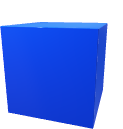
# 1.0 Computation related to CNN Architecture

**Background/ Concepts / Formulas to take note**

There are three types of layers in Convolutional Neural Network (CNN) namely: **CPF**

**\*\* Parameters are association of a neuron to another. The connection of that is basically the parameter or weights.**

1. Convolutional Layer –
   1. ****Concept: Through the convolution operations, expect the number of filters to increase while capturing more complex patterns, and expecting to see the height width of the image to reduce, and the depth to increase/elongate due to the increased number of filters used.

****

* 1. Concept: Each kernel or filter may be responsible to detect some patterns like horizontal line, vertical line, etc. A combination of the filters and the increase of it allows the network to learn more diverse and complex features.
  2. CNN is designed for hierarchical representations; layers nearer to the input will learn lower-level features, as we move deeper into the network, these layers will learn more abstract features (higher level features) because more filters contribute to this hierarchy for the network to learn these abstractions.

|  |  |  |  |
| --- | --- | --- | --- |
| Item | ConvLayer1 | ConvLayer2 | ConvLayer3 |
| Spatial Dimension of Activations/Feature Maps | 37 | 17 | 7 |
| Num of filters | 10 | 20 | 40 |
| Activations Depth | 10 | 20 | 40 |
| Total Number of Params | 280 | 1520 | 3040 |

* 1. Formulas: Remember by **NPF/S**. We can swap P with S, so achieve NSF. where N is the size of the input, P is the padding and F is the filter size, S is the stride value.
  2. Concept – learnable weights or parameters are referred to the same thing. And they “physically” reside at the filter/kernels. The feature extraction process: Filter is the one that performs the transformation of input to the output (activations/feature maps) So if needed to compute the total parameters of convLayerX, then we first can compute the number of parameters found in a single filter. Subsequently, sum all the number of filters to find the total number of learnable parameters.

To imagine the number of parameters in a single filter, we can imagine a cuboid (3D), where each cell of the volume is a parameter. A bias term Is usually added for each filter for potential adjustments. Therefore, the bias term also accounts for one additional bias term. Each filter has 1 + (filter height\* filter width\*filter channels/depth) Pooling Layer –

[deep learning - How to calculate the number of parameters for convolutional neural network? - Stack Overflow](https://stackoverflow.com/questions/42786717/how-to-calculate-the-number-of-parameters-for-convolutional-neural-network)

1. Fully Connected Layer / output layers –
   1. Fully connected layer
      1. By definition, Fully connected layer is a dense layer where every neuron of the previous layer is connected to every neuron is the current layer
      2. Output activations of Fully connected layer depend on the expected input number of classes to output layer.
      3. Has RELU activations to capture non-linear patterns.
   2. Output layer
      1. Output of the fully connected layer usually doesn’t allow to directly interpret probabilities. By definition, Output layer is the final layer of the neural network
      2. For a binary classification, the activation function of output layer is a sigmoid whereas it is a softmax activation for a multi-class classification.
      3. For regression, it is just a linear activation function (no activation?)
      4. Output activations of output layers depend on expected number of classes to predict
      5. e.g. 10 classes; (10,1)
      6. e.g. binary class; (1,1) – one neuron is suffice
      7. Each output is connected to not only n inputs but also 1x bias term. Suppose we have m outputs, then total number of parameters (line connections) is m\*(n+1)

ConvLayer1

Randomly selected first input image volume, of 39 x 39 x 3 where the height and width of image is 39, and a depth/channel of 3 (RGB) from the large collection of images from CIFAR10 dataset, compute the resulting activations and the number of learnable weights given number of filters at convLayer1, , and the stride value and there is no padding applied around the image ,

Resulting convolutional operation of filters with the input volume aka Activations volume is 37 x 37 x 10.

* Each filter (akin to a stamp to detect some pattern) will be used to “stamp through” a stack of transparent papers, since we have 32 filters, then the total number of stamped result (sheet) is also 32 as the output i.e. 32 different number of stamps.
* The resulting size of the output activations from the operation of this stamp/convolution is some height and some width. Fortunately, the height of the output is the width so we can compute as one unit using following formula:
  1. + 1 =

why need to + 1 ? when do we need to do ceiling?

~~Total number of learnable parameters from this ConvLayer1 is 37 x 37 x 32? => 43, 808 parameters.~~

ConvLayer2

Input of ConvLayer2 is the activations from ConvLayer1. Compute the number of learnable weights given number of filters at convLayer2, and the stride value and there is no padding applied around the input,.

Resulting convolutional operation of filters – ConvLayer2 activations is 17 x 17 x 20.

