Level 2 Investigation

Name: Soichiro Tanabe

Setup

In [1]:
```
import tensorflow as tf
import numpy as np
import csv
import os
import json
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets, metrics, Seque
from tensorflow.keras.layers import *
from IPython.display import HTML, display
import matplotlib.pyplot as plt
%matplotlib inline
```

Data format and structure

The dataset contains just over 50,000 images, each showing a 60m × 60m region of a forest in Germany. Each image shows mainly one species of tree. There are three levels of label for each image: the English name (level 3), forest management class (level 2), and leaf type (level 1).

Treesat label levels

> Image detail from Ahlswede *et al.* (2023) figure 4.

The dataset has been split into 70% training, 10% validation, and 20% testing images.

The images are 304 × 304 pixels and encoded as four-channel PNG images. We are mis-using the PNG's alpha (transparency) channel to represent the near-IR light band in the original images.

The data resides in the `/datasets/treesat/` directory. All the images are in the `/datasets/treesat/images/` directory. There are three `.csv` files that specify which images are in which split of the dataset. There are three text files that list the different labels at each level.

In [2]:
```
!ls /datasets/treesat
```

```
images                  level_3_vocabulary.txt  validation_file_labels.csv
level_1_vocabulary.txt  test_file_labels.csv
level_2_vocabulary.txt  train_file_labels.csv
```

If we look at one of the `.csv` files, we can see the image name and labels at each level.

In [3]:
```
!head /datasets/treesat/train_file_labels.csv
```

Similarly, we can see the distinct labels at each level.

In [4]:
```
!cat /datasets/treesat/level_2_vocabulary.txt
```

```
larch
cleared
beech
oak
douglas fir
pine
fir
short-lived deciduous
long-lived deciduous
spruce
```

Defining constants

`label_level` is one of 1, 2, or 3, and is the level of labels used in this notebook.

In [5]:
```
data_dir = '/datasets/treesat'
label_level = 2
```

In [6]:
```
IMAGE_RESCALE = (302, 302)
input_shape = (IMAGE_RESCALE[0], IMAGE_RESCALE[1], 4)
batch_size = 128
label_key = f'level_{label_level}'
```

Loading data

First, we load the image names and labels into the `file_label` datasets.

In [7]:
```
train_file_labels = tf.data.experimental.make_csv_dataset(
    os.path.join(data_dir, 'train_file_labels.csv'), batch_size=batch_siz
train_file_labels = train_file_labels.unbatch()
```

In [8]:
```
validation_file_labels = tf.data.experimental.make_csv_dataset(
    os.path.join(data_dir, 'validation_file_labels.csv'), batch_size=batc
validation_file_labels = validation_file_labels.unbatch()
```

In [9]:
```
test_file_labels = tf.data.experimental.make_csv_dataset(
    os.path.join(data_dir, 'test_file_labels.csv'), batch_size=batch_size
test_file_labels = test_file_labels.unbatch()
```

We load the vocabulary for this level of labels and create a `StringLookup` encoder that will convert each label into a one-hot vector.

In [10]:
```
encoder_vocab_file = os.path.join(data_dir, f'level_{label_level}_vocabul
label_encoder = StringLookup(vocabulary=encoder_vocab_file, num_oov_indic

num_classes = len(label_encoder.get_vocabulary())

label_lookup = {i: n for i, n in enumerate(label_encoder.get_vocabulary()
label_lookup
```

Out[10]:
```
{0: 'larch',
 1: 'cleared',
 2: 'beech',
 3: 'oak',
 4: 'douglas fir',
 5: 'pine',
 6: 'fir',
 7: 'short-lived deciduous',
 8: 'long-lived deciduous',
 9: 'spruce'}
```

With the labels defined, we know enough to pretty-print a confusion matrix.

In [11]:
```
def pretty_cm(cm):
    result_table  = '<h3>Confusion matrix</h3>\n'
    result_table += '<table border=1>\n'
    result_table += f'<tr><td> </td><td> </td><th colspan={len(
    result_table += '<tr><td> </td><td> </td>'

    for _, cn in sorted(label_lookup.items()):
        result_table += f'<td><strong>{cn}</strong></td>'
    result_table += '</tr>\n'

    result_table += '<tr>\n'
    result_table += f'<th rowspan={len(label_lookup) + 1}>Actual labels</

    for ai, an in sorted(label_lookup.items()):
        result_table += '<tr>\n'
        result_table += f'  <td><strong>{an}</strong></td>\n'
        for pi, pn in sorted(label_lookup.items()):
            result_table += f'  <td>{cm[ai, pi]}</td>\n'
        result_table += '</tr>\n'
    result_table += "</table>"
    # print(result_table)
    display(HTML(result_table))
```

We have a function that take a filename and text label and returns the image and one-hot encoded label.

In [12]:
```
# Adjusted load_image function to accept file_path and label separately
def load_image(file_path, label):
    # read the image from disk, decode it, resize it, and scale the pixel
```

```
    image = tf.io.read_file(file_path)
    image = tf.io.decode_png(image, channels=4)
    image = tf.image.resize(image, IMAGE_RESCALE)
    image /= 255.0

    # grab the label and encode it
    encoded_label = label_encoder(label)

    # return the image and the one-hot encoded label
    return image, encoded_label
```

Loading the images

Now we can load the images into datasets.

