CMP304 Project Report
Image Classifier using Convolutional Neural Network

Utkarsh Yadav 1905406

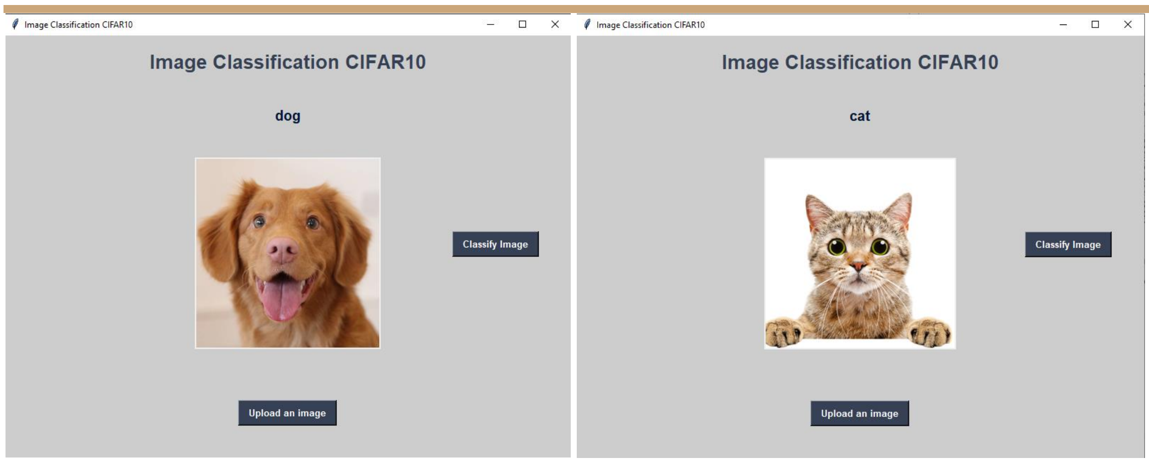


Image Classification CIFAR10
Image Classification CIFAR10
dog
Classify Image
Upload an image
Image Classification CIFAR10
Image Classification CIFAR10
cat
Classify Image
Upload an image

Figure 1Title Image

Introduction

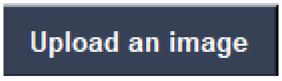
An image classifier was developed for this project. Classifying different types of images has always been hard for machines, this is due the complexity of image classification. Image classification is important as it gives a machine the power sight and the power to tell the difference between different objects, which in turn helps them with tasks like medical imaging, object identification in satellite images, traffic control systems, brake light detection, machine vision, and more.

One of the first image classification neural network made was LeNet developed by Yann LeChun in 1989 (LeChun, 1989), it used a convolutional neural network it also promoted the development of deep learning it had an accuracy of 66% on the MNIST dataset (LeChun, et al., 1998). It was the first of its kind which used feature detection for detecting spatial structure in an image.

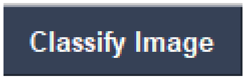
The next big image classifier was AlexNet which was designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton (Krizhevsky, et al., 2012). AlexNet introduced better non-linearity in the network using ReLU, it also introduced the concept of dropout as regularization, this made about 82% accurate on the CIFAR-10 dataset.

The application made is a very beginner level image classification. The image classifier uses a convolutional neural network with a feed-forward network architecture. Convolutional neural network is used for its ability to identify the spatial structure in an image. The CIFAR-10 (Krizhevsky, 2009) dataset is used to train the neural network. The CIFAR-10 dataset contains 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The keras API is used for loading and training the model in python. Tkinter graphics library is used for inputting custom images into the model for evaluation. The model returns a probability distribution of the predicted answer and the index of the highest number in the distribution is compared against the image class.

User Guide



Click upload image button on the GUI application and choose any of the 10 images



Click on classify to classify the image.

Links

GitHub repository link: https://github.com/utkarshyadav009/CMP304-Image-Classifier

YouTube video link: https://youtu.be/c9A7-P9fl-c

Data Specification

The most important thing required to build a good image classifier is the dataset used to train the neural network. A dataset that wasn't too big and had enough class variations was required for this problem. In the end the CIFAR-10 dataset (Krizhevsky, 2009) was used as it satisfied all the above requirements and is also a good start for beginners who are looking to learn about computer vision and convolutional neural network.

CIFAR-10

The CIFAR-10 dataset (Canadian Institute for Advanced Research) is a collection of 60,000 32x32 color images which are divided into 10 classes which are: -

