cnn_analysis

December 12, 2024

[1]:
```
# Install basic modules and make sure they are available with the latest pip␣
↪version.
# Always updating PIP could be either good or bad, you just have to choose one␣
↪base on the situation around.

# I use --quiet and --no-warn-script-location to hide the output of my␣
↪directory paths
import sys
import os

!{sys.executable} -m pip install --upgrade pip matplotlib numpy␣
↪tensorflow-macos tensorflow-metal scikit-learn --quiet␣
↪--no-warn-script-location
```

[8]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
import numpy as np
import os
from PIL import Image
import glob
import matplotlib.pyplot as plt
from tensorflow.keras import Model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping
from sklearn.utils.class_weight import compute_class_weight
```

[3]:
```
# Define dataset path
dataset_path = os.path.join(os.getcwd().replace("investigation",␣
↪"kaggledataset"), 'garbage_classification')

# Load all images and labels
image_data = []
labels = []
class_names = sorted(os.listdir(dataset_path))
```

```
print(f"Classes: {class_names}")

for class_idx, class_name in enumerate(class_names):
    class_folder = os.path.join(dataset_path, class_name)
    if os.path.isdir(class_folder):
        for img_file in glob.glob(os.path.join(class_folder, "*.jpg")):
            try:
                # Open the image, resize, and normalize
                img = Image.open(img_file).convert("RGB").resize((256, 256))
                image_data.append(np.array(img) / 255.0)  # Normalize to 0-1␣
↪range
                labels.append(class_idx)
            except Exception as e:
                print(f"Error loading image {img_file}: {e}")

# Convert to NumPy arrays
image_data = np.array(image_data, dtype="float32")
labels = np.array(labels)

# One-hot encode the labels
labels_one_hot = to_categorical(labels, num_classes=len(class_names))

# Split data into 80/20 train/validation
train_data, test_data, train_labels, test_labels = train_test_split(
    image_data, labels_one_hot, test_size=0.2, random_state=42, stratify=labels
)

print(f"Train data shape: {train_data.shape}")
print(f"Train labels shape: {train_labels.shape}")
print(f"Validation data shape: {test_data.shape}")
print(f"Validation labels shape: {test_labels.shape}")
```

```
Classes: ['battery', 'biological', 'brown-glass', 'cardboard', 'clothes',
'green-glass', 'metal', 'paper', 'plastic', 'shoes', 'trash', 'white-glass']
Train data shape: (12412, 256, 256, 3)
Train labels shape: (12412, 12)
Validation data shape: (3103, 256, 256, 3)
Validation labels shape: (3103, 12)
```

[4]:
```
num_classes = len(class_names)  # Number of classes

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(256, 256, 3))

# First convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
```

```python
x = layers.Conv2D(32, 3, activation=None, kernel_regularizer=tf.keras.
↪regularizers.l2(0.01))(img_input)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)
x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(64, 3, activation=None, kernel_regularizer=tf.keras.
↪regularizers.l2(0.01))(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 128 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(128, 3, activation=None, kernel_regularizer=tf.keras.
↪regularizers.l2(0.01))(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)
x = layers.MaxPooling2D(2)(x)

# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x = layers.Flatten()(x)
x = layers.Dense(512, activation=None, kernel_regularizer=tf.keras.regularizers.
↪l2(0.01))(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)
x = layers.Dropout(0.7)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(num_classes, activation='softmax')(x)
```

```
2024-12-12 14:07:03.933364: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M3 Max
2024-12-12 14:07:03.933461: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 48.00 GB
2024-12-12 14:07:03.933476: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 18.00 GB
2024-12-12 14:07:03.933737: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2024-12-12 14:07:03.933762: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
```

<undefined>)

[5]:
```
# Create model:
model = Model(img_input, output)

model.summary()

# Define optimizer
optimizer = RMSprop(learning_rate=0.0001)

model.compile(
  loss='categorical_crossentropy',
  optimizer=optimizer,
  metrics=['acc']
)
```

