UNIVERSITY OF MORATUWA

Faculty of Engineering

Group Assignment

Machine Vision – EN 4553

Name	Index Number	Contribution
Gajaanan S.	190185D	25%
Loshanan M.	190363X	25%
Luxshan S.	190364C	25%
Sajeepan T.	190539T	25%

Department of Electronic and Telecommunication Engineering

University of Moratuwa

11th of December 2023

Question (a): k-NN Classification

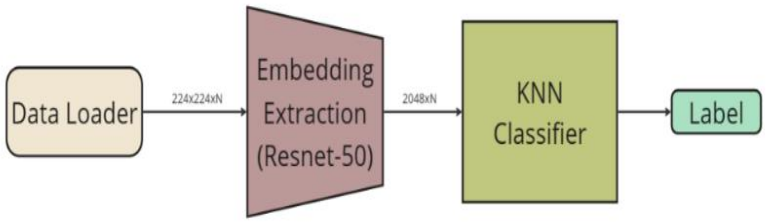

Figure 1: Model architecture

In this experiment, a pre-trained ResNet-50 model was utilized for image classification on the Oxford-IIIT Pet dataset. The split train and test datasets are used as training and testing sets, and the ResNet-50 model was employed to extract embeddings from the images. These embeddings were then used as feature vectors for training a k-NN (k-Nearest Neighbors) classifier.

The data loading process involved transforming the images to a consistent size of (224, 224) pixels and normalizing pixel values. The ResNet-50 model, pre-trained on ImageNet, was employed as a feature extractor by removing its final fully connected layer. The resulting embeddings were then flattened and used as input for the k-NN classifier.

The experiment achieved a classification accuracy of approximately 81.47% on the test set, demonstrating the effectiveness of the k-NN classifier in utilizing the extracted ResNet-50 embeddings for image classification. The precision and recall metrics further support the overall performance of the model, with precision being approximately 82.45% and recall matching the accuracy at 81.47%.

```
Accuracy: 0.8146633960207141
Precision: 0.824486255096862
Recall: 0.8146633960207141
```

Figure 2: Results of k-NN classification

Overall, this experiment showcases the transferability of features learned by a pre-trained ResNet-50 model for image classification tasks, emphasizing the potential of leveraging pre-trained models and k-NN classifiers for efficient and accurate image classification without the need for extensive fine-tuning.

Question (b): Linear Classification with Pre-trained Embedding.

In multi-class logistic regression, we used the pre-trained image encoder as the feature extractor part and added the custom classifier network to classify the 37 different pet breeds. Used TensorFlow framework for developing image classification model. The final developed network implemented using DenseNet201 pre-trained network with added classifier consists of GlobalAveragePooling2D(), Dropout(0.2) and Dense(37,activation='softmax) layers.

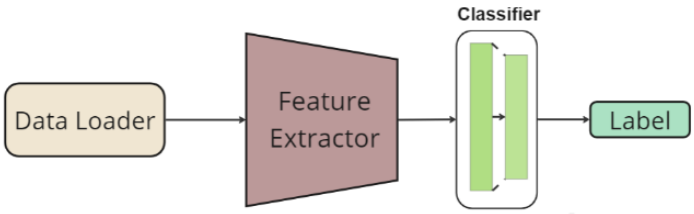

Figure 3: Model architecture

Figure 4: Sample dataset images with labels

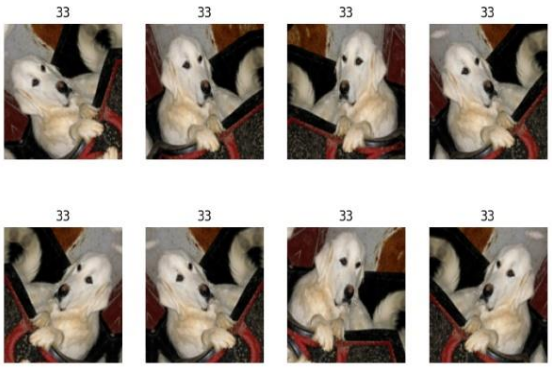

Figure 5: Data augmentation for random sample image

- The processed Oxford-IIIT Pet Dataset is used. The processed version has separate train and test datasets. The validation set is obtained as 20% of the training dataset. Finally, dataset splits as train, validation and test sets have image samples of 2944, 736 and 3669. Uses data augmentation techniques such as RandomFlip('horizontal') and RandomRotation() for training datasets.
- **Training details and results:** Use Adam optimizer, Sparse categorical cross-entropy loss, and final training was done for 40 epochs. In the training procedure, pre-trained model weights are set to be non-trainable and only the added classifier network was trained. Finally obtained a training accuracy of 99.69% and a validation accuracy of 87.50%.
- **Test results:** Test accuracy when evaluating the model on the test dataset was 85.91%

```
Layer (type)                 Output Shape              Param #   Trainable
=================================================================
input_2 (InputLayer)         [(None, 150, 150, 3)]     0         Y

rescaling (Rescaling)        (None, 150, 150, 3)       0         Y

densenet201 (Functional)     (None, 4, 4, 1920)        18321984  N

global_average_pooling2d (   (None, 1920)              0         Y
GlobalAveragePooling2D)

dropout (Dropout)            (None, 1920)              0         Y

dense (Dense)                (None, 37)                71077     Y

=================================================================
Total params: 18393061 (70.16 MB)
Trainable params: 71077 (277.64 KB)
Non-trainable params: 18321984 (69.89 MB)
```

