Abstract

The defects of a product or materials in the manufacturing industries are one of the problems that cause headaches and costs for manufacturers as well as companies around the world so far. It is costly in recalling defective products, costly in refurbishing or remanufacturing them, and also in human resources in monitoring and evaluating products before they are put on the market. Because of that, the application of research in artificial intelligence as well as computer vision for defect detection in the manufacturing industries was started a few years ago, with the aim of reducing production. defective products as well as minimizing costs such as recalling products, remaking them, or consuming human resources for product quality assurance. More than that, it is aimed at helping the product have a better and perfect level when it reaches the user. Therefore, in this thesis, I have researched and applied the research results of the forerunners in the field of Deep learning as well as computer vision, although it is not perfect, it also achieved a possible result. certain importance in the research and application of artificial intelligence, it plays a small part in the field of IoTs for the manufacturing industry. Specifically, I conducted research on the topic of defect detection on steel surfaces, it was taken from Kaggle's contest titled "Severstal: Steel Defect Detection", which is quite a big contest. was held 2 years ago at the time of writing this thesis. In the thesis, I went to solve the requirements of identifying and classifying defects on the steel surface through the image data set provided by the Russian corporation Severstal. More specifically, I have implemented mainly semantic segmentation and accompanied by some other techniques in deep learning to solve this problem.

Table of Content:

I. Introduction:

// TODO:

II. Objective:

// TODO:

III. Computer Vision:

// TODO:

IV. Background Knowledge

1. Artificial Neural Networks (ANN)

The concept of Artificial Neural Networks was built on the inspire of human brain biology characteristics, Cybenko [1] proved that in a neural network a single hidden layer containing a finite number of neurons is capable of approximating any continuous function to any desired precision. it included a group of machine learning models that are able to "learn" to execute or perform some specific tasks based on supplied examples. In general, these networks were created by a group of units or nodes connected each other like a simulating version of neurons in human brain, which called artificial neurons.

1.1. Neuron

Analog electrical signals flow inside biological neurons, while digital values present as the signals for artificial neurons. Each artificial neuron has an output being computed by using a function of the sum of the input signal values with generally added bias, the connection between the neurons as known as edges, in each connection contains an adjustable model parameter named weight that can be "learned" through a process called "model training". The strength of the synaptic will determines the interaction's level between the neurons, this function combines the neurons, bias and weights in the connections called "activation function",

and there are several activation functions that mean there is not only one fixed activation function for ANN, and we can choose the activation function for the network's output. These functions will be mentioned later in this thesis.

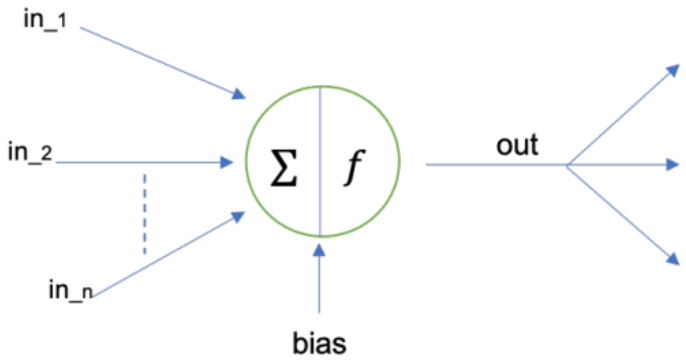


Figure 1: Activation Function.

1.2. Layer

In typical, all neurons are linked directly in network, on the other hand, they are grouped into subsets of neurons known as layers, which are generally connected in a sequential way based on the activation function in every node to compute nodes' outputs from node's inputs. In detail, the neurons in a layer are connected to a set of neurons in previous layer and to another one in next layer. Because of the modeling of ANNs as a connection of a neurons' structure in layer, neurons' outputs at a certain layer become neurons' inputs at the following layer, which networks are understood as feedforward neural networks. There are no loops in present layer, therefore the inputs are always sent forward. As a result, the neurons into a layer cannot be connected with each other. When all the neurons in a layer are connected to all other ones in previous layer and following layer, it is called "Fully Connected Layers", this state of the connection is kept in whole network as a series of fully connected layers that will be called "Fully Connected Neural Network".

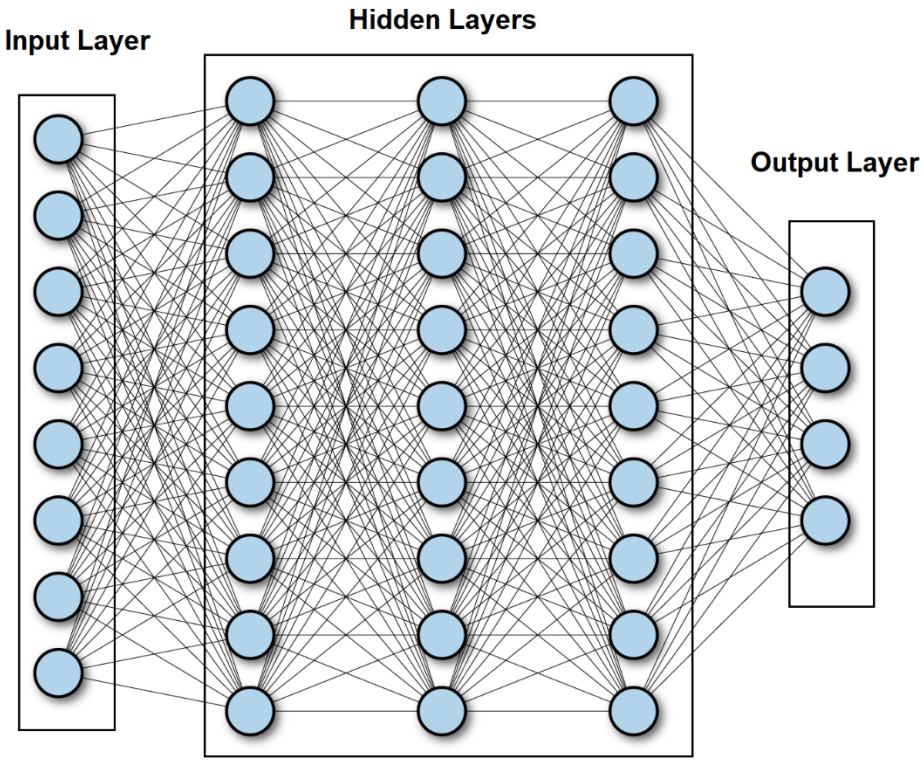


Figure 2: Fully Connected Neural Network.

As figure 2 shows above, there is an existing of 3 kinds of layer inside a network, they are input layer, hidden layers and output layer.

Input layer represent for the input of the network, the number of nodes as known as neurons are size of the input that call “input dimension”. The hidden layers lay between input lay and output layer, there are so many hidden layers in a network, however there is only one input layer and output layer, and the output layer represent for the output of the network. In addition, the bias contains in input and output layer, not in output layer. Although simple decisions are made at the first layer based on the input, sophisticated judgments are already made in the second layer depending on decisions made in the first layer. As one moves deeper into the network, more difficult and abstract judgments emerge. The phrase deep neural networks refer to a network with numerous hidden layers.

1.3. Forward and Backward (back propagation) steps

As the mentioning before, the signals travel from input layer to output layer traversing hidden layers, which operations performed over the values received from the nodes in previous layers (in all layers but in the input layer) and whole activation function results are sent forwards to the next layer till the output layer. However, it is not the end of process. After getting the result of output layer, there is a backwards named back-propagation applying in here as known as training step for ANN. The algorithm is used to effectively train a neural network through a technique called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases), which helps minimize the error for network's result, it is called "loss function" or "cost function" which is the evaluation between predicted output and expected output at the final point of forward step. Basically, this function calculates the gradient of the loss function to each parameter in network respectively that means it work from the top layer (output layer) to the bottom layer (input layer) to calculate the contribution of each parameter in that value [2], which update the weights until reaching the minimum values. Furthermore, the key point of calculating the minimum value for loss function is gradient descent, which is a process of determining the modifications needed to minimize the loss and optimize the model to a specific evaluation measure by calculating the derivative of the loss function [3].

1.4. Gradient Descent:

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks [4]. It is built on a convex function and iteratively changes its parameters to minimize a

given function to its local minimum. On the other words, Gradient descent begins by defining the initialized parameter values, and then iteratively adjusts the values to minimize the given cost-function. Deeply, a gradient is a function's derivative with several input variables. In terms of mathematics, a gradient is known as the function's slope, and it simply measures the change in all weights in relation to the change in error. The steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the model will stop learning. A gradient is a partial derivative with regard to its inputs.

Imaging that a blindfolded man wants to hike down to the bottom of a valley from the top of a hill with as few steps as possible. He might begin going down the hill by taking large steps in the sharpest direction, which he can do as long as he is not close to the bottom. However, as he gets closer to the bottom, his steps will become smaller and smaller in order to prevent overshooting it. The gradient can be used to mathematically characterize this process. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [5].

To reach the local minimum for gradient descent, we should choose a relevant value for the learning rate, which is not both too low and too high. This is very significant because if the steps are too large, it could fail to reach the local minimum since it bounces between the convex function of gradient descent. Besides, gradient descent will eventually reach the local minimum if we set the learning rate to a very low value, but it may take a lot of time [6]. So, the key point is choosing an optimal learning rate that helps model converge to the minimum value without spending much time.

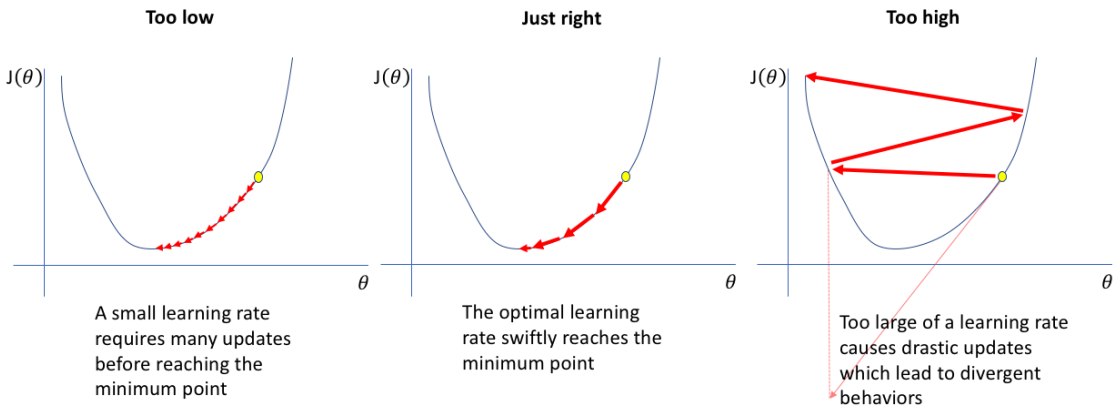


Figure 3: Learning rate choices that decide gradient descent result.

Gradient descent has three forms that differ in the amount of data used to compute the gradient of the objective function. We choose between the accuracy of the parameter update and the time it takes to complete an update based on the amount of data.

1.4.1. Batch Gradient Descent

Batch gradient descent calculates the sum of the error for each point in a training set and updates the model only after all training instances have been evaluated. This procedure is known as a training epoch. Though this batching supplies a computation efficiency, it can still take a long time to execute big training datasets because it must still keep all of the data in memory [5]. Batch gradient descent also frequently yields a stable error gradient and convergence, although that convergence point isn't always the best, locating the local minimum rather than the global minimum.

1.4.2. Stochastic Gradient Descent

Stochastic gradient descent (SGD) performs a training epoch for each example in the dataset and updates the parameters of each training example at a time. They are easier to remember because you only need to

hold one training example [5]. In spite of the fact that these frequent updates provide more detail and speed, they can lead in computational efficiency losses when it be compared to batch gradient descent. Despite having ability of noisy gradients production, these frequent updates can also aid in escape the local minimum and locating the global one.