```
Model: "functional"


 Layer (type)                    Output Shape                 Param #

 input_layer (InputLayer)        (None, 256, 256, 3)                0

 conv2d (Conv2D)                 (None, 254, 254, 32)             896

 batch_normalization             (None, 254, 254, 32)             128
 (BatchNormalization)

 re_lu (ReLU)                    (None, 254, 254, 32)               0

 max_pooling2d (MaxPooling2D)    (None, 127, 127, 32)               0

 conv2d_1 (Conv2D)               (None, 125, 125, 64)          18,496

 batch_normalization_1           (None, 125, 125, 64)             256
 (BatchNormalization)

 re_lu_1 (ReLU)                  (None, 125, 125, 64)               0

 max_pooling2d_1 (MaxPooling2D)  (None, 62, 62, 64)                 0

 conv2d_2 (Conv2D)               (None, 60, 60, 128)           73,856

 batch_normalization_2           (None, 60, 60, 128)              512
 (BatchNormalization)

 re_lu_2 (ReLU)                  (None, 60, 60, 128)                0
```

max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten (Flatten)	(None, 115200)	0
dense (Dense)	(None, 512)	58,982,912
batch_normalization_3 (BatchNormalization)	(None, 512)	2,048
re_lu_3 (ReLU)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 12)	6,156

Total params: 59,085,260 (225.39 MB)

Trainable params: 59,083,788 (225.39 MB)

Non-trainable params: 1,472 (5.75 KB)

[10]:

```
class_weights = compute_class_weight('balanced', classes=np.unique(labels),␣
↪y=labels)
class_weights_dict = dict(enumerate(class_weights))

early_stopping = EarlyStopping(monitor='val_acc', patience=20, verbose=1,␣
↪restore_best_weights=True)

def smooth_lr(epoch):
    base_lr = 0.0001
    decay = 0.9  # Slight decay every epoch
    return base_lr * (decay ** epoch)

lr_scheduler = LearningRateScheduler(smooth_lr)

# Cyclical Learning Rate
def clr(epoch):
    base_lr = 0.0001
    max_lr = 0.001
    step_size = 10
    cycle = np.floor(1 + epoch / (2 * step_size))
    x = np.abs(epoch / step_size - 2 * cycle + 1)
    lr = base_lr + (max_lr - base_lr) * max(0, (1 - x))
    return lr
```

```python
clr_callback = LearningRateScheduler(clr)

# Class weights
class_weights = compute_class_weight('balanced', classes=np.unique(labels),␣
 ↪y=labels)
class_weights_dict = dict(enumerate(class_weights))

validation_data = tf.data.Dataset.from_tensor_slices((test_data, test_labels)).
 ↪shuffle(1000).batch(32)
```

[11]:
```python
result = model.fit(
  train_data,
  train_labels,
  epochs=50,
  batch_size=32,
  validation_data=validation_data,
  verbose=1,
  class_weight=class_weights_dict,
  callbacks=[early_stopping, lr_scheduler]
)
```

```
Epoch 1/120

2024-12-12 14:09:56.575588: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.

388/388                48s 116ms/step -
acc: 0.4115 - loss: 11.7814 - val_acc: 0.3039 - val_loss: 6.5655 -
learning_rate: 1.0000e-04
Epoch 2/120
388/388                45s 115ms/step -
acc: 0.6012 - loss: 5.1413 - val_acc: 0.6262 - val_loss: 3.7608 - learning_rate:
9.0000e-05
Epoch 3/120
388/388                45s 115ms/step -
acc: 0.6557 - loss: 3.5513 - val_acc: 0.7061 - val_loss: 3.1458 - learning_rate:
8.1000e-05
Epoch 4/120
388/388                45s 115ms/step -
acc: 0.7098 - loss: 3.0106 - val_acc: 0.7151 - val_loss: 2.9199 - learning_rate:
7.2900e-05
Epoch 5/120
388/388                45s 116ms/step -
acc: 0.7545 - loss: 2.7232 - val_acc: 0.7048 - val_loss: 2.8962 - learning_rate:
6.5610e-05
Epoch 6/120
388/388                45s 116ms/step -
```

