

CAB420_A1B_Q2_Template

May 9, 2025

1 CAB420 Assignment 1B Question 2: Template

1.1 Overview

This notebook provides a helper function to load in the Oxford-IIIT Pets dataset suitable for classification and semantic segmentation, to help with Assignment 1B, Question 2.

It also provides an example of how to load in the MobileNetV3Small Network which you are required to fine tune for the second part of the question.

Please read the comments and instructions within this notebook. It has been carefully designed to help you with many of the tasks required.

Please make sure you read the assignment brief on canvas, and check the FAQ for other information.

```
[1]: !pip install --upgrade tensorflow_datasets
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import matplotlib.pyplot as plt
from matplotlib import gridspec

import tensorflow as tf
import keras
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D, concatenate, BatchNormalization, SpatialDropout2D, Activation, Flatten
from keras.models import Model

import numpy as np
import pandas as pd
import tensorflow_datasets as tfds
import glob
```

Requirement already satisfied: tensorflow_datasets in
/opt/conda/lib/python3.11/site-packages (4.9.8)

Requirement already satisfied: absl-py in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (2.2.2)

Requirement already satisfied: array_record<=0.5.0 in
/opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (0.7.2)

Requirement already satisfied: dm-tree in /opt/conda/lib/python3.11/site-

packages (from tensorflow_datasets) (0.1.9)
 Requirement already satisfied: etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (1.12.2)
 Requirement already satisfied: immutabledict in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (4.2.1)
 Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (1.26.4)
 Requirement already satisfied: promise in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (2.3)
 Requirement already satisfied: protobuf>=3.20 in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (5.29.4)
 Requirement already satisfied: psutil in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (5.9.8)
 Requirement already satisfied: pyarrow in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (19.0.1)
 Requirement already satisfied: requests>=2.19.0 in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (2.32.3)
 Requirement already satisfied: simple_parsing in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (0.1.7)
 Requirement already satisfied: tensorflow-metadata in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (1.17.1)
 Requirement already satisfied: termcolor in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (3.1.0)
 Requirement already satisfied: toml in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (0.10.2)
 Requirement already satisfied: tqdm in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (4.67.1)
 Requirement already satisfied: wrapt in /opt/conda/lib/python3.11/site-packages (from tensorflow_datasets) (1.17.2)
 Requirement already satisfied: einops in /opt/conda/lib/python3.11/site-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (0.8.1)
 Requirement already satisfied: fsspec in /opt/conda/lib/python3.11/site-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (2025.3.2)
 Requirement already satisfied: importlib_resources in /opt/conda/lib/python3.11/site-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (6.5.2)
 Requirement already satisfied: typing_extensions in /opt/conda/lib/python3.11/site-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (4.13.2)
 Requirement already satisfied: zipp in /opt/conda/lib/python3.11/site-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow_datasets) (3.21.0)
 Requirement already satisfied: charset_normalizer<4,>=2 in

```

/opt/conda/lib/python3.11/site-packages (from
requests>=2.19.0->tensorflow_datasets) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.11/site-
packages (from requests>=2.19.0->tensorflow_datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.11/site-packages (from
requests>=2.19.0->tensorflow_datasets) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.11/site-packages (from
requests>=2.19.0->tensorflow_datasets) (2025.1.31)
Requirement already satisfied: attrs>=18.2.0 in /opt/conda/lib/python3.11/site-
packages (from dm-tree->tensorflow_datasets) (25.3.0)
Requirement already satisfied: six in /opt/conda/lib/python3.11/site-packages
(from promise->tensorflow_datasets) (1.17.0)
Requirement already satisfied: docstring-parser<1.0,>=0.15 in
/opt/conda/lib/python3.11/site-packages (from
simple_parsing->tensorflow_datasets) (0.16)
Requirement already satisfied: googleapis-common-protos<2,>=1.56.4 in
/opt/conda/lib/python3.11/site-packages (from tensorflow-
metadata->tensorflow_datasets) (1.70.0)

```

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

```

E0000 00:00:1746782489.652478      129 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered

```

```

E0000 00:00:1746782489.657223      129 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered

```

1.1.1 Data loading and pre-processing functions

We first provide some helper functions to format the data in the way we need. You shouldn't need to change these, though you are welcome to if you like.

One thing you may want to do is create additional augmentation functions, and the `flip_lr_augmentation` function below could be used as a template to create additional augmentation types.

```

[2]: def preprocess_segmentation_mask(segmentation_mask):
    """preprocess the segmentation mask

    The original segmentation mask has three categories.
    foreground, background and outline
    This function will just convert it to foreground and background

    The original segmentation mask is also 1-index, so will convert it
    to 0-index.
    """

```

the original mask is represented as:
1 - edge of dog/cat and things like leashes etc.
2 - background
3 - foreground

we want to just keep the merge the edges and foreground of the doggo/catto, `and`

then treat it as a binary semantic segmentation task.

To achieve this, we will just subtract two, converting to values of `[-1, 0, 1]`,

and then apply the `abs` function to convert the `-1` values (edges) to the foreground.

Will also convert it to 32 bit float which will be needed for working with `tf`.

Why am I doing it this way?

A reasonable question. Initially I tried to do it with just normal array indexing,

but this is a bit more work since the mask is a tensorflow tensor and not a `np` array.

We could alternatively convert it to an array, perform indexing and then map it back,

but this would have a performance overhead, which wouldn't be a big deal, but still.

With all that being said, I am doing it for you, so you don't have to.

Args:

`segmentation_mask (array):`
original segmentation mask

Returns:

`preprocessed segmentation_mask`
"""

`return tf.abs(tf.cast(segmentation_mask, tf.float32) - 2)`

`def return_image_label_mask(ds_out):`

""" function to return image, class label and segmentation mask

The original dataset contains additional information, such as the filename, `and`

the species. We don't care about any of that for this work, so will discard them and just keep the original image as our input, and then a tuple of our outputs that will be the class label and the semantic segmentation mask.

Whilst we are here, we will also preprocess the segmentation mask.

Args:

ds_out: dict
original dataset output

Returns:

RGB image
tuple of class label and preprocessed segmentation mask
"""

preprocess the segmentation mask

seg_mask = preprocess_segmentation_mask(ds_out['segmentation_mask'])

image = tf.cast(ds_out['image'], tf.float32)

image = standardise_image(image)

return image, (ds_out['label'], seg_mask)

def mobilenet_preprocess_image(image):

"""Apply preprocessing that is suitable for MobileNetV3.

Simply scales to ranges [-1, 1]

you should use this preprocessing for both your model and the mobilenet_
model

"""

image = (image - 127.5) / 255.0

return image

def unprocess_image(image):

""" undo preprocessing above so can plot images"""

image = image * 255.0 + 127.5

return image

def preprocess_and_resize(image, output, image_size):

"""apply preprocessing steps above to images and resize images and maps

Each image in the dataset is of a different size. The resizing will make_
sure

each image is the same size.

"""

resize the image and the semantic segmentation mask

image = tf.image.resize(image, [image_size, image_size])

image = mobilenet_preprocess_image(image)

mask = tf.image.resize(output[1], [image_size, image_size])

return image, (output[0], mask)

def flip_lr_augmentation(image, output, flip_lr_prob):

```

""" function to return perform left-right flip augmentation

The function will flip the image along the left-right axis with
a defined probability.
"""

# randomly sample a value between 0 and 1
uniform_sample = tf.random.uniform([], minval=0, maxval=1)
# perform flip_lr with probability given by flip_lr_prob
flip_lr_cond = tf.math.less(uniform_sample, flip_lr_prob)
# output is a tuple of (class, segmentation_mask), pull out the
↪ segmentation mask
seg = output[1]

# wrapper fn for when we do the flip
def flip():
    flipped_image = tf.image.flip_left_right(image)
    flipped_seg = tf.image.flip_left_right(seg)
    return flipped_image, flipped_seg

# wrapper fn for when we do NOT flip
def no_flip():
    return image, seg

# apply augmentation
image, seg = tf.cond(flip_lr_cond, flip, no_flip)
# return the image, and output
return image, (output[0], seg)

def select_tasks(image, output, classification=True, segmentation=True):
    """select the tasks to include the data

    By default for each input there are two outputs. This function allows
    you to select which outputs to use, so the problem can be reduced to a
    single task problem for initial experimenting.
    """

    # both tasks
    if classification and segmentation:
        return image, output
    # just classification
    elif classification:
        return image, output[0]
    # just segmentation
    elif segmentation:
        return image, output[1]
    # neither task, doesn't really make sense, so return the image
    # for a self-supervised task