1.4.3. Mini-batch Gradient Descent

Mini-batch gradient descent combines batch gradient descent with stochastic gradient descent ideas. It divides the training dataset into small batches and updates each of those batches. This method strikes a balance between batch gradient descent's computing efficiency and stochastic gradient descent's speed [5].

Besides, gradient descent has its own set of challenges such as: Local minima and saddle points, vanishing and exploding gradients.

- Local minima resemble global minima in shape with the slope of the cost function increasing on each side of the current location.
- Saddle points happen when the negative gradient exists just on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name was inspired by horse's saddle;
- Vanishing gradients happens when the gradient is too small. The gradient continues to shrink as we move backwards during backpropagation, leading earlier levels of the network to learn more slowly than later layers. When this occurs, the weight parameters are updated until they become inconsequential, i.e., 0, resulting in an algorithm that is no longer learning.
- Exploding gradients occurs when the gradient becomes too big, resulting in an unstable model. In this instance, the model weights will get excessively huge and will be represented as “NaN” finally. One

approach to this problem is to use a dimensionality reduction technique, which can help to reduce model complexity.

2. Convolutional Neural Network

Convolutional Neural Networks (CNNs), like ANNs, being inspired by the structure of the human visual cortex [8]. CNNs, which are a type of neural network typically utilized for computer vision tasks in deep learning, include convolutional layers, pooling layers and fully connected layers. These networks are identical to traditional neural networks, except that instead of generic matrix multiplication, a convolution kernel is used in one or more of their layers [3]. These networks have the role of reduction the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Moreover, CNNs have ability to take image tensors as an input, identify which image characteristics or features are essential for classification or differentiation, and output these classifications [1, 7].

Therefore, the basic structure of a Convolutional Network in order starting from a Convolutional Layer or so-called convolution process (do not count the initial input layer), has the effect of picking out the 2-dimensional characteristics of the image input. Specifically, the features of the input original image are passed through the “convolutional layer”, at which there is a mechanism built up by matrix multiplication between the input image known as a matrix and one or more matrices with a certain size (usually smaller than the input matrix) being called “filters”, this matrix multiplication will be shifted by rows and columns with a certain rule that will give generate one or more new matrices based on the number of filter layers, the product from this process is called feature maps which contain the attributes of the original input image.

Followed by a pooling layer to help the network synthesize the features of the image in a deeply specific way (max pooling) or more objectively general (average pooling or mean pooling) from the output of the previous layer. Next, the size of the image has been significantly reduced that is very meaningful for memory in process of training model. Although the dimensions are decreased, the number of channels or feature maps are still kept unchanged. So, this process is as a compositing to reduce the size of the input image feature in length and width. At last, when the input size has been reduced to a reasonable value, the 2-dimensional image will be converted to a one-dimensional (one-dimensional vector) or known as "flatten", at which this process is implemented as a Fully Connected Layer, the number of units in this layer will correspond to the number of classes that defined for the network to distinguish as known as the final output of the network with a specific activation function for specific final result.

In general, using CNNs because of their capacity to learn translation-invariant patterns as well as spatial hierarchies utilizing appropriate filters. Learning translation invariant patterns means that once a pattern is learned in one area of a given image, the network will be able to recognize it anywhere else in the image. This improves image processing efficiency by requiring fewer training samples to learn generalizable representations [1].

Spatial hierarchies show how earlier layers in the network can learn tiny local patterns and grow into bigger patterns composed of these tiny patterns in the next layers, which also improves the network's ability to learn complicated and abstract concepts of visualization. In addition, this also allows these networks to deliver class predictions with far higher accuracy than vectorizing a complex image with pixel dependencies throughout. Therefore, CNNs have to be used for any image classification task so as to track these crucial properties and correlations between features [1].

2.1. Convolution Layer

Convolution is an operation that produces a new function by combining two functions of a real-valued argument. Let s(t) is the output estimate function based on time t, x(a) is the age-based input position function, and w(a) is the weighting function that prioritizes recent measurements. This is the general formula for convolution utilizing this defined function:

$$s(t) = \int x(a)w(t-a)da.$$

Figure 4: Convolution function.

In detail, the input is the function x(a), the kernel is the weighting function w(a), and the output s(t) is the feature map in the CNN. the CNNs use convolutions that are performed over two-dimensional tensors that are made up of the input images' height, width, and channels of colour. These techniques take out patches and alter the input to build a feature map with varied depth. One of the two critical criteria for defining these convolutions is the size of these extracted patches. The depth of the output feature map is the second parameter to consider. This is the number of filters built by the layer that can encode diverse features of the input data.

2D Convolutions are computed by moving a square window of defined height and width across all pixels of the input feature map, which connect to each local region or receptive field of the image corresponding to the size of the applied filters. For images, the input feature map has three dimensions: height, width and colour bands, which commonly correspond to red, green, and blue. When these feature maps perform 2D convolution, two dimensional patches of surrounding features are created.

These patches are then turned into a one-dimensional vector reflecting the output depth via a convolution kernel. After that, the vectors are reassembled spatially into a two-dimensional output map that corresponds to all points in the input map. The convolution process is depicted in the figure 4 below.

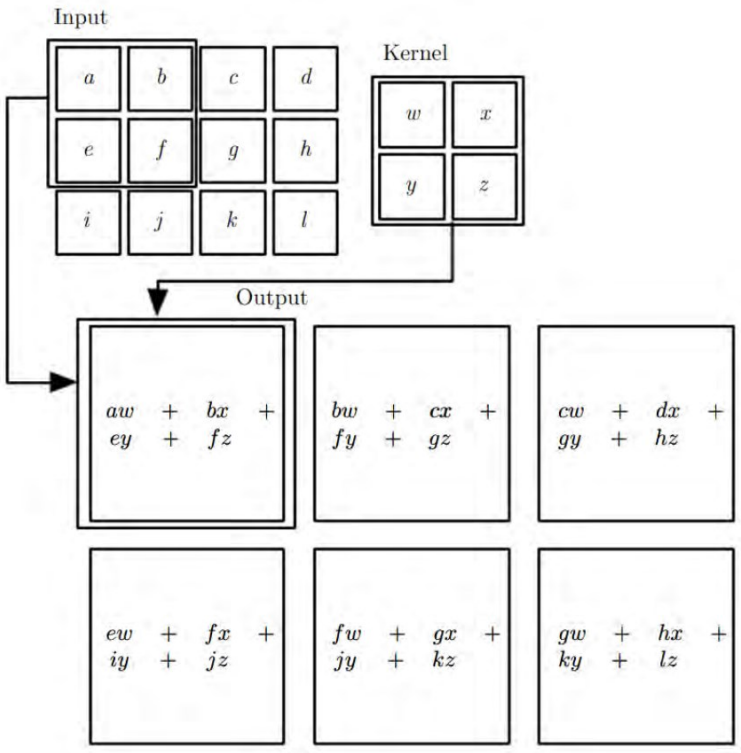


Figure 5: Feature maps are computed by stride mechanism of convolution process.

In other words, a feature map is the output of a convolutional layer, which is typically a three-dimensional tensor with dimensions width, height, and depth. To process inputs or feature maps, the convolutional layer employs a filter, similar to the filtering procedure in picture pre-processing. The primary distinction is that in standard image pre-processing, the filter weights (values) are hand-crafted, whereas in CNNs, during training, data helps they learn. Alternatively, the receptive field in the input feature map corresponds to the output of a convolutional layer, and the size of the receptive field is governed by the kernel size and

dilation. It is not essential to flatten images while using CNNs. However, flattening is required to feed the features maps, being the outputs of convolutional layers, to the fully connected layers. Fully connected layers are the same as the ANNs mentioned above, however ANN is a catch-all term for all artificial neural networks, including CNNs. Fully connected layers understand the linear and non-linear correlations between extracting features and classifying each sample into many abstract classes. This figure shows a simple CNN architecture.

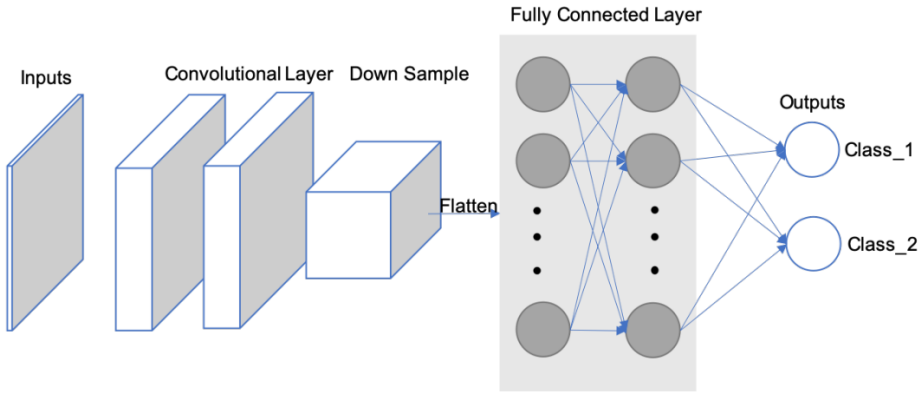


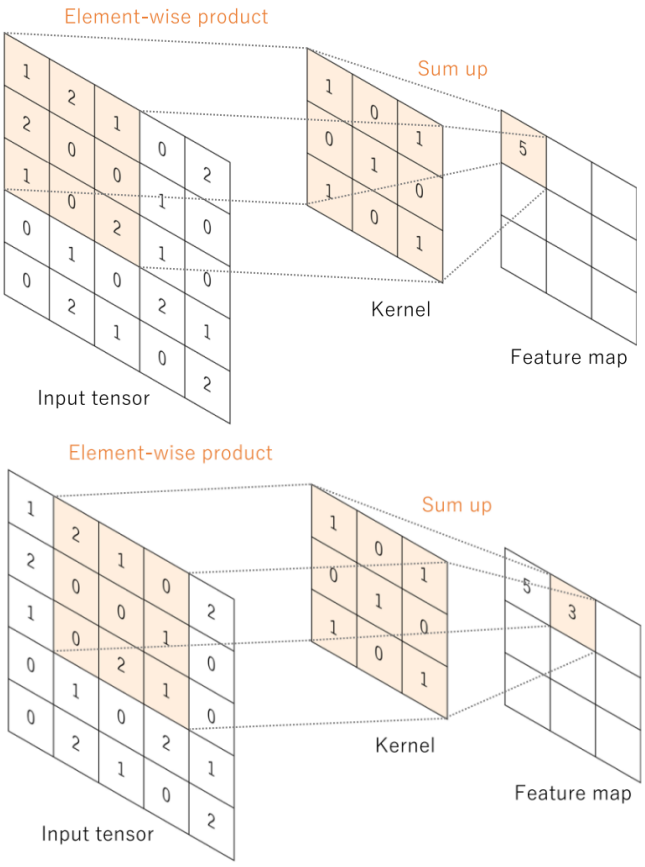
Figure 6: CNN Architecture.

The size of the output volume is controlled by hyperparameters which are depth, stride, and zero-padding.

The first component of the output volume decision is depth, which is the number of filters desiring to utilize, each learning to search for something different in the input. For example, if the first Convolutional Layer receives the raw image as an input, distinct neurons along the depth dimension may fire in the presence of different oriented edges or colour blobs. A column of depth is a group of neurons being all staring at the same region of the input [9].

Stride aids in picture and video data compression tuning. The stride of the convolution is a characteristic which can contribute to a different output size in CNNs, which is a convolution operation parameter that

defines the distance between patches derived from the input feature map. With a stride of 2, the output feature map's width and height are down-sampled by a factor of two with no padding. For example, if the stride of a neural network is set to 1, step by step, the filter will make a reflection onto the matrix of input, which move one pixel or one unit at a time from left to right in the row till the end of horizontal point and top to bottom in the column at the end of the matrix. Because the size of the filter influences the encoded output volume, stride is frequently a positive integer rather than a fraction or decimal.