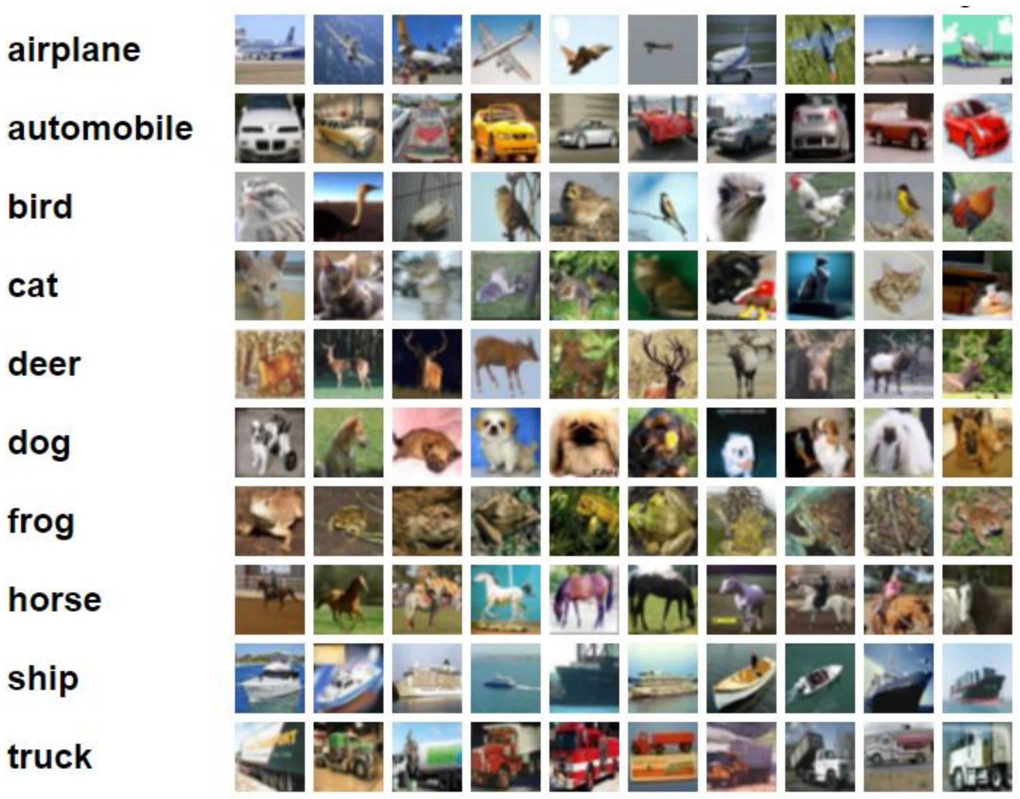


Figure 2 Dataset sample Images

Each class is made up of 6000 images. There are 50,000 training images and 10,000 test images, all the classes are mutually exclusive i.e. no overlap between classes. The dataset is divided into 5 training batches and one test batch, each containing 10,000 images. The test batch containes 1000 randomly selected images from each class.

Background

Convolutional Neural Network

A convolutional neural network was used to build the image classifier as it takes in account the spatial structure of an image. Convolutional neural network can be divided into 3 basic concepts:

Local receptive field

In an artificial fully connected network, the inputs are depicted as a vertical array of neurons, in contrast to this a convolutional network is depicted as a 2-D array of inputs whose values corresponds to the pixel intensity of the image (Nielsen, 2019).

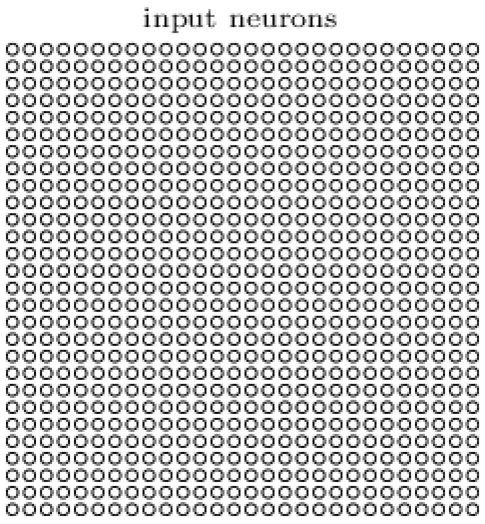


Figure 3 2D array of Input neurons

The input pixels are connected to a layer of hidden neurons but not every input pixel is connected to every hidden neuron, instead a small, localised region of the input image is chosen to connect with a single hidden neuron i.e., each neuron in the first hidden layer is connected to a small region of the input neurons. For example, for a region of 5x5 i.e., 25 input neurons a single connection to one hidden neuron is made.

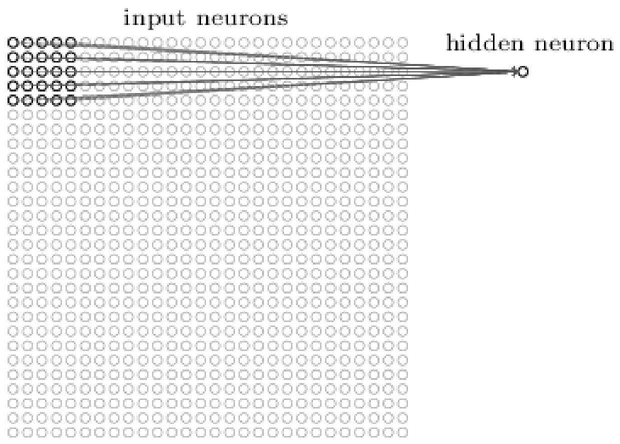


Figure 4 First hidden layer neuron

This region is called the local receptive field for the hidden neuron. The local receptive field is then slid across the entire input image, and for each local receptive field there is a hidden neuron. The sliding of local receptive field is called stride and the amount of pixel slid is called stride length for example:- here the stride length is one as the local receptive field slides on pixel to the right.