```
acc: 0.7950 - loss: 2.5181 - val_acc: 0.7093 - val_loss: 2.7528 - learning_rate:
5.9049e-05
Epoch 7/120
388/388             44s 115ms/step -
acc: 0.8259 - loss: 2.3311 - val_acc: 0.7203 - val_loss: 2.6476 - learning_rate:
5.3144e-05
Epoch 8/120
388/388             45s 115ms/step -
acc: 0.8573 - loss: 2.1735 - val_acc: 0.7151 - val_loss: 2.6300 - learning_rate:
4.7830e-05
Epoch 9/120
388/388             45s 115ms/step -
acc: 0.8778 - loss: 2.0065 - val_acc: 0.7396 - val_loss: 2.4540 - learning_rate:
4.3047e-05
Epoch 10/120
388/388             44s 114ms/step -
acc: 0.9000 - loss: 1.8737 - val_acc: 0.6781 - val_loss: 2.5367 - learning_rate:
3.8742e-05
Epoch 11/120
388/388             44s 114ms/step -
acc: 0.9145 - loss: 1.7454 - val_acc: 0.7235 - val_loss: 2.3802 - learning_rate:
3.4868e-05
Epoch 12/120
388/388             44s 114ms/step -
acc: 0.9352 - loss: 1.6379 - val_acc: 0.7045 - val_loss: 2.4514 - learning_rate:
3.1381e-05
Epoch 13/120
388/388             45s 115ms/step -
acc: 0.9488 - loss: 1.5374 - val_acc: 0.7348 - val_loss: 2.2860 - learning_rate:
2.8243e-05
Epoch 14/120
388/388             45s 115ms/step -
acc: 0.9582 - loss: 1.4438 - val_acc: 0.7451 - val_loss: 2.1651 - learning_rate:
2.5419e-05
Epoch 15/120
388/388             44s 114ms/step -
acc: 0.9636 - loss: 1.3721 - val_acc: 0.7103 - val_loss: 2.3354 - learning_rate:
2.2877e-05
Epoch 16/120
388/388             44s 113ms/step -
acc: 0.9683 - loss: 1.3024 - val_acc: 0.7328 - val_loss: 2.1075 - learning_rate:
2.0589e-05
Epoch 17/120
388/388             44s 113ms/step -
acc: 0.9773 - loss: 1.2313 - val_acc: 0.6851 - val_loss: 2.2281 - learning_rate:
1.8530e-05
Epoch 18/120
388/388             44s 113ms/step -
```