```

```

else:
    return image, image

class TrainForTime(keras.callbacks.Callback):
    """callback to terminate training after a time limit is reached

    Can be used to control how long training runs for, and will terminate
    training once a specified time limit is reached.
    """
    def __init__(
        self,
        train_time_mins=15,
    ):
        super().__init__()

        self.train_time_mins = train_time_mins
        self.epochs = 0
        self.train_time = 0
        self.end_early = False

    def on_train_begin(self, logs=None):
        # save the start time
        self.start_time = tf.timestamp()

    def on_epoch_end(self, epoch, logs=None):
        self.epochs += 1
        current_time = tf.timestamp()
        training_time = (current_time - self.start_time)
        if (training_time / 60) > self.train_time_mins:
            self.train_time = current_time - self.start_time
            self.model.stop_training = True
            self.end_early = True

    def on_train_end(self, logs=None):
        if self.end_early:
            print('training time exceeded and ending early')
            print(f'training ended on epoch {self.epochs}')
            print(f'training time = {self.train_time / 60} mins')

```

1.1.2 Data Loader

We will now put the above functions together into a data loader that we can use to feed directly to the network. You can use this directly as it is. However, you may modify it to add some additional functionality such as further data augmentations.

```
[3]: def load_oxford_pets(split,
                        batch_size=233,
                        classification=True,
                        segmentation=True,
                        shuffle=True,
                        augment=True,
                        image_size=300):
    """Load Oxford pets dataset for Assignment 1B

    Function handles loading of data for 1b, included processing of images and
    semantic segmentation masks. This function will
    organise the tensorflow dataset to return an output that is a tuple, where
    the tuple will be (classification_labels, segmentation_masks).

    Parameters
    -----
    split : string
        either train, val or test string
    classification : bool
        whether to include classification labels
    segmentation : bool
        whether to include semantic segmentation masks
    batch_size : int
        size of batches to use
    shuffle : bool
        whether to shuffle the dataset (WILL ONLY APPLY TO TRAIN)
    augment : bool
        whether to augment the dataset (WILL ONLY APPLY TO TRAIN)
    image_size : int
        new image size

    Returns
    -----
        tf.Dataset containing the Oxford pets dataset
    """

    # lets do some error checking first
    # Check for a valid dataset split, this must be train or test
    if (split != 'train') and (split != 'val') and (split != 'test'):
        raise ValueError('Arg for split must be either \'train\' or \'test\'')
    if (not classification) and (not segmentation):
        print("WARNING: One of the tasks (classification and segmentation) must
        ↪ be selected")
        print("Setting both to enabled")
        classification = True
        segmentation = True
```



```

# check that if using the val split, shuffle if false. If not, print a
↳warning and force shuffle to be false
if (split == 'val') and shuffle:
    print("WARNING: shuffle is set to true, but have specified split to be
↳\ 'val\ '")
    print('The shuffle argument will be ignored')
    shuffle = False

# check that if using the test split, shuffle if false. If not, print a
↳warning and force shuffle to be false
if (split == 'test') and shuffle:
    print("WARNING: shuffle is set to true, but have specified split to be
↳\ 'test\ '")
    print('The shuffle argument will be ignored')
    shuffle = False

# check that if using the val split, augment if false. If not, print a
↳warning and force augment to be false
if (split == 'val') and augment:
    print("WARNING: augment is set to true, but have specified split to be
↳\ 'val\ '")
    print('The augment argument will be ignored')
    augment = False

# check that if using the test split, augment if false. If not, print a
↳warning and force augment to be false
if (split == 'test') and augment:
    print("WARNING: augment is set to true, but have specified split to be
↳\ 'test\ '")
    print('The augment argument will be ignored')
    augment = False

# the dataset by default only has train and test splits. If val is
↳requested, pull the first 30% of the test set
if (split == 'val'):
    split = 'test[:30%]'
# the test set then becomes the remaining 70% of the original test set
elif (split == 'test'):
    split = 'test[30%:]'

# now start loading the dataset
ds = tfds.load('oxford_iiit_pet',
               split=split,
               with_info=False)

# remove unnecessary dataset info

```

```

ds = ds.map(return_image_label_mask)

# augmentation
# only apply if in the training split and augment has been set to True
if split == 'train' and augment:
    # apply a left-right flip with 50% probability
    flip_lr_prob = 0.5
    # flip operation
    ds = ds.map(lambda inp, out: flip_lr_augmentation(inp, out,
↪flip_lr_prob), num_parallel_calls=tf.data.AUTOTUNE)

    # more augmentation operations could go here .....

# Final processing of the data
# here we will resize the data, and add the preprocessing that is needed
↪for compatability with the mobilenet models.
ds = ds.map(lambda inp, out: preprocess_and_resize(inp, out, image_size))

# and now remove any tasks that we don't want. Note that we call this last
↪as it means that all the other functions
# can safely assume that data for both tasks is in the dataset
ds = ds.map(lambda inp, out: select_tasks(inp, out, classification,
↪segmentation))

# if in the training split, and shuffle is true, shuffle the data
if split == 'train' and shuffle:
    ds = ds.shuffle(1000)

# return the loaded and processed dataset
return ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)

```

1.1.3 Testing the provided data loader.

We'll now test the data loader and plot some examples to confirm it's working. **NOTE:** some poor defaults are specified below for image size and batch size. Set these to something more appropriate.

```

[4]: # testing the data loader and plotting some images.
# NOTE: the image size set here is all but definitely too large. You will need
# to chage this yourself to something that is suitable given your constraints
# NOTE: The batch size is also too large. This done on purpose force you to
# pick a suitable batch size yourself
image_size = 300
batch_size = 273
# load training data, note that shuffle and augment are true
train_class_seg = load_oxford_pets('train', classification=True,
↪segmentation=True, shuffle=True, augment=True,

```

```

                                batch_size=batch_size, image_size=image_size)
# load validation data, note that shuffle and augment are false (though if they
↳weren't, the data loader would force these to be false)
val_class_seg = load_oxford_pets('val', classification=True, segmentation=True,
↳shuffle=False, augment=False,
                                batch_size=batch_size, image_size=image_size)
# load testing data, note that shuffle and augment are false (though if they
↳weren't, the data loader would force these to be false)
test_class_seg = load_oxford_pets('test', classification=True,
↳segmentation=True, shuffle=False, augment=False,
                                batch_size=batch_size, image_size=image_size)

```

WARNING:absl:Variant folder /tmp/tensorflow_datasets/oxford_iiit_pet/4.0.0 has no dataset_info.json

Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /tmp/tensorflow_datasets/oxford_iiit_pet/4.0.0...

Dl Completed...: 0 url [00:00, ? url/s]

Dl Size...: 0 MiB [00:00, ? MiB/s]

Extraction completed...: 0 file [00:00, ? file/s]

IOPub message rate exceeded.

The Jupyter server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

`--ServerApp.iopub_msg_rate_limit`.

Current values:

ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)

ServerApp.rate_limit_window=3.0 (secs)

IOPub message rate exceeded.

The Jupyter server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

`--ServerApp.iopub_msg_rate_limit`.

Current values:

ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)

ServerApp.rate_limit_window=3.0 (secs)

IOPub message rate exceeded.

The Jupyter server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

```
`--ServerApp.iopub_msg_rate_limit`.
```

Current values:

```
ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
```

```
ServerApp.rate_limit_window=3.0 (secs)
```

IOPub message rate exceeded.

The Jupyter server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

```
`--ServerApp.iopub_msg_rate_limit`.
```

Current values:

```
ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
```

```
ServerApp.rate_limit_window=3.0 (secs)
```

IOPub message rate exceeded.

The Jupyter server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

```
`--ServerApp.iopub_msg_rate_limit`.
```

Current values:

```
ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
```

```
ServerApp.rate_limit_window=3.0 (secs)
```

```
I0000 00:00:1746782577.943900      129 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 1880 MB memory:  -> device: 0,
name: NVIDIA A16-4Q, pci bus id: 0000:02:03.0, compute capability: 8.6
Corrupt JPEG data: premature end of data segment
Corrupt JPEG data: 240 extraneous bytes before marker 0xd9
```

```
Shuffling /tmp/tensorflow_datasets/oxford_iiit_pet/incomplete.4NH8FR_4.0.0/
↳oxford_iiit_pet-train.tfrecord*...:...
```

```
Generating test examples...: 0 examples [00:00, ? examples/s]
```

```
Shuffling /tmp/tensorflow_datasets/oxford_iiit_pet/incomplete.4NH8FR_4.0.0/
↳oxford_iiit_pet-test.tfrecord*...: ...
```