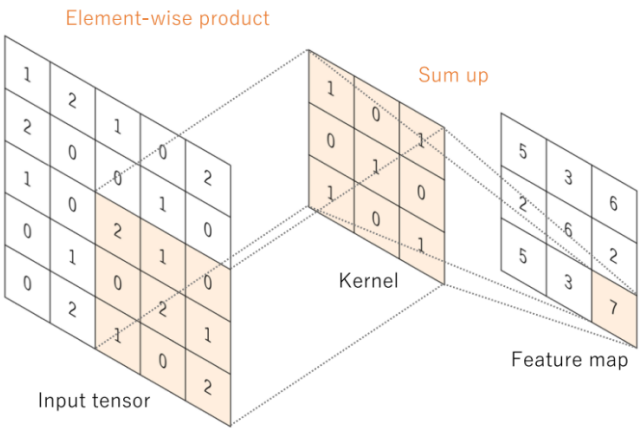


Figure 7: Generating feature map process.

It is sometimes useful to pad the input volume with zeros around the border. This zero-size padding's is a hyperparameter. The great thing about zero padding is that it allows to tweak the spatial size of the output volumes, which most typically using it to precisely preserve the spatial size of the input volume so as to be same of the input and output width and height [9].

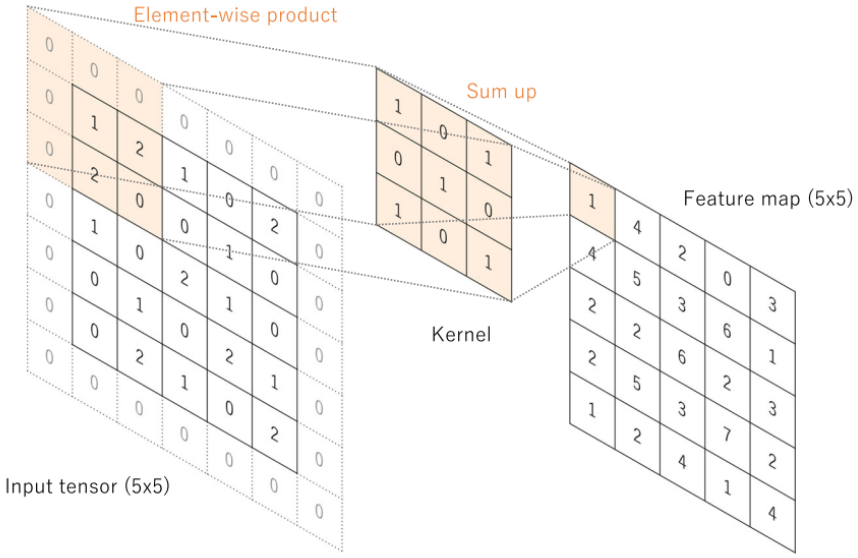


Figure 8: Generating feature map process with zero-padding.

Combination of these components will give out the formula for calculating the size of Convolutional layer:

$$[(N + 2P - F) / S + 1] \times [(N + 2P - F) / S + 1] \times C'$$

Figure 9: The formula of output size of Convolutional layer.

The spatial dimension of the output volume may be calculated as a function of the input volume size ***(N)*** which represent for the dimension by width and height of input image, the receptive field size of the Conv Layer neurons ***(F)*** as the dimension of filter by width and height, the amount of zero padding placed the border for input image ***(P)***, the positive integer number of strides with which they are applied ***(S)***, and the number of filter ***(C')***.

The results of a linear operation, such as convolution, are subsequently processed by a nonlinear activation function. Although smooth nonlinear functions such as the sigmoid or hyperbolic tangent (tanh) function were previously used because they are mathematical representations of biological neuron behaviour, the rectified linear unit (ReLU) function is now the most commonly used nonlinear activation function, which simply computes the function: $f(x) = \max(0, x)$

In convolutional neural networks, rectified linear units (ReLUs) provide three key advantages:

- The ReLUs efficiently transmit the gradient, reducing the risk of a vanishing gradient problem, which is prevalent in deep neural architectures.
- The ReLUs set negative values to zero, so solving the problem of cancellation and resulting in a much sparser activation volume at its output. Sparsity is helpful for a variety of reasons;

However, it is most commonly used to supply robustness to tiny adjustments in input such as noise.

- The ReLUs that including just simple computations (primarily comparisons) and are thus substantially more efficient to implement in convolutional neural networks.

2.2. Pooling Layer

There are also pooling layers between convolutional layers. They are used to down sample the feature maps and reduce the number of parameters. The later layers after feature extraction will need a large number of parameters because the depth is determined by the number of channels in the later layers often increasing exponentially. That increases the number of parameters and the amount of computation in the neural network. Therefore, to reduce the computational load we will need to reduce the dimensions of the input matrix block or reduce the number of layer units. Since each unit would be a representative result of applying a filter to find a specific feature, reducing the number of units would not be feasible. Reducing the input matrix block size by finding a representative value for each spatial region that the filter passes through will not change the main contours of the image but reduce the size of the image. Hence the matrix reduction process is applied.

Average pooling and max pooling are the two most commonly used types of pooling layers helping make this reduction purpose. As the suggestion of their names, average pooling takes out the average (mean) of the values on the feature map corresponding to a kernel size. Besides, max pooling calculates the maximum value in the neighbourhood corresponding to the kernel size position on the feature map. Both of them travel through the whole feature map by a stride size without

overlapping local region position. In addition, the pooling layers do not have any parameters which need to be learned during training and the stride in the convolutional layer is able to also be used to replace the pooling layers. Figure below shows the difference between these two pooling types.

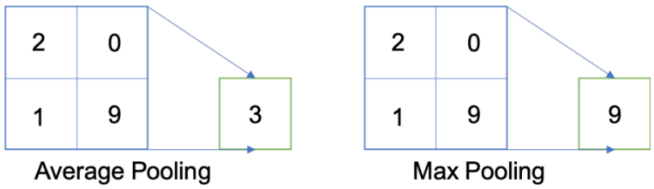


Figure 10: Average (Mean) and Max Pooling Layer.

2.3. Batch Normalization:

Another critical component of a CNN architecture is batch normalization (BN). Although this is not a required component in CNN, it is a relatively important component with many benefits for training the model to reach the best results. BN allows faster training and stabilization of deep neural networks by stabilizing the distribution of layer inputs during training. This approach mainly involves Internal Covariate Shift (ICS). To improve model training, it is important to reduce ICS by controlling the means and variances of the input data layers. In detail, BN normalizes the feature maps at the output of a convolutional layer in the same way which it normalizes the input pictures. BN makes subtractions of the batch mean and does divisions of the feature map values by the batch standard deviation to ensure that the distribution of feature maps is the same. As a result, BN lowers value shift in hidden layers and BN allows you additional flexibility in the initialization of kernel weights. Furthermore, because of guarantee of BN which activation does not diverge to very large values, greater learning rates can be employed. In addition, there is a similar to a dropout layer which also has some

regularization effects, because of adding noise to the feature maps. During training and evaluation, both the BN and dropout layers will behave differently.

When we would like the function of probability density (FPD) of the model to be closest the true real distribution of data as possible in deep learning. Unfortunately, in fact that we only have a narrow amount of training data at our disposal. As a result, the goal is to get the FPD of the model as near to the FPD of the train as possible by training on a dataset that replicates the domain of application in the real-world. A sufficiently big training dataset is required to get improved representation. When the training dataset is insufficiently large, there will be a significant difference in the error between the FPD of real and the FPD of train.

In other expressions, even if we produce a model FPD that is very close to the FPD of the train, it will not perform with the FPD of test skilfully, being a sample of the FPD of the real. This is referred to as the overfitting problem. Overfitting occurs when a model fails to fit subsequent observations. Dropout layer is applied to stay away from overfitting by randomly selecting some neurons to brush aside throughout the training process. A dropped-out node's incoming and outgoing edges are also eliminated. Thus, dropout not only reduces the problem of overfitting, but it also simultaneously speeds up training by eliminating training with all nodes.

2.4. Full Connected Layer

Typically, the activation function applied to the last connected layer differs from the others. Each job necessitates the selection of an appropriate activation function. A soft-max function is used as an activation function in the multiclass classification job to normalize output real values from the final fully connected layer to target class

probabilities, where each value ranges between 0 and 1 and all values will be made a sum to 1. Typical final layer activation function selections for various sorts of jobs.

In addition, the majority of the trainable parameters are located in fully connected layers in CNN. Fully connected layers can give out outputs that are categories for image classification or localization via bounding boxes [11]. Convolutional layer work can also be used as fully connected layers to execute the same duty more efficiently. Instead of a class, the result of an image segmentation job is a mask with the same resolution as the input pictures. We may use a 1x1 kernel as the output layer to reduce the depth to the same level as the target classes [10], giving the model pixel-wise classification capability.

Training a CNN consists of two steps: forward and backward. We feed the CNN with input information flow in the forward. To acquire the input for the following layer, feature maps are generated in each layer with predefined weights and biases. A loss value will be computed at the output layer by applying a loss function to the difference between the calculated output and the ground truth. The chain rule is utilized in the backward step to find out the gradients of loss value for each trainable parameter. After that, the gradients are utilized to adjust each parameter for the following iteration. This type of update is what allows CNNs to “learn”.

After processing enough iterations, at which the loss value converges to an acceptable small number that means it is approximately close to zero, the training process can be terminated. The loss function computes the difference or error between the outputs and the ground truth. Minimizing loss implies minimizing the difference between pdf(model) and pdf(train) when the loss function is applied as a guide. Moreover, the loss function will change depending on the application and objective. Because of its

ability of quantify the similarity/difference between two distributions, cross-entropy is a widely used loss function.

Otherwise, there are several optimizer options to help you decide how to update the network parameters. Adam and Stochastic Gradient Descent (SGD) are two options. In general, SGD will produce better results while learning at a slow pace. It will take several iterations to achieve a satisfactory outcome. SGD has a fixed learning rate that does not vary during training. Adam, on the other hand, converges quicker since it employs a vector of learning rates that are adjusted as learning continues during training progresses. They will be mentioned more detail in the next sections of Optimizer.

3. Activation Functions

Activation functions (AFs) in Neural Network are a component which is very essential in Deep Learning. AFs help determine the output of a Deep Learning model, the accuracy and computational efficiency of the training model – which will make a decision on the success or failure of a Neural Network system. AFs also have a great influence on the convergence and convergence speed of neural networks, or in some cases, AFs can prevent neural networks from converging in the first place.

AFs are presented in mathematical equations and this function is attached to each neuron in the network and determines whether it should be activated, based on whether the input data fits the model prediction, which also help normalize the output of each neuron to a range of 1 to 0 or between -1 and 1, depending on the type of activation function being used in the layer or network. Additionally, another aspect of AFs is that they have to have efficiency in computation because of being computed over thousands or even millions of neurons for each data sample. And

modern neural networks use backpropagation technique to train the model.

The main reason that neural network models stand out from machine learning models is their ability to solve non-linear data problems through AFs of hidden layers, at which execute the task of solving problems for complex nonlinear relationships between data features and model outputs.