```
acc: 0.9813 - loss: 1.1651 - val_acc: 0.7432 - val_loss: 2.0240 - learning_rate:
1.6677e-05
Epoch 19/120
388/388          44s 114ms/step -
acc: 0.9799 - loss: 1.1300 - val_acc: 0.7357 - val_loss: 2.0494 - learning_rate:
1.5009e-05
Epoch 20/120
388/388          44s 114ms/step -
acc: 0.9876 - loss: 1.0786 - val_acc: 0.7051 - val_loss: 2.1905 - learning_rate:
1.3509e-05
Epoch 21/120
388/388          44s 114ms/step -
acc: 0.9882 - loss: 1.0392 - val_acc: 0.7309 - val_loss: 1.9612 - learning_rate:
1.2158e-05
Epoch 22/120
388/388          44s 114ms/step -
acc: 0.9910 - loss: 1.0012 - val_acc: 0.7286 - val_loss: 2.0012 - learning_rate:
1.0942e-05
Epoch 23/120
388/388          44s 114ms/step -
acc: 0.9920 - loss: 0.9705 - val_acc: 0.7393 - val_loss: 1.8963 - learning_rate:
9.8477e-06
Epoch 24/120
388/388          44s 115ms/step -
acc: 0.9935 - loss: 0.9402 - val_acc: 0.7596 - val_loss: 1.8224 - learning_rate:
8.8629e-06
Epoch 25/120
388/388          44s 114ms/step -
acc: 0.9962 - loss: 0.9121 - val_acc: 0.6184 - val_loss: 2.4651 - learning_rate:
7.9766e-06
Epoch 26/120
388/388          44s 114ms/step -
acc: 0.9960 - loss: 0.8903 - val_acc: 0.7432 - val_loss: 1.8721 - learning_rate:
7.1790e-06
Epoch 27/120
388/388          44s 114ms/step -
acc: 0.9958 - loss: 0.8705 - val_acc: 0.7193 - val_loss: 1.9879 - learning_rate:
6.4611e-06
Epoch 28/120
388/388          44s 114ms/step -
acc: 0.9957 - loss: 0.8508 - val_acc: 0.7686 - val_loss: 1.6975 - learning_rate:
5.8150e-06
Epoch 29/120
388/388          44s 114ms/step -
acc: 0.9975 - loss: 0.8315 - val_acc: 0.7467 - val_loss: 1.7885 - learning_rate:
5.2335e-06
Epoch 30/120
388/388          44s 114ms/step -
```

```
acc: 0.9966 - loss: 0.8191 - val_acc: 0.7580 - val_loss: 1.6762 - learning_rate:
4.7101e-06
Epoch 31/120
388/388            44s 114ms/step -
acc: 0.9980 - loss: 0.8032 - val_acc: 0.7544 - val_loss: 1.7482 - learning_rate:
4.2391e-06
Epoch 32/120
388/388            44s 114ms/step -
acc: 0.9981 - loss: 0.7949 - val_acc: 0.7751 - val_loss: 1.6221 - learning_rate:
3.8152e-06
Epoch 33/120
388/388            44s 113ms/step -
acc: 0.9987 - loss: 0.7773 - val_acc: 0.7686 - val_loss: 1.6315 - learning_rate:
3.4337e-06
Epoch 34/120
388/388            44s 114ms/step -
acc: 0.9988 - loss: 0.7648 - val_acc: 0.7667 - val_loss: 1.6473 - learning_rate:
3.0903e-06
Epoch 35/120
388/388            44s 113ms/step -
acc: 0.9990 - loss: 0.7557 - val_acc: 0.7705 - val_loss: 1.6470 - learning_rate:
2.7813e-06
Epoch 36/120
388/388            44s 113ms/step -
acc: 0.9985 - loss: 0.7475 - val_acc: 0.7686 - val_loss: 1.6288 - learning_rate:
2.5032e-06
Epoch 37/120
388/388            44s 114ms/step -
acc: 0.9982 - loss: 0.7420 - val_acc: 0.7757 - val_loss: 1.6232 - learning_rate:
2.2528e-06
Epoch 38/120
388/388            44s 114ms/step -
acc: 0.9989 - loss: 0.7288 - val_acc: 0.7699 - val_loss: 1.6124 - learning_rate:
2.0276e-06
Epoch 39/120
388/388            44s 114ms/step -
acc: 0.9994 - loss: 0.7201 - val_acc: 0.7728 - val_loss: 1.5918 - learning_rate:
1.8248e-06
Epoch 40/120
388/388            44s 114ms/step -
acc: 0.9994 - loss: 0.7137 - val_acc: 0.7641 - val_loss: 1.6362 - learning_rate:
1.6423e-06
Epoch 41/120
388/388            44s 114ms/step -
acc: 0.9997 - loss: 0.7067 - val_acc: 0.7657 - val_loss: 1.6166 - learning_rate:
1.4781e-06
Epoch 42/120
388/388            44s 114ms/step -
```