In [13]:
```
train_data = train_file_labels.map(lambda fl: (fl['file_name'], fl[label_
                                       num_parallel_calls=tf.data.AUTOTUNE
train_data = train_data.shuffle(200)
train_data = train_data.map(load_image,
                                       num_parallel_calls=tf.data.AUTOTUNE
train_data = train_data.batch(batch_size)
train_data = train_data.prefetch(tf.data.AUTOTUNE)
```

In [14]:
```
validation_data = validation_file_labels.map(lambda fl: (fl['file_name'],
                                       num_parallel_calls=tf.data.AUTOTUNE
validation_data = validation_data.map(load_image,
                                       num_parallel_calls=tf.data.AUTOTUNE
validation_data = validation_data.batch(batch_size)
validation_data = validation_data.prefetch(tf.data.AUTOTUNE)
```

In [15]:
```
test_data = test_file_labels.map(lambda fl: (fl['file_name'], fl[label_ke
                                       num_parallel_calls=tf.data.AUTOTUNE
test_data = test_data.map(load_image,
                                       num_parallel_calls=tf.data.AUTOTUNE
test_data = test_data.batch(batch_size)
test_data = test_data.prefetch(tf.data.AUTOTUNE)
```

Exploring the data

We can now explore some elements of the data, such as some sample images and the distribution of classes.

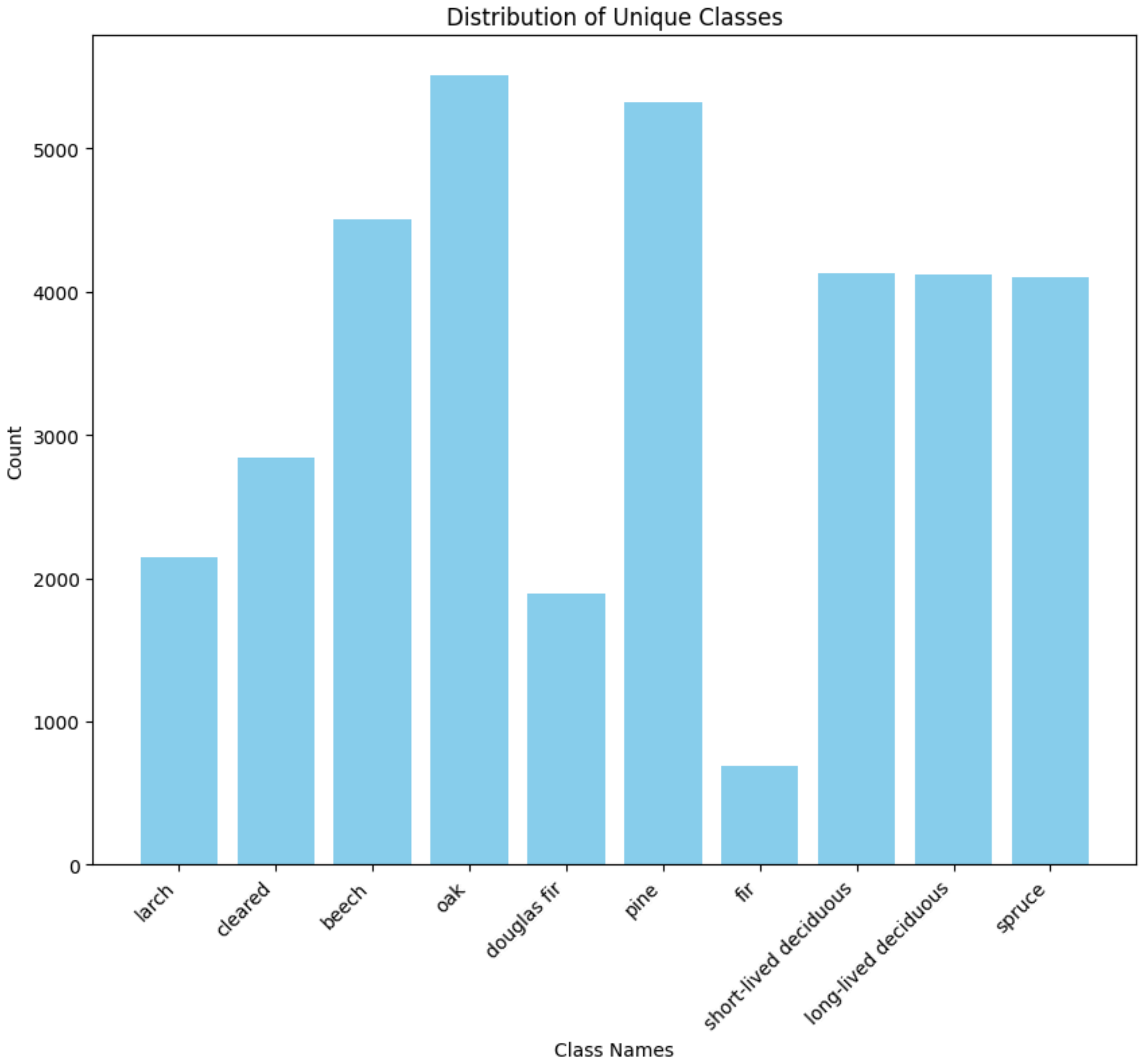
Distribution of classes

In [16]:
```
class_counts = train_file_labels.map(lambda fl: label_encoder(fl[label_ke
class_counts = class_counts.reduce(tf.zeros((num_classes,)),
                             lambda o, l: tf.math.add(o, l))
class_counts
```