Dataset oxford_iiit_pet downloaded and prepared to

/tmp/tensorflow_datasets/oxford_iiit_pet/4.0.0. Subsequent calls will reuse this data.

```
[5]: # lets plot a few now to see some good kittens/doggos
fig, axs = plt.subplots(2, 3, figsize=(8, 4), layout="constrained")
num_plot = 3
i = 0
```

```

# each sample of our dataset will be of the format
# image, outputs
# where outputs[0] = label
#     outputs[1] = segmentation mask
#
#
# lets get a single batch, and plot just a few of them
for image, output in train_class_seg.take(1).as_numpy_iterator():
    for i in range(num_plot):
        im = axs[0, i].imshow(np.squeeze(unprocess_image(image[i, ...])) / 255.
        ↪0)

        axs[0, i].set_title(output[0][i])
        axs[0, i].axis('off')
        im = axs[1, i].imshow(np.squeeze(output[1][i, ...]))
        axs[1, i].axis('off')

        print(output[1].shape)
        i += 1
        if i >= num_plot:
            break

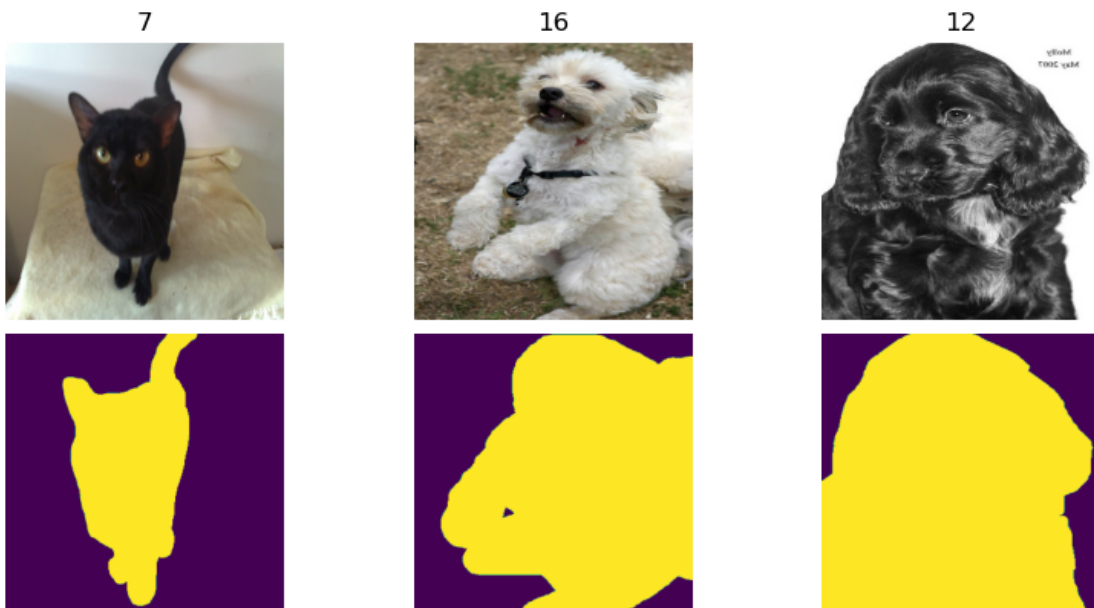
plt.savefig('doggos_cattos.png')

```

```

(273, 300, 300, 1)
(273, 300, 300, 1)
(273, 300, 300, 1)

```



The images are showing correctly.

NOTE: You can ignore the JPEG warning.

We can use the `classification` and `segmentation` flags to pull out just one output as well, as the below demonstrates.

```
[6]: # classification only; classification = True, segmentation = False (note batch_
      ↪size is 1 here)
train_class_only = load_oxford_pets('train', classification=True,
      ↪segmentation=False, shuffle=True, augment=True, batch_size=1,
      ↪image_size=image_size)
# segmentation only; classification = False, segmentation = True (note batch_
      ↪size is 1 here)
train_seg_only = load_oxford_pets('train', classification=False,
      ↪segmentation=True, shuffle=True, augment=True, batch_size=1,
      ↪image_size=image_size)

# test the classification only dataset
# pull out one element
inp, out = next(iter(train_class_only))
# print the output
print(out.numpy())

# test the segmentation only dataset
# pull out one element
inp, out = next(iter(train_seg_only))
# print just the output shape for the segmentation output
print(out.numpy().shape)
```

```
[1]
(1, 300, 300, 1)
```

While for the question you do need to train networks to do both tasks simultaneously, when you starting playing with the problem it might be easier to get things working for one task, and then add the second.

1.1.4 Loading MobileNetV3Small base for fine tuning

This model can be loaded directly from keras. By default, the model we download will be pre-trained on Imagenet dataset.

Note that we will need to set the preprocessing option when loading this base network to False. This is because the `include_preprocessing` step is implemented in the Datasets we defined above.

We also set `include_top=False`, to avoid loading our model with the final Dense classification layer which is used for the original Imagenet model.

More details are available in the keras documentation [here](#).

```
[7]: mobile_base = keras.applications.MobileNetV3Small(input_shape=(image_size,
    ↪image_size, 3),
                                           include_top=False,
                                           include_preprocessing=False)
```

```
/opt/conda/lib/python3.11/site-
packages/keras/src/applications/mobilenet_v3.py:452: UserWarning: `input_shape`
is undefined or non-square, or `rows` is not 224. Weights for input shape (224,
224) will be loaded as the default.
```

```
    return MobileNetV3(
```

For this task, can ignore the `input_shape` warning, though it is important to keep in mind the difference in size of data used for the pre-trained model and our data may have an impact on our model (what that impact might be is for you to investigate :). Depending on what input shape you select you may also be able to eliminate this.

For more information on fine-tuning models, can refer to many of the examples from class, or the [Keras documentation](#)

1.2 Question 2 Template

The following provides a starting point for your solution, and some suggestions on how to proceed.

PLEASE READ THE COMMENTS CAREFULLY There are a number of tips around how to proceed, and things to look out for.

1.2.1 Data loading

```
[8]: # Step 1: Data Loading with Proper Settings

image_size = 128    # Resized for computational efficiency
batch_size = 32     # Suitable for training on most GPUs/CPUs

# Load datasets with both classification and segmentation enabled
train_class_seg = load_oxford_pets(
    'train', classification=True, segmentation=True, shuffle=True, augment=True,
    batch_size=batch_size, image_size=image_size)

val_class_seg = load_oxford_pets(
    'val', classification=True, segmentation=True, shuffle=False, augment=False,
    batch_size=batch_size, image_size=image_size)

# Load test data for final evaluation
test_class_seg = load_oxford_pets(
    'test', classification=True, segmentation=True, shuffle=False,
    ↪augment=False,
    batch_size=batch_size, image_size=image_size)
```

```
[9]: # Load a single image from the raw dataset
raw_dataset = tfds.load('oxford_iiit_pet', split='train[:1]')
raw_example = next(iter(raw_dataset))

# Preprocess using your existing pipeline
image, (label, seg_mask) = return_image_label_mask(raw_example)
image_resized, (label_proc, mask_proc) = preprocess_and_resize(image, (label,
↪seg_mask), image_size=128)

# Convert tensors to numpy
raw_image_np = image.numpy().astype(np.uint8)
proc_image_np = ((image_resized.numpy() + 1) * 127.5).astype(np.uint8) # undo
↪MobileNet preprocessing

# Plot and compare
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plt.imshow(raw_image_np)
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(proc_image_np)
plt.title("Pre-processed Image")
plt.axis('off')

plt.tight_layout()
plt.savefig("original_vs_preprocessed.png")
plt.show()
```