```
acc: 0.9993 - loss: 0.7024 - val_acc: 0.7647 - val_loss: 1.6062 - learning_rate:
1.3303e-06
Epoch 43/120
388/388              44s 114ms/step -
acc: 0.9992 - loss: 0.6956 - val_acc: 0.7670 - val_loss: 1.5911 - learning_rate:
1.1973e-06
Epoch 44/120
388/388              45s 115ms/step -
acc: 0.9990 - loss: 0.6928 - val_acc: 0.7696 - val_loss: 1.6115 - learning_rate:
1.0775e-06
Epoch 45/120
388/388              44s 114ms/step -
acc: 0.9992 - loss: 0.6863 - val_acc: 0.7696 - val_loss: 1.5821 - learning_rate:
9.6977e-07
Epoch 46/120
388/388              44s 114ms/step -
acc: 0.9996 - loss: 0.6829 - val_acc: 0.7725 - val_loss: 1.5824 - learning_rate:
8.7280e-07
Epoch 47/120
388/388              44s 113ms/step -
acc: 0.9995 - loss: 0.6773 - val_acc: 0.7738 - val_loss: 1.5885 - learning_rate:
7.8552e-07
Epoch 48/120
388/388              44s 113ms/step -
acc: 0.9993 - loss: 0.6772 - val_acc: 0.7751 - val_loss: 1.5825 - learning_rate:
7.0697e-07
Epoch 49/120
388/388              44s 114ms/step -
acc: 0.9996 - loss: 0.6719 - val_acc: 0.7734 - val_loss: 1.5790 - learning_rate:
6.3627e-07
Epoch 50/120
388/388              45s 115ms/step -
acc: 0.9992 - loss: 0.6696 - val_acc: 0.7770 - val_loss: 1.5827 - learning_rate:
5.7264e-07
Epoch 51/120
388/388              45s 116ms/step -
acc: 0.9997 - loss: 0.6654 - val_acc: 0.7776 - val_loss: 1.5900 - learning_rate:
5.1538e-07
Epoch 52/120
388/388              44s 114ms/step -
acc: 0.9997 - loss: 0.6643 - val_acc: 0.7751 - val_loss: 1.5764 - learning_rate:
4.6384e-07
Epoch 53/120
388/388              44s 114ms/step -
acc: 0.9992 - loss: 0.6628 - val_acc: 0.7731 - val_loss: 1.5804 - learning_rate:
4.1746e-07
Epoch 54/120
388/388              44s 114ms/step -
```

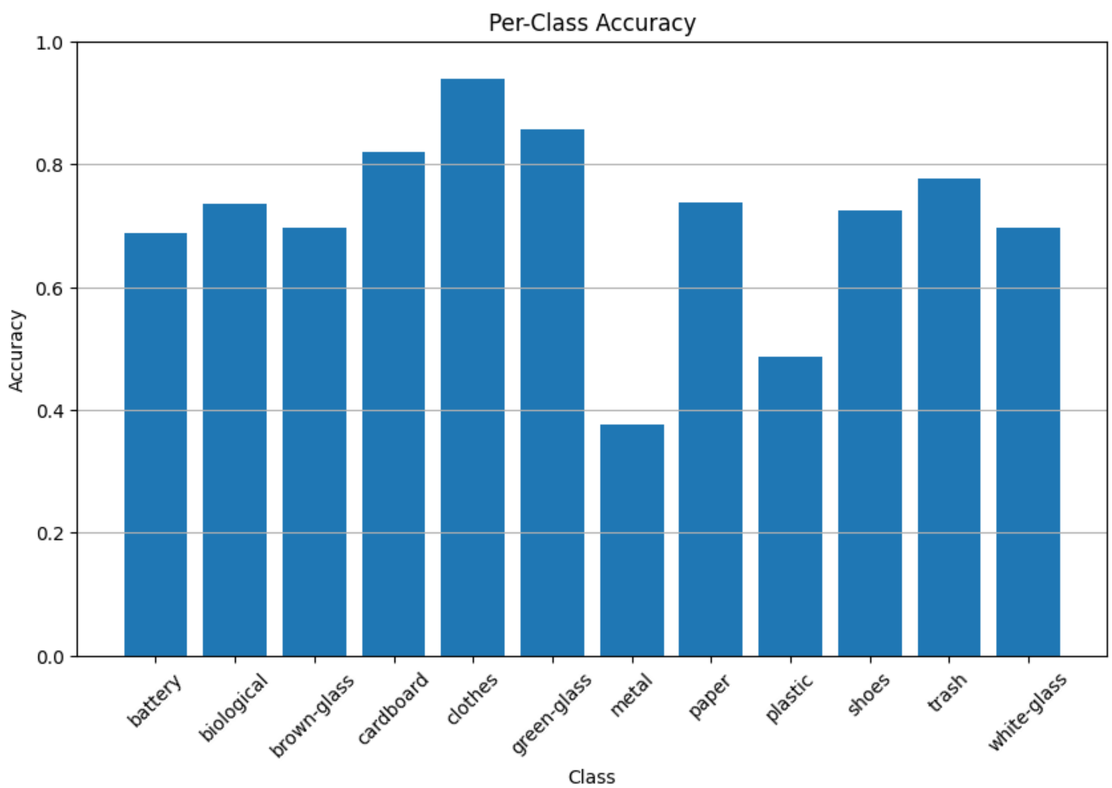
```
acc: 0.9990 - loss: 0.6615 - val_acc: 0.7731 - val_loss: 1.5827 - learning_rate:
3.7571e-07
Epoch 55/120
388/388            44s 114ms/step -
acc: 0.9994 - loss: 0.6585 - val_acc: 0.7770 - val_loss: 1.5738 - learning_rate:
3.3814e-07
Epoch 56/120
388/388            45s 115ms/step -
acc: 0.9996 - loss: 0.6563 - val_acc: 0.7763 - val_loss: 1.5791 - learning_rate:
3.0433e-07
Epoch 57/120
388/388            44s 114ms/step -
acc: 0.9998 - loss: 0.6538 - val_acc: 0.7757 - val_loss: 1.5752 - learning_rate:
2.7389e-07
Epoch 58/120
388/388            44s 114ms/step -
acc: 0.9997 - loss: 0.6530 - val_acc: 0.7776 - val_loss: 1.5740 - learning_rate:
2.4650e-07
Epoch 59/120
388/388            45s 115ms/step -
acc: 0.9993 - loss: 0.6522 - val_acc: 0.7767 - val_loss: 1.5674 - learning_rate:
2.2185e-07
Epoch 60/120
388/388            44s 113ms/step -
acc: 0.9993 - loss: 0.6554 - val_acc: 0.7767 - val_loss: 1.5693 - learning_rate:
1.9967e-07
Epoch 61/120
388/388            43s 112ms/step -
acc: 0.9997 - loss: 0.6503 - val_acc: 0.7751 - val_loss: 1.5736 - learning_rate:
1.7970e-07
Epoch 62/120
388/388            43s 112ms/step -
acc: 0.9994 - loss: 0.6497 - val_acc: 0.7767 - val_loss: 1.5749 - learning_rate:
1.6173e-07
Epoch 63/120
388/388            43s 112ms/step -
acc: 0.9998 - loss: 0.6476 - val_acc: 0.7767 - val_loss: 1.5748 - learning_rate:
1.4556e-07
Epoch 64/120
388/388            45s 115ms/step -
acc: 0.9998 - loss: 0.6467 - val_acc: 0.7767 - val_loss: 1.5684 - learning_rate:
1.3100e-07
Epoch 65/120
388/388            44s 114ms/step -
acc: 0.9994 - loss: 0.6463 - val_acc: 0.7767 - val_loss: 1.5740 - learning_rate:
1.1790e-07
Epoch 66/120
388/388            44s 114ms/step -
```

