Keras Dog-Cat Image Classification

Anh Duc Le – May 2020 – Cork, Ireland

Table of Contents

1. Introduction

This report is to provide a detailed explanation of applying convolutional neural network (CNN) for a classification task of classifying dogs vs cats by using the Kaggle dataset. The report is constructed as follows: Section 2 discussing the CNN's key features, Section 3 explaining the methodology used for the classification task, and finally Section 4 evaluating CNN's performance for the classification task.

2. Convolutional Neural Network

Convolutional Neural Network (CNN) is a feedforward neural network in which information flow takes place in one direction from an input layer to multiple hidden layers and finally an output layer. The hidden layers of a CNN commonly include convolutional layers, pooling layers and fully connected layers. Notably, convolutional and pooling layers are the most important features distinguishing CNN from other neural networks (Stanford University, 2020).

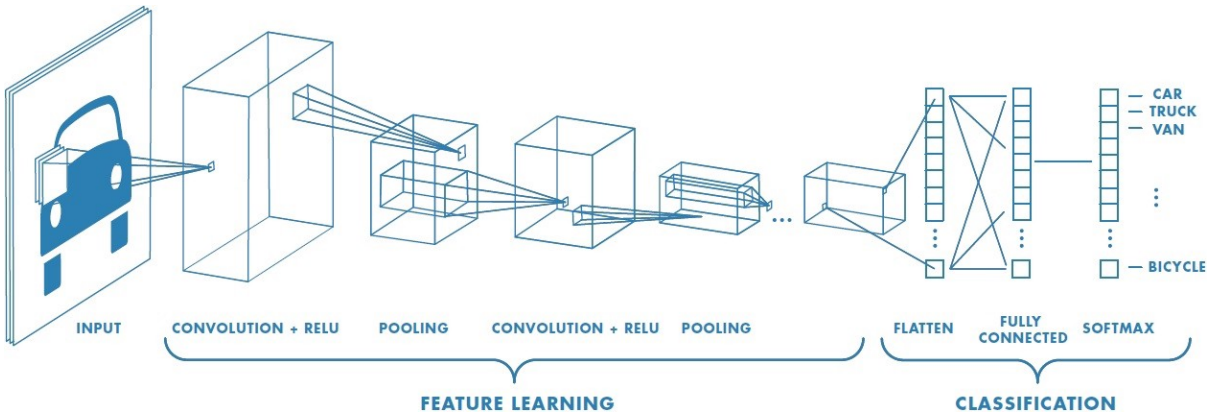


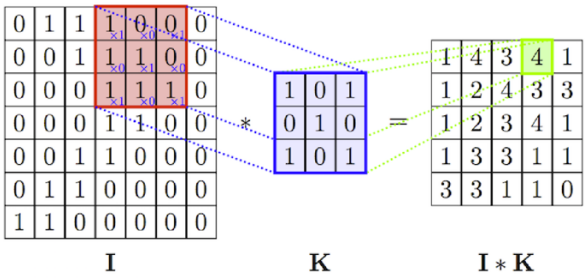
Figure 1: CNN architecture

2.1. CNN's Key features

2.1.1. Convolutional Layers

The convolutional layers extract different features of the input. The first convolutional layer extracts low-level features like edges, lines and corners. High-level layers extract high-level features. Image feature extraction is done by the convolution kernel which is described in equation form as below, where K is the convolution kernel, I

is the image data in the for of matrix, h is the height of the image, and w is the width of image (Murphy, 2018).



$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

A complete convolutional layer includes a number of kernels transforming an input image into multiple output images which are then fed into an activation function to produce feature maps as the output for the subsequent layers. The most common activation functions are sigmoid function, hyperbolic tangent function and ReLU function. The convolutional layers can be customised by specifying the feature extractions, i.e. number of kernels and biases to be convolved with the input image, the size of the kernels, and the stride which is the number of pixels the kernel is slid along each time it is moved across the image. The following examples show how different kernels can transform the input image into different outputs for the purpose of edge detection in the convolutional layers (Rawat & Wang, 2017) (Figure 2)

2.1.2. Pooling Layers

The pooling layer, which comes after the convolutional layer, is to reduce the resolution of the feature maps to reduce the number of parameters to be computed but to achieve invariance to translations in shape, size and scale. Two common types of pooling are average pooling and max pooling. The max pooling is to select the largest element within a fixed-sized receptive field such at

$$Y_{kij} = \max_{(p,q)\in\mathcal{R}_{ij}} x_{kpq}$$

Where Y_{kij} is the output of the pooling operations associated with the k-th feature map, x_{kpq} is the element at location (p, q) contained by the pooling region $\mathcal{R}_{ij}$, a receptive

field around the position (i, j) (Yu et al., 2014). The difference between max pooling and average one is illustrated in Figure 3 The max pooling selects the max value of each receptive field; meanwhile, the average pooling averages all values in each receptive field (Rawat & Wang, 2017).