Figure 6: Model summary details

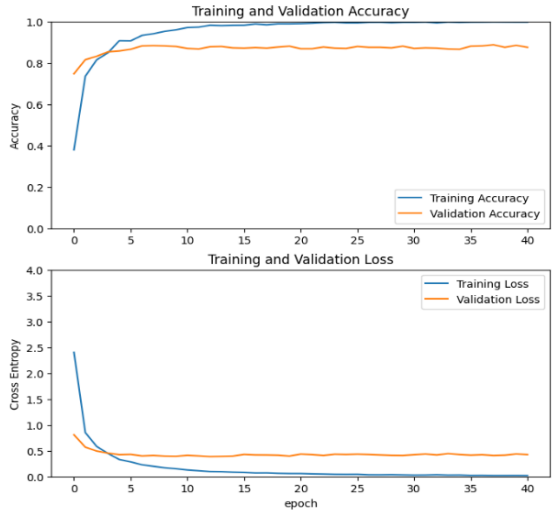

Figure 7: Plots of accuracy and loss

- The following table provides the comparisons of different developed network details and results on the train, validation and test accuracy.

Table1

Pre-trained Network	Added Classifier	Training details	Training epochs	Train acc	Val acc	Test acc
ResNet50	• Flatten() • Dense(1024, activation='relu') • Dropout(0.2) • Dense(37,activation='softmax')	• Adam Optimizer • Sparse categorical cross-entropy loss	30	97.08%	60.73%	56.72%
VGG16	• Flatten() • Dense(512, activation='relu') • Dropout(0.2) • Dense(37,activation='softmax')	• Adam Optimizer • Sparse categorical cross-entropy loss	20	65.63%	50.62%	45.72%
Xception	• GlobalAveragePooling2D() • Dropout(0.2) • Dense(37,activation='softmax')	• Adam Optimizer • Sparse categorical cross-entropy loss	20	99.80%	80.57%	80.73%
NASNetLarge	• GlobalAveragePooling2D() • Dropout(0.2) • Dense(37,activation='softmax')	• Adam Optimizer • Sparse categorical cross-entropy loss	15	99.63%	83.70%	82.47%
DenseNet201	• GlobalAveragePooling2D() • Dropout(0.2) • Dense(37,activation='softmax')	• Adam Optimizer • Sparse categorical cross-entropy loss	40	99.69%	87.50%	85.91%

The developed network contains a pre-trained model with different variations of added classifiers, optimizers, learning rates, and training epochs and the above table provides the final comparisons of the best categories under each section.

Question (c): Fine-tune an Image Classification Network.

Final network: Model implemented using DenseNet201 pre-trained network with added classifier consisting of GlobalAveragePooling2D(), Dropout(0.2) and Dense(37,activation='softmax) layers. Used the same final model network used in question (b).

Fine-tuning procedure: When fine-tuning a model, initially enabling training for the entire base model might lead to the loss of the valuable knowledge acquired during pretraining. To address this, a common approach is to start training only the top-level classifier with the pre-trained model's weights frozen (non-trainable) for a few epochs (same as question (b)). This initial phase allows the new classifier to adapt to the task. Subsequently, the pre-trained model's layers are made trainable, and the fine-tuning process continues. This method helps prevent large and potentially disruptive gradient updates caused by random weight initialization in the classifier.

- **Training details and results:** Use Adam optimizer, Sparse categorical cross-entropy loss, and final training was done with 2 initial epochs and 13 fine-tuned epochs. Obtained training accuracy of 100% and validation accuracy of 90.08%.
- **Test results:** Test accuracy when evaluating the model on the test dataset was 86.84%. Test accuracy was improved by 0.93% compared to question (b).

```
Layer (type)                 Output Shape              Param #   Trainable
=================================================================
input_2 (InputLayer)         [(None, 150, 150, 3)]     0         Y

rescaling (Rescaling)        (None, 150, 150, 3)       0         Y

densenet201 (Functional)     (None, 4, 4, 1920)        18321984  Y

global_average_pooling2d (   (None, 1920)              0         Y
GlobalAveragePooling2D)

dropout (Dropout)            (None, 1920)              0         Y

dense (Dense)                (None, 37)                71077     Y

=================================================================
Total params: 18393061 (70.16 MB)
Trainable params: 18164005 (69.29 MB)
Non-trainable params: 229056 (894.75 KB)
_________________________________________________________________
```

Figure 8: Fine-tune model summary details

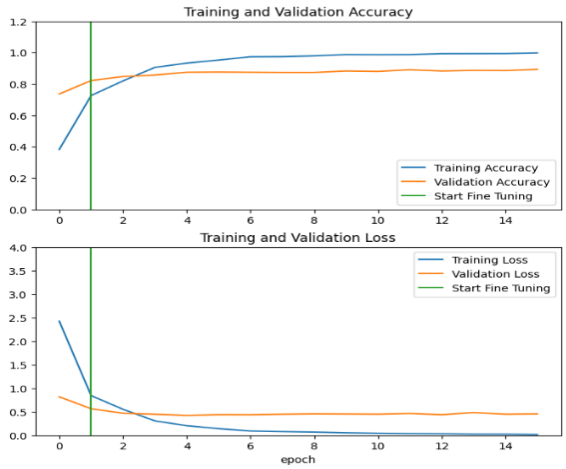

Figure 9: Plots of accuracy and loss

- The train, validation and test accuracy comparisons of different developed networks results are shown in the table below.

Table 2

Pre-trained Network	Train accuracy	Validation accuracy	Test accuracy
Xception	99.73%	84.78%	83.02%
NASNetLarge	100%	85.60%	85.01%
DenseNet201	100%	90.08%	86.84%

References

1. https://keras.io/guides/transfer_learning/
2. https://www.tensorflow.org/tutorials/images/transfer_learning
3. The processed version in https://www.tensorflow.org/datasets/catalog/oxford_iiit_pet