MIS780 Advanced AI For Business - Assignment 2 - T2 2024

Task Number 2 : Waste classification with image data

Student Name: Anagha prashanth Raje URS

Student ID: 223709844

Table of Content

Executive Summary

This project aimed to create a model for waste classification, utilizing a dataset containing 2,864 images across six distinct categories which are cardboard, glass, metal, paper, plastic, and vegetation. The primary objective was to explore different convolutional neural network (CNN) architectures, evaluate their performance, and identify the most accurate model for real-world deployment, utilizing a 70/30 training and testing data split.

Throughout the experiment, multiple CNN models were tested with variations in layers, filters, and hidden nodes. Among them, CNN 4 demonstrated model to arrive at best performance of 68.3% accuracy and a Kappa score of 0.619, showcasing an optimal balance between model complexity and generalization. CNN 4 excelled at classifying materials like vegetation, which had distinctive visual traits. However, it faced challenges in correctly classifying plastic due to its resemblance to glass and metal, resulting in higher misclassification rates.

The findings highlight opportunities for improvement, particularly in handling difficult-to-distinguish waste types. Enhancements such as more diverse data collection, advanced feature extraction, and model fine-tuning could further boost classification accuracy, thereby concluding that model effectiveness for real-world waste management applications is high.

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
tf.config.list_physical_devices('GPU')
```

```
[]
```

```
from google.colab import drive
drive.mount('/content/drive')

#to show folders
!ls "/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Cardboard  Glass  Metal  Paper  Plastic  Vegetation
```

Data Processing

```
import os
from typing import ParamSpecArgs

#specification of directions to point image file folders
Cardboard = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Cardboard'
Glass = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Glass'
Metal = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Metal'
Paper = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Paper'
Plastic = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Plastic'
Vegetation = '/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part2_WasteImages/Vegetation'

#Retrieving a list of all files withing the directory
```

```
Cardboard_file = os.listdir(Cardboard)
Glass_file = os.listdir(Glass)
Metal_file = os.listdir(Metal)
Paper_file = os.listdir(Paper)
Plastic_file = os.listdir(Plastic)
Vegetation_file = os.listdir(Vegetation)

#Displaying total count of files
print(f'Total files under Cardborad folder are: {len(Cardboard_file)}')
print(f'Total files under Glass folder are: {len(Glass_file)}')
print(f'Total files under Metal folder are: {len(Metal_file)}')
print(f'Total files under Paper folder are: {len(Paper_file)}')
print(f'Total files under Plastic folder are: {len(Plastic_file)}')
print(f'Total files under Vegetation folder are: {len(Vegetation_file)}')
```

```
Total files under Cardborad folder are: 461
Total files under Glass folder are: 420
Total files under Metal folder are: 547
Total files under Paper folder are: 500
Total files under Plastic folder are: 500
Total files under Vegetation folder are: 436
```

Defining a function to visualize the images

```
import os
import tensorflow as tf

##Initialize a list to hold th image data and corresponding labels.
data = []

#Loop through the files in the "Cardboard" directory
for file in os.listdir(Cardboard):
  #Verifying if the file has a '.jpeg' or '..jpg' extension
  if file.endswith('.jpeg') or file.endswith('.jpg'):
    #Using Tensorflow to load image
    img = tf.io.read_file(os.path.join(Cardboard, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    # assign a corresponding label to the file
    label = 'cardboard'
    # append both the image data and label to the data list
    data.append((img,label))

#Loop through the files in the "Glass" directory
for file in os.listdir(Glass):

 if file.endswith('.jpeg') or file.endswith('.jpg'):
    img = tf.io.read_file(os.path.join(Glass, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    label = 'glass'
    data.append((img,label))

#Loop through the files in the "Metal" directory
for file in os.listdir(Metal):

 if file.endswith('.jpeg') or file.endswith('.jpg'):
    img = tf.io.read_file(os.path.join(Metal, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    label = 'metal'
    data.append((img,label))

#Loop through the files in the "Paper" directory
for file in os.listdir(Paper):

 if file.endswith('.jpeg') or file.endswith('.jpg'):
    img = tf.io.read_file(os.path.join(Paper, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    label = 'paper'
    data.append((img,label))