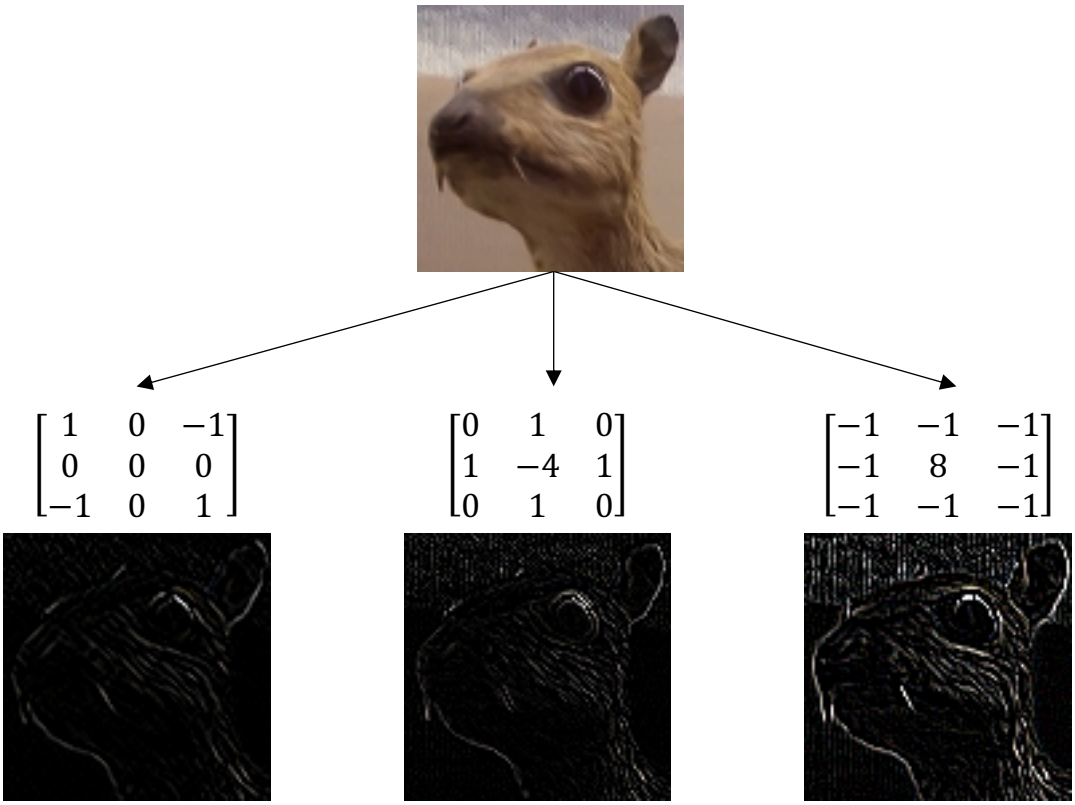


Figure 2. Convolution Kernels

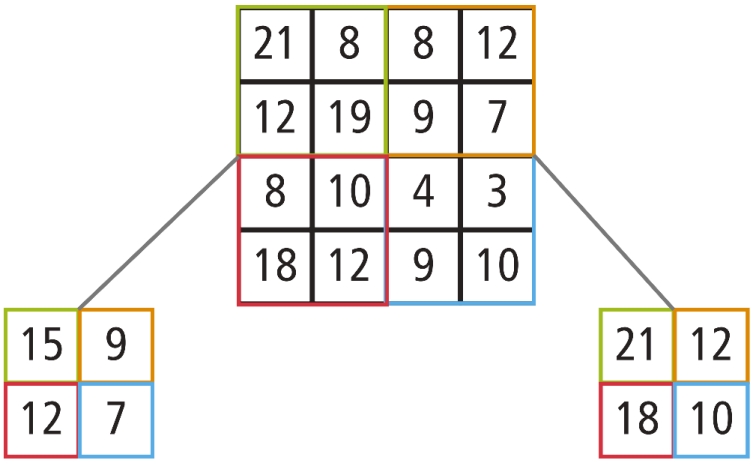


Figure 3: Average Pooling (left) vs. Max Pooling (right)

2.2. Other important factors

This section is talking about two important regularisation techniques which are commonly used in CNN to avoid the risk of overfitting.