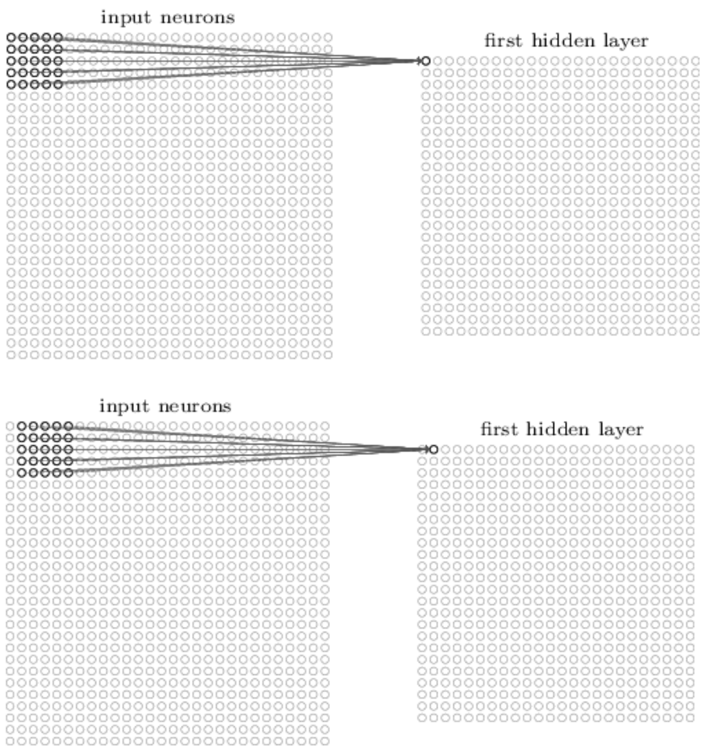


Figure 5 Stride example

For an input image of size 32x32, and 5x5 local receptive fields, there will be a total of 28x28 neurons in the hidden layer. All the neurons in the hidden layer have the same weights and bias meaning that all the neurons in the first hidden layer will detect the same exact feature. A feature can be defined as an input pattern that'll cause the neuron to activate. It can be the edge of an image or a certain shape in an image. This concept of same weights and bias is known as shared weights and bias.

Shared Weights and Bias

Each connection in the hidden layer has a 5x5 weights attached to it, which is calculated when the connection is made, and each neuron has an overall bias. So, for a hidden layer containing 28x28 neurons the output for the j,k^{th} hidden neuron is

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right) \quad eq(1)$$

σ	Neural network activation function
b	Shared value for the bias
$w_{l,m}$	5x5 array of shared weights
$a_{x,y}$	Input activation at position x, y

As mentioned above each neuron in the first hidden will have the same weights and bias which lets them detect the exact same feature, just at different locations in the input image. Shared weights are the reason why convolutional networks are well adapted to the translation invariance of images.

The local receptive field from the input layer to the hidden layer is also referred to as a map, and because this map detects input patterns or features, it's called a feature map. The shared weights and bias are referred to as a kernel or a filter.

To perform image classification more than a single feature map is required to identify multiple localized features in an input image. Therefore, a comprehensive convolutional layer is made up of multiple separate feature maps.

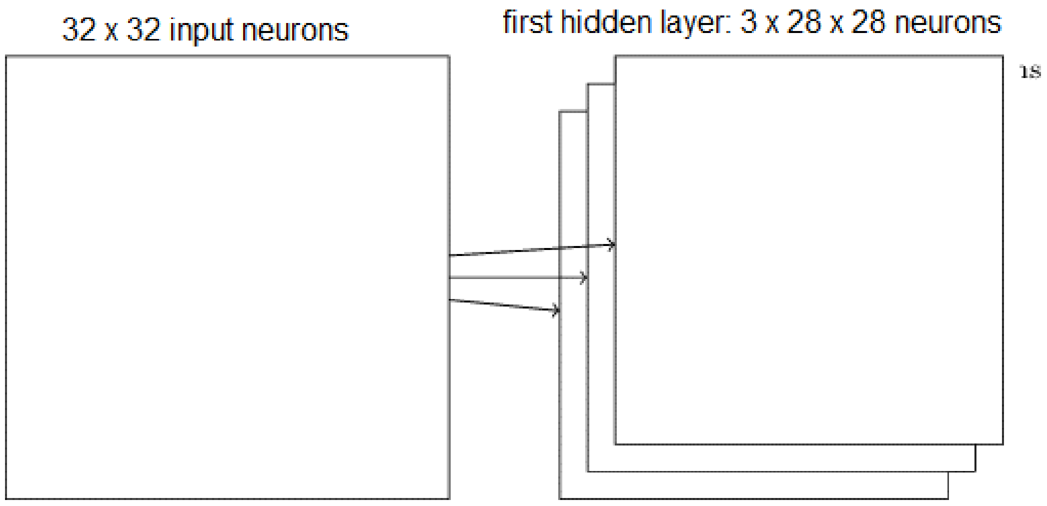


Figure 6 Multiple Feature maps

In the example shown above, there are 3 feature maps of size 28x28 for an input image of size 32x32. The feature maps are characterized by a set of 5x5 shared weights and single shared bias. Which results in the network being able to detect 3 distinct kinds of features.

