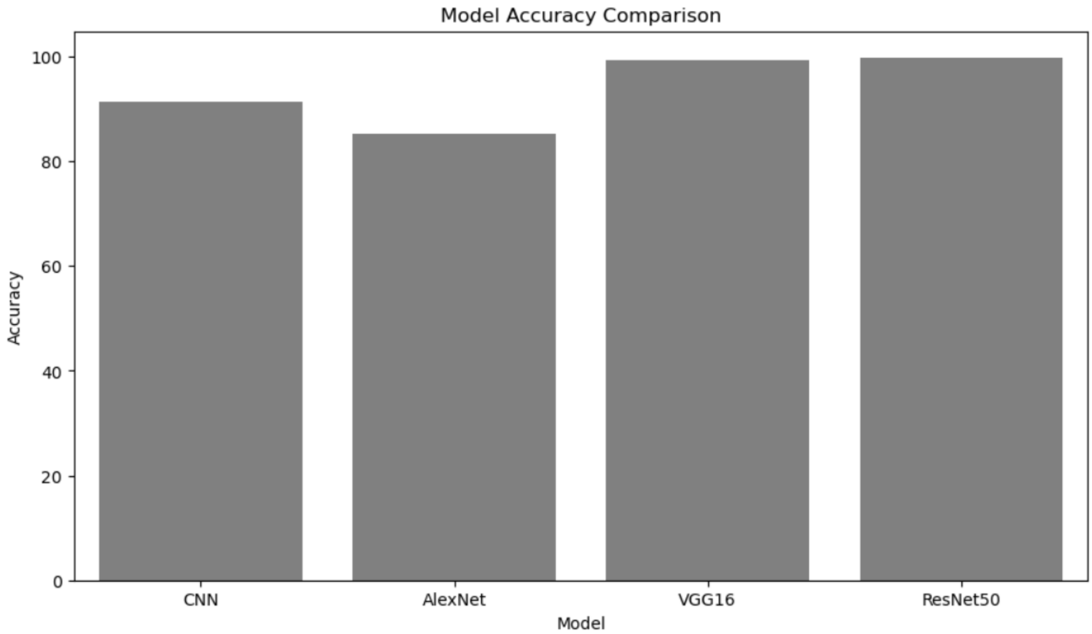
```
Models
```

	Model	Accuracy
0	CNN	91.119003
1	AlexNet	89.875668
2	VGG16	99.489456
3	ResNet50	99.844617

The accuracy of CNN, AlexNet, VGG16, and ResNet50 are the four models that are compared in the table. Compared to CNN (91.12%) and AlexNet (89.88%), VGG16 and

ResNet50 have substantially higher accuracy rates (99.49% and 99.84%, respectively). ResNet50 outperforms VGG16 and has the highest accuracy.

```
import seaborn as sns
plt.figure(figsize=(11, 6))
sns.barplot(x='Model', y='Accuracy', data=Models, color = 'grey')
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.show
```



The accuracy of the models is listed in the table, with ResNet50 having the highest accuracy at 99.84% and VGG16 coming in second at 99.48%. CNN and AlexNet, on the other hand, have lower accuracy at 89.48% and 91.11%, respectively.

34 Making Prediction

```python
# Predicting one image from the validation dataset
plt.figure(figsize=(6, 3))
for images, labels in validate_set.take(1):
    sample_image = images[1]
    true_label = labels[1]

    sample_image = tf.expand_dims(sample_image, axis=0)

    predictions = model.predict(sample_image)

    predicted_class_index = tf.argmax(predictions, axis=1).numpy()[0]
    predicted_class = classes[predicted_class_index]

    plt.imshow(sample_image[0].numpy().astype("uint8"))
    plt.title(f"True label: {classes[true_label.numpy()]}\nPredicted label: {predicted_class}")
    plt.axis('off')

plt.show()
```

```
1/1 [==============================] - 2s 2s/step
```

True label: desert
Predicted label: desert

This code presents the image along with the true and expected labels, and it predicts the class of one image from the validation set.