Out[16]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([2145., 2844., 4508., 5515., 1896., 5324.,  687., 4131., 4118.,
       4098.], dtype=float32)>
```

In [17]:

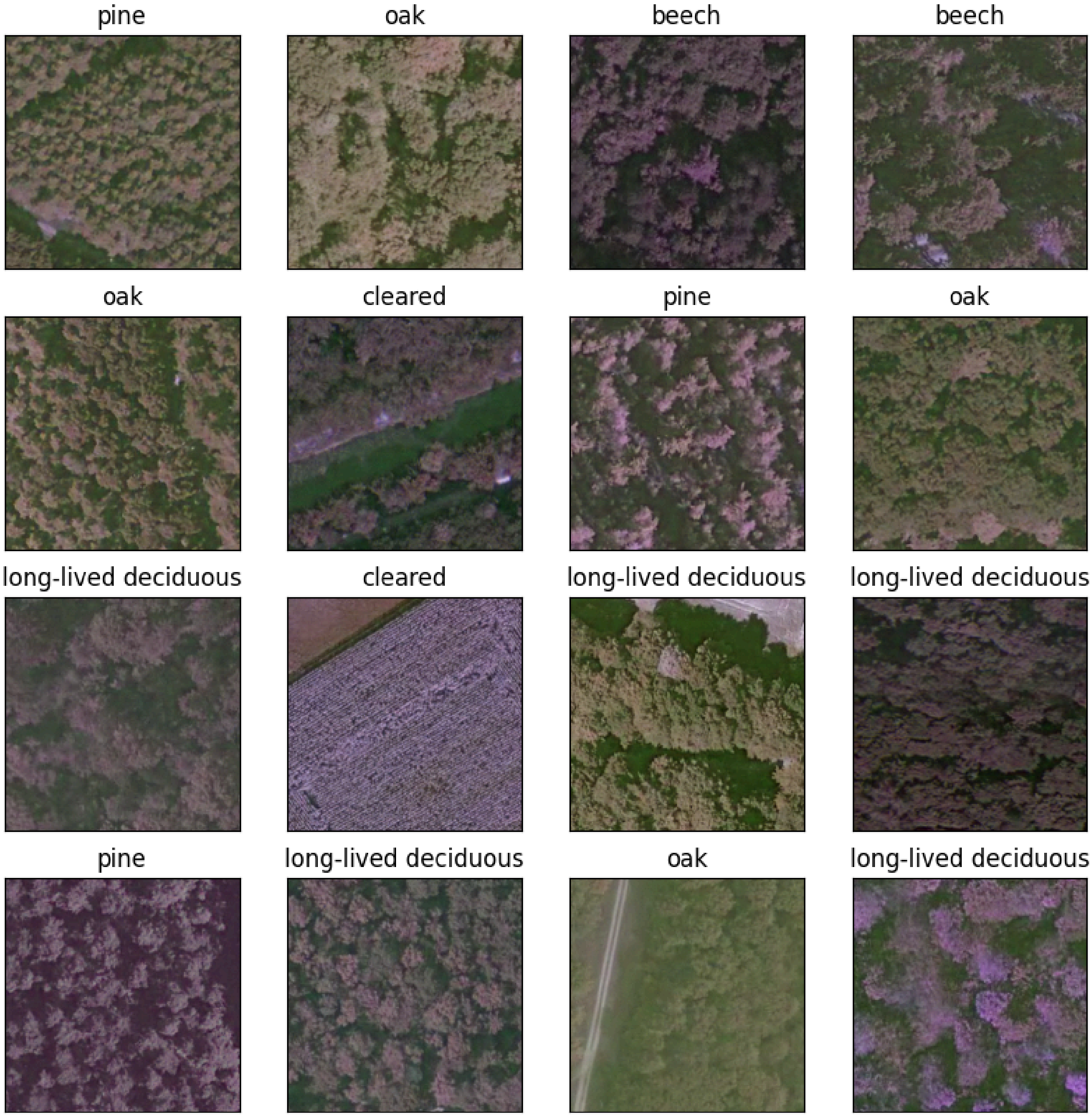
```
# Plot distribution using class names
plt.figure(figsize=(10, 8))
plt.bar(label_encoder.get_vocabulary(), class_counts.numpy(), color='skyb
plt.xlabel("Class Names")
plt.ylabel("Count")
plt.title("Distribution of Unique Classes")
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readabili
plt.show()  # Display the plotted distribution
```



Show some sample images

In [20]:

```
sample_images, sample_labels = train_data.as_numpy_iterator().next()

plt.figure(figsize=(10, 10))
for i in range(16):  # Adjust this number based on how many images you wa
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(sample_images[i, ..., :-1]) # drop the alpha channel, to m
#     plt.imshow(sample_images[i]) # image with alpha channel
    label_index = np.argmax(sample_labels[i])
    class_name = label_lookup[label_index]
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.title(class_name)
```



Create and train a base model

As the initial model, a simple CNN model was set up, with the following structure:

In [63]:
```
model_base_level_2 = Sequential([
    Conv2D(8, (5, 5), padding='same', activation='relu', input_shape=inpu
    Conv2D(8, (5, 5), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(4, 4)),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])