2.2.1. Batch Regularization

This operation takes the output of a particular layer, and rescales it so that it always has 0 means and standard deviations of 1 in every batch of training. This is proposed by Sergey Loffe and Christian Szegedy in 2015. The algorithm solves the problem where different batches of input might produce wildly different distributions of outputs an any given layer in the network. Because the adjustments to the weights through back-propagation depends on the activation of the units in every step of learning. This means that the network may progress very slowly through training.

2.2.2. Drop out

This regularization strategy allows model train many different networks on different parts of the data. Each time, the network that is trained is randomly chosen from the full network. This way, if part of the network becomes too sensitive to some noise in the data, other parts will compensate for this, because they haven't seen the samples with this noise. It also helps prevent different units in the same network from becoming overly correlated in their activity. In another word, if one unit learns to prefer horizontal orientations, another unit would be trained to prefer vertical ones.

3. Methodology

3.1. Data pre-processing

The Kaggle Data contains 2 folders including train and test1. The train folder, which has 25,000 images labelled cats and dogs, is split into 3 parts for training (50%), validation (25%) and testing (25%). For the test1 folder with 12,500 images without labels, it is used for making predictions whether images are cats' or dogs' (Figure 4)

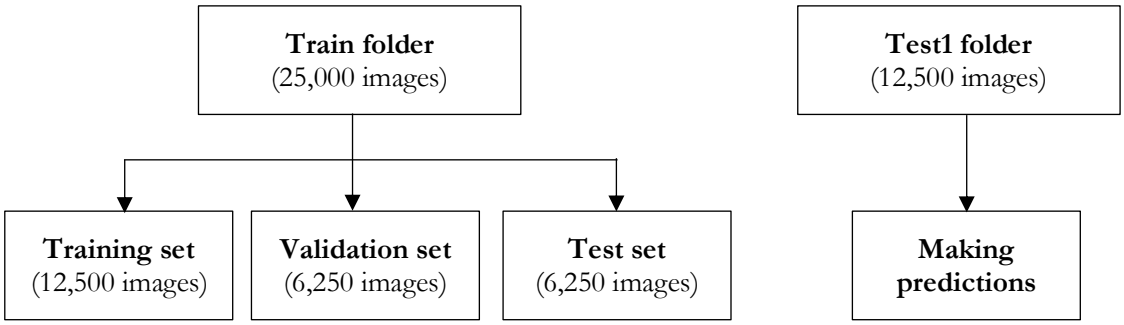


Figure 4. Data gathering

The next step is pre-processing images which should be formatted appropriately before being fed into the network by using the ImageDataGenerator class in the Keras library. The ImageDataGenerator is used to read the image JPEG files, decode the JPEG content to RGB grids of pixels with a predefined target size, i.e. 128×128 pixels in our case, convert those pixels into matrices, and finally rescale the pixel values between 0 and 255 to $[0, 1]$ interval which is preferable for the network to perform calculations.

Besides, to acquire good classification models being invariant to images' rotation, scale, viewpoint and occlusion, ImageDataGenerator is used to perform augmentation on the training data by generating more training data from the existing training samples via random transformations (Figure 5). This enables the classification models to learn different variations of the images so that the overfitting could be potentially avoided.

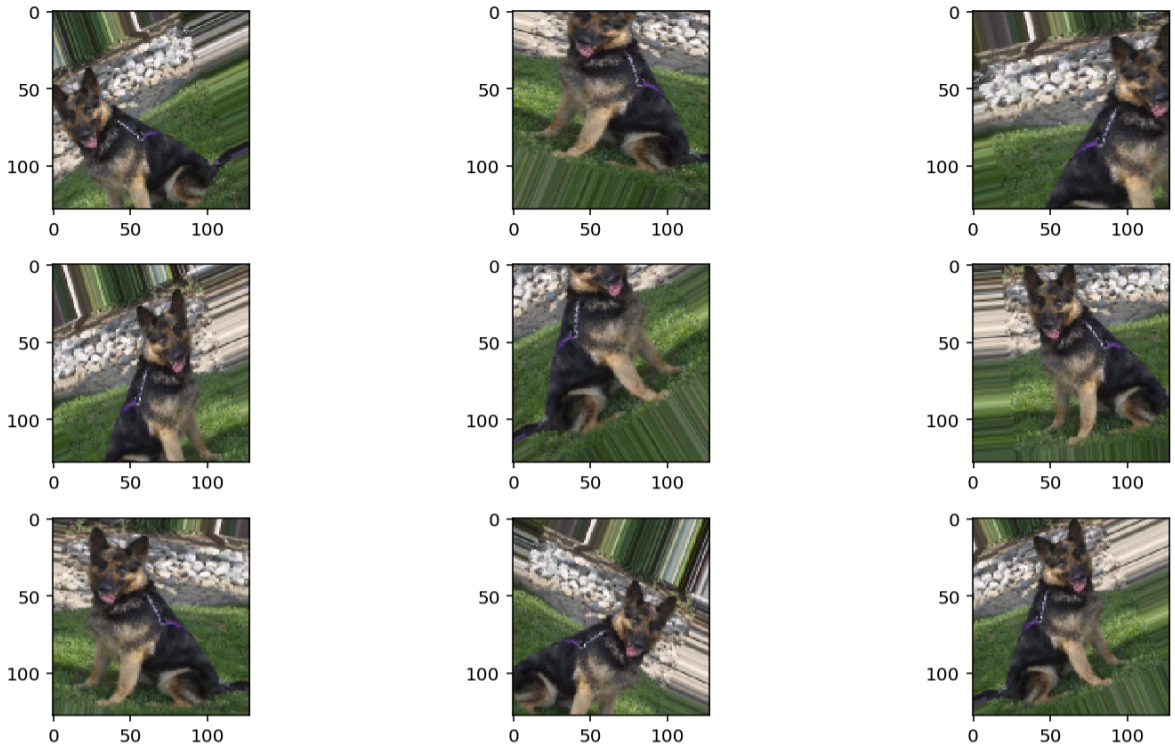


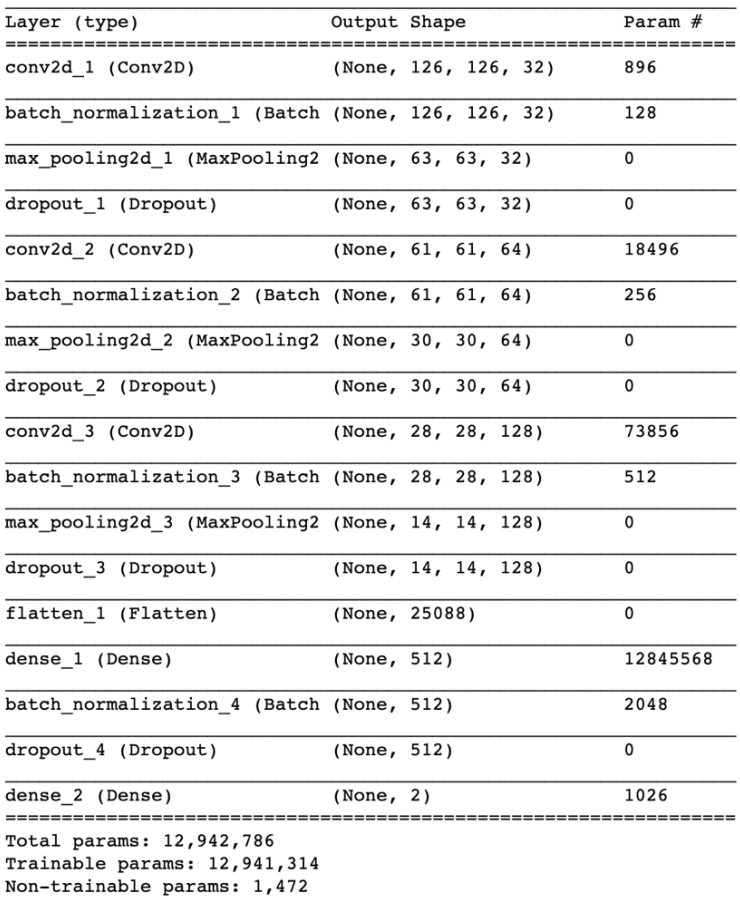
Figure 5. Image Augmentation

3.2. Building Models

In this section, two CNN models are used to perform the classification task. The first model is manually built, mainly featuring 3 convolutional layers (Model 1). The second model is the pre-trained VGG16 model connected with 2 densely-connected output layers (Model 2).

3.2.1. Manually built CNN model (Model 1)

Figure 6 shows the Model 1's architecture as follows



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 126, 32)	896
batch_normalization_1 (Batch	(None, 126, 126, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 63, 63, 32)	0
dropout_1 (Dropout)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_2 (Batch	(None, 61, 61, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 30, 30, 64)	0
dropout_2 (Dropout)	(None, 30, 30, 64)	0
conv2d_3 (Conv2D)	(None, 28, 28, 128)	73856
batch_normalization_3 (Batch	(None, 28, 28, 128)	512
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 128)	0
dropout_3 (Dropout)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 512)	12845568
batch_normalization_4 (Batch	(None, 512)	2048
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1026

Total params: 12,942,786
Trainable params: 12,941,314
Non-trainable params: 1,472

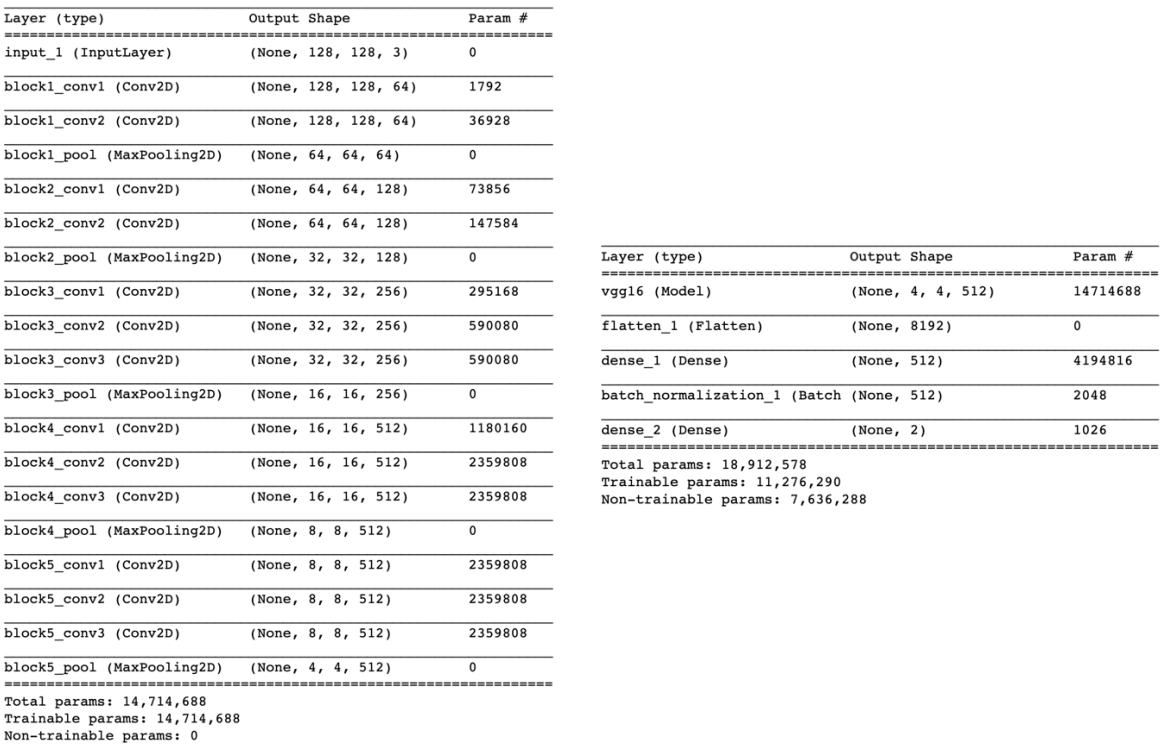
Figure 6. Model 1's architecture

The main features of Model 1 are the three 3x3-kernel convolutional layers having the number of the neuron units of 32, 64 and 128 in the sequential order. Those convolutional layers are followed by Batch Normalisation layers, 2x2 Max Pooling layers and Drop out layers. Finally, two densely-connected layers connected with the previous convolutional blocks are used to produce classification estimates.

To compile Model 1, the cross-entropy loss function and the RMSprop optimiser with the customised learning rate of 0.0001 are used. To avoid overfitting, early stopping is applied to monitor the validation's loss so that the training process will stop after 5 epochs of no decreasing validation's loss.

3.2.2. Customised pre-trained CNN model (Model 2)

Figure 7 shows the Model 2's architecture comprising of the of the VGG16 architecture (on the right) and two additional densely-connected layers on top (on the left).



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 128, 128, 3)	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36928
block1_pool (MaxPooling2D)	(None, 64, 64, 64)	0
block2_conv1 (Conv2D)	(None, 64, 64, 128)	73856
block2_conv2 (Conv2D)	(None, 64, 64, 128)	147584
block2_pool (MaxPooling2D)	(None, 32, 32, 128)	0
block3_conv1 (Conv2D)	(None, 32, 32, 256)	295168
block3_conv2 (Conv2D)	(None, 32, 32, 256)	590080
block3_conv3 (Conv2D)	(None, 32, 32, 256)	590080
block3_pool (MaxPooling2D)	(None, 16, 16, 256)	0
block4_conv1 (Conv2D)	(None, 16, 16, 512)	1180160
block4_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block4_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block5_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 512)	4194816
batch_normalization_1 (Batch	(None, 512)	2048
dense_2 (Dense)	(None, 2)	1026

Total params: 18,912,578
Trainable params: 11,276,290
Non-trainable params: 7,636,288

Figure 7. Model 2's architecture

The VGG16 architecture is a CNN architecture trained on the ImageNet dataset (1.4 million labelled images and 1,000 different classes). ImageNet contains many animal classes, including different species of cats and dogs. Therefore, using the pre-trained network might be suitable for our classification problem. To maximise the VGG16's capability, the pre-trained weights for first four convolutional blocks are kept the same since those first four blocks were already trained to recognise distinguishable features of cats and dogs. Weights of the last convolutional block are trained so that the model could adapt well to our dataset. For the two final densely-connected layers connected with the previous layers of VGG16, they are used to produce classification estimates.

To compile Model 2, the cross-entropy loss function and the RMSprop optimiser with an extremely small learning rate of 0.00001 are used. The reason for using a low learning rate is to limit the magnitude of the modifications for the last convolutional bocks. The number of epochs is 50.

4. Evaluations and Conclusions

4.1. Model 1

Figure 8 shows that the model was somehow successfully trained with the convergence of training loss and validation loss. Similarly, Figure 9 also indicates that the validation accuracy goes closely around with the training accuracy. However, there might be a need of increasing the number of epochs for early stopping since the model was just trained on 15 epochs. Test Model 1 on the test set of 6,250 images, accuracy is 76.48% and loss is 0.6506.

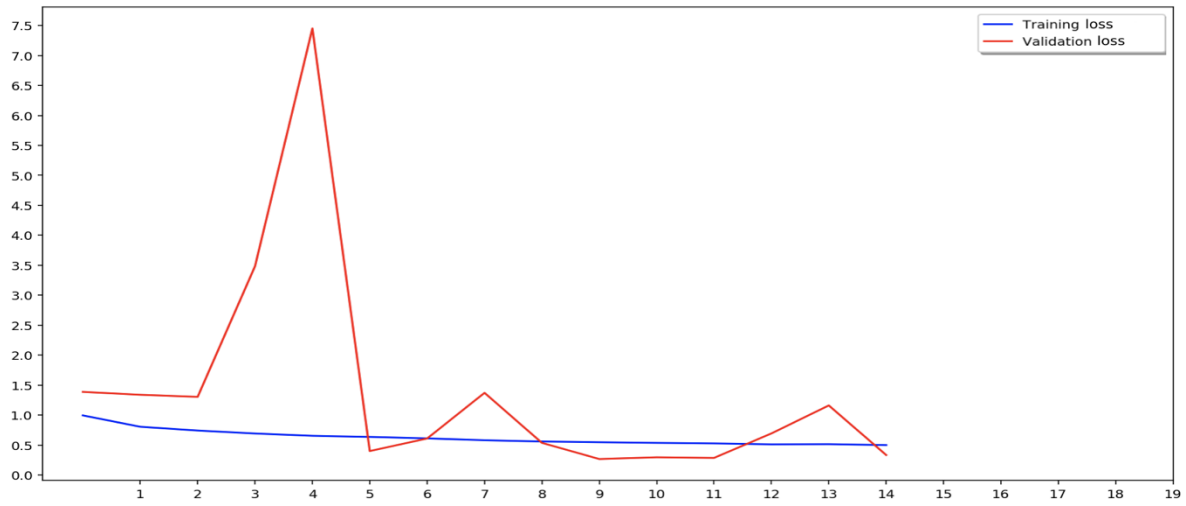


Figure 8. Training loss vs Validation loss during training process (Model 1)

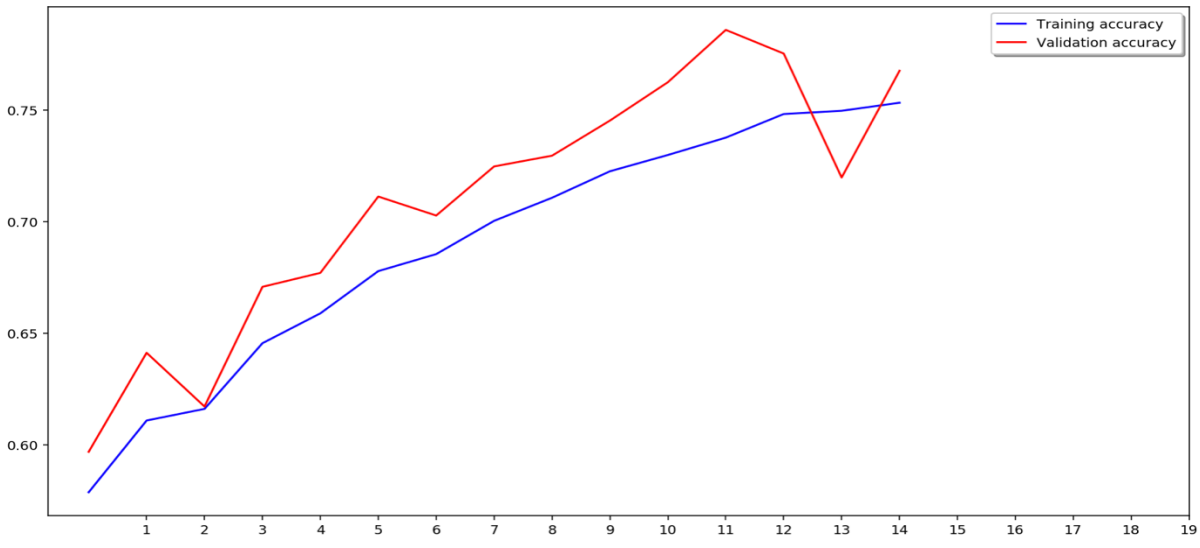


Figure 9. Training vs Validation accuracy during training process (Model 1)

4.2. Model 2

Figure 10 shows that the model might experience overfitting since the convergence of training loss and validation loss was not clearly shown. Similarly, Figure 11 also indicates that the validation accuracy diverges from the training accuracy. It prompts the need of adjusting Model 2' s specifications to yield better performance. Test Model 2 on the test set of 6,250 images, accuracy is 93.12% and a very small loss is of 0.0009.