As mentioned earlier, in a neural network, the input data, is fed to the neurons in the input layer. Each neuron has a weight and multiplying the input by the weight gives the output of the neuron, which is then passed on to the next layer. So, AFs are applied in neural networks to be on duty of computation for the weight input's sum and biases, being utilized to make a decision on whether or not a neuron can be made activated. It manipulates the supplied data via several gradient processing, or gradient descent in typical and afterwards give out a production for a neural network's output, including the parameters. These AFs are usually known as a reference for a transfer function in some documentation or research papers. Furthermore, depending on the function they deputize, AFs can be either linear or non-linear, and they are used to modulate the outputs of neural networks in a variety of areas ranging such as: object recognition, classification, segmentation, speech recognition, cancer detection systems, finger print detection, and so on… [12]

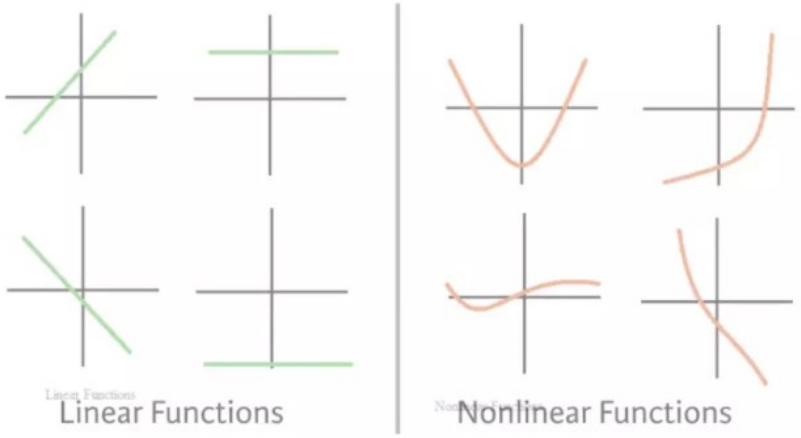


Figure 11: Linear and Non-linear functions.

Although in fact that there are many types of AFs, in this article I will only mention some of the AFs that are commonly used in current CNNs architectures such as: sigmoid, softmax, hyperbolic tangent (Tanh), rectified linear unit (ReLU) and Leaky rectified linear unit (LReLU).

3.1. Sigmoid function

The Sigmoid Function is also known as the Sigmoid curve. This is a mathematical function characterized by an S-shaped curve, which is one of the non-linear activation functions most widely used. If being familiar with some machine learning models, probably still remembering Logistic Regression - a one of the simple algorithms for binary classification problems, which is quite effective. The "soul" of Regression is this Sigmoid function. The sigmoid is a continuously nonlinear function, allows to pass real numbers as its input and gives a production of results in the range 0 to 1, is considered probabilistic in some problems. Suppose you trained a *Neural Network* to images classification of birds and dogs, what a classic problem, where bird is 1 and dog is 0. Basically, when your Model returns a value greater than 0.5

meaning the image is of a bird, or the value less than 0.5 means the image is of a dog.

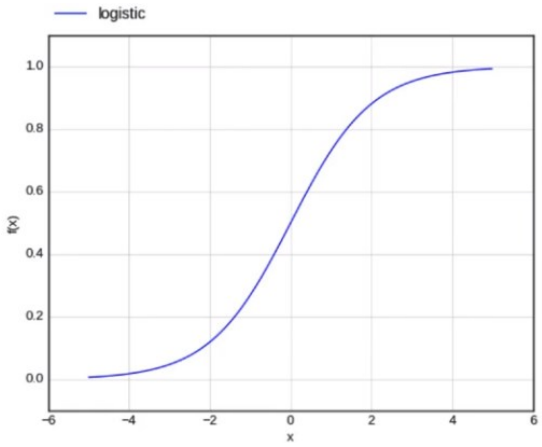


Figure 12: Sigmoid function.

In the Sigmoid function, a small adjustment from the input results in a no relative change for output. It gives a smoother and more continuous output than input, so, it has derivative in everywhere [13]. The Figure below shows the formular of sigmoid function:

$$f(x) = \left(\frac{1}{(1 + \ exp^{-x})} \right)$$

Figure 13: Sigmoid function equation.

In the formula, x represents for the network's output, and f(x) is its function, and exp stands for the Number of Euler. As a result, if the value of x approaches to the infinity of positive, the expected value of f(x) becomes 1. In contrast, if x approaches to the infinity of negative, the predicted value of f(x) becomes 0. And if the result of the sigmoid function is greater than 0.5, that label is classified as class 1 or positive side, and if it is less than 0.5, classified as class 0 or negative side.

On the other hand, the sigmoid has significant shortcomings such as sharp damp gradients while doing backpropagation starting from

deeper hidden layers back to input layers, gradient saturation, sluggish convergence, and output of having non-zero centred, which causes gradient updates to propagate in various passageways. Other types of AF, such as the hyperbolic tangent function (Tanh), have been recommended to address some of the shortcomings of the Sigmoid AF.

3.2. Hyperbolic tangent function (Tanh)

Another form of AF utilized in DL is the hyperbolic tangent function, which has several versions employed in DL applications [14]. The hyperbolic tangent function, often known as the tanh function, is a smoother zero-centred function with a range of -1 to 1, and its output is given by:

$$f(x) = \left(\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \right)$$

Figure 14: Tanh equation.

Because of supplying higher training performance for multilayer neural networks than the sigmoid function, the tanh function has become the favoured function over the sigmoid function [15]. Otherwise, the tanh function, like the sigmoid functions, was unable to overcome the vanishing gradient problem. The function's key advantage is that it generates zero-centred output, which aids in the operation of back-propagation.

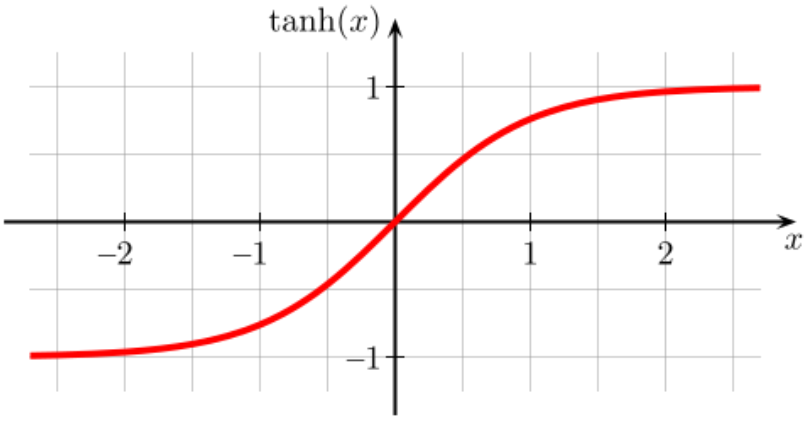


Figure 15: Tanh function with values moving in range [-1: 1].

The tanh function has the peculiarity of only being able to achieve a gradient of 1 only when the input's value is equal to 0, that equivalent to x equal zero. As a result, the tanh function generates some inactivated neurons during computation. The unworked neuron is a condition in which the activation weight rarely employed as a result of a zero gradient. In addition, this obstruction of the tanh function prompted additional study in activation functions to remedy the problem, giving rise to the ReLU activation function. Furthermore, the tanh function has mostly been employed in recurrent neural networks for natural language processing and speech recognition [16].

3.3. Softmax function

Softmax Function (SF) is an exponential averaging function that calculates the probability of an event occurring. In general, the SF calculates the probability of a class occurring out of the total of all possible classes. This probability will then be used to determine the target class for the inputs. Specifically, the SF turns a k-dimensional vector of any real values into a real-valued k-dimensional vector that sums to 1. The input value can be positive, negative, zero, or greater than 1, but the

SF will always turn them into a value in the range (0:1]. The SF is calculated following this formula:

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)}$$

Figure 16: Softmax function equation.

Where (x_i are vector of input values for the function from x_0 to x_n, in which all the values can be any real number, positive number, negative number or zero. Next, $exp^{(xi)}$ is the standard exponentiation function is applied to each input value returning a positive value greater than 0. This value will be very small if the input is negative, and very large if the input is positive and it is an uncertain number in the range (0:1] as the requirement of a probability.

As such, they can be called “probabilities”. If one of the input values is negative or very small, the SF turns them into a small probability. If an input is large, it will be converted to a large probability. But the probability is always greater than 0 and less than 1, or equal to 1. So, the sum of all probabilities is always 1 and the SF brings a lot of benefits such as:

- Optimized when calculating the maximum probability in the model parameter.
- The property of the SF makes it satisfactory for interpretation of probability, which is very beneficial in ML.
- Normalization of Softmax is a method to minimize the effect of extreme values or outliers in data with no having to modify the original data.

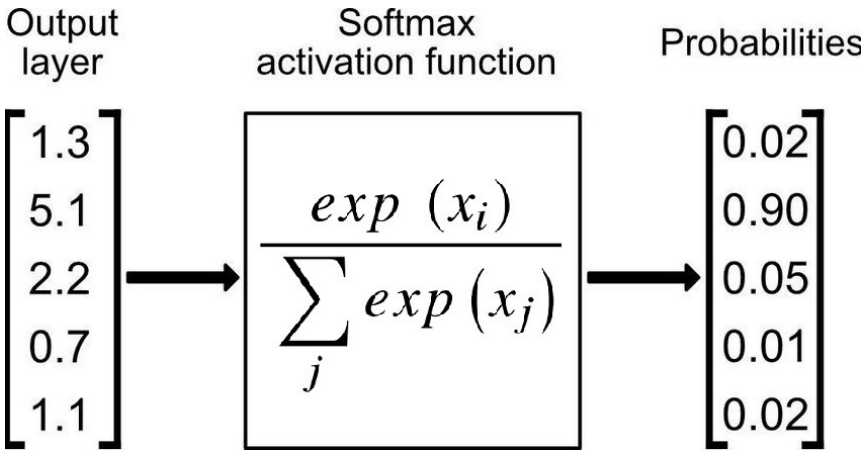


Figure 17: Softmax function applied in output layer of a network for classification task.

For the reasons mentioned above, the SF returns the value of probabilities for each class in multi-class models, with the target class having the highest likelihood. The SF may be found in practically all of the deep learning's output layers architectures where it is used. Adding, the Sigmoid and Softmax vary in that the Sigmoid is used for binary classification and the Softmax is utilized for the tasks of multivariate classification.

3.4. Rectified Linear Unit (ReLU) Function

In 2010, Nair and Hinton introduced the rectified linear unit (ReLU) activation function, which has been the most extensively utilized activation function for applications in the DL field, bringing a state-of-the-art result till currently [17]. In general, The ReLU AF has an expressive speed of learning AF proven to be the most extensively and successfully applied function [14]. In DL, it outperforms and generalizes the Sigmoid and Tanh activation functions [18]. In particular, there is a representation of a roughly linear function in ReLU function and hence retains the features of linear models that make them easier to optimize using the approaches of gradient decent. The mechanism of the ReLU AF, which applies a functioning of threshold to each element of input

having values smaller than zero to be replaced to zero, and the formular of the ReLU AF as follows:

$$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$$

Figure 18: Rectified Linear Unit function equation.

The ReLU will correct the inputs' values which are smaller than zero, driving them to zero and removing the issue about vanishing gradient that was noticed in the previous types of AFs. Mainly, the ReLU function is put to utilize into the hidden units of deep neural networks. Besides, another AF as sigmoid or softmax is offen applied in the network's output layers with popular purpose of classification in image [12] and speech recognition.

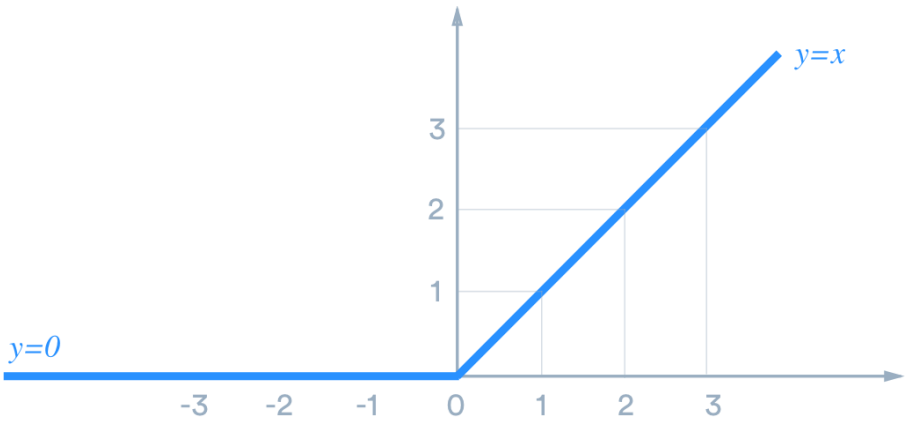


Figure 19: Rectified Linear Unit function.