model_base_level_2.summary()
```

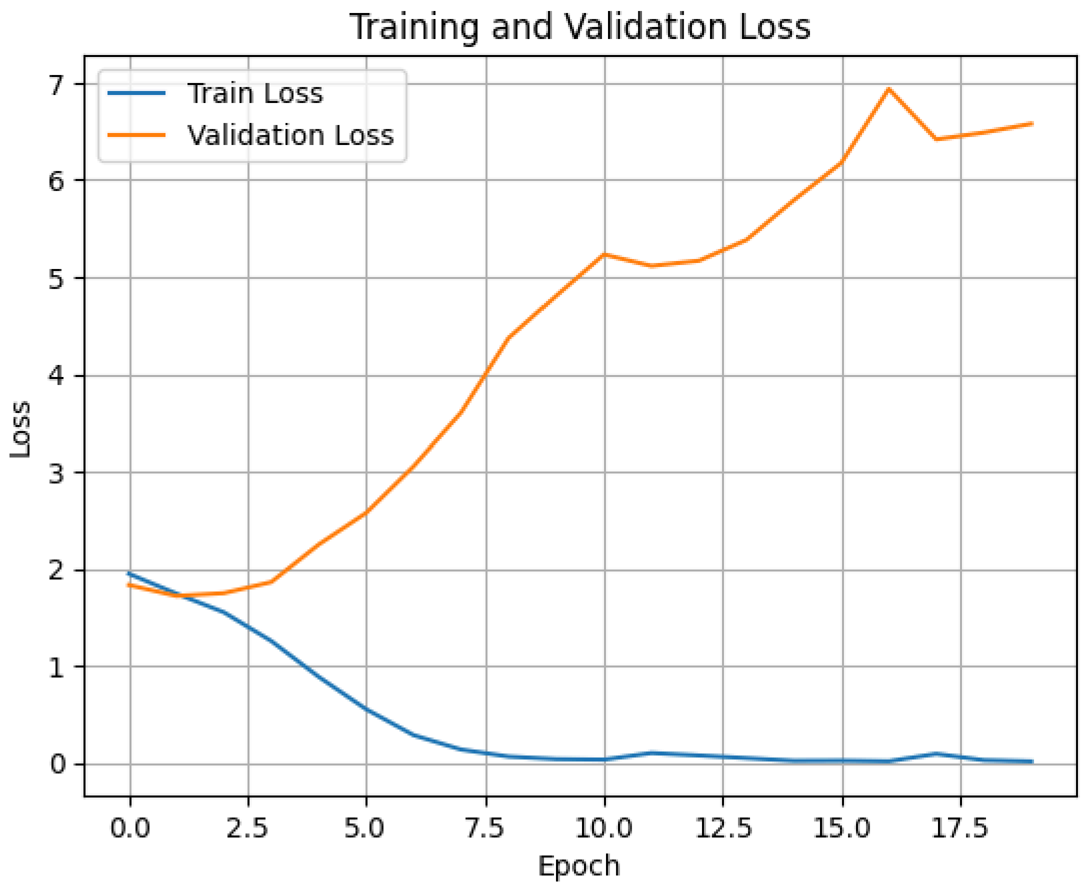
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 302, 302, 8)       808

 conv2d_19 (Conv2D)          (None, 302, 302, 8)       1608

 max_pooling2d_9 (MaxPoolin  (None, 75, 75, 8)         0
 g2D)

 flatten_5 (Flatten)         (None, 45000)             0

 dense_10 (Dense)            (None, 128)               5760128

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 5763834 (21.99 MB)
Trainable params: 5763834 (21.99 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

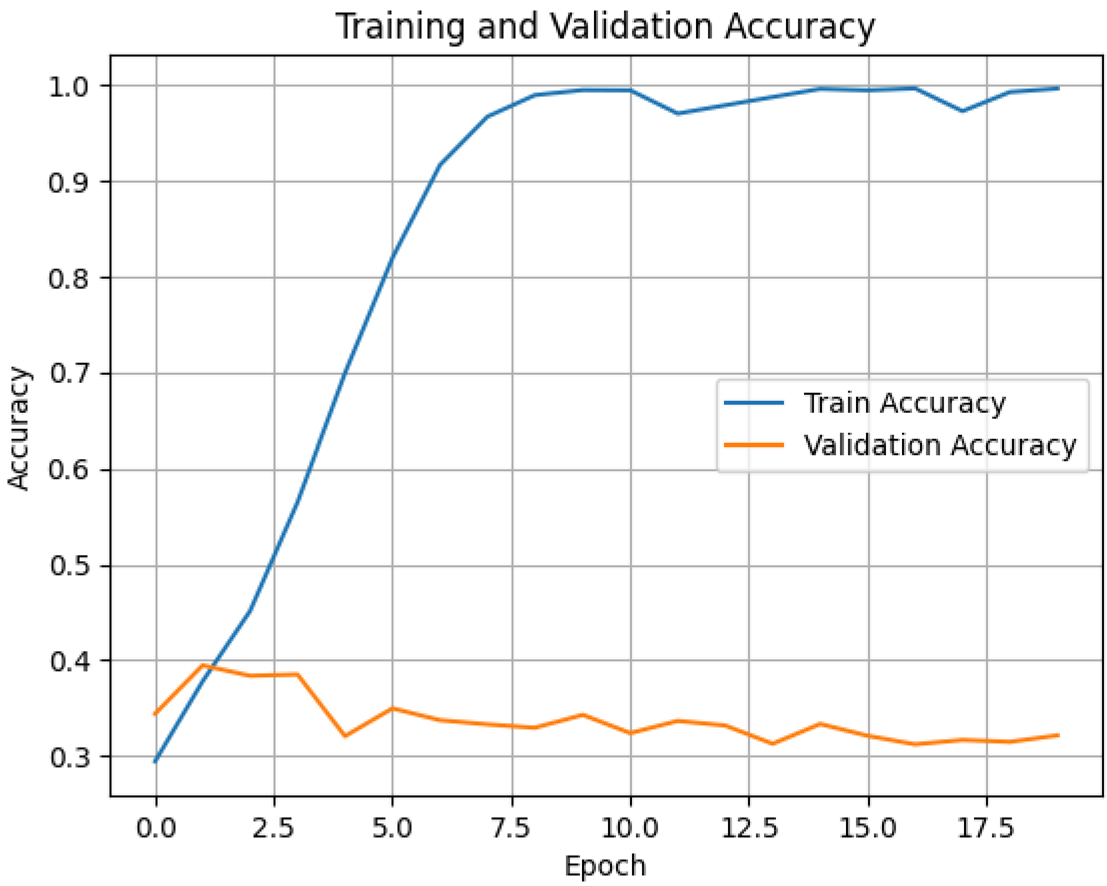
In [64]:
```
model_base_level_2.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [65]:
```
history_base_level_2 = model_base_level_2.fit(
    train_data,
    epochs=20,
    validation_data=validation_data,
    verbose=0
)
```