```
acc: 0.9991 - loss: 0.6467 - val_acc: 0.7744 - val_loss: 1.5735 - learning_rate:
1.0611e-07
Epoch 67/120
388/388              45s 115ms/step -
acc: 0.9995 - loss: 0.6484 - val_acc: 0.7767 - val_loss: 1.5699 - learning_rate:
9.5500e-08
Epoch 68/120
388/388              44s 114ms/step -
acc: 0.9994 - loss: 0.6462 - val_acc: 0.7741 - val_loss: 1.5728 - learning_rate:
8.5950e-08
Epoch 69/120
388/388              44s 114ms/step -
acc: 0.9995 - loss: 0.6450 - val_acc: 0.7751 - val_loss: 1.5703 - learning_rate:
7.7355e-08
Epoch 70/120
388/388              44s 113ms/step -
acc: 0.9997 - loss: 0.6443 - val_acc: 0.7747 - val_loss: 1.5718 - learning_rate:
6.9620e-08
Epoch 71/120
388/388              44s 113ms/step -
acc: 0.9990 - loss: 0.6466 - val_acc: 0.7738 - val_loss: 1.5714 - learning_rate:
6.2658e-08
Epoch 71: early stopping
Restoring model weights from the end of the best epoch: 51.
```

[12]:
```
##############################################
# Get predictions for the test data
predictions = model.predict(test_data)