There are many advantages when using ReLU. In detail, the fundamental advantage of ReLU in computation is that they bring a more expeditious computation because they do not need to quantify exponentials and divisions, resulting in increased overall computation performance [18]. creation of sparsity in the hidden units is another feature of the ReLU, which squishes the values from zero to maximum.

The ReLU has a problem in that it readily overfits when compared to the sigmoid function, despite the fact that the approach of dropout has been used to lessen the overfitting's influence of ReLUs and the rectified networks enhanced deep neural network performances [19].

There is a notable weakness in the ReLU, which it is occasionally brittle during training, leading to the death of the several gradients. This causes will make neurons to be dead and direct weight updates to inactivate data points in the future, hampered learning because of dead neurons result in zero activation [3]. So that, the leaky ReLU was offered as a solution to the dead neuron problem.

3.5. Leaky ReLU

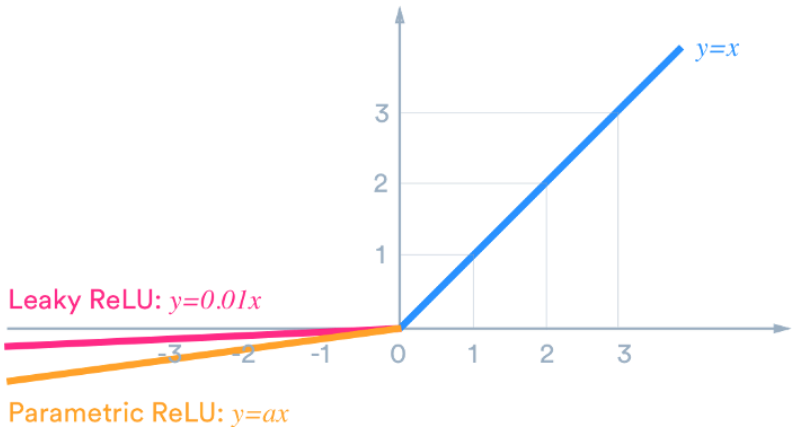


Figure 20: Leaky Rectified Linear Unit function.

The leaky ReLU was released in 2013 that give out several small negative slope to the ReLU to maintain and stay survived the weight updates throughout the phase of propagation. The building of the alpha parameter serves for the solution to issue of dead neuron of the ReLUs, which ensure that the gradients are not fixed zero anymore during training. The computation of the LReLU bases on the gradient with a tiny constant value for the negative gradient α in the range of 0.01; hence, the LReLU AF is computed following this formula:

$$f(x) = \alpha x + x = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases}$$

Figure 21: Leaky Rectified Linear Unit function equation.

The comparasion of the ReLU to an exception of no having zero gradients over the entire of time, which report a similar result. Therefore, excepting of distribution and sparsity in the comparasion of ReLU to Tanh, no having any noteworthy enhancement of result. As a result, Leaky ReLU offers two advantages:

- Because it lacks zero-slope portions, it solves the "dying ReLU" issue, so training process is expedited. And, the "mean activation" near to zero is evidence for speeding up training. (It aids in keeping the diagonal entries of the Fisher information matrix short, but it can disregard judiciously.) Unlike ReLU, leaky ReLU is more "balanced," and as a result, it may learn faster.
- Keep in mind that the outcome is not always consistent. Leaky ReLU isn't necessarily better than regular ReLU, and it should only be used as a last resort.

4. Residual Network (Resnet)

ResNet, an abbreviation for Residual Network, is a form of NN introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their paper "Deep Residual Learning for Image Recognition". The ResNet models were immensely successful with winning first place in the competition of the ILSVRC 2015 for classification with a top-5 error proportion of 3.57 percentage (An ensemble model), as well as first place in the competitions of ILSVRC and COCO 2015 in ImageNet both detection and localization, Coco detection, and Coco segmentation [20]. Furthermore, ResNet-101 was used to replace VGG-16 layers in Faster

R-CNN. They discovered a relative improvement with a proportion of 28 percentage. Resnet was developed because of the vanishing or exploding gradient issue on some state of the arts model architectures at that time.

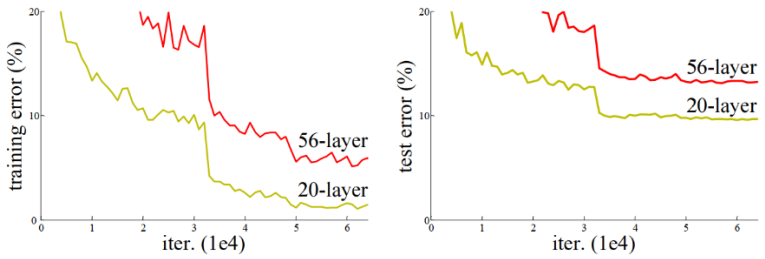


Figure 22: Performance of 20 layers and 50 layers plain networks on CIFAR-10.

Before the Resnet, with the advent of neural networks deeper and deeper by a common argument, which is "go deeper", growing up the depth of the network will be an improvement for the quality of the network. Based on that, several new architectures of deep neural network have been born such as AlexNet, GoogleNet or VGG, which apply the mechanism to increase the number of layers or number of layer's group. Because these added layers are able to solve complicated issues more efficiently as the non-identical layers could be trained for various duties to get high results of accuracy. Additionally, the number of stacked layers can enrich the features of the model. However, with an increasing number of layers of complexity in deeper CNNs, a new trouble has also emerged called vanishing gradients. In specific, as deeper networks may begin to converge, the accuracy of the network gradually saturates to a certain threshold and then slowly moves down during backpropagation step, and higher error increases may occur. This issue goes against the purpose of reducing error and increasing accuracy during network model training. In other words, the values go through the gradient step that are chain-rule by the calculation of partial derivatives method, which will quickly return a

very small result that approaches 0 after several chain-rule applications [22]. It will lead the learning of the network stopped soon, so this backpropagation will not be executed in previous layers, in this case, it may be stopped at an unknown hidden layer. This problem of training very deep networks has been alleviated with these Resnets which are made up from Residual Blocks (RBs).

4.1. Residual Blocks (RBs)

Resnet takes advantages of using RBs to increase model accuracy. The strength of this form of neural network is the concept of "skip connections" or skip paths, which is at the heart of the RBs. There are two ways for the skip connections to work. At first, they address the issue of vanishing gradients by creating an alternate path as a shortcut for the gradient to go through. Second, they also allow the model to learn a function of identity, which assures that the model's upper levels will have a performance not being worse than the layers lays underneath.

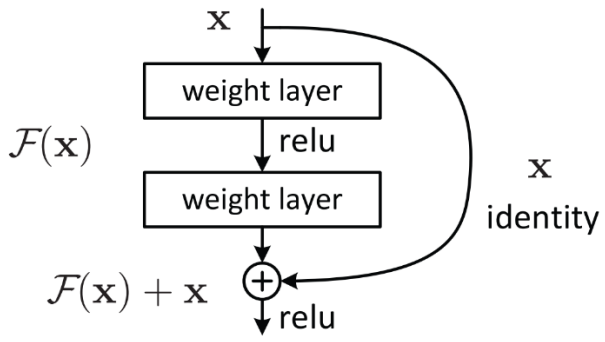


Figure 23: Residual Block Unit with skip connection.

The construction of an RB is depicted in the figure above. The ResNet is developed to create its stack of 2 layers being able to fit to F(x) = H(x) - x rather than training only stacked convolutional kernels have an ability to be suitable to the intended mapping. Supposing that, the original mapping has been recast as H(x) = F(x) + x [20]. They claim that optimizing the residual mapping substantially easier, which is F(x). In

spite of being the optimal option of the identity in an extreme circumstance, model will give out a result in F(x) being zero. Using such a framework, networks with more than 100 layers might be trained. Besides, this method appears to have a minor flaw when there is a different between input's dimensions and those output, which have ability to occur with pooling and convolutional layers. When the dimensions of F(x) differ from those of x, we have two options:

- To up-sample its dimensions, extra zero entries will be padded to the skip connection.

- To make the dimensions equal, the linear projection approach is utilized, which is accomplished by attaching more 1 x 1 convolutional layers to the input. In this example, the output is as follows: H(x) = F(x) + W(x). In this case, adding an extra parameter W.

In short, the RBs help the layers acquire a knowledge of identity functions substantially easier than. So that, ResNet raises up the performance of DNNs with additional neural layers while reducing the proportion of error. In other words, the skip paths combine the outputs of prior layers with the outputs of stacked layers, allowing for considerably deeper networks to be trained than previously feasible.

4.2. Resnet Architecture

4.2.1. Resnet – 34 Architecture

The Resnet-34 was the first architecture built, which included inserting shortcut connections into a plain network to transform it into counterpart of its residual network. The plain network in this case was influenced by VGG neural networks (VGG-16, VGG-19), while the convolutional networks had 3 x 3 filters. On the other hand, ResNets

have fewer filters and are less sophisticated than VGGNets. There is a comparison between the 34-layer ResNet attains 3.6 billion FLOPs of performance and larger VGG-19 with 19.6 billion FLOPs [20]. Resnet 34 takes a proportion of 18 percentage of VGG-19, thus, the VGG-19 has more filters and higher complexity than Resnet 34.

The Resnet 34 also adhered to two simple design principles that the layers must to have the same filter amounts for the same size of feature map's output, and the filter amounts is twofold if the size of feature map is cut off a half to maintain the time complexity for each layer [21]. And one more thing is 34 weighted layers being involved in it. In addition, this plain network plus shortcut connections. Despite being identical in the input and output dimensions, the identity shortcuts were employed by a direct route. There were two choices for considering when the dimensions raised up. The first one was the shortcut would continue to execute identity mapping while padding extra zero entries for an increasement of dimensions. The projection shortcut could also be implemented to fit dimensions.

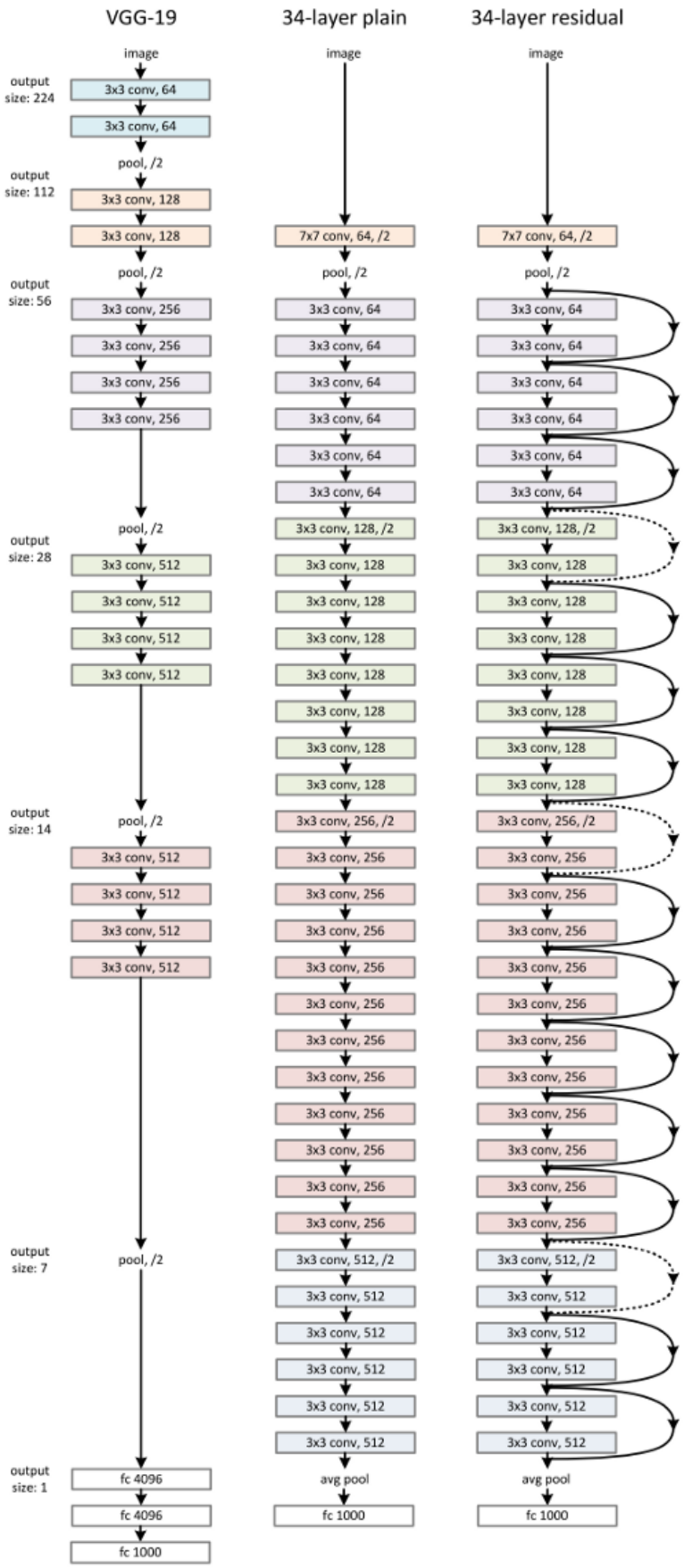


Figure 24: VGG-19 model, 34 Plain Network, Resnet 34.

