Melbourne School of Computing and Information Systems

COMP90086 Machine Learning - Assignment 2:

CNN for Image Classification

Tuan Khoi Nguyen - 1025294

Question 1: Network Parameters

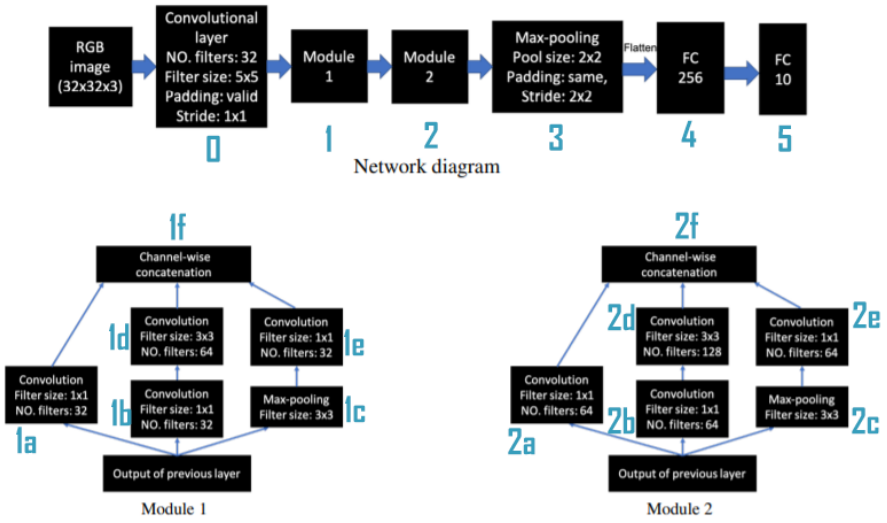

Figure 1: Labelling of the CNN layers

The formulation for different types of layers can be evaluated as follows:

- Output size (width/height):
  - Convoluational, fully connected layer: $\begin{cases} ceil(\frac{size(input)-size(kernel)+1}{size(stride)}) & \text{for same padding} \\ ceil(\frac{size(input)}{size(stride)}) & \text{for valid padding} \end{cases}$
  - Concatenation layer: Retain the size of the inputs, which also share the same size.
  - Fully Connected Layer: $size = w_{input} \times h_{input}$
- Number of output channels c:
  - Convolutional layer: $c = n_{filters}$, where $n_{filters}$ is the number of filters in the layer.
  - Max pooling layer: c remains the same.
  - Concatenation layer with connected inputs i_x: $c = \sum n_{channels}(i_x)\ for\ i_x \in inputs(layer)$
- Flattening: $size(output) = w_{input} \times h_{input} \times c_{input} = (w \times h \times c)_{input}$.
- Number of parameters p and multiplications m:
  - Convolutional layer: $\begin{cases} p = n_{filters} \times c_{input} \times w_{filter} \times h_{filter} = n_{filters} \times c_{input} \times (w \times h)_{filter} \\ h = c_{input} \times (w \times h)_{filter} \times (w \times h \times c)_{output} \end{cases}$
  - Max pooling layer: $p = m = 0$, due to being a transform layer.
  - Concatenation layer: $p = m = 0$, due to being a transform layer.
  - Fully connected layer:
    - $p = size(layer) \times size(input)$
    - $m = p$, given that the fully connected layer is equivalent to a single 1×1 filter.

Substituting the input variables to the corresponding equations will obtain the following table:

