University of Moratuwa

Department of Electronic and Telecommunication Engineering

Assignment 1

Machine Vision

Name	Index number	Percentage contribution
A. M. A. D. Adikari	190021A	25%
W. A. D. K. De Silva	190128H	25%
T. A. Peirispulle	190443T	25%
B. W. S Priyashan	190476V	25%

12th December, 2023

Contents

List of Figures

List of Tables

1 ResNet-50

1.1 Generating embeddings from pretrained ResNet-50

1.1.1 Importing the dataset

Tensorflow is used as the machine learning platform to complete this assignment. The dataset which is used in this assignment is the Oxford-IIIT Pet Dataset. We are using the processed version of the dataset available in *tensorflow_datasets*. The dataset is loaded using the tfds.load function.

We divided the train set to have a separate validation set since test partition is only used to report the final results (it should not be used to train the network or tune hyperparameters). The images are resized to (224,224,3)

1.1.2 Importing the pretrained network

Pretrained ResNet-50 model was imported from *keras.applications*. Here we import the model giving the parameter *include_top = False* as we drop the final dense layer and only import the feature extractor. The model is then freezed as we are using the model in inference mode.

```python
#import the ResNet50 base model
base_model = keras.applications.ResNet50(
    weights = "imagenet",
    input_shape = (224,224,3),
    pooling = 'avg',
    include_top =False,  #do not include the ImageNet classifier at the top
)

#Freeze the base model
base_model.trainable = False
```

Figure 1: Code snippet for loading loading the feature extractor from Resnet50

1.1.3 Generating embeddings

We divide the dataset into batches of 32. Then we generate the embeddings for the images available in the whole dataset. Embeddings are the outputs from the last Average Pooling layer of our model.

```
conv5_block3_add (Add)      (None, 7, 7, 2048)          0         ['conv5_block2_out[0][0]',
                                                                   'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activati  (None, 7, 7, 2048)          0         ['conv5_block3_add[0][0]']
on)

avg_pool (GlobalAveragePoo  (None, 2048)                0         ['conv5_block3_out[0][0]']
ling2D)

==================================================================================================
Total params: 23587712 (89.98 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________________________
```

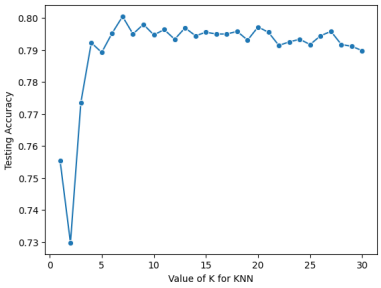
(a) Last layers of the model

```python
def preprocess(images, labels):
  return tf.keras.applications.resnet50.preprocess_input(images), labels

#train embeddings
ds_train = ds_train.map(preprocess)
train_embeddings = ds_train.map(model)

#validation embeddings
ds_validation = ds_validation.map(preprocess)
val_embeddings = ds_validation.map(model)

#test embeddings
ds_test = ds_test.map(preprocess)
test_embeddings = ds_test.map(model)
```

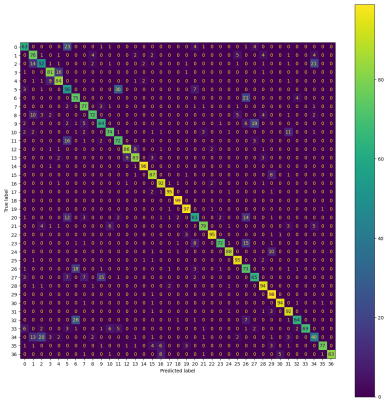
(b) generating the image embeddings

1.1.4 k-NN Classification

Then we standardize the embeddings using *StandardScaler* from *sklearn.preprocessing*. We use *KNeighborsClassifier* from *sklearn.neighbors* as our knn model, and after fitting the knn model with the X train and y train with different k values and plotting the accuracy values we choose k = 7 as the optimum k and generate predictions.



(c) Accuracy scores for different k values



(d) Confusion matrix for k=7

We achieved an accuracy score of 80.04% on the test set.