Original Image



Pre-processed Image




```
[10]: # Step 2: Define the Multi-Task Model

from tensorflow.keras import layers, models, Input

# Input layer for images
inputs = Input(shape=(image_size, image_size, 3))

# Shared convolutional base (encoder)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
x = layers.MaxPooling2D()(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D()(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D()(x)

[11]: # Flatten for classification
flat = layers.Flatten()(x)

# -----
# Output Head 1: Classification
# -----
# Predicts pet breed (37 classes in Oxford-IIIT Pets dataset)
class_output = layers.Dense(128, activation='relu')(flat)
class_output = layers.Dense(37, activation='softmax',
                             name='classification')(class_output)

# -----
# Output Head 2: Segmentation
# -----
# Decode using upsampling to predict binary mask
seg_x = layers.Conv2D(64, (3,3), activation='relu', padding='same')(x)
seg_x = layers.UpSampling2D()(seg_x)
seg_x = layers.Conv2D(32, (3,3), activation='relu', padding='same')(seg_x)
seg_x = layers.UpSampling2D()(seg_x)
seg_x = layers.Conv2D(16, (3,3), activation='relu', padding='same')(seg_x)
seg_x = layers.UpSampling2D()(seg_x)

# Output segmentation mask (1 channel with sigmoid)
seg_output = layers.Conv2D(1, (1,1), activation='sigmoid',
                             name='segmentation')(seg_x)

# Final model
model_a = models.Model(inputs=inputs, outputs=[class_output, seg_output])
```

```
[12]: # Step 3: Compile the Model
```

```
model_a.compile(
    optimizer='adam',
    loss={
        'classification': 'sparse_categorical_crossentropy',
        'segmentation': 'binary_crossentropy'
    },
    metrics={
        'classification': 'accuracy',
        'segmentation': 'accuracy'
    }
)
```

```
model_a.summary()
```

```
# -----
# Step 4: Train the Model
# -----
```

```
history_a = model_a.fit(
    train_class_seg,
    validation_data=val_class_seg,
    epochs=10
)
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d (Conv2D)	(None, 128, 128, 32)	896	input_layer_1[0]...
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 64, 64, 64)	18,496	max_pooling2d[0]...
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73,856	max_pooling2d_1[...

max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 16, 16, 64)	73,792	max_pooling2d_2[...]
up_sampling2d (UpSampling2D)	(None, 32, 32, 64)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 32, 32, 32)	18,464	up_sampling2d[0]...
up_sampling2d_1 (UpSampling2D)	(None, 64, 64, 32)	0	conv2d_4[0][0]
flatten (Flatten)	(None, 32768)	0	max_pooling2d_2[...]
conv2d_5 (Conv2D)	(None, 64, 64, 16)	4,624	up_sampling2d_1[...]
dense (Dense)	(None, 128)	4,194,432	flatten[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 128, 128, 16)	0	conv2d_5[0][0]
classification (Dense)	(None, 37)	4,773	dense[0][0]
segmentation (Conv2D)	(None, 128, 128, 1)	17	up_sampling2d_2[...]

Total params: 4,389,350 (16.74 MB)

Trainable params: 4,389,350 (16.74 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

I0000 00:00:1746782593.698083 259 service.cc:148] XLA service 0x7f05cc0047e0 initialized for platform CUDA (this does not guarantee that XLA will be used).

Devices:

I0000 00:00:1746782593.698123 259 service.cc:156] StreamExecutor device
(0): NVIDIA A16-4Q, Compute Capability 8.6
I0000 00:00:1746782594.078516 259 cuda_dnn.cc:529] Loaded cuDNN version
90300

5/115 3s 34ms/step -
classification_accuracy: 0.0264 - classification_loss: 3.6684 - loss: 4.3597 -
segmentation_accuracy: 0.5393 - segmentation_loss: 0.6913

I0000 00:00:1746782601.869983 259 device_compiler.h:188] Compiled cluster
using XLA! This line is logged at most once for the lifetime of the process.

115/115 19s 73ms/step -
classification_accuracy: 0.0363 - classification_loss: 3.6107 - loss: 4.2401 -
segmentation_accuracy: 0.6274 - segmentation_loss: 0.6293 -
val_classification_accuracy: 0.0554 - val_classification_loss: 3.4973 -
val_loss: 3.9825 - val_segmentation_accuracy: 0.7667 - val_segmentation_loss:
0.4874

Epoch 2/10

115/115 6s 43ms/step -
classification_accuracy: 0.0885 - classification_loss: 3.4185 - loss: 3.9118 -
segmentation_accuracy: 0.7603 - segmentation_loss: 0.4932 -
val_classification_accuracy: 0.0872 - val_classification_loss: 3.3533 -
val_loss: 3.8048 - val_segmentation_accuracy: 0.7882 - val_segmentation_loss:
0.4570

Epoch 3/10

115/115 6s 44ms/step -
classification_accuracy: 0.1274 - classification_loss: 3.2080 - loss: 3.6571 -
segmentation_accuracy: 0.7872 - segmentation_loss: 0.4491 -
val_classification_accuracy: 0.1326 - val_classification_loss: 3.2542 -
val_loss: 3.6871 - val_segmentation_accuracy: 0.7997 - val_segmentation_loss:
0.4388

Epoch 4/10

115/115 6s 43ms/step -
classification_accuracy: 0.2050 - classification_loss: 2.9451 - loss: 3.3756 -
segmentation_accuracy: 0.7988 - segmentation_loss: 0.4305 -
val_classification_accuracy: 0.1444 - val_classification_loss: 3.1539 -
val_loss: 3.5675 - val_segmentation_accuracy: 0.8082 - val_segmentation_loss:
0.4165

Epoch 5/10

115/115 6s 43ms/step -
classification_accuracy: 0.2788 - classification_loss: 2.6333 - loss: 3.0511 -
segmentation_accuracy: 0.8056 - segmentation_loss: 0.4178 -
val_classification_accuracy: 0.1789 - val_classification_loss: 3.0987 -
val_loss: 3.5025 - val_segmentation_accuracy: 0.8127 - val_segmentation_loss:
0.4100

Epoch 6/10

115/115 6s 43ms/step -
classification_accuracy: 0.3441 - classification_loss: 2.2848 - loss: 2.6896 -

```

segmentation_accuracy: 0.8139 - segmentation_loss: 0.4048 -
val_classification_accuracy: 0.1807 - val_classification_loss: 3.2426 -
val_loss: 3.6521 - val_segmentation_accuracy: 0.8126 - val_segmentation_loss:
0.4104
Epoch 7/10
115/115          8s 57ms/step -
classification_accuracy: 0.4263 - classification_loss: 2.0248 - loss: 2.4289 -
segmentation_accuracy: 0.8140 - segmentation_loss: 0.4041 -
val_classification_accuracy: 0.1644 - val_classification_loss: 3.6054 -
val_loss: 3.9899 - val_segmentation_accuracy: 0.8246 - val_segmentation_loss:
0.3870
Epoch 8/10
115/115          8s 57ms/step -
classification_accuracy: 0.5456 - classification_loss: 1.6575 - loss: 2.0483 -
segmentation_accuracy: 0.8209 - segmentation_loss: 0.3907 -
val_classification_accuracy: 0.1798 - val_classification_loss: 3.5810 -
val_loss: 3.9692 - val_segmentation_accuracy: 0.8217 - val_segmentation_loss:
0.3989
Epoch 9/10
115/115          7s 43ms/step -
classification_accuracy: 0.6153 - classification_loss: 1.3243 - loss: 1.7199 -
segmentation_accuracy: 0.8200 - segmentation_loss: 0.3956 -
val_classification_accuracy: 0.1889 - val_classification_loss: 3.8099 -
val_loss: 4.1786 - val_segmentation_accuracy: 0.8306 - val_segmentation_loss:
0.3811
Epoch 10/10
115/115          6s 42ms/step -
classification_accuracy: 0.6855 - classification_loss: 1.0683 - loss: 1.4391 -
segmentation_accuracy: 0.8321 - segmentation_loss: 0.3708 -
val_classification_accuracy: 0.1753 - val_classification_loss: 4.0927 -
val_loss: 4.4505 - val_segmentation_accuracy: 0.8356 - val_segmentation_loss:
0.3687

```

```

[13]: #
# From-scratch DCNN
#
# Put your from-scratch DCNN here. This network will have one input (the input_
↪ image), and two outputs (the class, and the segmentation map). You can
# think of your model as having three components:
# - A backbone, which is going to take your input and encode that. This will_
↪ possibly be collection of (probably mostly) 2D convolution layers
#   and max pooling layers (and for which you may be able to find something_
↪ fairly fit for purpose in lecture or prac examples).
# - A classification head, which will take the backbone output and via one or_
↪ more dense layers do the classification task. This will likely only
#   be a few layers at most.

```

```

# - A decoder head, which will do the semantic segmentation. This will look a
↳lot like the back half of an autoencoder.
#
# As usual, the network does not need to be overly complex, but you will need
↳to briefly explain your selection of this model, and you should avoid
# networks so simple that they perform very badly.
#
# Take note of the order that the outputs appear in the data loader. The data
↳loader will give you (classification_output, segmentation_output), so
# your network should have the outputs in the same order, and your losses
↳should also be in the same order. With your losses, note that:
# * For the classification task ground truth data is being given as class
↳labels, so you should use Sparse Categorical Cross Entropy.
# * For the segmentation loss, there are a few options you could use, but a
↳good starting point is simply binary cross entropy, essentially
# every pixel is a binary classification task (foreground or background)
# * The order of your losses is critical. The dataset object will give you
↳(classification_targets, segmentation_targets). This means that you're
# network should it's outputs, and loss functions, in the same order - if you
↳get this order wrong you will get weird errors
#
# When first developing your model, you may want to start by getting a single
↳output of the network going, and then adding the second output. For
# example, you may build a network with the image input and the just the
↳classification head, get that running, and then add the decoder head. You can
# use the classification and segmentation flags in the data loader to turn off
↳one of the outputs.