c: Number of channels, w: width size, h: height size

Layer	**Type**	**Output size**	**Parameters** $= n_{filters} \times c_{input} \times w_{filter} \times h_{filter}$	**Multiplications** $= c_{input} \times (w \times h)_{filter} \times (w \times h \times c)_{output}$
0	Convolutional, no padding	$w = h = ceil(\frac{32-5+1}{1}) = 28$ $c = 32$ Final size: (28,28,32)	$32 \times 3 \times 5 \times 5 = 2400$	$3 \times 5 \times 5 \times 28 \times 28 \times 32 = 1881600$
1a & 1b	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 32$ Final size: (28,28,32)	$32 \times 32 \times 1 \times 1 = 1024$	$32 \times 1 \times 1 \times 28 \times 28 \times 32 = 802816$
1c	Max pooling, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 32$ Final size: (28,28,32)	0	
1d	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 64$ Final size: (28,28,64)	$64 \times 32 \times 3 \times 3 = 18432$	$32 \times 3 \times 3 \times 28 \times 28 \times 64 = 14450688$
1e	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 32$ Final size: (28,28,32)	$32 \times 32 \times 1 \times 1 = 1024$	$32 \times 1 \times 1 \times 28 \times 28 \times 32 = 802816$
1f	Channel-wise Concatenation of 1a,1d,1e	$w = h = 28$ $c = 32 + 64 + 32 = 128$ Final size: (28,28,128)	0	
2a & 2b	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 64$ Final size: (28,28,64)	$64 \times 128 \times 1 \times 1 = 8192$	$128 \times 1 \times 1 \times 28 \times 28 \times 64 = 6422528$
2c	Max pooling, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 128$ Final size: (28,28,128)	0	
2d	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 128$ Final size: (28,28,128)	$128 \times 64 \times 3 \times 3 = 73728$	$64 \times 3 \times 3 \times 28 \times 28 \times 128 = 57802752$
2e	Convolutional, padding	$w = h = ceil(\frac{28}{1}) = 28$ $c = 64$ Final size: (28,28,64)	$64 \times 128 \times 1 \times 1 = 8192$	$128 \times 1 \times 1 \times 28 \times 28 \times 64 = 6422528$
2f	Channel-wise Concatenation of 2a,2d,2e	$w = h = 28$ $c = 64 + 128 + 64 = 256$ Final size: (28,28,256)	0	
3	Max pooling, padding, flatten	$w = h = ceil(\frac{28}{2}) = 14$ $c = 256$ Final size: (14,14,256) **Flattened size:** $14 \times 14 \times 256 = 50176$	0	
4	Fully Connected Layer	256	$256 \times 50716 = 12845056$	
5	Fully Connected Layer	10	$10 \times 256 = 2560$	

Table 1: All layers and its calculations, ordered from initial input to final output

Question 2: Classification on CIFAR-10 Subset

2.1 Evaluation on CNN accuracy

The result from figure 2 shows that testing accuracy lies very low at around 0.3, comparing to the training accuracy that reaches close to 1, suggesting that the model is overfitting, a common problem in CNN models, as neural networks tend to try and get the most information from the training dataset.

This vulnerability happens due to small number of instances in training data and large feature space in the model, which makes the training dataset having many redundant or irrelevant features and not being able to generalise well, hence not being representative for testing dataset. Furthermore, given that the data is not regularised, the data will have significant variance. As a result of these, the model will only fit to the trend of the training dataset, leading to wrong predictions on the testing dataset that result in lower accuracy.

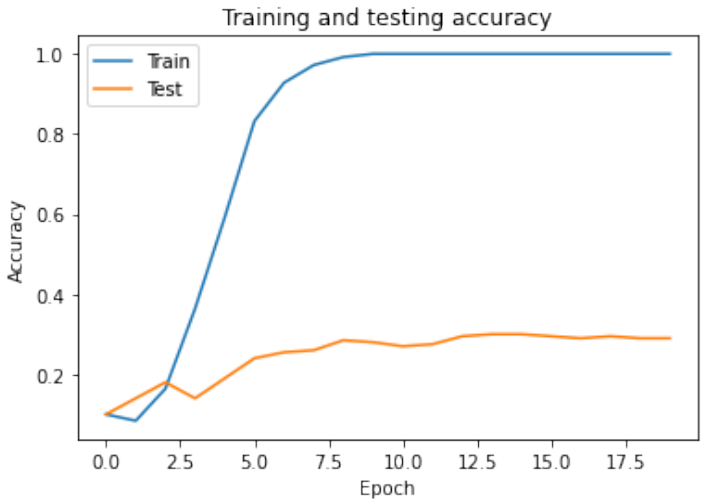

Figure 2: Accuracy of CNN model under different epochs.

2.2 Evaluation on MLP accuracy

The results from figure 3 shows low accuracy in both training and testing. Testing accuracy is also lower than training accuracy, which once again confirms that the neural network model is overfitting due to learning information from a low number of training instances and large number of parameters.

The lower and unstable accuracy in both training and testing proves that the model does not learn enough information from the data to distinguish instance labels. This not only makes the instances hard to identify, but also cause high randomness of the accuracy throughout each epoch. To make the instances more distinguishable without affecting the model architecture, feature extraction is needed in order to reduce dimensionality and aid the MLP model in learning more valuable information.