In [66]:
```
import matplotlib.pyplot as plt

# Assuming history contains the training history returned by model.fit
train_loss = history_base_level_2.history['loss']
val_loss = history_base_level_2.history['val_loss']
train_acc = history_base_level_2.history['accuracy']
val_acc = history_base_level_2.history['val_accuracy']

# Plot loss
plt.plot(train_loss, label='Train Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot accuracy
plt.plot(train_acc, label='Train Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.grid(True)
plt.show()
```





Cache results for later

In [67]:
```
model_base_level_2.save('model_base_level_2.keras')

with open('history_base_level_2.json', 'w') as f:
    json.dump(history_base_level_2.history, f)
```

In [68]:
```
model_base_level_2 = tf.keras.models.load_model('model_base_level_2.keras

with open('history_base_level_2.json') as f:
    history_base_level_2 = json.load(f)
```

Evaluating the model

In [69]:
```
model_base_level_2.evaluate(test_data, return_dict=True)
```

```
79/79 [==============================] - 26s 321ms/step - loss: 6.7639 - a
ccuracy: 0.3158
```

Out[69]:
```
{'loss': 6.7638983726501465, 'accuracy': 0.3157685697078705}
```

In [70]:
```
test_predictions = model_base_level_2.predict(test_data)
```

```
79/79 [==============================] - 22s 280ms/step
```

In [71]:
```
predict_labels = np.argmax(test_predictions, axis=1)
```

In [72]:
```
test_labels = np.array(list(test_data.unbatch().map(lambda x, y: y).as_nu
test_labels = np.argmax(test_labels, axis=1)
```

In [73]:
```
cm = tf.math.confusion_matrix(test_labels, predict_labels).numpy()
pretty_cm(cm)
```

Confusion matrix

		Pre								
		larch	cleared	beech	oak	douglas fir	pine	fir	short-lived deciduous	long-l decidu
Actual labels	larch	27	35	96	106	26	102	4	74	
	cleared	25	71	142	137	39	138	10	88	
	beech	53	79	181	203	65	204	14	146	
	oak	57	117	225	241	88	254	13	167	
	douglas fir	22	30	72	79	31	87	5	65	
	pine	56	102	236	221	81	208	12	164	
	fir	6	14	32	35	9	35	1	21	
	short-lived deciduous	40	82	154	187	66	169	15	114	
	long-lived deciduous	49	86	171	191	63	203	11	153	
	spruce	38	80	158	186	69	184	10	146	

Implementing more intricate models.

The more intricate model was structured similarly to that in Level 1, aiming to enable the neural network to learn more levels of global abstract structures and capture more intricate patterns within the data, as indicated by Ramesh. S (2018) and Saturn Cloud (2023) as well as to encourage the network to learn more robust and generalised representations of the data, as indicated by the Open University (2023).

In [24]:
```
# Define the model architecture with additional convolutional layers
model_2_level_2 = Sequential([
    Conv2D(16, (5, 5), padding='same', activation='relu', input_shape=inp
    Conv2D(16, (5, 5), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(4, 4)),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(rate=0.5),
    Dense(num_classes, activation='softmax')
])
```