4.2.2. Resnet – 50, Resnet – 101, and Resnet – 152 Architecture

Although the Resnet50 architecture is based on the above model, there is one significant variation. In this situation, the RB was re-created as a design of bottleneck appearance cause of concerning about the requirement of time for training the layers. Instead of using two layers, a three-layer stack was used for building of RB. As a result, each of the Resnet34's 2-layer blocks was taken a place of a 3-layer bottleneck block for the constructure of Resnet 50, which is as known as “Deeper Bottleneck Architecture”. This model is generally more accurate than the 34-layer ResNet model with performing of 3.8 billion FLOPS compared with 3.6 billion FLOPS [20].

Furthermore, Larger Residual Networks are the 101-layer ResNet101 and ResNet152, which are built with additional 3-layer blocks as well as Resnet50 [21]. Even with a increasement of network depth, the 152-layer ResNet reaches to 11.3 billion FLOPS, and has a specifically lower complexity than VGG-16 or VGG-19 nets with the number of FLOPS about 15.3 and 19.6 billion in respective.

5. EfficientNet

Following the previous success of AlexNets, GoogleNets, VGGs, Resnets, Gpipe, RCNN, or Mask RCNN with outstanding results in image recognition contests as well as the impressive accuracy in image recognition tasks bringing in reality. EfficientNet was born in 2019 by Quoc V. Le et al from GoogleBrain. Based on inspiration from the process of scaling up ConvNets, these researchers systematically studied it and came up with a completely new solution compared to the previous state of the arts solutions, which is scaling up the main component of the network model in 3 dimensions: in depth - scaling to add more layers to

the model, in width - scaling to add more channels for layers, in resolution - scaling in resolution being also known as the size of the layers, feature maps, they called it the "compound scaling method"[25].

Before going into more detail about EfficientNet, we will learn about the reason why the Model Scaling definition of the previous network models are the fundamental foundation to construct the EfficientNet.

5.1. Model Scaling

In fact, the state of the arts Models is built by Model scaling method, and "go deeper" is also one of this method. ResNet is able to be scaled up or down by modifying the layers of network - in depth, whereas WideResNet and MobileNets have ability to scale based on adjustment of the network's channels - in width. Besides, the network model is also widely acknowledged that larger input image sizes improve accuracy while incurring the burden of additional FLOPS [25]. Although several previous researches have proven that network depth and width are both important for the power expression of the ConvNets. It is still not easy to productively scale a ConvNet for a better achiement of accuracy and efficiency.

In Depth: Many ConvNets, including Resnets and Inception Net, scale network depth in this manner. The assumption is that deeper ConvNet is capable to capture more affluent and more complicated characteristics of image while also generalizing successfully on new tasks. However, because of the vanishing gradient problem, deeper networks are harder to train. In spite of having numerous approaches as

ResNet-skip connections and batch normalization to mitigate the training issue, the accuracy gain of very deep networks will drop, for example: Despite having so many more layers than ResNet-101, ResNet-1000 has equal accuracy [23].

In Width: For small scale models like MobileNet, MobileNetv2, and MnasNet, scaling network width is widely applied. Wide residual networks of Zagoruyko and Komodakis in 2016 explored how wider networks can take more fine-grained features and are easier to train [25]. On the other hand, outstandingly wide but shallow networks, having a difficulty in catching the features in higher level. Therefore, when networks expand substantially wider with bigger channels, the accuracy quickly saturates [23].

In Resolution: ConvNets may be able to catch more fine-grained patterns with higher resolution input photos. Early beginning with 224x224 with ConvNets, later ConvNets often employ 299x299 or 331x331. And in 480x480 resolution [23], GPipe just achieved state-of-the-art accuracy of ImageNet. In object detection ConvNets, higher resolutions, such as 600x600, are also commonly employed [23]. The results of scaling network resolutions are shown in the figure below, where greater resolutions indeed enhance accuracy, but the accuracy gain reduces for very high resolutions (r = 1:0 means resolution 224x224 and r = 2:5 denotes resolution 560x560).

Summary of this thing, although the accuracy of the model will be enhanced when scaling up any single dimension of network width, depth, or resolution, the accuracy gain reduces with larger models.

5.2. Compound Scaling Method – EfficientNet

Based on the above 3 options of Model scaling method, the EfficientNet's researchers have come up with the compound scaling method which is a uniform combination of 3-dimensional expanding the width, depth and resolution of the network, with a set of the ratio is fixed, this aims to create a balance in the dimensions of all 3 dimensions [25].

$$N = F_k \odot F_{k-1} \odot ..... \odot F_1(X_1) = \bigodot_{j=1...k} F_j(X_1)$$

$$N = \bigodot_{j=1...s} F_i^{L_i} X_{\langle H_i, W_i, C_i \rangle}$$

Figure 25: Common ConvNets equation.

The above formula presents for commonly structure of network model base on ConvNets layer, the ConvNets layer *i* was defined as an equation $Y_i = F_i(X_i)$, where F_i is an operator, Y_i is an output of tensor, X_i is tensor input, spatial dimensions are H_i and W_i as width and height, and the C_i represents for the number of channels, all of these things make a combination for a list of layers. In fact, the Convnet layers are divided into a plenty of stages and all the layers in each state have the same architecture such as Resnet [25]. F_i^{Li} presents F_i layer in L_i time of iterations in state *i*. However, EfficentNet expands three of the dimensions that make a no change to F_i. Because of fixing *Fi*, model scaling makes issue of the design simple for new constraints of material, in spite of continuing to exist a large design space in each layer, there is a consistence of expanding with constant ratio in all layers to reduce the design space, which serve for purpose of maximizing the model accuracy, in the circumstances, having any given resource constraints [25]. Formula below shows the optimal solution of the model scaling, where (*w, d, r)*

are coefficients for scaling network width, depth, and resolution, *F^i, L^i, H^i, W^i, C^i* are predefined parameters in baseline network.

And the model can achieve better accuracy and efficiency, more specifically this ratio is followed by this principle:

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Figure 26: The setting of coefficient values in compound scaling method.

where α, β, γ are constants that can be affirmative by a tiny grid search. Visionally, φ (Phi) is a coefficient is able to adjust to add more the number of available resources for model scaling by user's designation, while α; β; γ are respective to the size of width, depth and resolution that are additional to the network [25]. Strikingly, the FLOPS of a regular convolution operation is proportional to d, w^2, r^2, i.e., increasing network depth twice, will make a double increasement of FLOPS too, however, if there is a double gaining of network width or resolution, it will be a cause of a quad-increasement for FLOPS [25]. So that, convolution operations often prevail over the cost of reckoning in ConvNets, when making a scale for a ConvNet with the above equation, it will nearly improve overall FLOPS by $(\alpha \cdot \beta^2 \cdot \gamma^2)^{\varphi}$. And the default value of equation was set: $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ in order to be with any new φ, the total FLOPS will approximately increase by 2^{φ}.

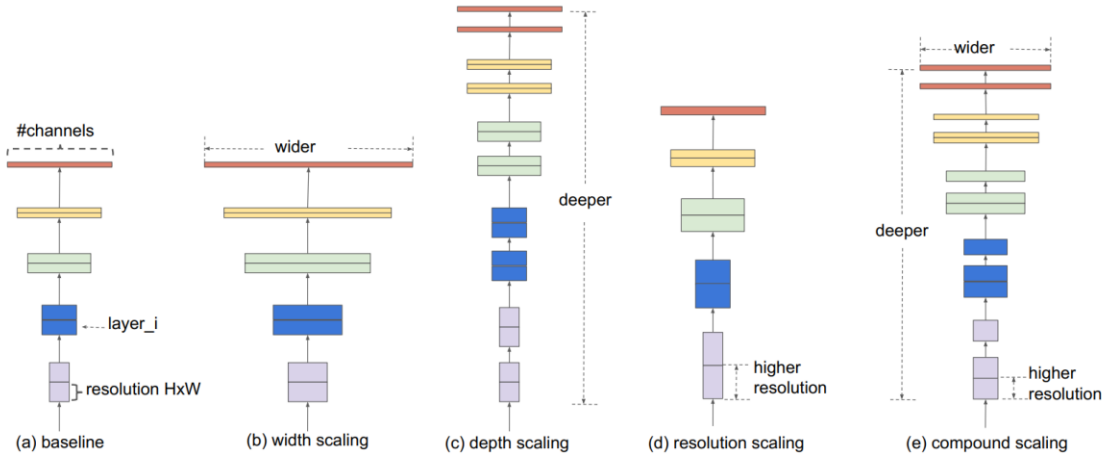


Figure 27: Model Scaling: (a) baseline network; (b) – (d): the networks only scale up one dimension of width, depth, or resolution; (e): compound scaling network has a scaling uniform of 3 dimensions with a fixed ratio.

5.3. EfficientNet Architecture

Because model scaling did not modify layer operators *F^i* of baseline network and had a good baseline network is also critical. The EfficientNet was built based on existing ConvNets with scaling method, in specific, they are MobileNet and Resnet, the mechanism is with a manipulation of a multi-objective neural architecture search, it made an optimization in both accuracy and FLOPS.

Indeed, manipulating $ACC(m) \times [F\ LOPS(m)=T]^{w}$ as a goal of optimization, where *ACC(m)* plays the accuracy and *FLOPS(m)* plays FLOPS of model *m*, *T* is the target FLOPS and w = -0.07 is a hyperparameter for adjusting the trade-off between accuracy and FLOPS [25]. This compound scaling method was applied with two steps:

• Firstly, there is a fixed φ = 1, assumption of having an availability of double resources and more, then making a small grid search of α; β; γ based on Equation 2 and 3. In particular, we find the best values for

EfficientNet-B0 are α = 1:2, β = 1:1, γ = 1:15, under constraint of α · β2 · γ2 ≈ 2.

• Secondly, α; β; γ will be fixed as constants and expanded baseline network with different φ using Equation 3, to obtain EfficientNet-B1 to B7.

$$\max_{d,w,r} \quad Accuracy\big(\mathcal{N}(d,w,r)\big)$$

$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\big(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\big)$$

$$\text{Memory}(\mathcal{N}) \leq \text{target\_memory}$$

$$\text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}$$

Figure 28: Optimal formula of scaling up method.

Outstandingly, it is feasible to attain even higher performance by looking for α; β; γ directly around a large model, but the search cost becomes prohibitively expensive for larger models. This method overcomes this problem by conducting a single search on the tiny baseline network at first step and then using the same coefficients of expanding for all other models at second step [25].