1. Like previous computation of ConvLayer1 where we find the depth of output activations which is basically the number of filters used in ConvLayer2 => 20
2. To find the resulting size of convolutional operation, we can use the following formula:

ConvLayer3

Input of ConvLayer3 is the activations from ConvLayer2. Compute the number of learnable weights given number of filters at convLayer2, and the stride value and there is no padding applied around the input,.

Resulting convolutional operation of filters – ConvLayer3 activations is 7 x 7 x 40.

1. Like previous computation of ConvLayer2 where we find the depth of output activations which is basically the number of filters used in ConvLayer3 => 40
2. To find the resulting size of convolutional operation, we can use the following formula:

[C4W2L09 Transfer Learning (youtube.com)](https://www.youtube.com/watch?v=FQM13HkEfBk)

Transfer learning is the task of taking a knowledge of performing a certain specific task, and transfers to performing another task. This leveraging of prior knowledge is popular because it works very well especially on an architecture that was trained on a lot of task A data – and then fine tune for performing task B using a small dataset.

Example, suppose an architecture that was previously trained on ImageNet dataset, we can remove the last layer of the architecture and its weights and then reattach a new layer or a few more hidden layers and train it (freezing the front part of the architecture); training it with another small dataset (say Xray dataset).

A diagram of a software development process

Description automatically generated



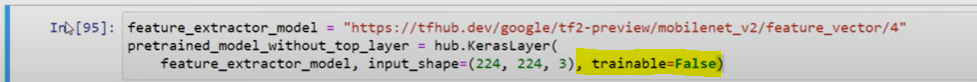
A diagram of a traffic light

Description automatically generated

A screenshot of a computer

Description automatically generated

Freezing the model – works by the following trainable = false. And then adding a last layer.



A screenshot of a computer

Description automatically generated

It will work well since the network that was pretrained on ImageNet dataset had very highly reported accuracy, must have learnt how to recognize curve/edges/or part of an object. Fine tuning on a pretrained network allows us to save time on training an entire from scratch because the low-level features of task A is helpful for task B. Had we retrained from scratch on the task B dataset, it would take a lot more epochs/iterations to get a good performance. A CNN that was trained from scratch on top of data augmentation strategy took 30 epochs. In order to obtain an 85% accuracy. However, by training only the last layer only took 2 epochs to reach 85%! Huge time saving and computational resources.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

[Transfer Learning | Deep Learning Tutorial 27 (Tensorflow, Keras & Python) (youtube.com)](https://www.youtube.com/watch?v=LsdxvjLWkIY)

One case that transfer learning would not make sense is that the architecture was pretrained on a relatively small dataset. Or even the inputs of task A and task B are different – different as in different types but size is acceptable.

Look up tensorflowhub to take a look at different types of pretrained network for reference.

# 2.0 Loss Functions

[133 - What are Loss functions in machine learning? (youtube.com)](https://www.youtube.com/watch?v=-qT8fJTP3Ks)

Definition of Loss Function: Loss function (LeeRoy) or also known as Objective function (lecture notes), Cost Function (Andrew Ng) or Error Function provides the quantification of the error ~~difference~~ between the predicted output y\_pred and target value y.

[Intuitively Understanding the Cross Entropy Loss (youtube.com)](https://www.youtube.com/watch?v=Pwgpl9mKars)

Regression: Common Loss Function

Linear regression commonly uses Mean Squared Error (MSE) as the loss function. The goal to having the best fit is to minimize the objective function defined (MSE).

Classification: Common Loss Function

For classification problems, cross entropy is the most commonly used loss function

Intuition of Cross Entropy loss, is the minimization of the KL divergence -> minimize the distance between P and Q. Let the true class distribution P\*(y | xi) be P and predicted class distribution P(y | xi;theta) be Q,

Then using the KL divergence formula

Question:

1. Why is it xi?
2. What does P(y | xi) really mean? Isit trying to find out what is the probability of predicting class label given data input xi?
3. What happens when I log the probability that’s less than zero?
4. What happens when I make a wrong prediction (aka true class distribution very diff from predicted class distribution)

A diagram of mathematical equations

Description automatically generated

Other intuition:

1. Measures uncertainty
2. Suppose the prediction distribution [0.77, 0.1, 0.01, 0.01] vs the ground truth distribution [1, 0, 0, 0]

Others:

1. Session09 Not the first time we saw this form, regression
   1. SST = SSR + SSE
   2. SSE is Sum of squared error
   3. what is SSR? Sum of squared regression
   4. SST is the total sum of squared
   5. R^2 is the ratio of SSR over SST. How much the variation is explainable by predictor variable.