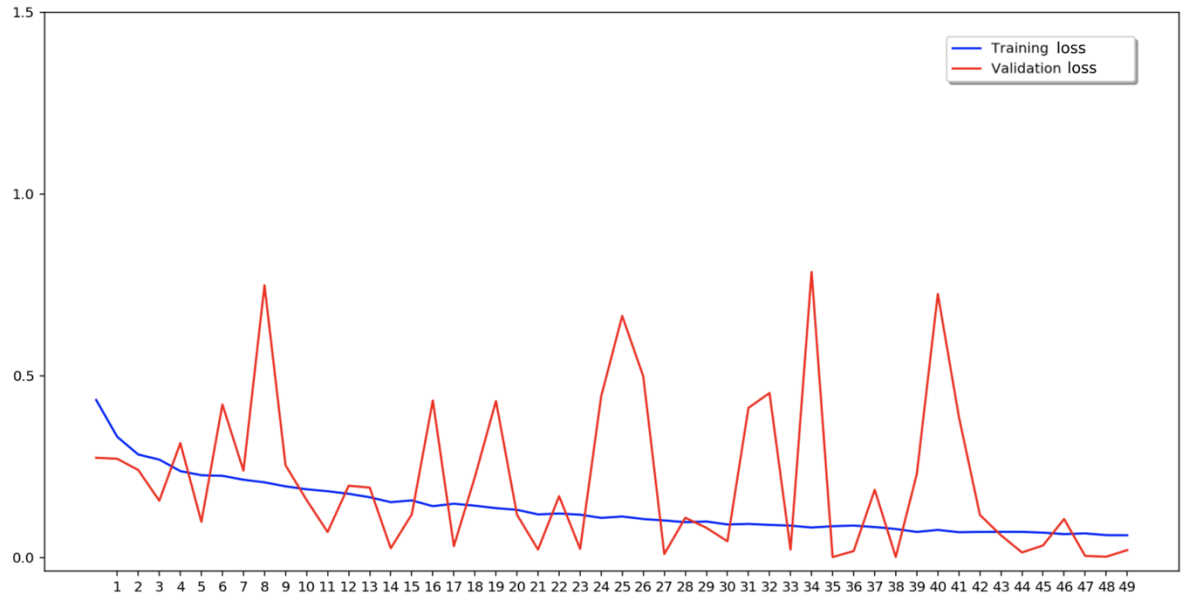


Figure 10. Training loss vs Validation loss during training process (Model 2)

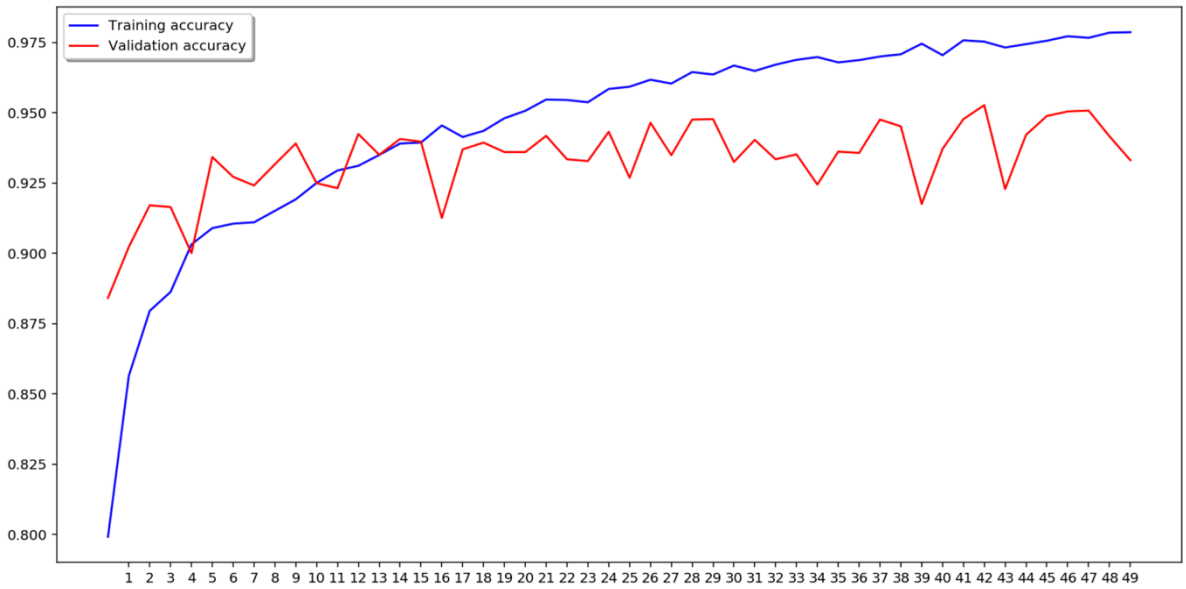


Figure 11. Training vs Validation accuracy during training process (Model 2)

It is clear that Model 2 has better performance than Model 1 although Model 2 exhibits patterns of overfitting to a certain extent. Model 2 might reveal the benefits of using the pre-trained CNN models which were already trained on the similar classification task. However, there is a strong need of adjusting Model 2's specifications such as applying additional drop-out and batch normalisation layers to avoid overfitting problem.

References

Murphy, S. (2018). *Telecommunications signal performance classification using Deep Learning techniques*. Cork: Cork Institute of Technology.

Rawat, W. & Wang, Z. (2017). Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*. 29(9), pp.2352-2449.

Stanford University, (2020). *Convolutional Neural Networks for Visual Recognition* [Online]. Available from: https://cs231n.github.io/convolutional-networks/ [Accessed 14th May 2020].