# For model training, we have provided a callback, TrainForTime, which will
↳terminate training after some time limit is reached. You could use this
# as follows:
#
# train_time = 15 # 15-minute training limit
# my_awesome_model.fit(train_class_seg, epochs=epochs, verbose=True,
↳validation_data=val_class_seg, callbacks=[TrainForTime(train_time)])
#
# You don't have to use this, but you may wish to use this to ensure training
↳doesn't take too long.
# Note that if you don't train until convergence, you will need to explain what
↳you observe with training and what the implications of this are.

```

1.2.2 MODEL B – TRANSFER LEARNING WITH MOBILENETV3SMALL

```

[14]: from tensorflow.keras.applications import MobileNetV3Small
from tensorflow.keras import layers, models, Input

```

```
[15]: # Step 1: Load Pretrained MobileNetV3Small as Backbone

# We exclude the top classification layer and use pretrained ImageNet weights
mobilenet_backbone = MobileNetV3Small(
    input_shape=(image_size, image_size, 3),
    include_top=False,
    weights='imagenet'
)

# Freeze backbone to start with (you can unfreeze later to fine-tune)
mobilenet_backbone.trainable = False

# Input layer
inputs = Input(shape=(image_size, image_size, 3))

# Pass input through MobileNetV3Small
x = mobilenet_backbone(inputs)

# Shared feature map output by backbone
# Shape here is (None, 4, 4, 576) due to downsampling

[16]: # Classification Head
# -----
flat = layers.GlobalAveragePooling2D()(x)
class_head = layers.Dense(128, activation='relu')(flat)
class_output = layers.Dense(37, activation='softmax',
    ↪name='classification')(class_head)

# Segmentation Head (Upsampling Decoder)
# -----
# Use transpose convolutions or upsampling layers to reach original image size
    ↪(128x128)
# Segmentation Head (Updated to match 128x128 output)
seg_head = layers.Conv2D(256, (3,3), padding='same', activation='relu')(x)
seg_head = layers.UpSampling2D()(seg_head) # 4x4 → 8x8
seg_head = layers.Conv2D(128, (3,3), padding='same',
    ↪activation='relu')(seg_head)
seg_head = layers.UpSampling2D()(seg_head) # 8x8 → 16x16
seg_head = layers.Conv2D(64, (3,3), padding='same', activation='relu')(seg_head)
seg_head = layers.UpSampling2D()(seg_head) # 16x16 → 32x32
seg_head = layers.Conv2D(32, (3,3), padding='same', activation='relu')(seg_head)
seg_head = layers.UpSampling2D()(seg_head) # 32x32 → 64x64
seg_head = layers.Conv2D(16, (3,3), padding='same', activation='relu')(seg_head)
seg_head = layers.UpSampling2D()(seg_head) # 64x64 → 128x128

# Output segmentation mask
```

```
seg_output = layers.Conv2D(1, (1,1), activation='sigmoid',  
↪name='segmentation')(seg_head)
```

```
[17]: # Final Model (Multi-Output)

model_b = models.Model(inputs=inputs, outputs=[class_output, seg_output])

# Compile with appropriate loss functions and metrics
model_b.compile(
    optimizer='adam',
    loss={
        'classification': 'sparse_categorical_crossentropy',
        'segmentation': 'binary_crossentropy'
    },
    metrics={
        'classification': 'accuracy',
        'segmentation': 'accuracy'
    }
)

# Print model summary
model_b.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 128, 128, 3)	0	-
MobileNetV3Small (Functional)	(None, 4, 4, 576)	939,120	input_layer_3[0]...
conv2d_6 (Conv2D)	(None, 4, 4, 256)	1,327,360	MobileNetV3Small...
up_sampling2d_3 (UpSampling2D)	(None, 8, 8, 256)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 8, 8, 128)	295,040	up_sampling2d_3[...
up_sampling2d_4 (UpSampling2D)	(None, 16, 16, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 64)	73,792	up_sampling2d_4[...

up_sampling2d_5 (UpSampling2D)	(None, 32, 32, 64)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 32)	18,464	up_sampling2d_5[...]
up_sampling2d_6 (UpSampling2D)	(None, 64, 64, 32)	0	conv2d_9[0][0]
global_average_poo... (GlobalAveragePool...)	(None, 576)	0	MobileNetV3Small...
conv2d_10 (Conv2D)	(None, 64, 64, 16)	4,624	up_sampling2d_6[...]
dense_1 (Dense)	(None, 128)	73,856	global_average_p...
up_sampling2d_7 (UpSampling2D)	(None, 128, 128, 16)	0	conv2d_10[0][0]
classification (Dense)	(None, 37)	4,773	dense_1[0][0]
segmentation (Conv2D)	(None, 128, 128, 1)	17	up_sampling2d_7[...]

Total params: 2,737,046 (10.44 MB)

Trainable params: 1,797,926 (6.86 MB)

Non-trainable params: 939,120 (3.58 MB)

```
[18]: # Step 2: Train the Model

from tensorflow.keras.callbacks import Callback
import time

# Custom callback to stop training after a set number of minutes
class TrainForTime(Callback):
    def __init__(self, minutes=15):
        super().__init__()
        self.train_for = minutes * 60 # convert to seconds

    def on_train_begin(self, logs=None):
```

```

        self.start_time = time.time()

    def on_batch_end(self, batch, logs=None):
        if time.time() - self.start_time > self.train_for:
            self.model.stop_training = True
            print("\nTraining stopped due to time constraint.")

# Set training time limit
train_time_limit = 15 # in minutes

# Train the model with a time-based callback
history_b = model_b.fit(
    train_class_seg,
    validation_data=val_class_seg,
    epochs=20, # can be higher, but will stop early if time exceeded
    callbacks=[TrainForTime(train_time_limit)]
)

```

Epoch 1/20

```

115/115          35s 153ms/step -
classification_accuracy: 0.0304 - classification_loss: 3.6411 - loss: 4.2043 -
segmentation_accuracy: 0.7178 - segmentation_loss: 0.5632 -
val_classification_accuracy: 0.0227 - val_classification_loss: 3.6134 -
val_loss: 4.1404 - val_segmentation_accuracy: 0.7391 - val_segmentation_loss:
0.5267

```

Epoch 2/20

```

115/115          8s 54ms/step -
classification_accuracy: 0.0280 - classification_loss: 3.6130 - loss: 4.1242 -
segmentation_accuracy: 0.7472 - segmentation_loss: 0.5112 -
val_classification_accuracy: 0.0254 - val_classification_loss: 3.6105 -
val_loss: 4.1222 - val_segmentation_accuracy: 0.7474 - val_segmentation_loss:
0.5118

```

Epoch 3/20

```

115/115          8s 54ms/step -
classification_accuracy: 0.0233 - classification_loss: 3.6112 - loss: 4.1146 -
segmentation_accuracy: 0.7528 - segmentation_loss: 0.5034 -
val_classification_accuracy: 0.0318 - val_classification_loss: 3.6108 -
val_loss: 4.1048 - val_segmentation_accuracy: 0.7579 - val_segmentation_loss:
0.4942

```

Epoch 4/20

```

115/115          8s 54ms/step -
classification_accuracy: 0.0238 - classification_loss: 3.6114 - loss: 4.1048 -
segmentation_accuracy: 0.7572 - segmentation_loss: 0.4934 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0984 - val_segmentation_accuracy: 0.7620 - val_segmentation_loss:
0.4879

```

Epoch 5/20

```

115/115          8s 55ms/step -

```

classification_accuracy: 0.0185 - classification_loss: 3.6114 - loss: 4.1023 -
segmentation_accuracy: 0.7591 - segmentation_loss: 0.4909 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6108 -
val_loss: 4.1120 - val_segmentation_accuracy: 0.7544 - val_segmentation_loss:
0.5017

Epoch 6/20

115/115 8s 55ms/step -

classification_accuracy: 0.0238 - classification_loss: 3.6113 - loss: 4.0998 -
segmentation_accuracy: 0.7617 - segmentation_loss: 0.4885 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0937 - val_segmentation_accuracy: 0.7644 - val_segmentation_loss:
0.4830

Epoch 7/20

115/115 8s 55ms/step -

classification_accuracy: 0.0208 - classification_loss: 3.6114 - loss: 4.0938 -
segmentation_accuracy: 0.7646 - segmentation_loss: 0.4824 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0890 - val_segmentation_accuracy: 0.7666 - val_segmentation_loss:
0.4783

Epoch 8/20

115/115 8s 54ms/step -

classification_accuracy: 0.0254 - classification_loss: 3.6114 - loss: 4.0820 -
segmentation_accuracy: 0.7723 - segmentation_loss: 0.4705 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0924 - val_segmentation_accuracy: 0.7671 - val_segmentation_loss:
0.4816

Epoch 9/20

115/115 8s 55ms/step -

classification_accuracy: 0.0190 - classification_loss: 3.6114 - loss: 4.0874 -
segmentation_accuracy: 0.7695 - segmentation_loss: 0.4759 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0989 - val_segmentation_accuracy: 0.7654 - val_segmentation_loss:
0.4879

Epoch 10/20

115/115 8s 53ms/step -

classification_accuracy: 0.0181 - classification_loss: 3.6113 - loss: 4.0789 -
segmentation_accuracy: 0.7743 - segmentation_loss: 0.4677 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0796 - val_segmentation_accuracy: 0.7733 - val_segmentation_loss:
0.4690

Epoch 11/20

115/115 8s 54ms/step -

classification_accuracy: 0.0176 - classification_loss: 3.6113 - loss: 4.0757 -
segmentation_accuracy: 0.7757 - segmentation_loss: 0.4644 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0790 - val_segmentation_accuracy: 0.7740 - val_segmentation_loss:
0.4687

Epoch 12/20

115/115 8s 53ms/step -
classification_accuracy: 0.0247 - classification_loss: 3.6113 - loss: 4.0740 -
segmentation_accuracy: 0.7771 - segmentation_loss: 0.4627 -
val_classification_accuracy: 0.0300 - val_classification_loss: 3.6108 -
val_loss: 4.0724 - val_segmentation_accuracy: 0.7793 - val_segmentation_loss:
0.4621
Epoch 13/20
115/115 8s 55ms/step -
classification_accuracy: 0.0207 - classification_loss: 3.6113 - loss: 4.0711 -
segmentation_accuracy: 0.7788 - segmentation_loss: 0.4598 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6108 -
val_loss: 4.0760 - val_segmentation_accuracy: 0.7754 - val_segmentation_loss:
0.4652
Epoch 14/20
115/115 8s 54ms/step -
classification_accuracy: 0.0184 - classification_loss: 3.6114 - loss: 4.0701 -
segmentation_accuracy: 0.7801 - segmentation_loss: 0.4586 -
val_classification_accuracy: 0.0227 - val_classification_loss: 3.6109 -
val_loss: 4.0689 - val_segmentation_accuracy: 0.7803 - val_segmentation_loss:
0.4587
Epoch 15/20
115/115 8s 54ms/step -
classification_accuracy: 0.0191 - classification_loss: 3.6119 - loss: 4.0673 -
segmentation_accuracy: 0.7811 - segmentation_loss: 0.4554 -
val_classification_accuracy: 0.0282 - val_classification_loss: 3.6109 -
val_loss: 4.0673 - val_segmentation_accuracy: 0.7830 - val_segmentation_loss:
0.4566
Epoch 16/20
115/115 7s 47ms/step -
classification_accuracy: 0.0267 - classification_loss: 3.6113 - loss: 4.0662 -
segmentation_accuracy: 0.7817 - segmentation_loss: 0.4548 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0595 - val_segmentation_accuracy: 0.7853 - val_segmentation_loss:
0.4488
Epoch 17/20
115/115 6s 41ms/step -
classification_accuracy: 0.0164 - classification_loss: 3.6117 - loss: 4.0646 -
segmentation_accuracy: 0.7819 - segmentation_loss: 0.4529 -
val_classification_accuracy: 0.0263 - val_classification_loss: 3.6109 -
val_loss: 4.0600 - val_segmentation_accuracy: 0.7880 - val_segmentation_loss:
0.4491
Epoch 18/20
115/115 6s 40ms/step -
classification_accuracy: 0.0225 - classification_loss: 3.6117 - loss: 4.0648 -
segmentation_accuracy: 0.7820 - segmentation_loss: 0.4531 -
val_classification_accuracy: 0.0227 - val_classification_loss: 3.6109 -
val_loss: 4.0558 - val_segmentation_accuracy: 0.7860 - val_segmentation_loss:
0.4455

Epoch 19/20

115/115 6s 40ms/step -

classification_accuracy: 0.0169 - classification_loss: 3.6113 - loss: 4.0565 -

segmentation_accuracy: 0.7874 - segmentation_loss: 0.4451 -

val_classification_accuracy: 0.0282 - val_classification_loss: 3.6109 -

val_loss: 4.0616 - val_segmentation_accuracy: 0.7849 - val_segmentation_loss: 0.4509

Epoch 20/20

115/115 6s 40ms/step -

classification_accuracy: 0.0188 - classification_loss: 3.6114 - loss: 4.0619 -

segmentation_accuracy: 0.7831 - segmentation_loss: 0.4505 -

val_classification_accuracy: 0.0263 - val_classification_loss: 3.6108 -

val_loss: 4.0575 - val_segmentation_accuracy: 0.7867 - val_segmentation_loss: 0.4465

1.2.3 Evaluation

Model A

```
[19]: from sklearn.metrics import classification_report, confusion_matrix, \
      accuracy_score, f1_score

# -----
# Extract Test Labels
# -----
y_class_test = np.concatenate([y[0] for x, y in test_class_seg.
                               ↪as_numpy_iterator()])
y_seg_test = np.concatenate([y[1] for x, y in test_class_seg.
                              ↪as_numpy_iterator()])

# -----
# Get Predictions from the Model

# -----
class_preds, seg_preds = model_a.predict(test_class_seg)

# -----
# Classification Evaluation
# -----
y_class_pred = np.argmax(class_preds, axis=1)
print("Classification Report for Model A:")
print(classification_report(y_class_test, y_class_pred))

# -----
# Segmentation Evaluation
# -----
seg_preds_flat = seg_preds.reshape(-1) > 0.5
seg_gt_flat = y_seg_test.reshape(-1) > 0.5
```

```

print("Segmentation Confusion Matrix for Model A:")
print(confusion_matrix(seg_gt_flat, seg_preds_flat))
print("Segmentation Pixel Accuracy for Model A:", accuracy_score(seg_gt_flat,
    ↪seg_preds_flat))
print("Segmentation F1 Score for Model A:", f1_score(seg_gt_flat,
    ↪seg_preds_flat))

```

81/81

4s 45ms/step

Classification Report for Model A:

	precision	recall	f1-score	support
0	0.22	0.29	0.25	69
1	0.14	0.29	0.19	66
2	0.02	0.01	0.02	70
3	0.23	0.34	0.27	65
4	0.10	0.07	0.08	75
5	0.38	0.22	0.28	67
6	0.21	0.33	0.25	70
7	0.40	0.32	0.36	60
8	0.12	0.13	0.13	67
9	0.28	0.08	0.12	64
10	0.15	0.05	0.07	66
11	0.42	0.15	0.22	68
12	0.08	0.01	0.02	73
13	0.16	0.14	0.15	72
14	0.17	0.29	0.22	76
15	0.21	0.17	0.19	64
16	0.19	0.15	0.17	67
17	0.29	0.26	0.27	70
18	0.29	0.20	0.23	71
19	0.30	0.23	0.26	70
20	0.19	0.30	0.23	66
21	0.13	0.22	0.16	64
22	0.35	0.50	0.41	70
23	0.22	0.14	0.17	77
24	0.28	0.18	0.22	73
25	0.18	0.09	0.12	66
26	0.24	0.16	0.19	73
27	0.23	0.37	0.28	71
28	0.26	0.31	0.28	70
29	0.31	0.43	0.36	75
30	0.12	0.35	0.18	68
31	0.14	0.16	0.15	75
32	0.26	0.14	0.18	73
33	0.16	0.16	0.16	67
34	0.12	0.11	0.12	63
35	0.12	0.06	0.08	78

	36	0.25	0.19	0.22	69
accuracy				0.20	2568
macro avg	0.21	0.21	0.20	0.20	2568
weighted avg	0.21	0.20	0.20	0.20	2568

Segmentation Confusion Matrix for Model A:

```
[[20701626  3520574]
 [ 3084314 14767598]]
```

Segmentation Pixel Accuracy for Model A: 0.8430177682656737

Segmentation F1 Score for Model A: 0.8172420407213221

Model B

```
[20]: # -----
# Extract Test Labels
# -----
y_class_test = np.concatenate([y[0] for x, y in test_class_seg.
    ↳as_numpy_iterator()])
y_seg_test = np.concatenate([y[1] for x, y in test_class_seg.
    ↳as_numpy_iterator()])

# -----
# Get Predictions from the Model

# -----
class_preds, seg_preds = model_b.predict(test_class_seg)

# -----
# Classification Evaluation
# -----
y_class_pred = np.argmax(class_preds, axis=1)
print("Classification Report for Model B:")
print(classification_report(y_class_test, y_class_pred))

# -----
# Segmentation Evaluation
# -----
seg_preds_flat = seg_preds.reshape(-1) > 0.5
seg_gt_flat = y_seg_test.reshape(-1) > 0.5

print("Segmentation Confusion Matrix for Model B:")
print(confusion_matrix(seg_gt_flat, seg_preds_flat))
print("Segmentation Pixel Accuracy for Model B:", accuracy_score(seg_gt_flat,
    ↳seg_preds_flat))
print("Segmentation F1 Score for Model B:", f1_score(seg_gt_flat,
    ↳seg_preds_flat))
```

81/81

16s 168ms/step

Classification Report for Model B:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	69
1	0.00	0.00	0.00	66
2	0.00	0.00	0.00	70
3	0.00	0.00	0.00	65
4	0.00	0.00	0.00	75
5	0.00	0.00	0.00	67
6	0.00	0.00	0.00	70
7	0.00	0.00	0.00	60
8	0.00	0.00	0.00	67
9	0.00	0.00	0.00	64
10	0.00	0.00	0.00	66
11	0.00	0.00	0.00	68
12	0.00	0.00	0.00	73
13	0.00	0.00	0.00	72
14	0.00	0.00	0.00	76
15	0.00	0.00	0.00	64
16	0.00	0.00	0.00	67
17	0.00	0.00	0.00	70
18	0.00	0.00	0.00	71
19	0.00	0.00	0.00	70
20	0.00	0.00	0.00	66
21	0.00	0.00	0.00	64
22	0.00	0.00	0.00	70
23	0.00	0.00	0.00	77
24	0.00	0.00	0.00	73
25	0.00	0.00	0.00	66
26	0.00	0.00	0.00	73
27	0.03	1.00	0.05	71
28	0.00	0.00	0.00	70
29	0.00	0.00	0.00	75
30	0.00	0.00	0.00	68
31	0.00	0.00	0.00	75
32	0.00	0.00	0.00	73
33	0.00	0.00	0.00	67
34	0.00	0.00	0.00	63
35	0.00	0.00	0.00	78
36	0.00	0.00	0.00	69
accuracy			0.03	2568
macro avg	0.00	0.03	0.00	2568
weighted avg	0.00	0.03	0.00	2568