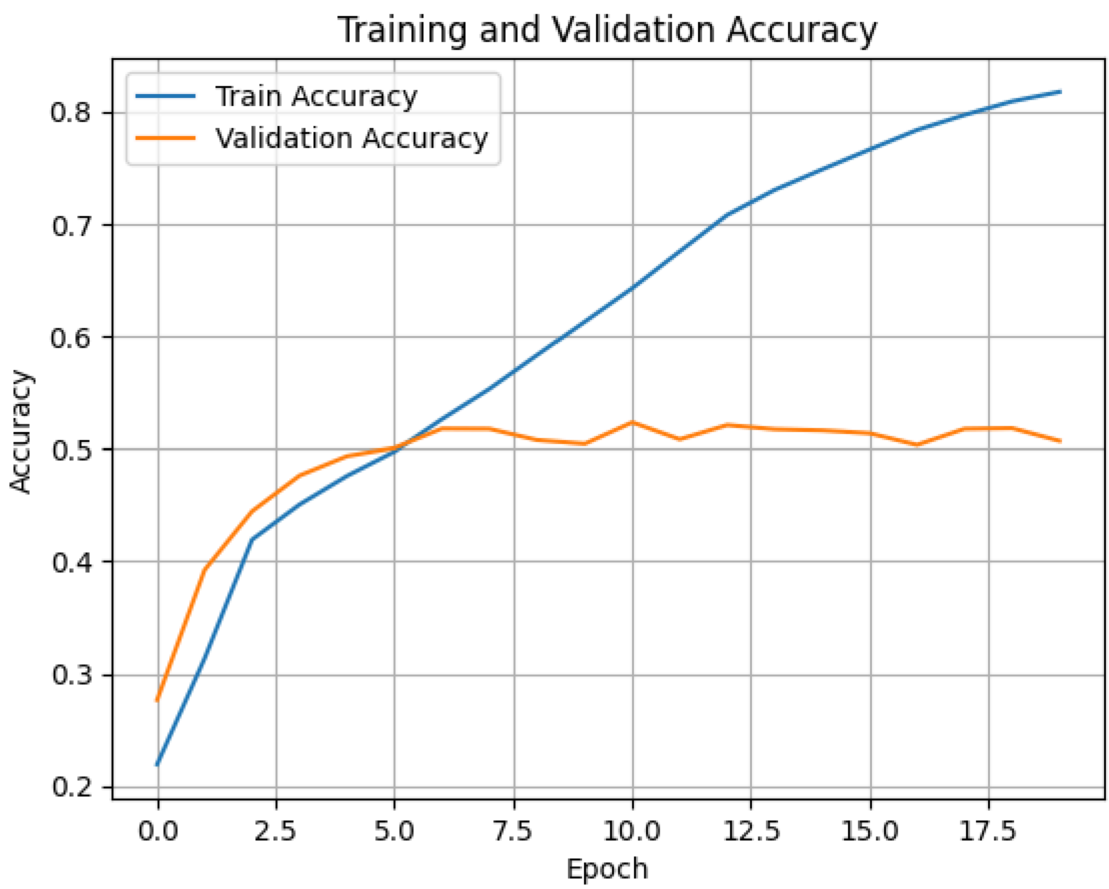
```python
# Print the summary of the improved model
model_2_level_2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 302, 302, 16)      1616

 conv2d_5 (Conv2D)           (None, 302, 302, 16)      6416

 max_pooling2d_2 (MaxPoolin  (None, 75, 75, 16)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 75, 75, 32)        4640

 conv2d_7 (Conv2D)           (None, 75, 75, 32)        9248

 max_pooling2d_3 (MaxPoolin  (None, 37, 37, 32)        0
 g2D)

 conv2d_8 (Conv2D)           (None, 37, 37, 64)        18496

 conv2d_9 (Conv2D)           (None, 37, 37, 64)        36928

 max_pooling2d_4 (MaxPoolin  (None, 18, 18, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 20736)             0

 dense_4 (Dense)             (None, 128)               2654336

 dropout (Dropout)           (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 2732970 (10.43 MB)
Trainable params: 2732970 (10.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [25]:
```python
model_2_level_2.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history_2_level_2 = model_2_level_2.fit(
    train_data,
    epochs=20,
    validation_data=validation_data,
    verbose=0
)
```

In [26]:
```python
# Extract the training and validation loss and accuracy from the dictiona
train_loss_2 = history_2_level_2.history['loss']
val_loss_2 = history_2_level_2.history['val_loss']
train_acc_2 = history_2_level_2.history['accuracy']
val_acc_2 = history_2_level_2.history['val_accuracy']
```

```python
# Plot loss
plt.plot(train_loss_2, label='Train Loss')
plt.plot(val_loss_2, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot accuracy
plt.plot(train_acc_2, label='Train Accuracy')
plt.plot(val_acc_2, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



Cache results for later

In [50]:
```
model_2_level_2.save('model_2_level_2.keras')

with open('history_2_level_2.json', 'w') as f:
    json.dump(history_2_level_2.history, f)

model_2_level_2 = tf.keras.models.load_model('model_2_level_2.keras')

with open('history_2_level_2.json') as f:
    history_2_level_2 = json.load(f)
```

Evaluating the model

In [51]:
```
model_2_level_2.evaluate(test_data, return_dict=True)
```

```
79/79 [==============================] - 22s 275ms/step - loss: 2.1504 - a
ccuracy: 0.5077
```

Out[51]: {'loss': 2.1503591537475586, 'accuracy': 0.5076907873153687}

In [52]:
```
test_predictions_2 = model_2_level_2.predict(test_data)
```

```
79/79 [==============================] - 23s 286ms/step
```

In [53]:
```
predict_labels_2 = np.argmax(test_predictions_2, axis=1)
```

In [54]:
```
test_labels_2 = np.array(list(test_data.unbatch().map(lambda x, y: y).as_
test_labels_2 = np.argmax(test_labels_2, axis=1)
```

```
In [55]: cm_2 = tf.math.confusion_matrix(test_labels_2, predict_labels_2).numpy()
pretty_cm(cm_2)
```

Confusion matrix

Pre

		larch	cleared	beech	oak	douglas fir	pine	fir	short-lived deciduous	long-l decidu
Actual labels	larch	47	46	102	90	46	107	5	83	
	cleared	60	84	138	96	48	179	4	98	
	beech	77	145	186	181	75	229	9	169	
	oak	110	155	208	214	96	252	8	218	
	douglas fir	35	52	70	91	40	101	5	52	
	pine	96	132	194	205	111	255	11	172	
	fir	14	18	24	23	6	41	1	26	
	short-lived deciduous	61	111	154	154	80	193	11	155	
	long-lived deciduous	78	121	158	211	78	220	8	159	
	spruce	87	115	171	158	79	181	5	139	

```
In [ ]:
```

Data augmentation

The overfitting was observed in both the model with the intricate model. To mitigate overfitting, one useful technique is to employ data augmentation (Open University, 2023). Therefore, modified training data using random rotation up to 10% and random zoom up to 20% in data augmentation were applied to the intricate model.