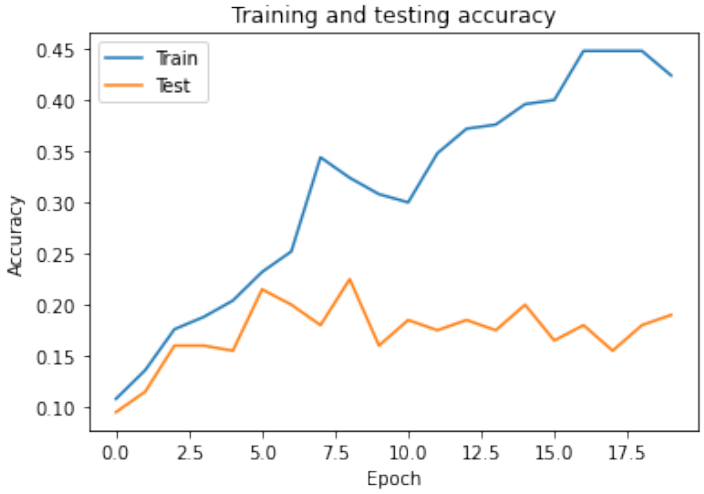

Figure 3: Accuracy of MLP model under different epochs.

2.3 Feature extraction with VGG19

VGG19 is a commonly used feature extraction method for Convolutional Neural Networks. Consisting of pre-trained weights from ImageNet dataset, using VGG19 architecture helps to modify instances effectively, as well as reducing the number of parameters needed, which helps in less overfitting and more feature representation. Normalization is also done prior to fitting, so that the instances will have more consistent features.

To select the most suitable layer for retrieval, feature extraction is done on all layers of the VGG19 architecture and compared against each other on the accuracy at final epoch, as shown in Figure 4. Running the code script shows that the 1st Convolutional Layer at the 5th block (`block5_conv1`) has the highest testing accuracy of 0.53, and therefore will be used as the extraction layer for the CIFAR-10 subset.

Figure 4: The testing accuracy through different epochs of VGG19 layers

Outcome

With the output layer chosen, the MLP model is run again on the extracted dataset, showing the following result on Figure 5.

As seen in Figure 5, both accuracy are improved: 0.45 to 1 for training accuracy, and from 0.2 to 0.5 for testing accuracy. This shows that VGG19 feature extraction is effective in exploiting distinguishable features of the instances. The testing accuracy still remains lower than training accuracy due to overfitting, therefore, future improvement may include increasing a diverse amount of labelled instances to the training dataset to increase its representation, or apply further feature reduction methods.

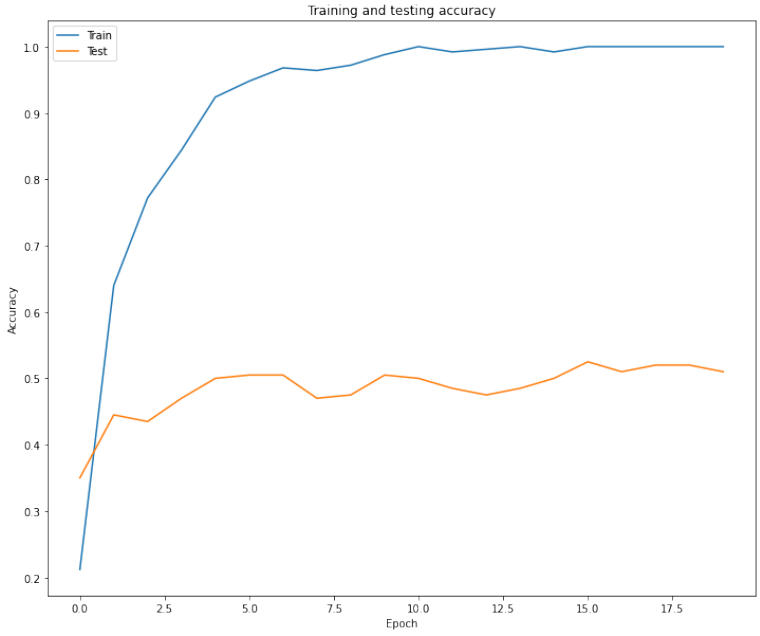

Figure 5: Accuracy of MLP model through each epoch, after applying VGG19 feature extraction