2 Linear classification (multi-class logistic regression) with the pre-trained embeddings

To classify images in our dataset into the given 37 labels, we then added a dense layer to our pre-trained model with 37 output nodes. The ResNet model was freezed and the model is fitted using the train embeddings, so that only the last fully-connected layer will be trained. A dropout layer was also added and tuned along with the type of optimizer and learning rate considering the model's response to the validation set to reduce overfitting. We finally used SGD as the optimizer and used 40 epochs for training as it gave us the best results.

Learning rate	0.005
Number of epochs	40
Accuracy of Regression model	0.891
Optimizer	SGD
Loss function	SparseCategoricalCrossentropy

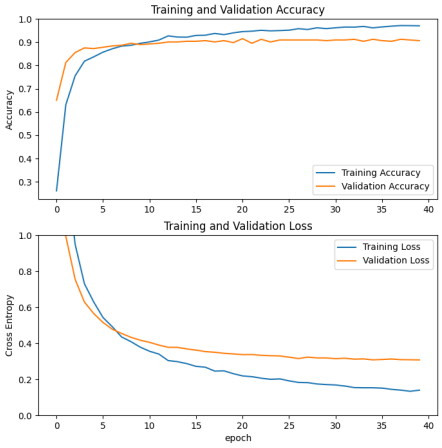
(e) Parameters

```python
#build the model end to end
inputs = tf.keras.Input(shape=(224, 224, 3))
x = preprocess_input(inputs)
x = model(x, training=False)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

(f) Code snippet for full model

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 model_2 (Functional)        (None, 2048)              23587712

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 37)                75813

=================================================================
Total params: 23663525 (90.27 MB)
Trainable params: 75813 (296.14 KB)
Non-trainable params: 23587712 (89.98 MB)
```

(g) Summary of the model



(h) Accuracy plots

We achieved an accuracy score of 89.1% on the test set.

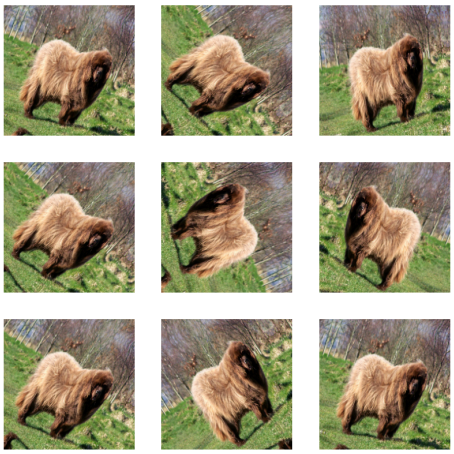
3 Fine-tuning the network end-to-end

3.1 Data augmentation

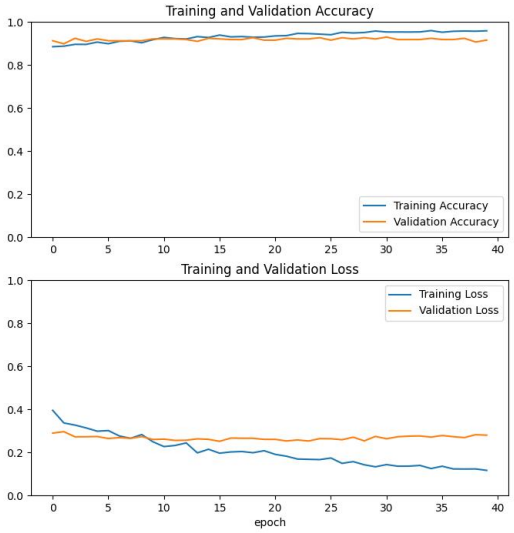
A data augmentation layer was added to the model which would randomly rotate and flip the images fed into the model. This exposes model to a considerably wider range of variation, even when the foundational dataset doesn't change.

3.1.1 Final model

We freezed the initial 140 layers of the full 176 layers in the ResNet model. The initial layers pick up extremely basic and generic properties that apply to nearly any kind of image. The characteristics get more and more unique to the dataset that the model was trained on as we move further up. Rather than replacing the general learning, the aim of fine-tuning is to modify these specialised features to function with the new dataset.



(i) Example augmented image



(j) Train and validation accuracies

```
#model with data augmentation and dropout
inputs = tf.keras.Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

(k) final model

```
tune_start_at = 140

# Freeze all the layers before the 140th layer
for layer in base_model.layers[:tune_start_at]:
  layer.trainable = False
```

(l) Fine tuning layers

We achieved an accuracy score of 89.72% on the test set.