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Figure 29: Optimal formula of scaling up method.

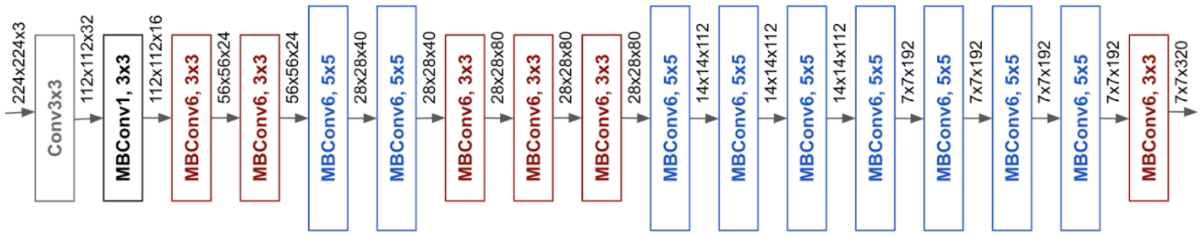


Figure 30: Efficient Net – B0.

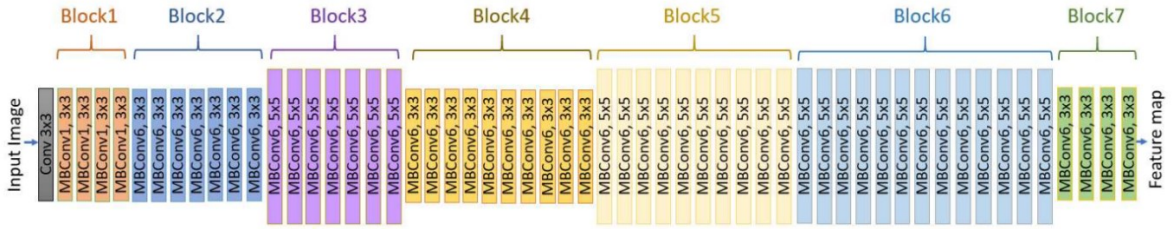


Figure 31: Efficient Net – B7.

As a result, EfficientNets give out an outstanding performance to other ConvNets. Specifically, EfficientNet-B7 got over the best existing accuracy of GPipe with using 8.4x less parameters than and running 6.1x faster on inference [24], it was about 66 million parameters and 37 billion FLOPS that reach to a proportion of 84.3 percentage of the accuracy. Adding, there was a comparision of ResNet-50 and EfficientNet-B4 that made an improvement of the peak of accuracy from 76.3 percent to 83.0 percent, increasing about 6.7 percent with similar FLOPS. Besides ImageNet, EfficientNets also transfer well and achieve state of-the-art accuracy on 5 out of 8 widely used datasets, while reducing parameters by up to 21x than existing ConvNets.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Figure 32: EfficientNets Performance Result on ImageNet.

6. Fully Convolutional Networks

One of the turning points in the field of image detection was the introduction of the fully convolutional network (FCN), which is known as a variant of CNN for image segmentation work [10]. The FCN has a difference from ordinary CNNs in which the fully connected layers are replaced by convolution layers at the end of the network. So, there is a generation of a network making a computation of a nonlinear filter for each layer's output vectors. As a result, not only can the completed network function on an input of any size and make a production of an output with respective spatial dimensions, but this also allows the classification network to generate a heatmap of the class of desire. In addition, adding layers and a spatial loss to the network results in an efficient machine for end-to-end dense learning [10]. Figure below shows the transformation of FCN.

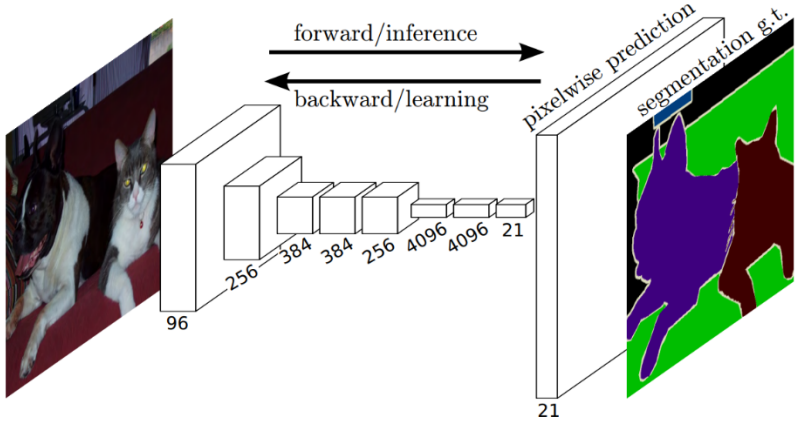


Figure 33: Fully Convolutional Network serves for semantic segmentation tasks

Not only do FCNs use to make a flexibility with the capacity to take multiple input image sizes, but it has also been shown to have a more efficiency for dense predictions of learning via in-network up-sampling [10]. Furthermore, the FCN may preserve spatial information from the input that is very important for semantic segmentation task because of involving both localization and classification for the task. Although an FCN has an ability to accept any size input image, there is a decreasement of the output resolution by being applied convolutions without any padding. These were added to make filters minimal and requirements of calculation manageable. As a result, the coarse output is reduced in size by a factor equal to the pixel stride of the output unit's receptive field [10].

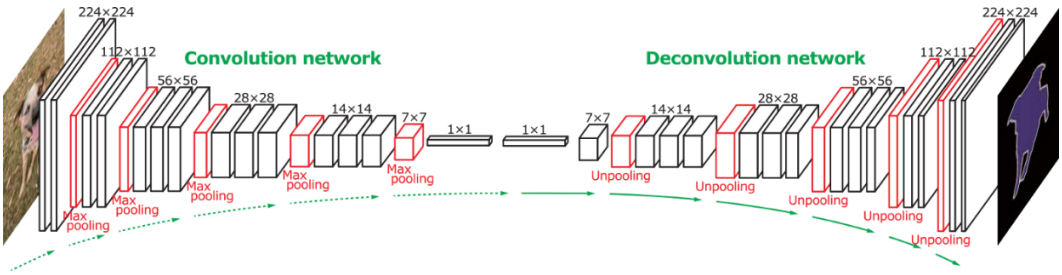


Figure 34: Fully Convolutional Network Architecture

In other words, the FCN is divided in 2 parts that are convolution network (down-sample) as an encoder step and the following is deconvolution network (up-sample) known as decoder step, the convolution network part is very similar with casual CNN architecture so we will go deep down to up-sample part.

In Semantic Segmentation, after using the method of convoluted features extraction to decrease the input's resolution, the following step is to upscale the low resolution turning back to the original resolution of the input image. Pooling is the process of converting a bunch of values to a solitary value, whereas up-sample is the process of converting a solitary value to a bunch of values and it can be named Unpooling, like Pooling, the unpooling can be carried out in a variety of ways [26].

1. **Nearest Neighbour:** Choosing a value and populating it into the surrounding cells (number of cells depending on the increasement in resolution), figure below shows the copy process of all values in matrix 2 x 2 to every cell with size of 2×2 square in new matrix 4 x 4 [26].

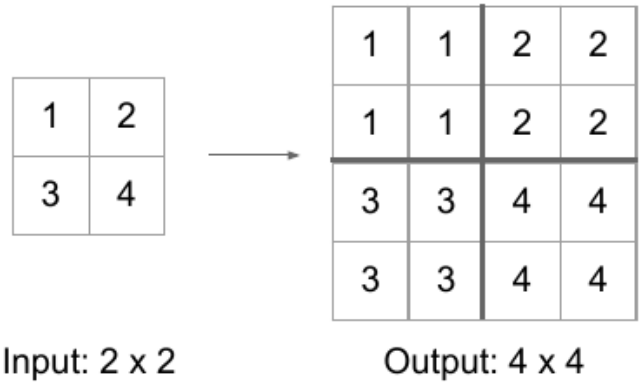


Figure 35: Nearest Neighbour Unpooling

2. **Bed of Nails**: Inserting the value in turn at a predefined cell and adding the rest with 0, the figure below is an example of

filling the value from the top-left cell of each 2 × 2 square in new matrix with size 4 x 4 [26].

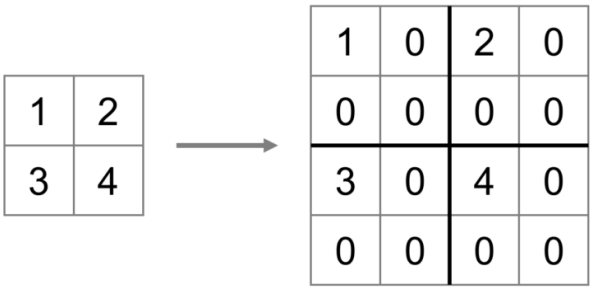


Figure 36: Bed of Nails Unpooling

3. **Max Unpooling:** In CNN, the max-pooling layer pulls the maximum values from the picture regions covered by the filter in order to down-sample the data, and the unpooling layer provides the value to the location from which the values were picked up in order to up-sample the data [26]. However, these approaches are not data-dependent, which means they are able to learn nothing from data; they are just programmed computations making them become as a specific task and costly in terms of calculation timing. The generic answer for these situations is transposed convolutions.

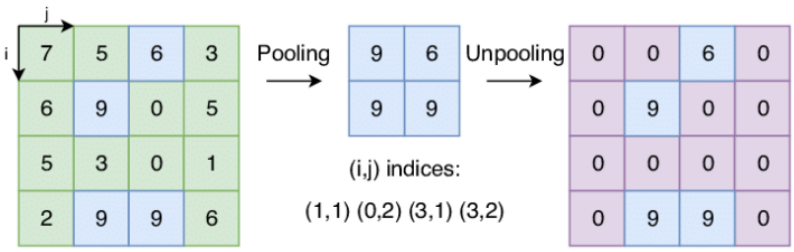


Figure 37: Max Unpooling

Transposed Convolution, also known wrongly as Deconvolution, is the inverse action of Convolution. A transposed convolution layer, as opposed to convolution, is used to up-sample the decreased resolution

feature back to its original resolution. To obtain the final output from the lower resolution features, a set of strides and padding values is learned. Transposed convolution is superior than other Up-sampling techniques because of being unlike earlier techniques (Nearest Neighbour, Bed of Nails, Max Unpooling), the transposed convolution is a learnable up-sampling technique being highly efficient. The image below describes the method in a very simple way [26].

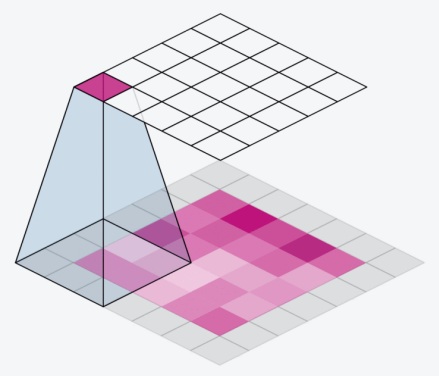


Figure 38: Transposed Convolution

Next, because of an existing main problem with in-network down-sampling in an FCN, which is the reduction of the resolution of the input by a huge proportion, making it impossible to recreate finer features during up-sampling even with complicated techniques like Transpose Convolution. As a result of getting a coarse output. One solution is to include "skip connections" in the up-sampling from previous layers stage and then making a sum of total the two feature maps. These skip connections supply enough information to following layers to produce correct borders of segmentation. This combination of fine and coarse layers yields local forecasts with virtually accurate global (spatial) structure. The skip connection learns how to integrate coarse and fine layer information. So, there is a performance of grids with varying degrees of spatial coarseness for layers, with the intermediate convolution layers of FCN deleted for clarity [26].