# Convert predictions and true labels from one-hot to class indices
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)

# Calculate overall accuracy
overall_accuracy = np.sum(predicted_classes == true_classes) / len(true_classes)
print(f"Overall Test Accuracy: {overall_accuracy:.2f}")

# Calculate per-class accuracy
num_classes = len(class_names)
class_accuracies = []
for class_index in range(num_classes):
    indices = np.where(true_classes == class_index)[0]
    class_correct = np.sum(predicted_classes[indices] == true_classes[indices])
    class_accuracy = class_correct / len(indices) if len(indices) > 0 else 0
    class_accuracies.append(class_accuracy)

# Plot per-class accuracy
```

```
plt.figure(figsize=(10, 6))
plt.bar(class_names, class_accuracies)
plt.title("Per-Class Accuracy")
plt.xlabel("Class")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis="y")
plt.show()
```

```
97/97              2s 17ms/step
Overall Test Accuracy: 0.78
```



[13]:
```
from sklearn.metrics import classification_report, accuracy_score
import numpy as np

# Get predictions for the test data
predictions = model.predict(test_data)

# Convert predictions and true labels from one-hot encoding to class indices
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)
```

```python
# Compute overall accuracy
accuracy = accuracy_score(true_classes, predicted_classes)
print(f"Overall Accuracy: {accuracy:.2f}")

# Compute classification report (includes Precision, Recall, F1-Score)
report = classification_report(true_classes, predicted_classes,␣
 ↪target_names=class_names)
print("Classification Report:")
print(report)
```

```
97/97              2s 16ms/step
Overall Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

     battery       0.72      0.69      0.70       189
  biological       0.75      0.74      0.74       197
 brown-glass       0.75      0.70      0.72       122
   cardboard       0.75      0.82      0.78       178
     clothes       0.92      0.94      0.93      1065
 green-glass       0.84      0.86      0.85       126
       metal       0.63      0.38      0.47       154
       paper       0.79      0.74      0.76       210
     plastic       0.56      0.49      0.52       173
       shoes       0.69      0.72      0.71       395
       trash       0.66      0.78      0.71       139
 white-glass       0.58      0.70      0.63       155

    accuracy                           0.78      3103
   macro avg       0.72      0.71      0.71      3103
weighted avg       0.78      0.78      0.77      3103
```

[ ]: