Ollscoil Chathair Bhaile Átha Cliath
Dublin City University

Submission Cover Sheet Please use block capitals	
Student Name	**Andrey Totev**
Student ID No.	**20214021**
Module Code	**EE544**
Degree	**MSc.**
Programme	**MCM-AI**
Year	**2021/2022**
Date	**12 Mar 2022**
CA Materials Link (where applicable)	https://drive.google.com/file/d/1n4E1TtCax-LJVA7un1UZA-WGC4A9_KuU/view?usp=sharing

For use by examiners only (students should not write below this line)

1: Binary Classification [Training from Scratch using ImageNette]

Train/validation/test split

In order to keep performance results comparable to those of other models in the Imagenette leaderboard, the model is trained strictly on the ‘train’ dataset and ‘val’ will be used as the test set. Since this is a new model with lots of hyperparameters to tune and experiments to conduct, the validation set needs to be relatively large [1]. Therefore, ‘train’ is split into 80/20 for training/validation. This leads to a final train/validation/test split of approximately 50/20/30.

For the purposes of the binary classification task, ‘golf ball’ and ‘parachute’ image classes are selected [2]. All training examples are loaded from the ground truth file - noisy_imagenette.csv - which comes with the dataset. All irrelevant (based on class) and test examples are filtered out and the remaining ones are randomly split into 80/20 training/validation sets using scikit-learn's utility method [3]. Since the classes in the dataset are balanced, no stratification is required.

To reflect the newly created validation set on the file system, the respective files are moved into a new ‘validation’ directory under their respective class’ subdirectories. No other files need to be moved since the directory-based generator flows used in the model training are configured to only operate on ‘golf ball’ and ‘parachute’ images.

```
if not path.isdir(VALIDATION_DIR):
    ground_truth = pd.read_csv(LABELS_FILE)
    ground_truth = ground_truth[ground_truth['noisy_labels_0'].isin(CLASSES)]
```

```
    test_df = ground_truth[ground_truth['is_valid']==True]
    imagenette_train = ground_truth[ground_truth['is_valid']==False]
    train_df, val_df = train_test_split(imagenette_train, test_size=0.2) # the dataset is balanced
    val_df = val_df.rename(columns={'path': 'orig_path'})
    val_df['path'] = val_df['orig_path'].str.replace('train/', 'validation/')
    val_df.apply(lambda v: os.renames(path.join(DATA_DIR, v['orig_path']), path.join(DATA_DIR,
v['path'])), axis=1)
    del val_df['orig_path']
```

Listing 1: Split out validation set on the file system

Baseline CNN model and training

The following CNN architecture fulfills the required baseline structure:

```
if not path.isfile(baseline_model_file):
    os.makedirs(path.dirname(baseline_model_file), exist_ok=True)

    # Define model architecture
    inputs = layers.Input(shape=IMG_SIZE + (3,))
    x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(512, activation='relu')(x)
    output = layers.Dense(1, activation='sigmoid')(x)
    baseline_model = keras.Model(inputs=inputs, outputs=output)
    baseline_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 150, 150, 3)]     0

_________________________________________________________________
conv2d_8 (Conv2D)            (None, 148, 148, 32)      896

_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 74, 74, 32)        0

_________________________________________________________________
conv2d_9 (Conv2D)            (None, 72, 72, 64)        18496

_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 36, 36, 64)        0

_________________________________________________________________
conv2d_10 (Conv2D)           (None, 34, 34, 128)       73856

_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 17, 17, 128)       0
```

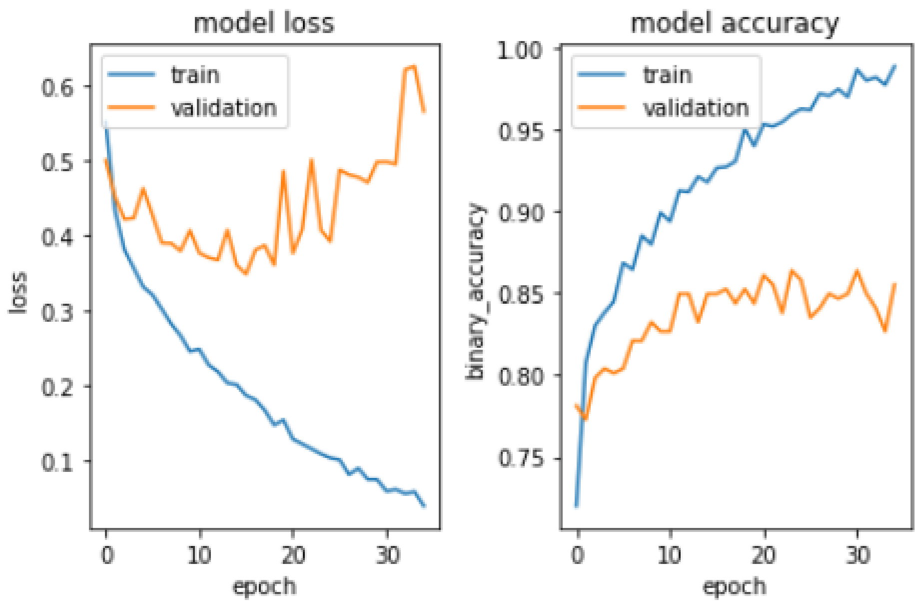
```
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Listing 2: Baseline CNN architecture

The model is trained using a Keras ImageDataGenerator that scales the RGB values down to the [0, 1] range for faster convergence [4]. The generator is used in a directory based ‘flow’ restricted to the two relevant classes and resizing the images to the shape of the model’s input tensor. In order to preserve the best performing model, a ModelCheckpoint callback is used to track the validation loss between the epochs and persist the model with lowest loss as an HDF5 file. Since this is a binary classification task, the loss function used is binary crossentopy and the metric is binary accuracy. The training is performed over 35 epochs using an RMSprop optimizer with a relatively low learning rate of 0.0001.

```
# Data generation flow from train/validation directory
def create_flow(datagen, path, batch_size):
    return datagen.flow_from_directory(
        path,
        target_size=IMG_SIZE,
        classes=CLASSES,
        class_mode='binary',
        batch_size=batch_size
    )

# Train model
epochs_count = 35
datagen = ImageDataGenerator(rescale=COLOUR_SCALE)
train_flow = create_flow(datagen, TRAIN_DIR, BATCH_SIZE)
val_flow = create_flow(datagen, VALIDATION_DIR, BATCH_SIZE)
save_best_cb = callbacks.ModelCheckpoint(filepath=baseline_model_file,
                                            monitor='val_loss', mode='min', save_best_only=True,
                                            verbose=False) # set to True to see best model's epoch
baseline_model.compile(
    optimizers.RMSprop(lr=1e-4),
    'binary_crossentropy',
    metrics=[metrics.BinaryAccuracy()]
)
```

```
history = baseline_model.fit(train_flow, steps_per_epoch=train_flow.samples // BATCH_SIZE,
                    validation_data=val_flow, validation_steps=val_flow.samples // BATCH_SIZE,
                    epochs=epochs_count,
                    callbacks=[save_best_cb],
                    verbose=False)
plot_model_history(history)
evaluate_model(baseline_model_file)
```




```
Test loss: 0.52, accuracy: 87.37%
```

Listing 3: Baseline CNN architecture

As seen in the training history plots, the minimal validation loss is observed at the 15th epoch approximately, after which it starts to increase and the validation accuracy gradually starts to decrease. The model accuracy with the unseen test data is 87.37%.

Improved model architecture and data pre-processing

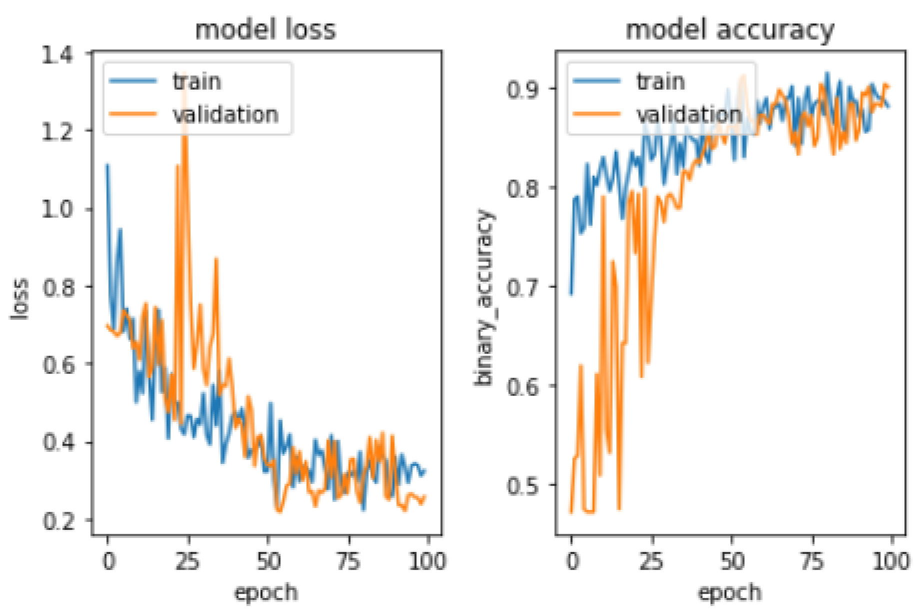
The modifications done to the baseline setup are aiming to improve the model's ability to generalize on unseen examples despite the limited training data. The improvements are threefold:

- Data augmentation. A subset of images from each batch are modified by applying random levels of the following transformations: rotation, width shift, height shift, shear, zoom, horizontal flip. Because of the artificially increased number of training examples, the model is trained three times longer, i.e. over 100 epochs
- Batch normalization. A type of layer which learns to normalize the activations from its preceding layer by maintaining a moving average of their mean and variance within each batch [4]. Similar to other normalization techniques, batch normalization helps with gradient propagation which speeds up training on larger (augmented in this case) data sets
- Dropout. Even though data augmentation is used, the generated images are similar to the ones from the small original training set. To further prevent the network from overfitting, it is regularized via two Dropout layers (rate 0.5) following the Flatten layer and the Dense(512) layer

```
# Define model architecture
inputs = layers.Input(shape=IMG_SIZE + (3,))
x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(64, (3, 3), activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(128, (3, 3), activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(128, (3, 3), activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(1, activation='sigmoid')(x)
improved_model = keras.Model(inputs=inputs, outputs=output)
improved_model.summary()

# Train model
epochs_count = 100
train_gen = ImageDataGenerator( # TODO document choices
    rescale=COLOUR_SCALE,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
train_flow = create_flow(train_gen, TRAIN_DIR, BATCH_SIZE)
val_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE), VALIDATION_DIR, BATCH_SIZE)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
batch_normalization_4 (Batch (None, 148, 148, 32)      128
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
```

```
conv2d_13 (Conv2D)           (None, 72, 72, 64)        18496     
_________________________________________________________________
batch_normalization_5 (Batch (None, 72, 72, 64)        256       
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 34, 34, 128)       73856     
_________________________________________________________________
batch_normalization_6 (Batch (None, 34, 34, 128)       512       
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 15, 15, 128)       147584    
_________________________________________________________________
batch_normalization_7 (Batch (None, 15, 15, 128)       512       
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 6272)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,454,529
Trainable params: 3,453,825
Non-trainable params: 704
```




```
Test loss: 0.33, accuracy: 88.54%
```

Listing 4: Data augmentation, batch normalization, dropout

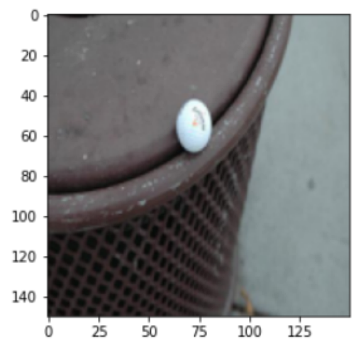
As seen in the training history plots, the minimal validation loss is observed at the 52nd epoch approximately - three times longer training in comparison to the baseline approach. After this point, both validation loss and accuracy plateau. The model accuracy with the unseen test data is 88.54% representing a performance improvement of more than 1% over the baseline.

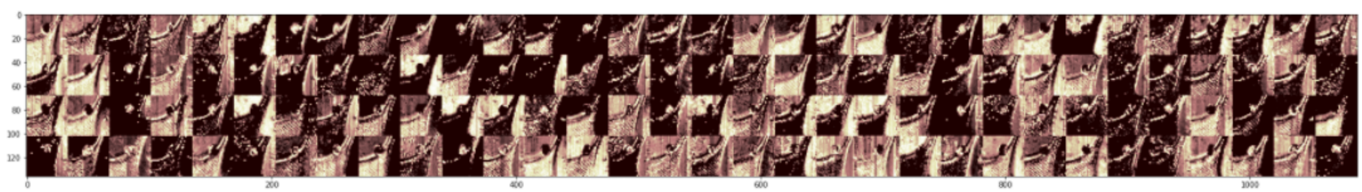
Visualization of intermediate activations

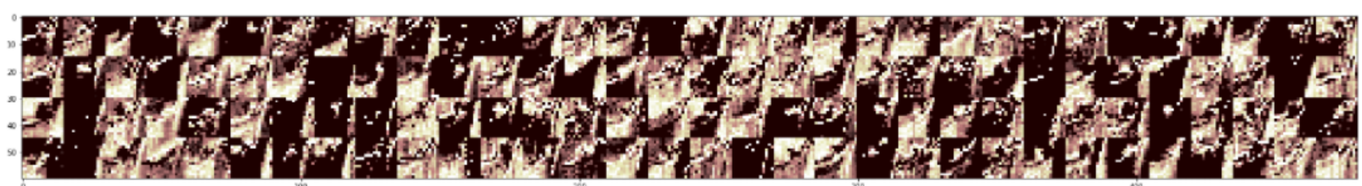
The intermediate CNN activations are retrieved using a common technique where the layers of the trained model are re-used in a new Model object with its outputs - set to be the output tensors of all CNN layers. When the predict() method is called, the activations of all CNN layers are returned - 32, 64, 128, and 128 channels respectively. The values are then min-max scaled in the [0, 255] range to prepare for drawing. Finally, they are displayed in a 32-column grid, grouped by CNN layer.

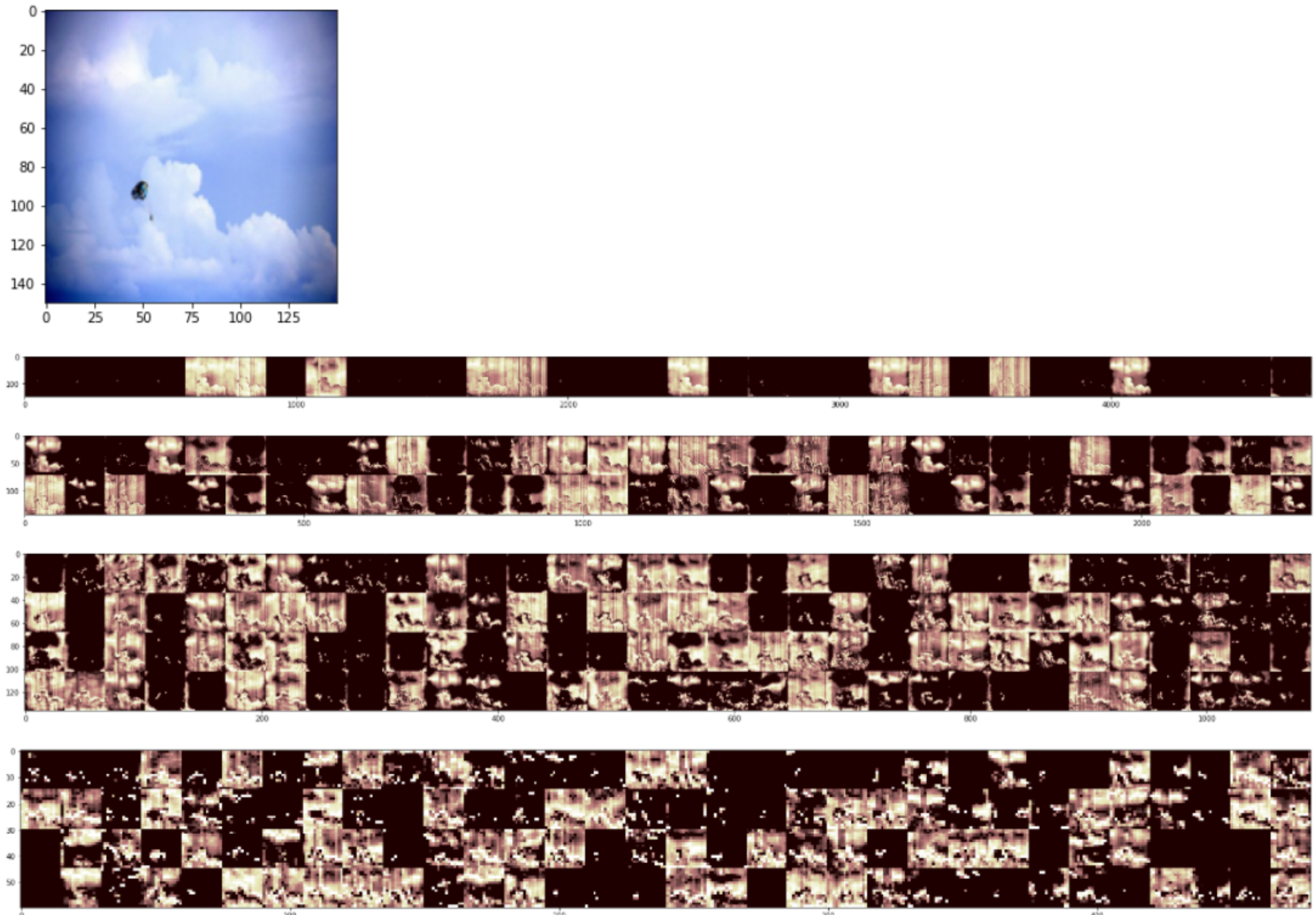
```
model = models.load_model(path.join(MODELS_DIR, 'imagenette', 'improved.h5'))

layer_outputs = []
for each in model.layers:
    if 'conv2d' in each.name:
        layer_outputs.append(each.output)
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

scaler = MinMaxScaler(feature_range=(0, 255)) # makes activation values ready for drawing
channels_per_row = 32
for c in CLASSES:
    img = load_random_image(path.join(TRAIN_DIR, c))
    plt.imshow(img)
    plt.show()

    img = np.expand_dims(img, axis=0) # array with a single image
    activations = activation_model.predict(img)
    for layer_activation in activations:
        layer_channel_count = layer_activation.shape[-1]
        img_size = layer_activation.shape[1]
        row_count = layer_channel_count // channels_per_row
        display_grid = np.zeros((img_size * row_count, channels_per_row * img_size))
        for row in range(row_count):
            for col in range(channels_per_row):
                channel_image = layer_activation[0, # CNN layer has a single output tensor
                                                 :, :, # the activations for each output channel
                                                 row * channels_per_row + col] # row and offset
                # scale the activations and populate in display grid
                channel_image = scaler.fit_transform(channel_image)
                display_grid[row * img_size : (row + 1) * img_size, col * img_size : (col + 1) *
img_size] = channel_image

        # draw
```

```
    plt.figure(figsize=(display_grid.shape[1] / img_size, display_grid.shape[0] / img_size))
    plt.imshow(display_grid, aspect='auto', cmap='pink')
    plt.show()
```









Listing 5: Visualizing intermediate CNN activations: bottom to top layers

The objects to classify are relatively small in the selected examples. This makes it difficult for the bottom channels to focus precisely on the area of the object. (For example, closer-up golf ball images tend to cause channels in the bottom convolution to directly “light up” the ball itself.) Only in the second and especially the third layer some of the channels highlight the object directly - looking as a small bright dot on a dark background. It is less certain what features are detected by the top convolution - perhaps the specific texture of a golf ball or the wavy surface of a parachute.

The above observations seem contradictory to the common understanding that top layers capture higher-level features and bottom ones detect more primitive features like edges. However, this is a relatively shallow architecture, classifying simpler to recognize and distinguish objects, so more investigation is required to confirm how the individual layers have “specialized” during training.

2: Dog Breed Classification using Fine-Tuning based Transfer Learning

Train/validation/test split

In order to keep performance results comparable to those of other models in the Imagewoof leaderboard, the model is trained strictly on the 'train' dataset and 'val' will be used as the test set. Since the model is predefined and there are not many hyperparameters to tune, the validation set does not need to be large. Therefore, 'train' is split into 90/10 for training/validation. This leads to a final train/validation/test split of approximately 60/10/30.

The method for splitting the data and organizing on the file system is identical to the previous task, except that the directory flow's class mode is 'categorical', thus representing the labels as one-hot encoded vectors.

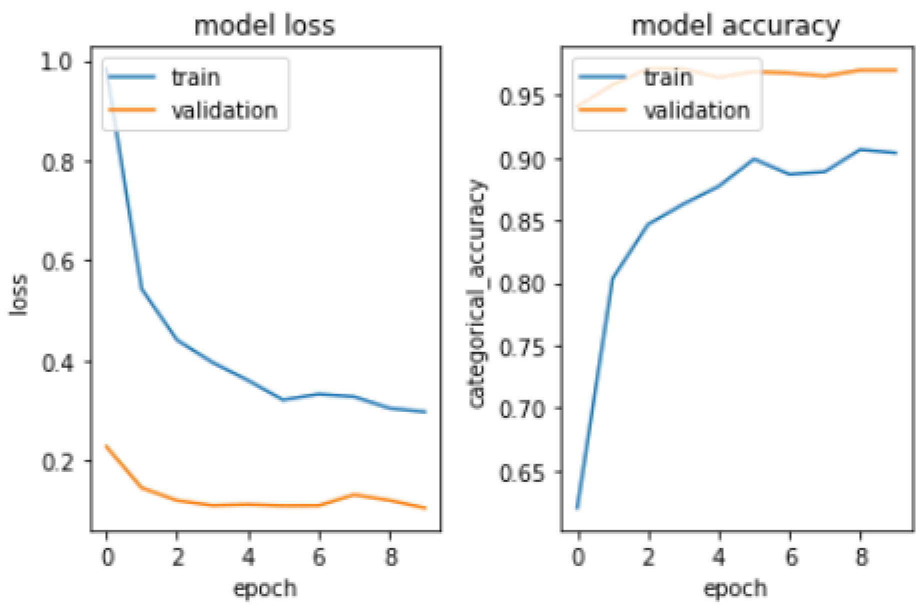
Fine-tuning InceptionV3

The InceptionV3 variant trained on ImageNet is used since the ImageWoof2 dataset is based on ImageNet. Fine-tuning is performed in two stages. In the first stage, InceptionV3 is instantiated, its layers are frozen, and its top layers are replaced with a custom classification component consisting of a GlobalAveragePooling2D bottleneck, and two Dense layers with Dropout of 0.5 each. The final layer is a Dense(5) with softmax activation used to predict one of the five classes. Training the classifier component is done for 10 epochs using the same data augmentation and a ModelCheckpoint callback as in the previous task. The loss function used is categorical_crossentropy and the metric is CategoricalAccuracy since this is a multi-class classification task. The learning rate used in the RMSprop optimizer for this stage is higher (0.001) because all weights are trained from a randomly initialized state.

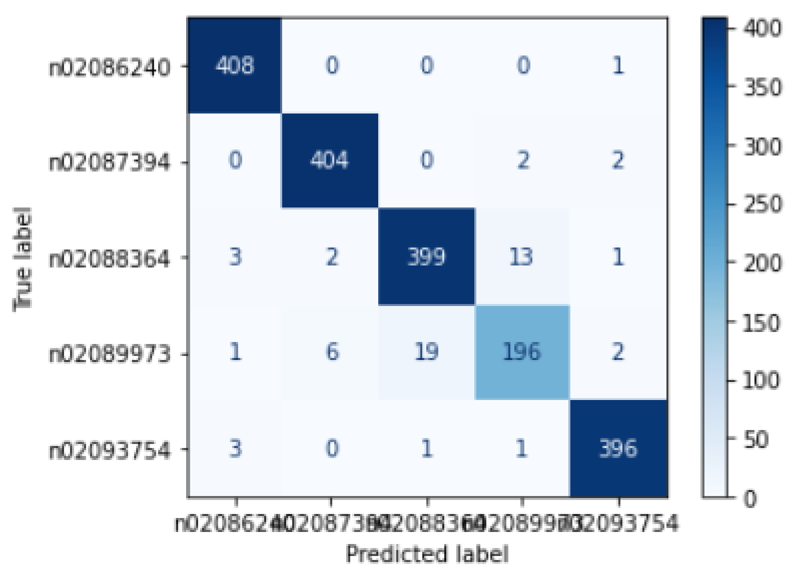
```
# Create InceptionV3 model
inceptionv3 = keras.applications.InceptionV3(weights='imagenet', include_top=False)
inceptionv3.trainable = False

# Add new top layers for classification
inputs = keras.Input(shape=IMG_SIZE + (3,))
x = inceptionv3(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(32, activation='relu')(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(5, activation='softmax')(x)

finetuned_model = keras.Model(inputs=inputs, outputs=output)
finetuned_model.compile(
    optimizers.RMSprop(lr=1e-3),
    'categorical_crossentropy',
    metrics=[metrics.CategoricalAccuracy()]
)
finetuned_model.summary()
```

```python
# Train model
epochs_count = 10
train_gen = ImageDataGenerator( # TODO document choices
    rescale=COLOUR_SCALE,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
train_flow = create_flow(train_gen, TRAIN_DIR, BATCH_SIZE)
val_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE), VALIDATION_DIR, BATCH_SIZE)
save_best_cb = callbacks.ModelCheckpoint(filepath=model_file,
                                         monitor='val_loss', mode='min', save_best_only=True,
                                         verbose=False) # set to True to see best model's epoch
history = finetuned_model.fit(train_flow, steps_per_epoch=train_flow.samples // BATCH_SIZE,
                    validation_data=val_flow, validation_steps=val_flow.samples // BATCH_SIZE,
                    epochs=epochs_count,
                    callbacks=[save_best_cb],
                    verbose=False)
plot_model_history(history)
evaluate_model(model_file)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 299, 299, 3)]     0
_________________________________________________________________
inception_v3 (Model)         multiple                  21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 64)                131136
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 165
=================================================================
Total params: 21,936,165
Trainable params: 133,381
Non-trainable params: 21,802,784
```




```
Evaluating against data in ./imagewoof2-320/val: Loss=0.12, Accuracy=96.94%
```



Listing 1: Stage 1 - training the classifying component

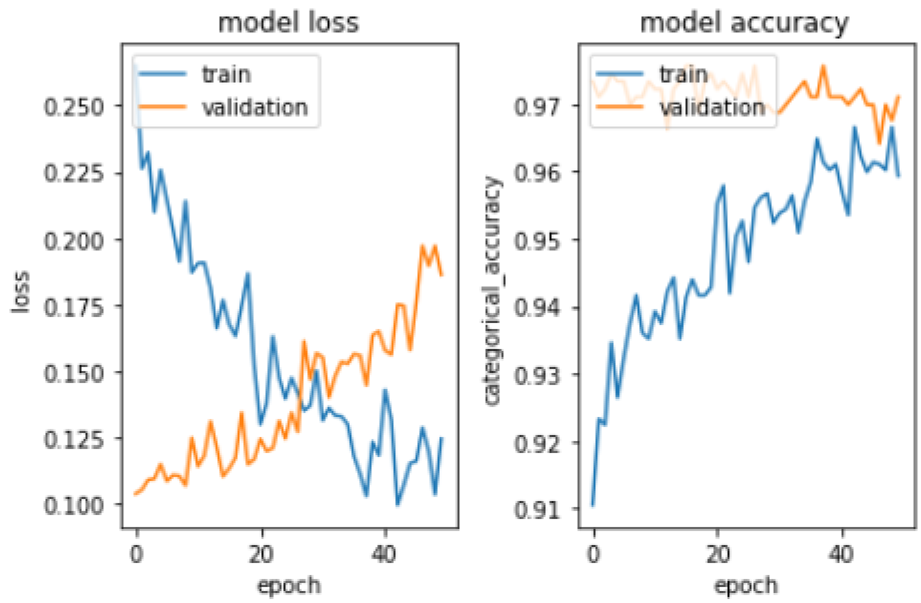
As seen from the training history plots, InceptionV3 (ImageNet variant) is very efficient at feature extraction allowing the new classifier to reach peak test accuracy of 96.94% only after the third epoch. Visualizing the confusion matrix for the test set shows notable difficulty in predicting class n02089973 (‘English foxhound’) which is understandable given that this class has approximately twice less training examples. This imbalance may be addressed by duplicating the English foxhound images during training (i.e. to sample at a higher rate compared to the remaining classes) or update the loss function to incur greater loss on misclassifying the underrepresented class. (Both approaches are not in the scope of this assignment.)

In the second stage of the training, the best weights from the previous stage are loaded from disk, and the top two inception blocks are made trainable. The learning rate is then decreased to 0.00001 in order to not make excess changes to the InceptionV3 weights during training and impact the performance negatively.

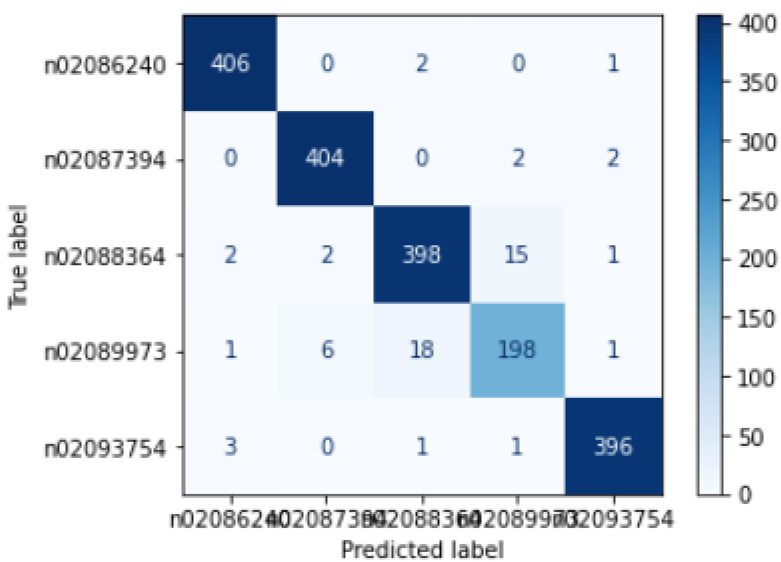
```
# Load best weights
finetuned_model.load_weights(model_file)

# Freeze all layers up to the top two inception blocks
inceptionv3.trainable = True
first_trainable_layer = 250
for layer in inceptionv3.layers[:first_trainable_layer]:
    layer.trainable = False
for layer in inceptionv3.layers[first_trainable_layer:]:
    layer.trainable = True

# Fine tune model
epochs_count = 50
finetuned_model.compile(
    optimizers.RMSprop(lr=1e-5),
    'categorical_crossentropy',
    metrics=[metrics.CategoricalAccuracy()]
)
finetuned_model.summary()
history = finetuned_model.fit(train_flow, steps_per_epoch=train_flow.samples // BATCH_SIZE,
                    validation_data=val_flow, validation_steps=val_flow.samples // BATCH_SIZE,
                    epochs=epochs_count,
                    callbacks=[save_best_cb],
                    verbose=False)
plot_model_history(history)

# Evaluate against test dataset
evaluate_model(model_file)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 299, 299, 3)]     0
_________________________________________________________________
inception_v3 (Model)         multiple                  21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 64)                131136
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 5)                 165
=================================================================
Total params: 21,936,165
Trainable params: 10,674,821
Non-trainable params: 11,261,344
```




```
Evaluating against data in ./imagewoof2-320/val: Loss=0.13, Accuracy=96.88%
```



Listing 2: Stage 2 - fine tuning

As seen from the training history plot for 50 epochs, the model begins to overfit the training data from the very start despite using data augmentation, dropout. Overfitting happens at a very rapid pace despite the use of low learning rate. These observations lead to the conclusion that it is very easy for a powerful model such as InceptionV3 to start overfitting small data sets and in such situations it is better to use it as a feature extractor, i.e. without fine tuning.

Testing on images "from the wild"

In order to test the fine-tuned model on images from the wild, 10 images per class were collected via searching with Google. The images were stored in a directory structure compatible with the Keras directory-based flow.

Figure 1: Sample royalty-free images from pexels.com used in model evaluation

```
Evaluating against data in ./imagewoof2-in_the_wild: Loss=0.10, Accuracy=96.00%
```

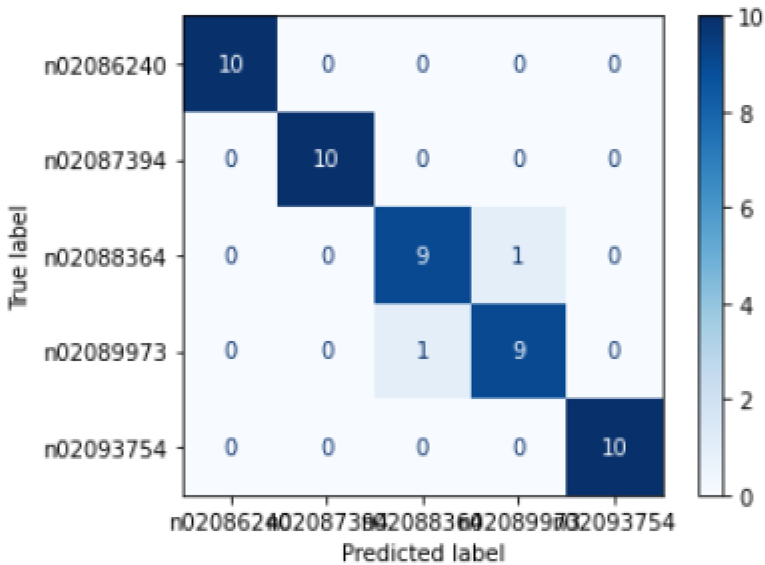


Figure 2: Model evaluation against collected images from the wild

The model accuracy was 96% on the new data set. The two misclassified images were English foxhound and beagle, which were also the most misclassified in the ImageWoof2 test set according to its confusion matrix.

References

[1] Medium. 2022. How to split data into three sets (train, validation, and test) And why?. [online] Available at: <https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c> [Accessed 12 March 2022].
[2] 2022. [online] Available at: <https://image-net.org/challenges/LSVRC/2014/browse-synsets.php> [Accessed 12 March 2022].
[3] scikit-learn. 2022. sklearn.model_selection.train_test_split. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html> [Accessed 12 March 2022].
[4] Chollet, F. (2017), Deep Learning with Python , Manning .

Appendix A - Hardware

The work was performed on the following system configuration:

Asus ROG GX701GWR-EV042T
CPU: Core i7-9750H
RAM: 32GB
HDD: 1TB SSD
GPU: NVidia RTX 2070 8GB

Appendix B - Listings

Problem 1

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


import pandas as pd
from os import path
from os import environ
import os
import random as random
from tensorflow.keras.preprocessing import image as imgproc
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import layers
from tensorflow.keras import metrics
from tensorflow.keras import optimizers
from tensorflow.keras import models
from tensorflow.keras import callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
from sklearn.preprocessing import MinMaxScaler

get_ipython().run_line_magic('matplotlib', 'inline')


# ### Initialisation

# In[ ]:


# Attempt to make runs more reproducible
seed_value=20212042
print("Using random seed: %d" % seed_value)
environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value) # tensorflow 2.x
```

```python
print("GPUs Available:", tf.config.list_physical_devices('GPU'))
print("Tensorflow version:", tf.__version__)


# ### Constants

# In[ ]:


DATA_DIR = './imagenette2-320'
TRAIN_DIR = path.join(DATA_DIR, 'train')
VALIDATION_DIR = path.join(DATA_DIR, 'validation') # a split off 'train' used as validation set during
NN training
TEST_DIR = path.join(DATA_DIR, 'val') # the original Imagenette test dir
MODELS_DIR = path.join('./models')
LABELS_FILE = path.join(DATA_DIR, 'noisy_imagenette.csv')
CLASS1 = 'n03445777' # e.g. n03445777 -> golf ball
CLASS2 = 'n03888257' # e.g. n03888257 -> parachute
CLASSES = [CLASS1, CLASS2]
IMG_SIZE = (150, 150)
COLOUR_SCALE = 1/255.
BATCH_SIZE = 32


# ### Helper functions

# In[ ]:


# Data generation flow from train/validation directory
def create_flow(datagen, path, batch_size):
    return datagen.flow_from_directory(
        path,
        target_size=IMG_SIZE,
        classes=CLASSES,
        class_mode='binary',
        batch_size=batch_size
    )

def evaluate_model(model_file):
    model = models.load_model(model_file)
    test_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE), TEST_DIR, BATCH_SIZE)
    loss_accuracy = model.evaluate(test_flow, steps=test_flow.samples // BATCH_SIZE, verbose=False)
    print('Test loss: %.2f, accuracy: %.2f%%' % (loss_accuracy[0], loss_accuracy[1] * 100.))

def plot_model_history(history):
    # Loss
```

```
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')

    # Binary Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history.history['binary_accuracy'])
    plt.plot(history.history['val_binary_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('binary_accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.tight_layout()
    plt.show()

def load_random_image(filepath):
    img_file = random.choice(os.listdir(filepath))
    img = imgproc.load_img(path.join(filepath, img_file))
    img = img.resize(IMG_SIZE)
    img_array = imgproc.img_to_array(img)
    return img_array * COLOUR_SCALE


# ### Make train/validation split and organize directory structure accordingly

# In[ ]:


if not path.isdir(VALIDATION_DIR):
    ground_truth = pd.read_csv(LABELS_FILE)
    ground_truth = ground_truth[ground_truth['noisy_labels_0'].isin(CLASSES)]
    test_df = ground_truth[ground_truth['is_valid']==True]
    imagenette_train = ground_truth[ground_truth['is_valid']==False]
    train_df, val_df = train_test_split(imagenette_train, test_size=0.2) # the dataset is balanced
    val_df = val_df.rename(columns={'path': 'orig_path'})
    val_df['path'] = val_df['orig_path'].str.replace('train/', 'validation/')
    val_df.apply(lambda v: os.renames(path.join(DATA_DIR, v['orig_path']), path.join(DATA_DIR,
v['path'])), axis=1)
    del val_df['orig_path']


# ### Define and train baseline CNN
```

```python
# In[ ]:


ImageDataGenerator(rescale=COLOUR_SCALE)
baseline_model_file = path.join(MODELS_DIR, 'imagenette', 'baseline.h5')
if not path.isfile(baseline_model_file):
    os.makedirs(path.dirname(baseline_model_file), exist_ok=True)

    # Define model architecture
    inputs = layers.Input(shape=IMG_SIZE + (3,))
    x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(512, activation='relu')(x)
    output = layers.Dense(1, activation='sigmoid')(x)
    baseline_model = keras.Model(inputs=inputs, outputs=output)
    baseline_model.summary()

    # Train model
    epochs_count = 35
    datagen = ImageDataGenerator(rescale=COLOUR_SCALE)
    train_flow = create_flow(datagen, TRAIN_DIR, BATCH_SIZE)
    val_flow = create_flow(datagen, VALIDATION_DIR, BATCH_SIZE)
    save_best_cb = callbacks.ModelCheckpoint(filepath=baseline_model_file,
                                             monitor='val_loss', mode='min', save_best_only=True,
                                             verbose=False) # set to True to see best model's epoch
    baseline_model.compile(
        optimizers.RMSprop(lr=1e-4),
        'binary_crossentropy',
        metrics=[metrics.BinaryAccuracy()]
    )
    history = baseline_model.fit(train_flow, steps_per_epoch=train_flow.samples // BATCH_SIZE,
                       validation_data=val_flow, validation_steps=val_flow.samples // BATCH_SIZE,
                       epochs=epochs_count,
                       callbacks=[save_best_cb],
                       verbose=False)
    plot_model_history(history)

# Evaluate against test dataset
evaluate_model(baseline_model_file)
```

```
# ### Use data augmentation to improve model performance

# In[ ]:


improved_model_file = path.join(MODELS_DIR, 'imagenette', 'improved.h5')
if not path.isfile(improved_model_file):
    os.makedirs(path.dirname(improved_model_file), exist_ok=True)

    # Define model architecture
    inputs = layers.Input(shape=IMG_SIZE + (3,))
    x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(512, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    output = layers.Dense(1, activation='sigmoid')(x)
    improved_model = keras.Model(inputs=inputs, outputs=output)
    improved_model.summary()

    # Train model
    epochs_count = 100
    train_gen = ImageDataGenerator( # TODO document choices
        rescale=COLOUR_SCALE,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    train_flow = create_flow(train_gen, TRAIN_DIR, BATCH_SIZE)
    val_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE), VALIDATION_DIR, BATCH_SIZE)
    save_best_cb = callbacks.ModelCheckpoint(filepath=improved_model_file,
```

```
                                             monitor='val_loss', mode='min', save_best_only=True,
                                             verbose=False) # set to True to see best model's epoch
    improved_model.compile(
        optimizers.RMSprop(lr=1e-4),
        'binary_crossentropy',
        metrics=[metrics.BinaryAccuracy()]
    )
    history = improved_model.fit(train_flow, steps_per_epoch=val_flow.samples // BATCH_SIZE,
                       validation_data=val_flow, validation_steps=val_flow.samples // BATCH_SIZE,
                       epochs=epochs_count,
                       callbacks=[save_best_cb],
                       verbose=False)
    plot_model_history(history)

# Evaluate against test dataset
evaluate_model(improved_model_file)


# ### Visualize intermediate activations

# In[ ]:


model = models.load_model(path.join(MODELS_DIR, 'imagenette', 'improved.h5'))

layer_outputs = []
for each in model.layers:
    if 'conv2d' in each.name:
        layer_outputs.append(each.output)
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

scaler = MinMaxScaler(feature_range=(0, 255)) # makes activation values ready for drawing
channels_per_row = 32
for c in CLASSES:
    img = load_random_image(path.join(TRAIN_DIR, c))
    plt.imshow(img)
    plt.show()

    img = np.expand_dims(img, axis=0) # array with a single image
    activations = activation_model.predict(img)
    for layer_activation in activations:
        layer_channel_count = layer_activation.shape[-1]
        img_size = layer_activation.shape[1]
        row_count = layer_channel_count // channels_per_row
        display_grid = np.zeros((img_size * row_count, channels_per_row * img_size))
        for row in range(row_count):
            for col in range(channels_per_row):
```

```python
                channel_image = layer_activation[0, # CNN layer has a single output tensor
                                                 :, :, # the activations for each output channel
                                                 row * channels_per_row + col] # row and offset in the 
display grid
                # scale the activations and populate in display grid
                channel_image = scaler.fit_transform(channel_image)
                display_grid[row * img_size : (row + 1) * img_size, col * img_size : (col + 1) * 
img_size] = channel_image

        # draw
        plt.figure(figsize=(display_grid.shape[1] / img_size, display_grid.shape[0] / img_size))
        plt.imshow(display_grid, aspect='auto', cmap='pink')
        plt.show()
```

Problem 2

```python
#!/usr/bin/env python
# coding: utf-8

# In[ ]:



import pandas as pd
from os import path
from os import environ
import os
import random as random
from tensorflow.keras.preprocessing import image as imgproc
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import layers
from tensorflow.keras import metrics
from tensorflow.keras import optimizers
from tensorflow.keras import models
from tensorflow.keras import callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

```
import random

get_ipython().run_line_magic('matplotlib', 'inline')


# ### Initialisation

# In[ ]:


# Attempt to make runs more reproducible
seed_value=20212042
print("Using random seed: %d" % seed_value)
environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value) # tensorflow 2.x

print("GPUs Available:", tf.config.list_physical_devices('GPU'))
print("Tensorflow version:", tf.__version__)


# ### Constants

# In[ ]:


DATA_DIR = './imagewoof2-320'
TRAIN_DIR = path.join(DATA_DIR, 'train')
VALIDATION_DIR = path.join(DATA_DIR, 'validation') # a split off 'train' used as validation set during NN training
TEST_DIR = path.join(DATA_DIR, 'val') # the original Imagenette test dir
MODELS_DIR = path.join('./models')
LABELS_FILE = path.join(DATA_DIR, 'noisy_imagewoof.csv')
BREEDS = { # mappings from: https://image-net.org/challenges/LSVRC/2014/browse-synsets.php
    'n02086240': 'Shih-Tzu',
    'n02087394': 'Rhodesian ridgeback',
```

```
    'n02088364': 'beagle',
    'n02089973': 'English foxhound',
    'n02093754': 'Border terrier'
}
CLASSES = ['n02086240', 'n02087394', 'n02088364', 'n02089973', 'n02093754'] #
BREEDS.keys()
IMG_SIZE = (299, 299)
COLOUR_SCALE = 1/255.
BATCH_SIZE = 32


# ### Helper functions

# In[ ]:


def plot_model_history(history):
    # Loss
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')

    # Categorical Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history.history['categorical_accuracy'])
    plt.plot(history.history['val_categorical_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('categorical_accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.tight_layout()
    plt.show()

def load_random_image(filepath):
```

```
    img_file = random.choice(os.listdir(filepath))
    img = imgproc.load_img(path.join(filepath, img_file))
    img = img.resize(IMG_SIZE)
    img_array = imgproc.img_to_array(img)
    return img_array * COLOUR_SCALE

# Data generation flow from train/validation directory
def create_flow(datagen, path, batch_size):
    # TODO document choices
    return datagen.flow_from_directory(
        path,
        target_size=IMG_SIZE,
        classes=CLASSES,
        class_mode='categorical',
        batch_size=batch_size
    )

def evaluate_model(model_file, data_path=TEST_DIR):
    model = models.load_model(model_file)
    test_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE), data_path,
1)

    loss_accuracy = model.evaluate(test_flow, steps=test_flow.samples,
verbose=False)
    print('Evaluating against data in %s: Loss=%.2f, Accuracy=%.2f%%' %
(data_path, loss_accuracy[0], loss_accuracy[1] * 100.))

    test_flow.reset()
    y_true = []
    for i in range(test_flow.samples):
        _, y = test_flow.next()
        y_true.append(np.argmax(y, axis=1))
    y_true = np.array(y_true)

    test_flow.reset()
    y_pred = model.predict(test_flow, steps=test_flow.samples, verbose=False)
    y_pred = np.argmax(y_pred, axis=1)
```

```
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=CLASSES)
    disp.plot(cmap=plt.cm.Blues)
    plt.show()


# ### Make train/validation split and organize directory structure accordingly

# In[ ]:


if not path.isdir(VALIDATION_DIR):
    ground_truth = pd.read_csv(LABELS_FILE)
    ground_truth = ground_truth[ground_truth['noisy_labels_0'].isin(CLASSES)]
    test_df = ground_truth[ground_truth['is_valid']==True]
    imagenette_train = ground_truth[ground_truth['is_valid']==False]
    train_df, val_df = train_test_split(imagenette_train, test_size=0.2) # the
dataset is balanced
    val_df = val_df.rename(columns={'path': 'orig_path'})
    val_df['path'] = val_df['orig_path'].str.replace('train/', 'validation/')
    val_df.apply(lambda v: os.renames(path.join(DATA_DIR, v['orig_path']),
path.join(DATA_DIR, v['path'])), axis=1)
    del val_df['orig_path']


# ### Define and train InceptionV3-based CNN

# In[ ]:


ImageDataGenerator(rescale=COLOUR_SCALE)
model_file = path.join(MODELS_DIR, 'imagewoof', 'inceptionv3_based.h5')
if not path.isfile(model_file):
    os.makedirs(path.dirname(model_file), exist_ok=True)

    # Create InceptionV3 model
    inceptionv3 = keras.applications.InceptionV3(weights='imagenet',
include_top=False)
```

```
    inceptionv3.trainable = False

    # Add new top layers for classification
    inputs = keras.Input(shape=IMG_SIZE + (3,))
    x = inceptionv3(inputs, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    output = layers.Dense(5, activation='softmax')(x)

    finetuned_model = keras.Model(inputs=inputs, outputs=output)
    finetuned_model.compile(
        optimizers.RMSprop(lr=1e-3),
        'categorical_crossentropy',
        metrics=[metrics.CategoricalAccuracy()]
    )
    finetuned_model.summary()

    # Train model
    epochs_count = 10
    train_gen = ImageDataGenerator( # TODO document choices
        rescale=COLOUR_SCALE,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    train_flow = create_flow(train_gen, TRAIN_DIR, BATCH_SIZE)
    val_flow = create_flow(ImageDataGenerator(rescale=COLOUR_SCALE),
VALIDATION_DIR, BATCH_SIZE)
    save_best_cb = callbacks.ModelCheckpoint(filepath=model_file,
                                             monitor='val_loss', mode='min',
save_best_only=True,
```

```
                                                verbose=False) # set to True to see
best model's epoch
    history = finetuned_model.fit(train_flow, steps_per_epoch=train_flow.samples
// BATCH_SIZE,
                        validation_data=val_flow,
validation_steps=val_flow.samples // BATCH_SIZE,
                        epochs=epochs_count,
                        callbacks=[save_best_cb],
                        verbose=False)
    plot_model_history(history)
    evaluate_model(model_file)

    # Load best weights
    finetuned_model.load_weights(model_file)

    # Freeze all layers up to the top two inception blocks
    inceptionv3.trainable = True
    first_trainable_layer = 250
    for layer in inceptionv3.layers[:first_trainable_layer]:
        layer.trainable = False
    for layer in inceptionv3.layers[first_trainable_layer:]:
        layer.trainable = True

    # Fine tune model
    epochs_count = 50
    finetuned_model.compile(
        optimizers.RMSprop(lr=1e-5),
        'categorical_crossentropy',
        metrics=[metrics.CategoricalAccuracy()]
    )
    finetuned_model.summary()
    history = finetuned_model.fit(train_flow, steps_per_epoch=train_flow.samples
// BATCH_SIZE,
                        validation_data=val_flow,
validation_steps=val_flow.samples // BATCH_SIZE,
                        epochs=epochs_count,
                        callbacks=[save_best_cb],
                        verbose=False)
```

```
    plot_model_history(history)

# Evaluate against test dataset
evaluate_model(model_file)



# ### Evaluate against unseen images

# In[ ]:



evaluate_model(model_file, './imagewoof2-in_the_wild')
```