Segmentation Confusion Matrix for Model B:

/opt/conda/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1565:

UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))  
/opt/conda/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1565:  
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels  
with no predicted samples. Use `zero_division` parameter to control this  
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))  
/opt/conda/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1565:  
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels  
with no predicted samples. Use `zero_division` parameter to control this  
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
[[21232240 2989960]
```

```
[ 5721220 12130692]]
```

Segmentation Pixel Accuracy for Model B: 0.7929562957858742

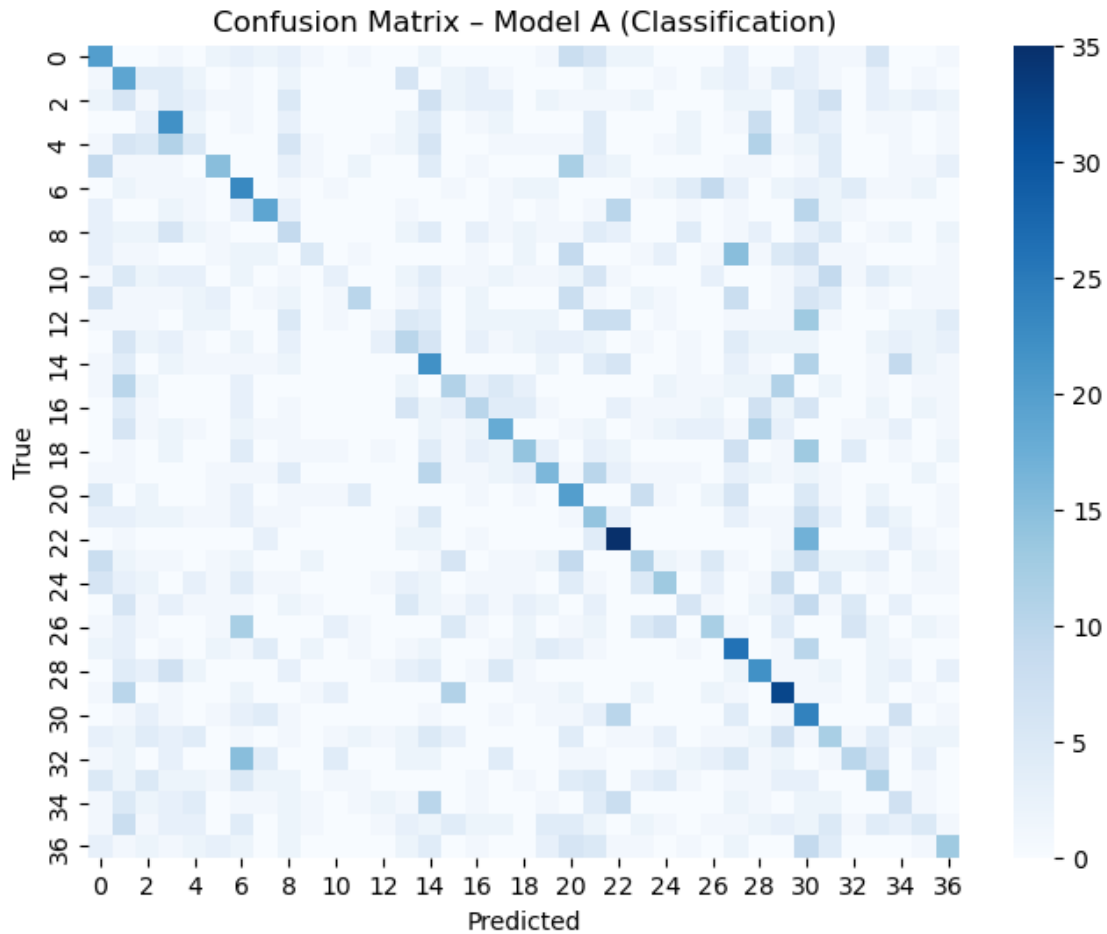
Segmentation F1 Score for Model B: 0.7358051985280853

Confusion Matrix for Model A (Classification)

```
[22]: import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion_matrix  
  
# Predicted class labels for Model A  
class_preds_a, _ = model_a.predict(test_class_seg)  
y_class_pred_a = np.argmax(class_preds_a, axis=1)  
  
# True class labels  
y_class_test = np.concatenate([y[0] for x, y in test_class_seg.  
    ↪as_numpy_iterator()])  
  
# Confusion matrix  
cm_a = confusion_matrix(y_class_test, y_class_pred_a)  
  
plt.figure(figsize=(8,6))  
sns.heatmap(cm_a, cmap='Blues', cbar=True)  
plt.title("Confusion Matrix - Model A (Classification)")  
plt.xlabel("Predicted")  
plt.ylabel("True")  
plt.savefig("model_a_confusion_matrix.png")  
plt.show()
```

81/81

2s 27ms/step



Confusion Matrix for Model B (MobileNetV3Small)

```
[23]: # Predict class labels for Model B
class_preds_b, _ = model_b.predict(test_class_seg)
y_class_pred_b = np.argmax(class_preds_b, axis=1)

# Ground truth
y_class_test = np.concatenate([y[0] for x, y in test_class_seg.
                               ↪as_numpy_iterator()])

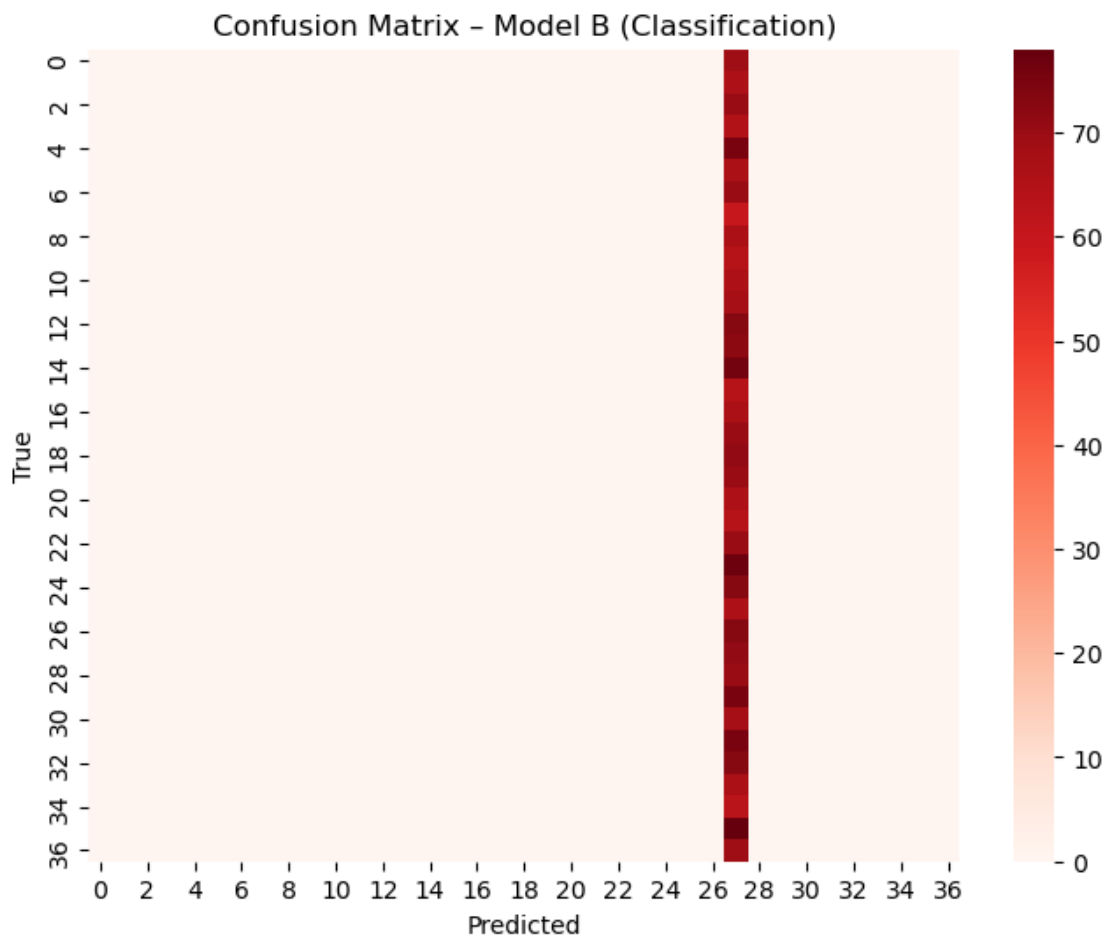
# Confusion Matrix
cm_b = confusion_matrix(y_class_test, y_class_pred_b)

# Plot
plt.figure(figsize=(8,6))
sns.heatmap(cm_b, cmap='Reds', cbar=True)
plt.title("Confusion Matrix - Model B (Classification)")
plt.xlabel("Predicted")
```

```
plt.ylabel("True")
plt.savefig("model_b_confusion_matrix.png")
plt.show()
```

81/81

2s 27ms/step



Segmentation Comparison – Original, Ground Truth, A vs. B

```
[24]: # Show side-by-side segmentation predictions from both models
images, (labels, masks) = next(iter(test_class_seg.take(1)))

# Predictions
_, seg_preds_a = model_a.predict(images[:2])
_, seg_preds_b = model_b.predict(images[:2])

for i in range(2):
    fig, axs = plt.subplots(1, 4, figsize=(12, 3))
    axs[0].imshow((images[i].numpy() + 1) / 2) # Rescale from [-1, 1] to [0, 1]
```

```

axs[0].set_title("Original")

axs[1].imshow(masks[i].numpy().squeeze(), cmap='gray')
axs[1].set_title("Ground Truth")

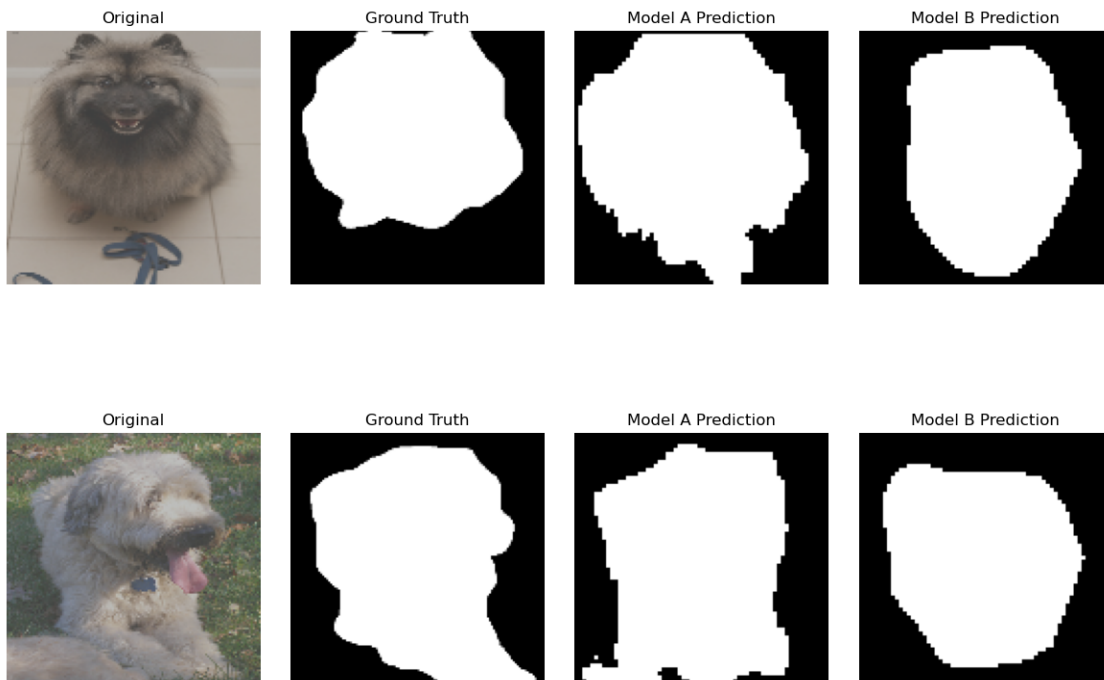
axs[2].imshow(seg_preds_a[i].squeeze() > 0.5, cmap='gray')
axs[2].set_title("Model A Prediction")

axs[3].imshow(seg_preds_b[i].squeeze() > 0.5, cmap='gray')
axs[3].set_title("Model B Prediction")

for ax in axs:
    ax.axis('off')
plt.tight_layout()
plt.savefig(f"segmentation_comparison_{i}.png")
plt.show()

```

1/1 2s 2s/step
1/1 9s 9s/step



Accuracy Over Epochs (Model A vs. Model B)

```

[25]: # Accuracy over epochs for both classification and segmentation
plt.figure(figsize=(8, 5))

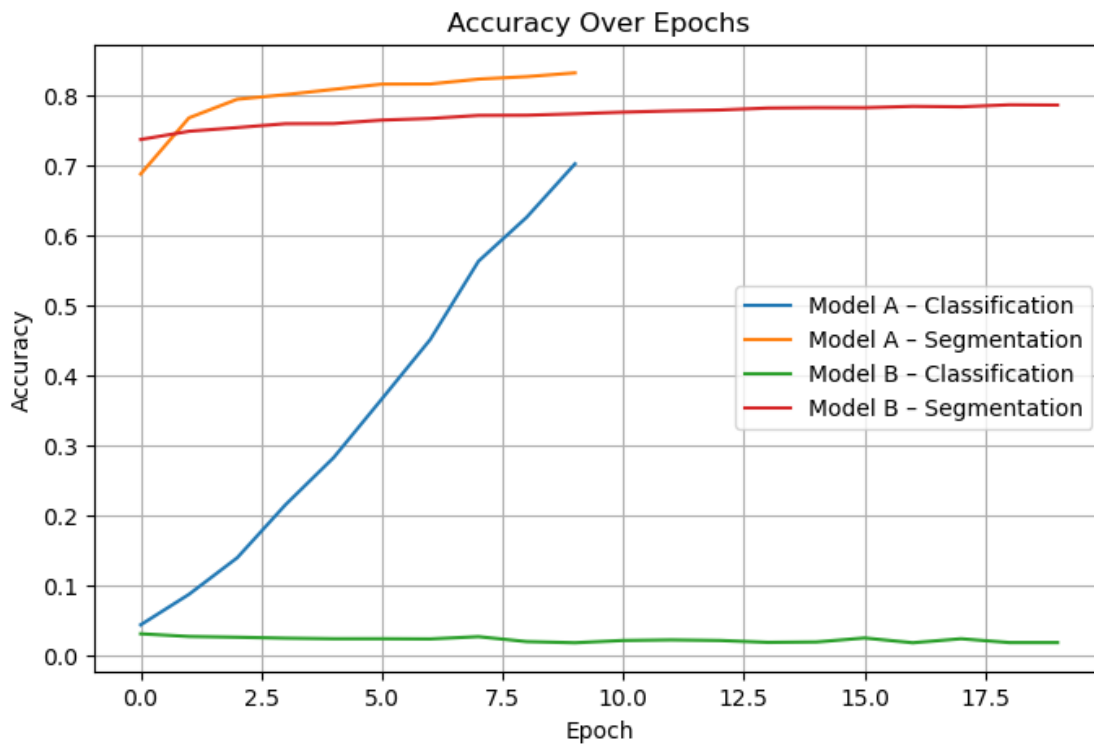
```

```

plt.plot(history_a.history['classification_accuracy'], label='Model A - Classification')
plt.plot(history_a.history['segmentation_accuracy'], label='Model A - Segmentation')
plt.plot(history_b.history['classification_accuracy'], label='Model B - Classification')
plt.plot(history_b.history['segmentation_accuracy'], label='Model B - Segmentation')

plt.title("Accuracy Over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig("accuracy_comparison.png")
plt.show()

```



[21]: *# For your write-up, you should include:*
- A discussion of what pre-processing (i.e. resizing, colour conversion, augmentation, etc.) you apply to the data and why.
- Details of two implemented methods. This should include a details of the final "from-scratch" approach and justification

for the chosen design, and details of changes made to MobileNetV3Small for
→the "fine-tuned" approach. Details on how the
models are trained are also to be provided.
- An evaluation that compares the two models for the two tasks
→(classification and semantic segmentation). Your evaluation
should discuss overall model performance, how it differs between the two
→approaches, and include figures if/where necessary.
- A discussion of methods that were explored to improve performance for both
→models and mitigate identified issues, and potentially
other methods that were considered but not implemented due to computational
→constraints. See the assignment brief for further
details.
Your write-up should be supported by appropriate figures and tables. Figures
→and tables should have numbers and meaningful captions.
Note that figures and tables are not included in the page limits.

SEE THE ASSIGNMENT BRIEF ON CANVAS FOR MORE DETAILS AND NOTE THAT A NOTEBOOK
→FILE DOES NOT CONSTITUTE A VALID SUBMISSION.
YOU SHOULD WRITE UP YOUR RESPONSE IN A SEPARATE DOCUMENT