#Iterate over the files located in the "Plastic" folder
for file in os.listdir(Plastic):

 if file.endswith('.jpeg') or file.endswith('.jpg'):
    img = tf.io.read_file(os.path.join(Plastic, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    label = 'plastic'
```

```
        data.append((img,label))

#Loop through the files in the "Vegetation" directory
for file in os.listdir(Vegetation):

 if file.endswith('.jpeg') or file.endswith('.jpg'):
    img = tf.io.read_file(os.path.join(Vegetation, file))
    img = tf.image.decode_jpeg(img,channels=3)
    img = tf.image.resize(img,(50,50))
    label = 'vegetation'
    data.append((img,label))



#random module importing
import random
#Randomize the data and split it into training and testing sets.
random.shuffle(data)
train_data, test_data = data[:int(len(data)*0.7)], data[int(len(data)*0.7):]
```

Assigning data to X_train, X_test, Y_train, y_test, and convert these datasets into NumPy arrays for subsequent CNN model training.

```
#Retrieve the image data and labels from the training dataset.
x_train, y_train = zip(*train_data)

#Retrieve the image data and labels from the testing dataset.
x_test, y_test = zip(*test_data)

#Transform the image data and labels into NumPy arrays.
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
```

Normalise the input data for X-train and X-test

```
from re import A

# Transforming the integers into 32-bit floating-point values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#Scale every pixel value across the entire vector for each input.
x_train /= 255
x_test /= 255

#Printing the shape of reshaped data
print("Training matrix shape", x_train.shape)
print("Testing matrix shape", x_test.shape)
```

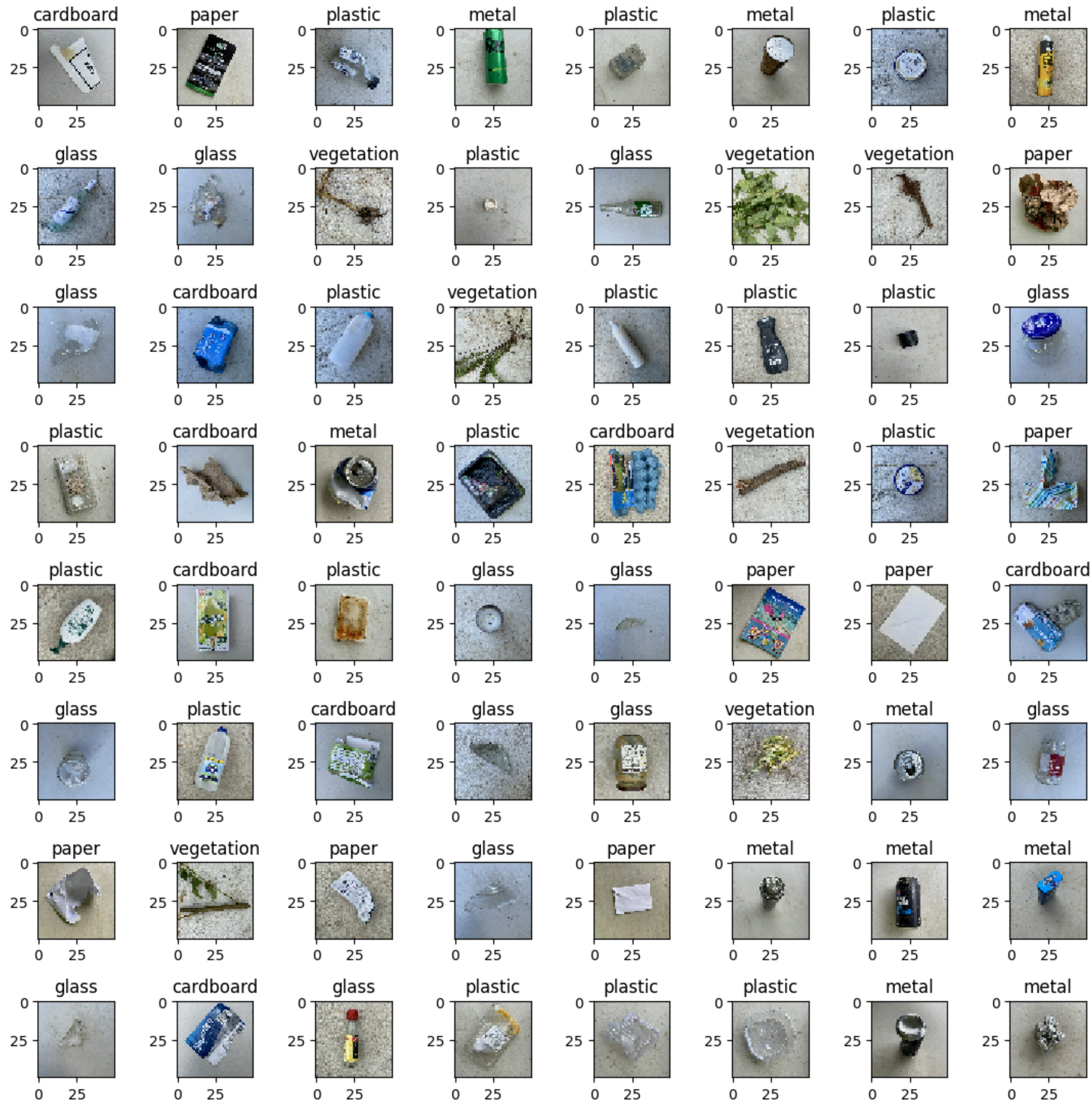
```
Training matrix shape (2004, 50, 50, 3)
Testing matrix shape (860, 50, 50, 3)
```

Implementing distinct integer encoding for the six categories: Cardboard, Glass, Metal, Paper, Plastic, and Vegetation.