Modify dataset

```
In [27]: data_augmentation = tf.keras.Sequential([
    layers.RandomRotation(factor=0.1),  # Random rotation by up to 10% of
    layers.RandomZoom(height_factor=0.2, width_factor=0.2)  # Random zoom
])


# Augmented all the images in our train_data
aug_train_data = train_data.map(lambda x, y: (data_augmentation(x, traini
                    num_parallel_calls=tf.data.AUTOTUNE)
aug_train_data = aug_train_data.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Data Model with augmentation

In [32]:
```
# Define the model architecture with additional convolutional layers
model_3_level_2 = Sequential([
    Conv2D(16, (5, 5), padding='same', activation='relu', input_shape=inp
    Conv2D(16, (5, 5), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(4, 4)),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(rate=0.5),
    Dense(num_classes, activation='softmax')
])

# Print the summary of the improved model
model_3_level_2.summary()
```

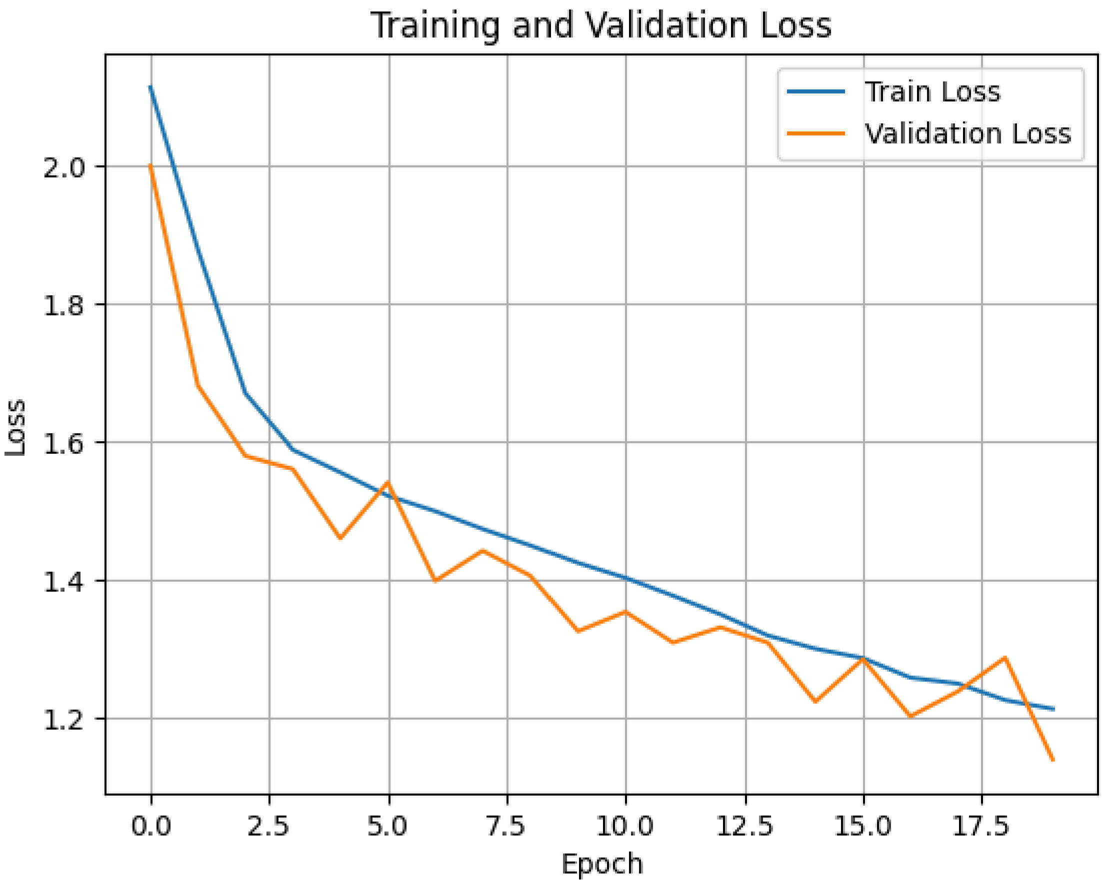
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_12 (Conv2D)          (None, 302, 302, 16)      1616      
                                                                 
 conv2d_13 (Conv2D)          (None, 302, 302, 16)      6416      
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 75, 75, 16)        0         
 g2D)                                                            
                                                                 
 conv2d_14 (Conv2D)          (None, 75, 75, 32)        4640      
                                                                 
 conv2d_15 (Conv2D)          (None, 75, 75, 32)        9248      
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 37, 37, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_16 (Conv2D)          (None, 37, 37, 64)        18496     
                                                                 
 conv2d_17 (Conv2D)          (None, 37, 37, 64)        36928     
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 18, 18, 64)        0         
 g2D)                                                            
                                                                 
 flatten_4 (Flatten)         (None, 20736)             0         
                                                                 
 dense_8 (Dense)             (None, 128)               2654336   
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_9 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 2732970 (10.43 MB)
Trainable params: 2732970 (10.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

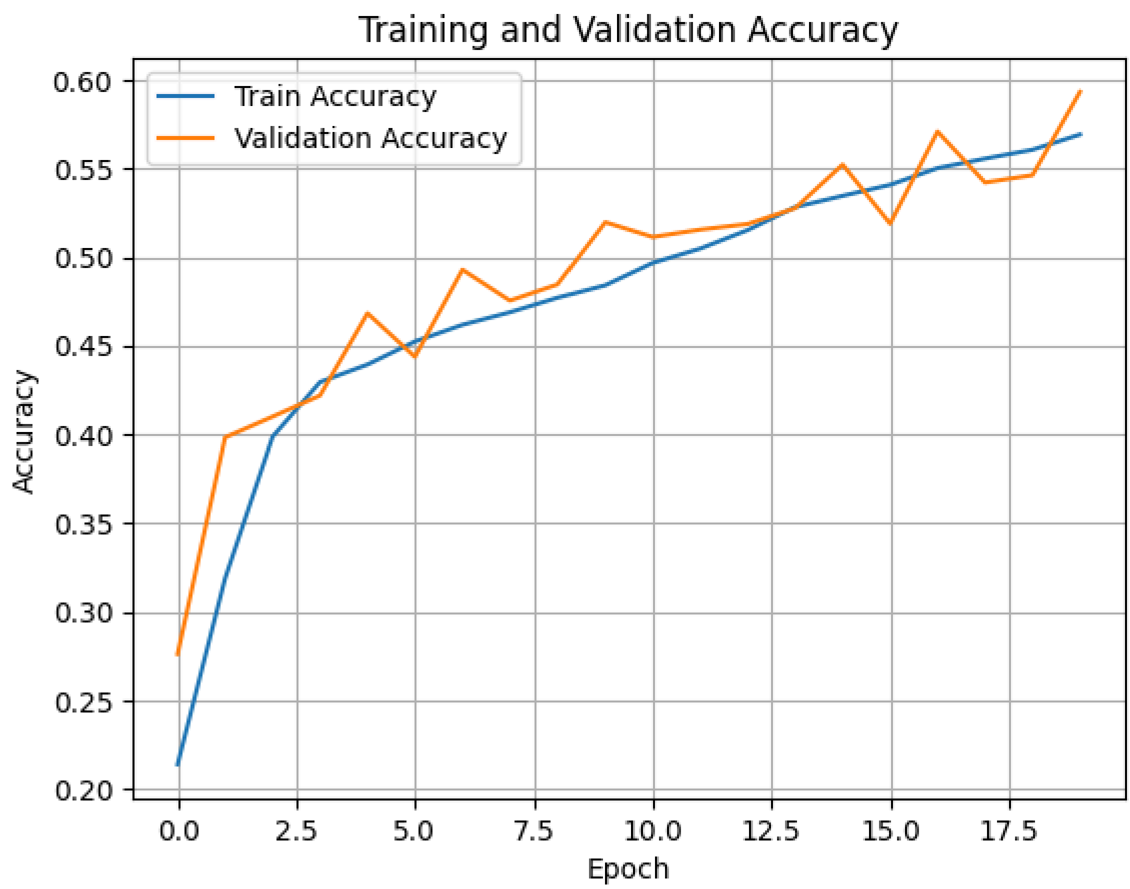
In [33]:
```
model_3_level_2.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [34]:
```
history_3_level_2 = model_3_level_2.fit(
    aug_train_data,
    epochs=20,
    validation_data=validation_data,
    verbose=0
)
```