7. U-Net

The U-Net architecture is a variant of the FCN that is commonly used for semantic segmentation duties. Ronneberger et al. created the first U-Net architecture in 2015 to conduct picture segmentation and localisation for biological applications [27]. U-Net is superior to traditional CNNs since they can give localization as well as classification in their output. In this scenario, localization implies labelling each single pixel from an image with a certain class. It is also more favourable than FCN because U-Net can function with less training photos while producing more exact segmentations. This is accomplished by constructing up-sampling layers with a huge amount of feature channels allowing context information to be propagated to layers having better resolution. Additionally, the segmentation networks of U-Net are the most appropriate for this application because they are the most easily scalable and sizeable FCN. Specially, they can avoid a compromise between localization and contextual information, and are frequently employed in cutting-edge semantic segmentation approaches [27].

7.1. U-Net Architecture

The U-Net architecture is made up of a path of contraction at the input and a path of expansion at the output. Ronneburger's paper introduces the basic architecture of this network, the first path is the contraction path, also known as the encoder path, which is essentially a stack of convolution, activation, and pooling layers designed to take the context from the input image. The encoder output, which is smaller than the input, is gradually expanded in the expansion path. With transposed convolutions, the expansion path has another name is decoder path which allows for exact localization. Through a series of up-convolutions and concatenation with appropriate feature maps from the contracting path,

the expansion pathway integrates high level features with spatial information. The architecture is depicted in figure 39 [27].

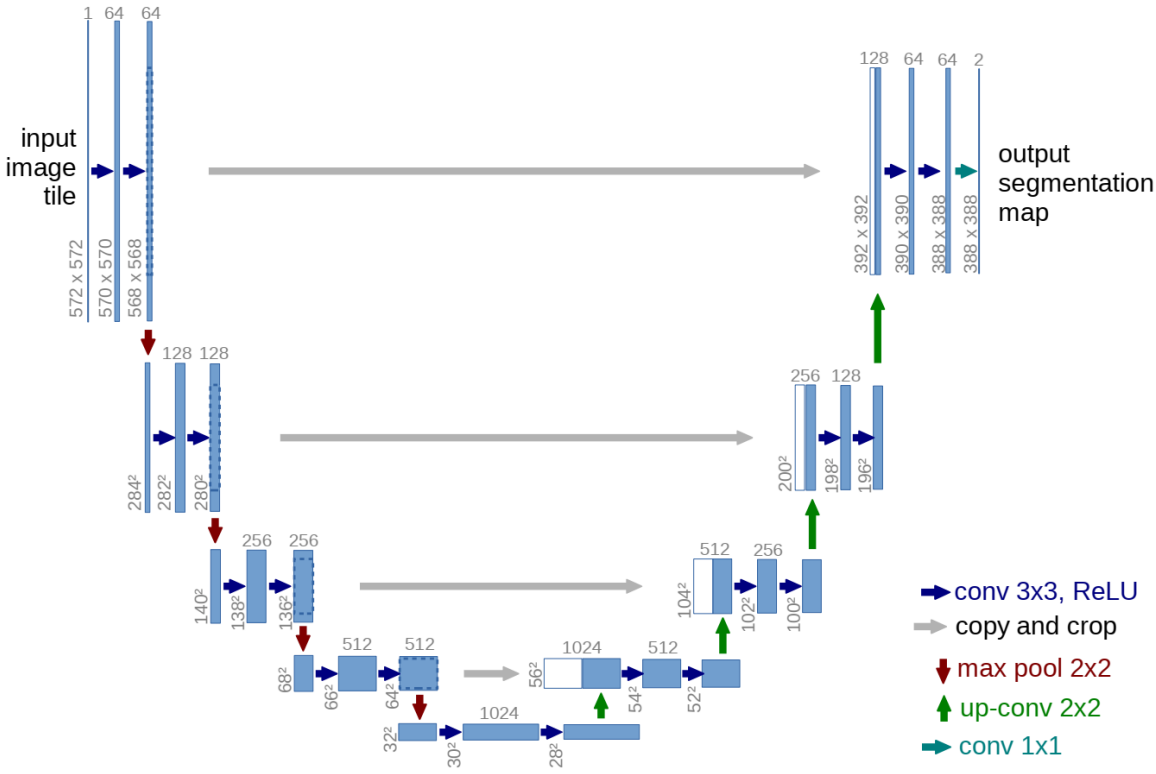


Figure 39: U-Net Architecture (The representation of a multi-channel feature map is each blue box. The number of channels is indicated on the box's top. The x-y size is specified at the box's lower left side. White boxes represent feature maps that have been replicated. The arrows represent the various operations).

The encoder route of U-Net is typical CNN design, including recurring 3x3 convolutions and operations of max pooling with stride 2 for down-sampling which double amount of feature channels before appearing of up-sampling the feature map and performing a 2x2 convolution on the expanding route. In detail, these actions remove the feature channels amount in a half before concatenating them with the feature map being cropped from the route of contraction, this

concatenation of the U-Net is the different point with the skip connection of FCN. After that, an extra convolution layer is added to map each feature vector to the expected number of classes [27]. The U-Net can collect image context by employing the contracting route, on the other side, the symmetric expanding route allows for exact localization. By integrating the characteristics with high resolution which come from the contracting path being up-sampled output, localization is activated. Added, the propagation of context information to the layers with higher resolution is allowed by the model's up-sampling section with a huge amount of feature channels [27].

Based on the feature maps with low level from the encoder contain more spatial information which is helpful to the analysis of complicated scenes containing multiple objects and their relative configurations, there is a combination of the feature maps at intermediate low level from EfficientNet or Resnet and the feature maps at intermediate high level from U-Net decoder. The network is able to transfer context information to higher resolution layers due to the huge amount of feature channels in the up-sampling section. Because of the symmetric characteristics in conventional U-Net. In my work, I have tried to use EfficientNets and Resnets as the encoders in contraction route instead of regular set of convolution layers. The decoder module is similar to the primordial U-Net. The detailed implementation for the combination of these architectures will be mentioned at the following section named “implementation”.

8. Optimization Algorithms (Optimizers)

During the training phase, we frequently modify and change our model's parameters (weights) in order to try to minimize its loss function and help produce predictions as optimal and accurate as possible. As a

result, optimizers were developed to assist in resolving this issue. They make a combination between the loss function and model parameters by making updates of the model in response to the loss function's output. In other terms, optimizers are techniques or approaches which are utilized to help error function (loss function) reaching a minimum value or to make production efficiency peaking a maximum value [28]. The optimizers are built from mathematical functions that are influenced by learnable parameters of the model such as Weights and Biases. Additionally, the optimizers assist in determining how to adjust the weights and learning rate of a neural network in order to minimise losses. In a nutshell, the optimizers will mould your model into its most accurate possible form by tinkering with the weights [28]. And the loss function serves as a guide to the terrain, informing the optimizer whether it is travelling in the correct or incorrect path. In previous section about ANN, I already mentioned to some optimizers such as Gradient Decent, Stochastic Gradient Decent, Mini-Batch Gradient Decent. So, in this section, I will continue to talk about other common algorithms such as SGD with Momentum, Adaptive Learning Rate methods (AdaGrad, RMSprop, Adam).

8.1. Stochastic Gradient Descent with Momentum

SGD has difficulty navigating ravines, which are widespread around local optima and are regions where the surface curves considerably more steeply in one dimension than another. In these settings, SGD oscillates across the ravine's slopes, making only sluggish progress along the bottom toward the local optimum [28]. That is the reason why momentum was established to reduce extreme volatility in SGD and soften the convergence. It promotes convergence in the relevant direction while decreasing fluctuation in the irrelevant direction. This approach employs another hyperparameter known as momentum, denoted by the symbol "γ".

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Figure 40: Momentum Formula

Some implementations swap the equations' signs. Typically, the momentum term γ is set to 0.9 or a similar number. We basically use momentum to drive a ball down a hill. As it rolls downhill, the ball gains momentum, increasing faster and faster (until it achieves its terminal velocity at the point being resistant by the air, i.e. $\gamma < 1$) [29]. The term of momentum increasement for dimensions have the point of gradients in the same directions and decreases for dimensions have change directions in gradients. Consequently, there is an achievement of quicker convergence and less oscillation. Although momentum helps eliminate oscillations and high variance in the parameters and converges faster than gradient descent, there is one more hyper-parameter which must be added and adjusted manually and precisely [28].

8.2. Adaptive Learning Rates Methods:

The difficulty with employing learning rate schedules is that their hyperparameters must be established in advance and are strongly dependent on the type of model and issue. Another issue is that all parameter updates use the same value of learning rate [29]. If the data is sparse, we desire to update the parameters to a different extent. With this cause, AdaGrad, Adadelta, RMSprop, and Adam are adaptive learning rate algorithms that offer an alternative solution for traditional SGD. These per-parameter learning rate approaches give a heuristic approach without the need for costly manual tuning of hyperparameters for the learning rate schedule.

AdaGrad, in summary, conducts greater changes for more sparse parameters and smaller changes for less sparse ones. It performs well with sparse data and in training large-scale neural networks. However, during training deep neural networks, its monotone learning rate is frequently excessively aggressive and stops learning too soon, that means AdaGrad makes learning rate changing adaptively in iterations and has an ability to train spare data as well, but it will make dead neuron issue when the neural network is deep the learning rate becomes very small number [29].

So, AdaDelta is an Adagrad addition that aims to minimize its belligerent, monotonically declining learning rate, when using AdaDelta, it is not essential to set a default learning rate, but the cost for computation is expensive [29].

In addition, RMSprop makes a very modest adjustment to the Adagrad method in an attempt to minimize its aggressive, monotonically falling learning rate. Specifically, the RMSprop helps learning rate automatically be tweaked and picks a different learning rate for every single parameter, however, it makes the training model very slow [29].

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Figure 41: RMSprop function

The learning rate is also divided by an exponentially decaying average of squared gradients by RMSprop. It is recommended that γ be set to 0.9, with 0.001 as an acceptable default value for the learning rate η [29].

Adam stands for Adaptive Moment Estimation, is one of the most well-known and often used gradient descent optimization methods, which is a technique for calculating adaptive learning rates for every single parameter, stores the decaying average of previous gradients, comparable to momentum, as well as the decaying average of previous squared gradients, similar to RMS-Prop and AdaDelta. It incorporates the benefits of both strategies. Furthermore, Adam brings some benefits that are easy to implement, computationally efficient, using little memory requirements, converging rapidly, rectifying vanishing learning rate and high variance, but its computation is costly [28].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

Figure 42: Adam function

In figure above, *m(t)* and *v(t)* are values of the first moment which is the Mean and the second moment which is the uncentered variance of the gradients respectively. There is a set of default values for β1 = 0.9, β2 = 0.999, and $\epsilon = 10^{-8}$.

REFERENCES:

[1] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Math. Control Signals Systems*, 1989.

[2] F. Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, USA, 1st edition, 2017

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning (Adaptive Computation and Machine Learning Series*. MIT Press, Cambridge, MA, USA, 2016.

[4] https://www.ibm.com/cloud/learn/gradient-descent

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*

[6] https://builtin.com/data-science/gradient-descent

[7] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1451 - 1460, March 2018.

[8] Kunihiko Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.*

[9] https://cs231n.github.io/convolutional-networks/

[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[13] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in Natural to Artificial Neural Computation. IWANN 1995. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 195–201.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[15] B. Karlik and A. Vehbi, "Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks," International Journal of Artificial Intelligence and Expert Systems (IJAE), vol. 1, no. 4, pp. 111–122, 2011.

[16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language Modeling with Gated Convolutional Networks," arXiv, 2017.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," Haifa, 2010, pp. 807–814.

[18] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, and G. E. Hinton, "On rectified linear units for speech processing," in International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 3517–3521

[19] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in International Conference on Machine Learning, 2011

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition"

[21] https://viso.ai/deep-learning/resnet-residual-neural-network/

[22] https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[23]https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95

[24] https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[25] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"

[26] https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/

[27] *Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation".*

[28] https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[29] https://ruder.io/optimizing-gradient-descent/index.html#momentum