```
print('The class format of the first element in the training dataset, in its original form, is: ',y_train[0], '\n')

import numpy as np
#create a NumPy array with category strings
categories = np.array(['cardboard','glass','metal','paper','plastic','vegetation'])

#create a mapping from category strings to integers
category_map = {'cardboard': 0,'glass': 1,'metal': 2,'paper': 3,'plastic': 4,'vegetation': 5}

#Encode the categories
y_train = np.array([category_map[category] for category in y_train])
y_test = np.array([category_map[category] for category in y_test])

print ('the unique integer maping encoding format of the class of the first element in the training dataset is: ',y_train[0])
```

```
The class format of the first element in the training dataset, in its original form, is:  cardboard

the unique integer maping encoding format of the class of the first element in the training dataset is:  0
```

```
#Adjust the default figure size for all plots generated in the program
plt.rcParams['figure.figsize'] = (11,11)

labels = ['cardboard','glass','metal','paper','plastic','vegetation']

for i in range(64):
  #the plt.subplot() function requires three integers argumnets:the number of rows, the number of columns, and the index of the subplot.
    plt.subplot(8,8,i+1)
  #the `plt.imshow()`function displays the image at index `i` from the `a_train` array as a grayscale image, with no interpolation appli
    plt.imshow(x_train[i],interpolation='none')
    plt.title("{}".format(labels[int(y_train[i])]))

plt.tight_layout()
```



Predictive modelling

3. Deep Learning Model Construction

Load the required libraries for CNN construction

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, Flatten
from tensorflow.keras.layers import MaxPooling2D, Activation, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, Callback, EarlyStopping
from tensorflow.keras.optimizers import SGD, RMSprop, Adam, Nadam
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras import regularizers
```

Construct several models with different architectures, such as varying the number of layers and nodes.

```
#Define img_rows, img_cols, and channels
img_rows, img_cols = 50,50
channels = 3

num_classes = 6
class_names = ['cardboard','glass','metal','paper','plastic','vegetation']
```

```
#CNN model with two Convolution layers, one Pooling layer with max pooling,
#which are stacked on top of a traditional ANN model (with the same architecture as the model 1)
def model_2():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=(img_rows, img_cols, channels)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()
    return model
```

Define callback to record training performnace.

```
# Keras callbacks (when Tensorboard installed)
keras_callbacks = [EarlyStopping(monitor='val_loss', patience=20, verbose=0)]
```

4. Model Execution

We select one of the above models to carry out the experiment.

```
model = model_2()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	896
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
dropout (Dropout)	(None, 24, 24, 32)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 128)	2,359,424
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 6)	774

Total params: 2,361,094 (9.01 MB)
Trainable params: 2,361,094 (9.01 MB)

Compile and fit the model using RMSprop optimizer.

```
!pip install tensorflow
```

```
import tensorflow as tf
```

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes=6)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=6)

model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001,weight_decay=1e-6),
              metrics=['accuracy'])

hist = model.fit(x_train, y_train,
      batch_size=128,
      epochs=100,
      verbose=2,
      validation_data=(x_test, y_test),
      validation_split=0.2,
      callbacks=keras_callbacks)
```

```
16/16 - 9s - 571ms/step - accuracy: 0.7854 - loss: 0.6137 - val_accuracy: 0.6988 - val_loss: 0.8287
Epoch 42/100
16/16 - 9s - 579ms/step - accuracy: 0.7730 - loss: 0.6180 - val_accuracy: 0.6721 - val_loss: 0.8593
Epoch 43/100
16/16 - 6s - 348ms/step - accuracy: 0.8044 - loss: 0.5662 - val_accuracy: 0.6895 - val_loss: 0.8347
Epoch 44/100
16/16 - 9s - 577ms/step - accuracy: 0.8039 - loss: 0.5661 - val_accuracy: 0.6802 - val_loss: 0.8750
Epoch 45/100
16/16 - 7s - 457ms/step - accuracy: 0.7929 - loss: 0.5757 - val_accuracy: 0.6663 - val_loss: 0.8945
Epoch 46/100
16/16 - 8s - 496ms/step - accuracy: 0.7894 - loss: 0.5945 - val_accuracy: 0.6953 - val_loss: 0.8157
Epoch 47/100
16/16 - 7s - 446ms/step - accuracy: 0.8298 - loss: 0.4965 - val_accuracy: 0.6860 - val_loss: 0.8843
Epoch 48/100
16/16 - 8s - 513ms/step - accuracy: 0.8268 - loss: 0.4815 - val_accuracy: 0.6744 - val_loss: 0.8596
Epoch 49/100
16/16 - 7s - 411ms/step - accuracy: 0.8179 - loss: 0.4985 - val_accuracy: 0.7035 - val_loss: 0.8033
Epoch 50/100
16/16 - 4s - 281ms/step - accuracy: 0.8368 - loss: 0.4632 - val_accuracy: 0.6547 - val_loss: 0.9307
Epoch 51/100
16/16 - 5s - 312ms/step - accuracy: 0.8353 - loss: 0.4638 - val_accuracy: 0.6767 - val_loss: 0.8598
Epoch 52/100
16/16 - 7s - 445ms/step - accuracy: 0.8373 - loss: 0.4592 - val_accuracy: 0.6628 - val_loss: 0.9014
Epoch 53/100
16/16 - 9s - 532ms/step - accuracy: 0.8478 - loss: 0.4327 - val_accuracy: 0.6907 - val_loss: 0.8500
Epoch 54/100
16/16 - 7s - 419ms/step - accuracy: 0.8563 - loss: 0.4115 - val_accuracy: 0.6581 - val_loss: 0.9524
Epoch 55/100
16/16 - 9s - 580ms/step - accuracy: 0.8523 - loss: 0.4121 - val_accuracy: 0.6802 - val_loss: 0.9119
Epoch 56/100
16/16 - 6s - 368ms/step - accuracy: 0.8588 - loss: 0.4063 - val_accuracy: 0.6884 - val_loss: 0.8746
Epoch 57/100
16/16 - 10s - 613ms/step - accuracy: 0.8668 - loss: 0.3889 - val_accuracy: 0.6721 - val_loss: 0.8890
Epoch 58/100
16/16 - 6s - 362ms/step - accuracy: 0.8767 - loss: 0.3574 - val_accuracy: 0.6791 - val_loss: 0.9141
Epoch 59/100
16/16 - 11s - 676ms/step - accuracy: 0.8613 - loss: 0.3733 - val_accuracy: 0.7047 - val_loss: 0.8706
Epoch 60/100
16/16 - 8s - 524ms/step - accuracy: 0.8812 - loss: 0.3366 - val_accuracy: 0.6663 - val_loss: 0.9012
Epoch 61/100
16/16 - 7s - 468ms/step - accuracy: 0.8792 - loss: 0.3516 - val_accuracy: 0.6558 - val_loss: 1.0615
Epoch 62/100
16/16 - 8s - 493ms/step - accuracy: 0.8852 - loss: 0.3246 - val_accuracy: 0.6477 - val_loss: 0.9917
Epoch 63/100
16/16 - 6s - 406ms/step - accuracy: 0.8867 - loss: 0.3250 - val_accuracy: 0.6907 - val_loss: 0.8935
Epoch 64/100
16/16 - 8s - 519ms/step - accuracy: 0.8852 - loss: 0.3089 - val_accuracy: 0.6860 - val_loss: 0.9305
Epoch 65/100
16/16 - 7s - 451ms/step - accuracy: 0.8837 - loss: 0.3185 - val_accuracy: 0.6477 - val_loss: 0.9443
Epoch 66/100
16/16 - 8s - 511ms/step - accuracy: 0.9007 - loss: 0.2829 - val_accuracy: 0.6826 - val_loss: 0.9141
Epoch 67/100
16/16 - 7s - 449ms/step - accuracy: 0.9132 - loss: 0.2704 - val_accuracy: 0.6442 - val_loss: 1.0646
Epoch 68/100
16/16 - 8s - 502ms/step - accuracy: 0.8992 - loss: 0.2813 - val_accuracy: 0.6698 - val_loss: 0.9946
Epoch 69/100
16/16 - 7s - 424ms/step - accuracy: 0.9057 - loss: 0.2879 - val_accuracy: 0.6826 - val_loss: 0.9374
```

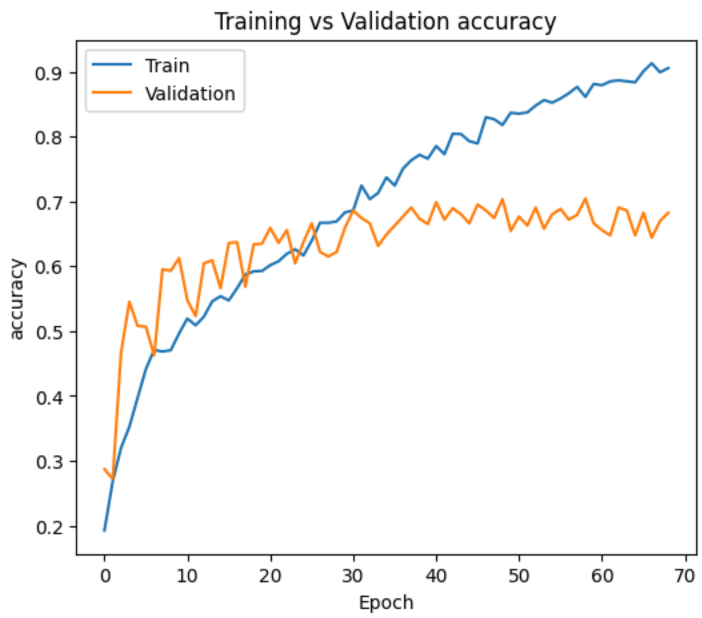
Evaluate the model

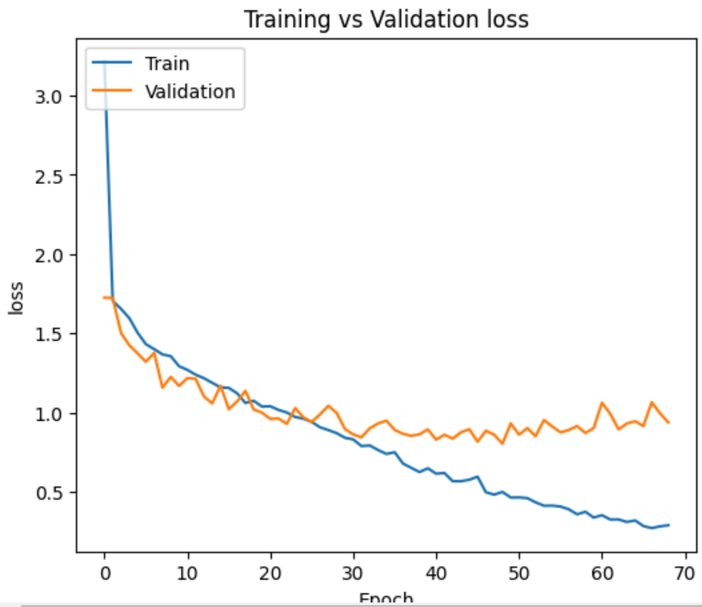
```
# Evaluate on training data
train_score = model.evaluate(x_train, y_train, verbose=0)
print('Train loss:', round(train_score[0], 4))
print('Train accuracy:', round(train_score[1], 4), '\n')

# Evaluate on test data
test_score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', round(test_score[0], 4))
print('Test accuracy:', round(test_score[1], 4))
```

```
Train loss: 0.0714
Train accuracy: 0.9955

Test loss: 0.9374
Test accuracy: 0.6826
```

```
def plot_hist(h, xsize=6, ysize=5):
    # Prepare plotting
    fig_size = plt.rcParams["figure.figsize"]
    plt.rcParams["figure.figsize"] = [xsize, ysize]

    # Get training and validation keys
    ks = list(h.keys())
    n2 = math.floor(len(ks)/2)
    train_keys = ks[0:n2]
    valid_keys = ks[n2:2*n2]

    # summarize history for different metrics
    for i in range(n2):
        plt.plot(h[train_keys[i]])
        plt.plot(h[valid_keys[i]])
        plt.title('Training vs Validation '+train_keys[i])
        plt.ylabel(train_keys[i])
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.draw()
        plt.show()

    return
```

```
import pandas as pd
import math
plot_hist(pd.DataFrame(hist.history))
```





Training vs Validation Accuracy:

The training accuracy improves steadily, reaching about 90% by the end, meaning the model learns the training data well. However, the validation accuracy fluctuates between 60-70%, showing it struggles more with unseen data. This gap hints that the model might be overfitting, as it's learning the specifics of the training set but isn't as reliable on new data.

Training vs Validation Loss:

The training loss keeps dropping, which is great for fitting the training data. But the validation loss levels off and starts fluctuating after 20 epochs, signaling the model might be memorizing the training data rather than learning patterns that generalize well.

Compuatation of accuracy, precision, recall, f1-score and support

```
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score

# Make predictions on the test set
y_pred = model.predict(x_test)

# Convert the predicted labels to continuous-multioutput format
y_pred_continuous = np.round(y_pred)

# Convert the predicted labels to multiclass format
y_pred_multiclass = np.argmax(y_pred, axis=1)
y_test_multiclass = np.argmax(y_test, axis=1)

# Calculate the kappa score
kappa = cohen_kappa_score(y_test_multiclass, y_pred_multiclass)
```

```
print("The result of Kappa is :", round(kappa, 3))

# Generate the classification report
report = classification_report(y_test_multiclass, y_pred_multiclass, target_names= class_names)

# Print the report
print("The result of the classification report is: \n ",report)
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
The result of Kappa is : 0.619
The result of the classification report is:
               precision    recall  f1-score   support

   cardboard       0.66      0.58      0.62       154
       glass       0.60      0.76      0.67       118
       metal       0.71      0.60      0.65       166
       paper       0.71      0.70      0.70       147
     plastic       0.53      0.54      0.54       136
  vegetation       0.87      0.94      0.91       139

    accuracy                           0.68       860
   macro avg       0.68      0.69      0.68       860
weighted avg       0.68      0.68      0.68       860
```

Best and Worst Performing Classes

Best Performing Class: Vegetation

**Recall: 0.94 Precision: 0.87 F1-Score: 0.91

Why it Performed Well:

Distinct Visual Features: Vegetation has unique traits—like the green hues and textured patterns of leaves—that make it stand out. The model finds it easy to pick up on these features, leading to more accurate predictions.

Consistency Across Samples: Unlike other materials, vegetation tends to look pretty uniform in the dataset, which helps the model generalize better and avoid mistakes.

Less Confusion with Other Classes: Since vegetation looks quite different from materials like plastic or metal, the model doesn't get confused as often, resulting in fewer misclassifications.

High-Quality Images: The images for vegetation might have been clearer, with less noise or glare, making it easier for the model to correctly identify this class.

Worst Performing Class: Plastic

Recall: 0.54 Precision: 0.53 F1-Score: 0.54

Why it Struggled:

Looks Like Other Materials: Plastics often look similar to glass or metal, especially when they're reflective or smooth, causing the model to confuse them with other materials.

Wide Variety of Appearances: Plastics come in many forms—clear, colored, shiny, dull—making it hard for the model to find a consistent pattern to rely on.

Lighting and Reflections: Plastics can reflect light or appear translucent, depending on the image's lighting. These variations can throw the model off, leading to errors in classification.

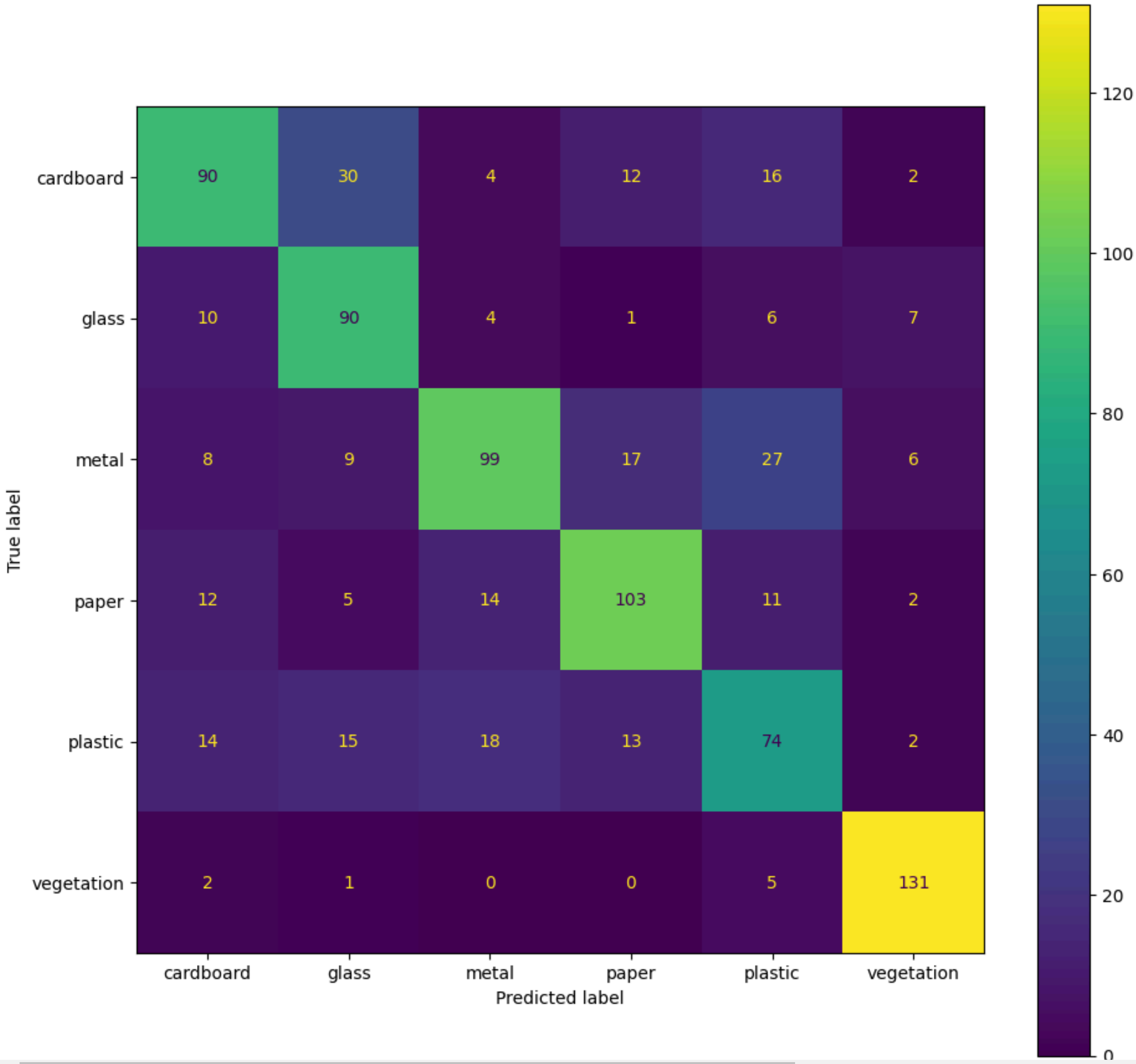
Complex Shapes: Plastics can take on many different shapes and textures, making it harder for the model to categorize them compared to more uniform materials like vegetation.

```
y_pred
```

```
array([[6.69764120e-08, 2.77063350e-07, 1.87194757e-07, 4.67115857e-09,
        5.08455926e-08, 9.99999344e-01],
       [1.11000679e-01, 4.91135865e-01, 1.67446211e-02, 3.04815471e-01,
        6.88799694e-02, 7.42342323e-03],
       [5.58609469e-03, 2.17043348e-02, 1.75818820e-02, 1.10304672e-02,
        2.17223912e-02, 9.22374845e-01],
       ...,
       [2.26369411e-05, 2.06947107e-06, 5.78787969e-03, 9.89577770e-01,
        4.60961740e-03, 4.09527487e-08],
       [7.01622746e-04, 1.06049774e-04, 5.82019973e-04, 2.65927811e-04,
        3.33999225e-04, 9.98010457e-01],
       [5.13722480e-05, 4.00234749e-05, 7.34120025e-04, 9.97729838e-01,
        1.44338259e-03, 1.19366837e-06]], dtype=float32)
```

Generating confusion matrix for inspection

```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(
    y_test_multiclass,
    y_pred_multiclass)

# Create a ConfusionMatrixDisplay object
display = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=class_names)

# Create a figure with a larger size
fig = plt.figure(figsize=(11, 11))

# Create a subplot within the figure
ax = fig.subplots()

# Plot the confusion matrix as a heatmap
display.plot(ax=ax)

# Show the plot
plt.show()
```



Explaining plot

The confusion matrix shows the model does well at identifying vegetation, getting 131 out of 139 correct, and metal, with 99 correct out of 166. However, it struggles with plastic, correctly predicting only 74 out of 136, often mistaking it for metal (18 times) and glass (15 times). Cardboard is also frequently confused with glass (30 cases). This highlights where the model performs well, like with vegetation, and where it needs improvement, especially in telling apart materials like plastic, glass, and metal, which look similar.

```
def plot_images(ims, figsize=(12,12), cols=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        if (ims.shape[-1] != 3):
            ims = ims = ims[:,:,:,0]
    f = plt.figure(figsize=figsize)
    rows=len(ims)//cols if len(ims) % cols == 0 else len(ims)//cols + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')


img_range = range(20)
imgs = x_test[img_range]
true_labels = [class_names[np.argmax(x)] for x in y_test[img_range]]
predictions = model.predict(imgs.reshape(len(img_range), img_rows, img_cols, channels))
pred_labels = [class_names[np.argmax(x)] for x in predictions]
titles = [pred_labels[x]+('' if true_labels[x] == pred_labels[x] else ' ('+true_labels[x]+')') for x in img_range]
plot_images(imgs, cols=5, figsize=(11,11), titles=titles)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```

Experiments Report

Model	Layers	Filter	Hidden Nodes	Kernalsiz	Test accuracy	Kappa	Optimizers abd activation
CNN 1	3	64,128	64	3,3	0.561	0.422	SGD
CNN 2	4	32,64,128	256	3,3	0.495	0.351	
CNN 3	2	32,64	128	3,3	0.527	0.401	
CNN 4	2	32,64	128	3,3	0.683	0.619	Adam, ReLU
CNN 5	6	64,128	256	3,3	0.479	0.298	
CNN 6	10	6,41,28,256	512	5,5	0.611	0.522	Adam, ReLU