# 3.0 Gini and Entropy

# Build Decision Tree using Gini Index Solved Numerical Example Machine Learning by Dr. Mahesh Huddar

[Build Decision Tree using Gini Index Solved Numerical Example Machine Learning by Dr. Mahesh Huddar (youtube.com)](https://www.youtube.com/watch?v=zNYdkpAcP-g)

What is the probability of a random instance observed to be incorrectly classified. The lower probability suggests that the probability of a random instance to be incorrectly classified is low and as such, the node is pure. The node which has zero entropy suggests it is a terminal node or leaf node which do not need any splits.

In short,

Generally, entropy is associated to something **negative -> max impurity of 1, min impurity of 0.**

**Or max impurity of 1 but max purity of 0.**

LOW probability of a random instance to be incorrectly classified -> Entropy is LOW-> Node is PURE

High probability of a random instance to be incorrect classified -> Entropy is HIGH -> Node is IMPURE

Binary Entropy Formula:



Given A1 has [29+, 25-], the split of A1

Outcome = True -> [21+, 5-]

Outcome = False -> [8+, 30-]

[Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning | by Shailey Dash | Towards Data Science](https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c)

* Need to quantify the entropy at the node level, root node:

Use entropy to quantify the impurity first given by Binary Entropy Formula

- Total instances = 29 + 25 = 54 -> therefore

=> pluck

Need to quantify at the leaf level, for each leaf/no child node:

Due to A1 split

0.74

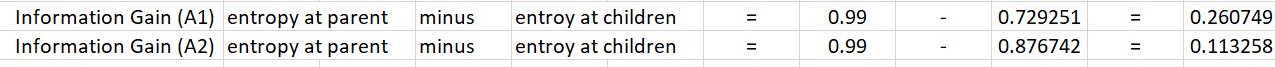
Due to A2 split

=

* To compute the Information Gain which is defined as a measure of how well the split of the training data according to their classifications. Information gain also known as the change in impurity or change in entropy.

A screenshot of a graph

Description automatically generated



Comparing I(A1) and I(A2) -> Choose attribute1 A1 with the most information gain. The change in entropy is the highest from A1 split from 0.99 => 0.73 vs A2 split from 0.99 to only => 0.88. Impurity difference is higher at A1 which is a good thing cause of being more ‘pure’.

# 4.0 Ensemble

5.0

Q1 Data Normalization (4 marks)

2 marks for each method

1. Method 1(1): Scale the data to be between 0 and 1 (both inclusive) [0,1] using min-max scaling as part of the preprocessing to mitigate numerical stability (exploding gradients) and speeding up the training process.
   1. How does it mitigate .. and … speed up?
   2. How to implement this function instead of just 255? Needed for question 2 to 3 only
2. Method 1(2): Normalize to follow a normal distribution by deducting the mean of cifar10 divided by the standard deviation of Xnew.
   1. What is the purpose of this normalization? What is the difference between both method1 and method2?
   2. How to implement this function for just this question– note: no need to do for q2 to q4.

\*\* can use chatgpt

1. Build a MLP to achieve >50% acc.
   1. Load and normalize the CIFAR10 dataset
      1. For mlp only
   2. Flatten the image
   3. Design and Build the model
      1. Hyperparameter tuning
         1. Learning rate (standard)
         2. Batch size (standard)
         3. Number of epochs (tune this)
         4. Size of layers (neurons per layer) (as per question)
      2. Choose appropriate activation function for hidden layer
         1. ReLU. And why. Sigmoid will have vanishing gradient problem especially for deep neural network. Gradients of the sigmoid range between 0 to 1. But if u do carefully it is at most 0.25(?). Updating of the weights during backpropagation involves chain rule in multiplying all the layers gradients and multiplying very small numbers (mathematically <1) would result in a very small update to the weights! Earlier layers dL/dw1 is a series of multiplications -> result in very small value update VS later layers closer to output -> is less likely to experience because chain rule is also smaller.
         2. [Vanishing and exploding gradients | Deep Learning Tutorial 35 (Tensorflow, Keras & Python) (youtube.com)](https://www.youtube.com/watch?v=qowp6SQ9_Oo)
      3. Choose appropriate activation function for output layer
      4. Compile model with appropriate loss function
      5. Compile model with appropriate optimizer
   4. Train the model on CIFAR 10 training set
   5. Validate its performance on the test set
   6. Report accuracy of the model on test set and discuss the model’s performance and any potential improvements.
2. Build a CNN to achieve >70%
   1. Load and normalize the CIFAR10 dataset
   2. Reshape the image (cannot use flatten)
   3. Design and build the model
      1. Why choose
         1. Kernel sizes
            1. From lecture notes: increasing size of filters (start from 3x3, and more filters?)
         2. Pooling sizes
         3. Architecture’s Depth
         4. Dropout, batch normalization (for overfitting and ensuring stability training)
      2. Compile model with appropriate loss function
      3. Compile model with appropriate optimizer
   4. Report the accuracy of the model on test set and Compare the performance of CNN to MLMLP model and discuss why CNN performs differently