In [35]:
```
# Extract the training and validation loss and accuracy from the dictiona
train_loss_3 = history_3_level_2.history['loss']
val_loss_3 = history_3_level_2.history['val_loss']
train_acc_3 = history_3_level_2.history['accuracy']
val_acc_3 = history_3_level_2.history['val_accuracy']

# Plot loss
plt.plot(train_loss_3, label='Train Loss')
plt.plot(val_loss_3, label='Validation Loss')
```

```python
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot accuracy
plt.plot(train_acc_3, label='Train Accuracy')
plt.plot(val_acc_3, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```





Cache results for later

In [36]:
```
model_3_level_2.save('model_3_level_2.keras')

with open('history_3_level_2.json', 'w') as f:
    json.dump(history_3_level_2.history, f)
```

In [37]:
```
model_3_level_2 = tf.keras.models.load_model('model_3_level_2.keras')

with open('history_3_level_2.json') as f:
    history_3_level_2 = json.load(f)
```

Evaluating the model

In [38]:
```
model_3_level_2.evaluate(test_data, return_dict=True)
```

```
79/79 [==============================] - 27s 338ms/step - loss: 1.1308 - a
ccuracy: 0.5984
```

Out[38]:
```
{'loss': 1.1307989358901978, 'accuracy': 0.5983923673629761}
```

In [39]:
```
test_predictions_3 = model_3_level_2.predict(test_data)
```

```
79/79 [==============================] - 23s 287ms/step
```

In [40]:
```
predict_labels_3 = np.argmax(test_predictions_3, axis=1)
```

In [41]:
```
test_labels_3 = np.array(list(test_data.unbatch().map(lambda x, y: y).as_
test_labels_3 = np.argmax(test_labels_3, axis=1)
```

```
cm_3 = tf.math.confusion_matrix(test_labels_3, predict_labels_3).numpy()
pretty_cm(cm_3)
```

Confusion matrix

										Pre
		larch	cleared	beech	oak	douglas fir	pine	fir	short-lived deciduous	long-l decidu
Actual labels	**larch**	12	64	93	137	20	116	3	57	
	cleared	16	68	131	200	40	166	6	57	
	beech	23	112	172	269	44	237	9	107	
	oak	24	125	241	363	61	259	6	124	
	douglas fir	10	47	81	116	23	95	1	42	
	pine	25	132	231	329	62	244	9	101	
	fir	3	25	31	42	12	30	0	19	
	short-lived deciduous	17	104	142	247	46	209	3	90	
	long-lived deciduous	16	111	182	288	54	197	8	113	
	spruce	19	92	174	262	36	169	9	98	

Reference

Ramesh. S (2018) ' A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST', Medium, 7 May 2018 [Online]. Available at https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7 (Accessed 11 May 2024).

Saturn Cloud (2023) How to Improve Accuracy in Neural Networks with Keras, 6 July 2023 [Online] Available at https://saturncloud.io/blog/how-to-improve-accuracy-in-neural-networks-with-keras/ (Accessed 11 May 2024).

The Open University (2023) 5 Training of CNNs, TM358 Weeks 8-11 Block 2: Image recognition with CNNs [Online]. Available at https://learn2.open.ac.uk/mod/oucontent/view.php?id=2152484&section=5.1 (Accessed 11 May 2024).


```
In [ ]:
```