Here's a visual example taken from the "Visualizing and Understanding Convolutional Networks" paper by Matthew D. Zeiler and Rob Fergus (Zeiler & Fergus, 2013).

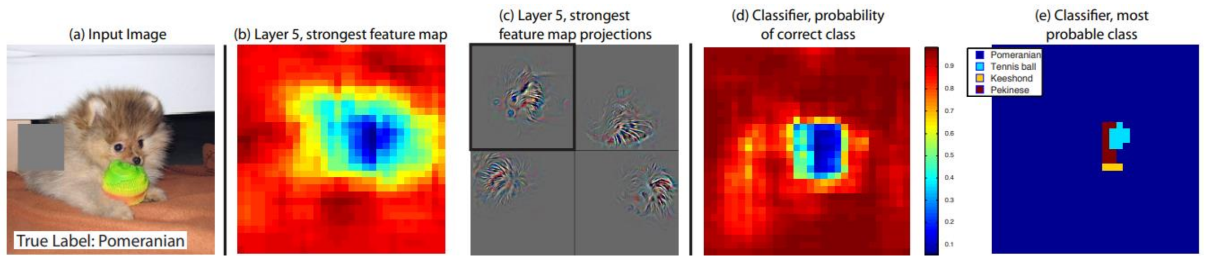


Figure 7Feature detection example

The image above shows the different kinds of features a convolutional layer looks for, the layer also has a unique spatial structure which is not random by any means. The features are divided into a light and dark sub region, meaning that the network does learn the spatial structure of any given input image.

One of the biggest advantages of shared weights and biases is that it greatly reduces the number of parameters involved in a convolutional network. In a conventional artificial neural network with an input layer having 32 x 32 neurons i.e., 1024 input neurons and a modest 25 hidden neurons the total number of weights will come to be 1024 x 25 which is 25,600 with 25 biases making the total number of parameters to be 25,625. In stark contrast to that in a convolutional network with 20 feature maps, each having 5x5 shared weights plus a single bias, making the total number of parameters 26. Which makes the total number of parameters in the entire convolutional layer to be 20x26 which is 520. Which is a 97.9% decrease in the total parameters. This in turn makes the training for a convolutional neural network faster in comparison to an artificial fully connected neural network.

Pooling

A pooling layer is used after the convolutional layer which simplifies the output from the convolutional layer. It takes the output of each of the feature maps and creates a compressed version of it. For example, each unit in a pooling layer encapsulates a 2x2 region in the hidden layer. One of the most common pooling methods is called max pooling, in this the pooling unit outputs the maximum activation in the 2x2 region of the hidden layer, as shown in the diagram below.

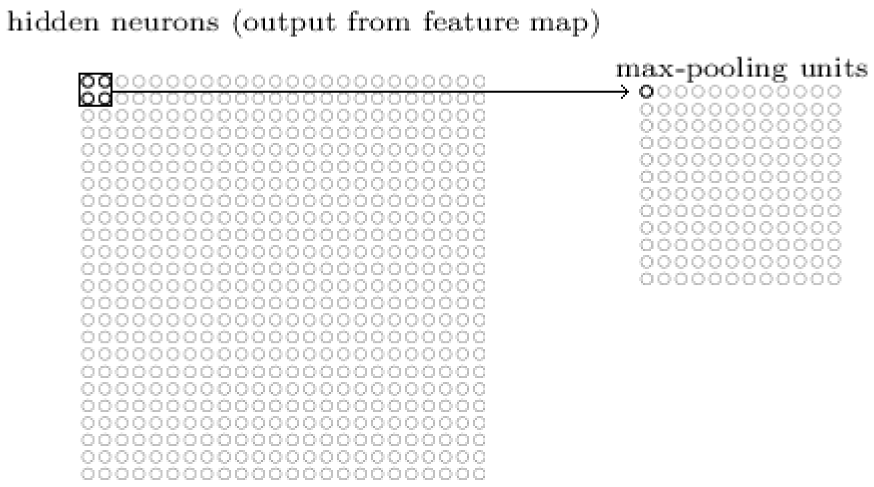


Figure 8 Max-pooling

For a feature map of size 28x28 hidden neurons the amount of the neurons left after pooling would be 14x14 neurons in turn halving the number of neurons. So, putting everything together the convolutional layer would look something like this:-

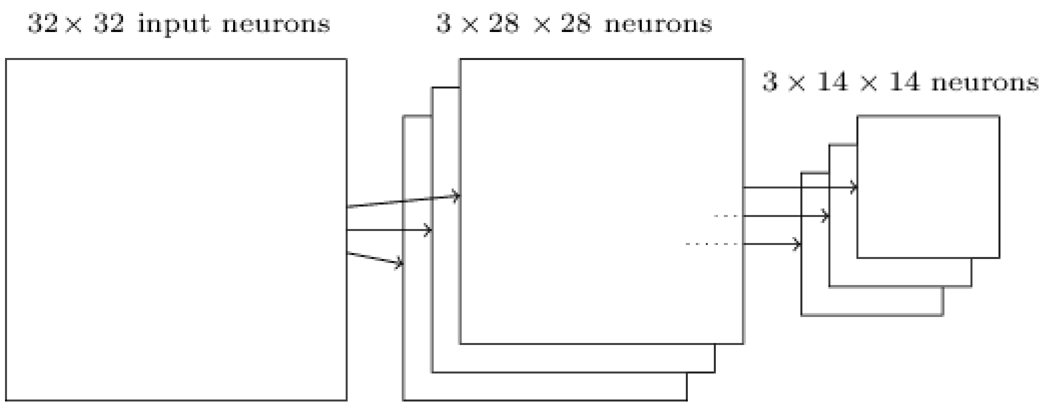


Figure 9 Convolutional Layer

To put it simply max pooling selects the most prominent features from a feature map, this reduces the number of pooled features which helps reducing the number of parameters needed in a layer. The reason the number of parameters in a layer is important is as the convolutional neural network becomes deeper the speed of each layer is extremely important for an overall faster training time.

Cost Function

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2 \quad eq(2)$$

x	Training inputs.
$y(x)$	Desired output for input ***x.***
w	Collection of all weights in the network.
b	All the biases.
n	Total number of training inputs.
a	Vector of outputs from the network where x is input.
C	Quadratic cost/loss function.

The quadratic function $C(w, b)$ is non-negative, and the cost $C(w, b)$becomes smaller i.e., $C(w, b) \approx 0$ when $y(x)$ is approximately equal to the output, a, for all training input x (Nielsen, 2019). This means if the cost function is approaching zero the prediction is closer to the desired output, on the contrary if the cost function is not approaching zero the prediction is not closer to the desired output. The goal of the training is to minimise cost as a function of weights and biases i.e., setting the weights and biases which will minimise cost as small as possible. It is achieved using the Gradient Descent algorithm.

Gradient Descent

Gradient Descent is a way to converge towards a local minimum of a cost function. A common way to visualize this is to use a visual diagram of a ball rolling down at the bottom of a valley.

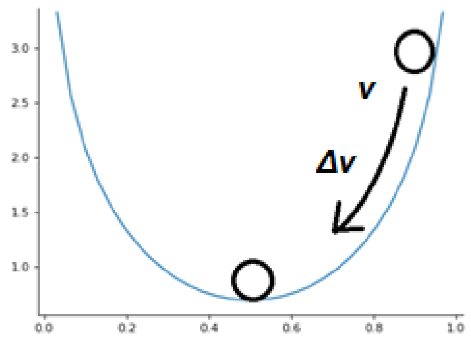


Figure 10 Gradient Descent visuals

Suppose the ball moves in the direction v and the change in distance is Δv. Using calculus, the change in C i.e., ΔC can be represented as follows: $\Delta C \approx \frac{\partial C}{\partial v} \Delta v$ where $\frac{\partial C}{\partial v}$ can be written as ∇C making the equation $\Delta C \approx \nabla C . \Delta v$, ∇C is called the gradient vector. This equation is used to see how to choose Δv so that ΔC becomes negative. For example choosing Δv as $-n\nabla C$ where n is a small positive parameter also known as the learning rate gives $\Delta C \approx -n\nabla C . \nabla C = -n||\nabla C||^2$, as $||\nabla C||^2$ is always ≥ 0 the value of ΔC will always be ≤ 0. Changing the value of v with respect to $-n\nabla C$ will keep decreasing the value of C until a global minimum is achieved. The value of n should be small enough so that $\Delta C > 0$, but not too small as it would make the changes Δv miniscule.

Applying this rule to the weights and biases of the neural network will give these following equations:

$$w = w - n\frac{\partial C}{\partial w} \quad eq(3)$$

$$b = b - n\frac{\partial C}{\partial b} \quad eq(4)$$

The problem with this approach is that for each training input x a computation of the gradient ∇C_x must be performed, meaning when the number of training input increases the computation time increases as well. To counteract this stochastic gradient descent can be used to speed up learning.

Stochastic Gradient Descent

The idea behind stochastic gradient descent is randomly picking out a small number of training inputs and computing them with the gradient function, which gives a good estimation of the true gradient, hence speeding gradient descent and learning. To further elaborate, suppose a small number of training inputs are chosen randomly, let's call that m. The training inputs can be $X_1, X_2 \ldots . X_m$, this can be referred as a mini batch. It is expected that the average value of ∇C_{X_i} where i is the number of elements is roughly equal to ∇C_x if the sample size of m is large enough. Putting all of that into an equation result in this, $\frac{\sum_{i=1}^{m} \nabla C_{X_i}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$, meaning the value of ∇C can be written as $\nabla C = \frac{1}{m}\sum_{i=1}^{m} \nabla C_{X_i}$, this confirms that the overall gradient can be estimated by computing a randomly chosen mini batch.

Now applying this new equation to the weights and biases in the neural network gives these following equations:-

$$w = w - \frac{n}{m}\sum_i \frac{\partial C_{X_i}}{\partial w} \quad eq(5)$$

$$b = b - \frac{n}{m}\sum_i \frac{\partial C_{X_i}}{\partial b} \quad eq(6)$$

After training the neural network with one mini batch, another mini batch is chosen randomly for training, this is done until all the training inputs are used, this is called an epoch of training.

Cross-entropy

The above $eq(2)$ cost function is a quadratic cost function, and it can be a victim of a phenomenon known as learning slowdown (Choi, 2016). This is due to the structural characteristics of the gradient descent method. As explained in (*Figure 10*) when the ball is dropped from a high place i.e., the gradient is large, the ball moves faster but as soon as the gradient decreases i.e., when the ball is near the lowest point, the movement slows down and learning slowdown occurs in which improvement in learning is minimised, even if more learning is performed.

To counter learning slowdown the quadratic cost function can be replaced by a different cost function which is known as cross-entropy. To understand cross-entropy, let’s use a simple example:-

Suppose a neuron with several input variables $x_1, x_{2\ldots,}$ with corresponding weights $w_1, w_2 \ldots .,$ and a bias b is getting trained.

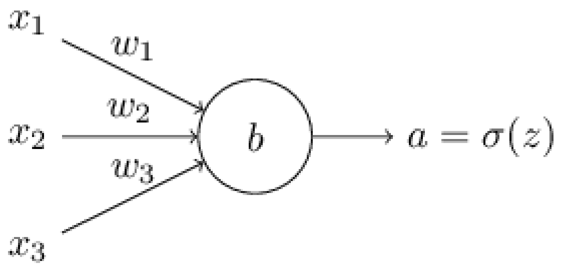


Figure 11 Single neuron

The output $a = \sigma(z)$ where $z = \sum_j w_j x_j + b$ is the weighted sun of the inputs.

The cross-entropy function can be defined as:-

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)] \quad eq(7)$$

There are two properties that make this a cost function, first is that its non-negative i.e., $c > 0$ this can be seen as all the individual terms in the sum are negative and since both logarithms are in the range of 0 to 1 and there is a minus sign at the start of the function. Secondly, if the output of the neuron is closer to the desired output for all training input x, then the value of the cost function will be close to zero. To prove this let's use an example, suppose $y = 0$ and $a \approx 0$ for some input x. In this case the neuron is doing a good job on the provided input. The first term in $eq(7)$ $y \ln a$ is zero and the second term $(1 - y)\ln(1 - a)$ is close to zero.

In conclusion the cross-entropy function is positive and tends to move towards zero as the neuron gets closer to the desired output y for all training input x. To prove how cross-entropy solves the learning slowdown problem, let's take the partial derivative of the $eq(7)$ and substitute $a = \sigma(z)$ and apply chain rule twice. This is done for w_j where j is the number of weights.

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial C}{\partial w_j} =$$

$$-\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j \quad eq(8)$$

Putting everything over a common denominator gives:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)\big(1-\sigma(z)\big)}(\sigma(z) - y) \quad eq(9)$$

Using basic algebra, the expression $\sigma'(z)$ can be written as $\sigma(z)\big(1-\sigma(z)\big)$ where $\sigma(z)$ is the definition of the sigmoid function which is $\frac{1}{(1+e^{-z})}$. The terms $\sigma'(z)$ and $\sigma(z)\big(1-\sigma(z)\big)$ cancels out and equation simplifies to be:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y) \quad eq(10)$$

This equation signifies that the rate at which leaning it performed is controlled by $\sigma(z) - y$ i.e., the error in the output meaning larger the error faster the learning. It avoids the learning slowdown caused by the $\sigma'(z)$ term in the analogous equation for the quadratic cost (Nielsen, 2019).The term $\sigma'(z)$ gets cancelled out when cross-entropy is performed which in turns eradicated the learning slowdown phenomenon.

The equation is very similar for the bias as well.

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z) - y) \quad eq(11)$$

Activation Functions

ReLU Activation

Before the convolutional layer is max pooled an activation function is applied to it, this is done to prevent the exponential growth of the computation required to run the network and the activation function used for this image classifier is ReLU or Rectified Linear Unit. ReLU is used as it is linear activation function which can act like a non-linear function if the input data goes below zero, it is also compatible with multi-layer neural network and if the size of the network increases the cost of adding more ReLU will increase linearly. The formula for ReLU is as follows: -

$$y = \max(0, x) \quad eq(12)$$

An easy way to explain the above formula is by this if statement

```
$if\ input > 0:$
    $return\ input$
$else:$
    $return\ 0$
```

The function is linear (i.e., has a constant rate of change) for input values greater than zero, yet it is nonlinear if the input values are negative.

Softmax Activation

The softmax activation function is used as the activation function for the output layer, it's used for its compatibility with multi-classification model. The softmax function turns a vector of numbers into a vector of probabilities the sum of which is always one (Basta, 2020). It outputs a vector that represents a probability distribution of a list of potential outcomes.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad eq(13)$$

σ	Sigmoid
$\vec{z}$	Input vector
e^{z_i}	Standard exponential function for input vector
K	Number of classes in the multi-class classifier
e^{z_j}	Standard exponential function for output vector

Suppose an input vector x with size 3 is passed through the softmax activation and each element represents the hand drawn digit for the numbers $\{0, 1, 2\}$.

$x = \{2.0, 1.0, 0.1\}$ the output y for x input will be $y(x) = \{0.7, 0.2, 0.1\}$, getting the max element of that output shows that the number is predicted to be zero.

In the end the final equation for the any predicted output comes out to be this:-

$$\sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n w_n + b) \ eq(14)$$

Where w is the weight a is the activated input and b is the bias. (3Blue1Brown, 2017)

Methodology

The application uses python for its simplicity and consistency, it also provides access to great libraries and framework that are helpful for a machine learning project. Google colab is used for loading the dataset and training the model as it is a hosted Jupyter notebook service which provides a simple, streamlined environment, google colab is also faster for training deep models in comparison to a standard laptop. The keras API is used for loading and training the dataset as it is designed to quickly define deep learning models. Tkinter graphics library is used to create a graphical user interface for custom image inputs which are then evaluated by the model.

Explanation

Data Preparation

Using the keras API the CIFAR-10 dataset is loaded into a numpy array which has 4 different variables which are trainX, trainY, testX, and testY. The data is converted into type float32, after which it is normalised. The reason the data needs to be normalised is since the data consists of coloured images with RGB value that range from 0-255. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information (likebupt, et al., 2021).

Model Creation

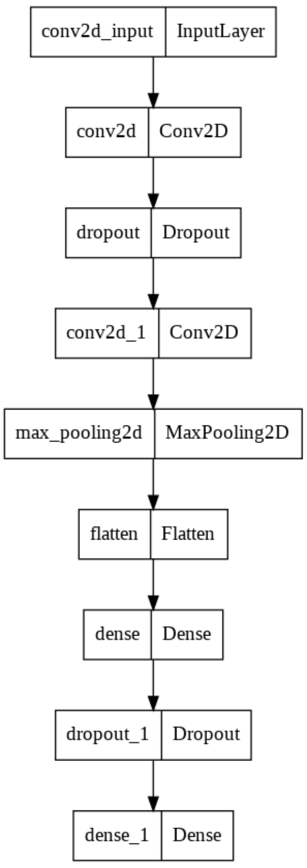


Figure 12 Layers visualization

The model uses the sequential method of creation as it allows to create a model layer-by-layer.

InputLayer	The first layer in the network is the input layer which takes in an input image of size 32x32
Conv2D	Conv2D which stands for 2-dimensional convolutional layer, the local receptive field or the filter used is 3x3 the output of this layer is going to be a feature map
Dropout	it is used to prevent overfitting. Overfitting is a modelling error which occurs while constructing an overly complex model. This layer drops out a random set of activations from the previous layer by setting them to zero.
MaxPooling2D	it has a pool size of 2x2, used to decrease the computational complexity of the network.
Flatten	Is used to convert the feature map into a 1-dimensional array
Dense	Is used to initialise a fully connected network.

This model is then saved as an .h5 file and loaded into a separate script using

```
from keras.models import load_model
```

An array of classes is made which has all the CIFAR-10 dataset classes, this is used to evaluate the right image type. Tkinter library is used GUI and before the image is evaluated it is resized to 32x32. The model then returns an array which contains a probability distribution of the predicted class. The argmax function is used to get the index of the largest number in the array and that index is used to display the image class type.

Results

The results of the testing varied from the type of images provided and the number of epochs the model was ran for.

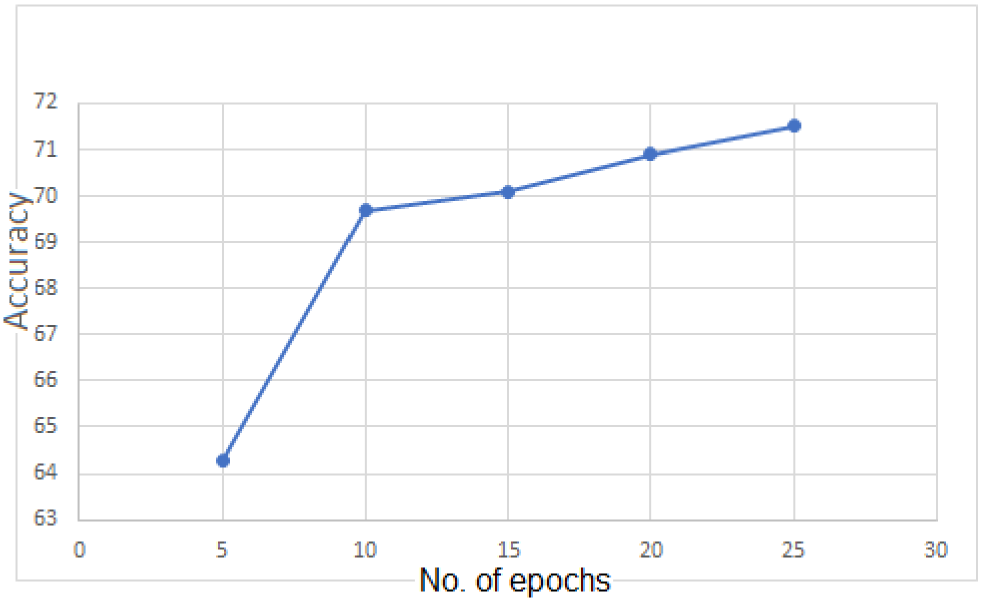


Figure 13 Accuracy and epoch graph

This graph shows the upward trend of accuracy when the number of epochs is increased.

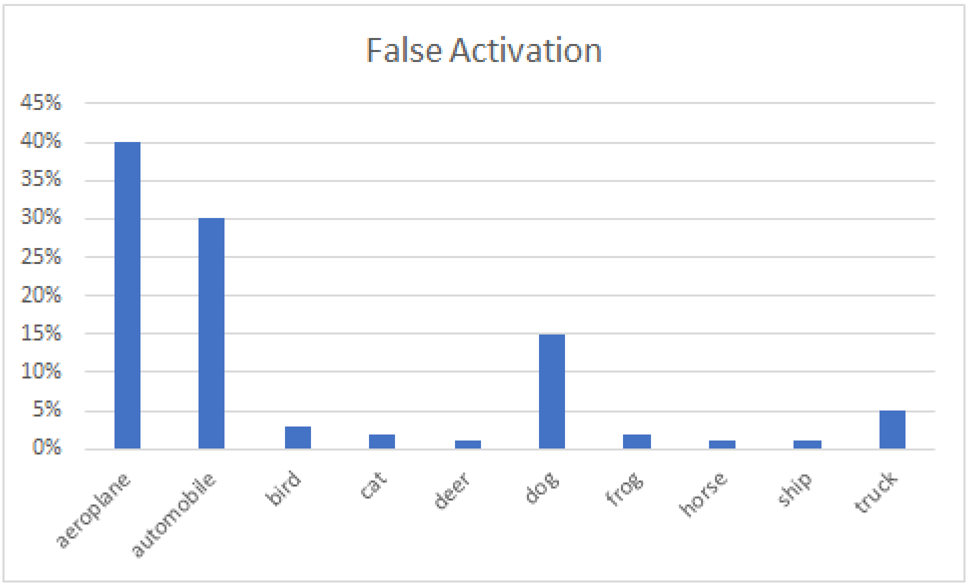


Figure 14 False activation graph

This graph shows which of classes were the most falsely activated over 5 different images in each class.

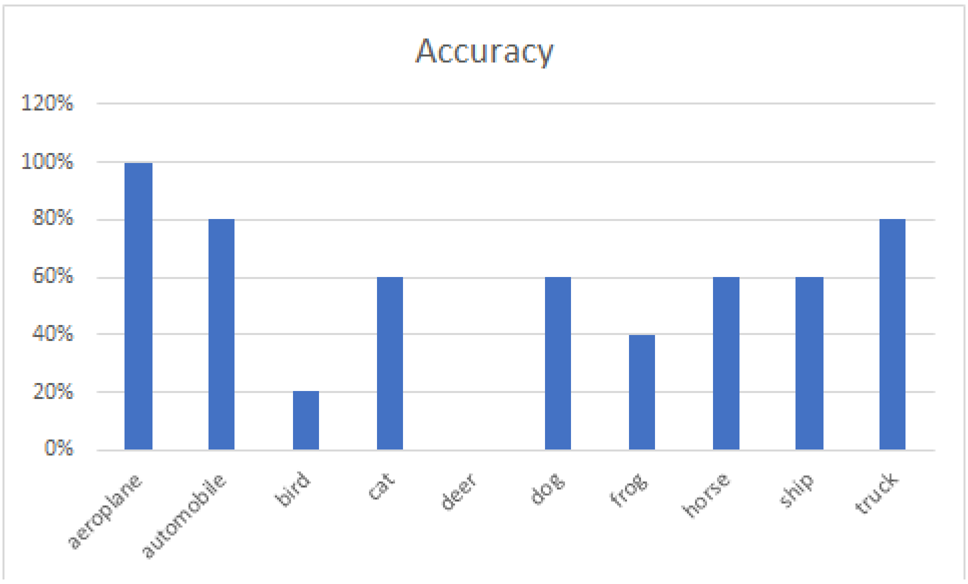


Figure 15 Class Accuracy

This graph shows the accuracy of the model of 5 different images for each class.

Discussion

As seen by the results above images of a deer were never correctly recognised even though it was falsely recognised once, after investigating the only reason that was found was the resizing of the images as the test images from the dataset were predicted correctly. Most of the deer images were falsely predicted as a dog, it might be due to the similar body structure of a dog and a deer. Aeroplane is the most falsely activated image and most of the bird images were falsely activated as aeroplane, this might be due to the similarity between the wings of a bird and an aeroplane. Another trend observed while testing was images with a white background were predicted more accurately.

Conclusion

Computer vision gives a machine the sense of sight, which allows it to perform tasks that are time consuming or just plain boring. Working on this project helped me understand how important computer vision can be for machines as it helps them do the things that we as humans are incapable of, or things that are too time consuming. Things like facial detection of an entire crowd of people, converting handwritten digits and alphabets into an electronic format.

Learning about convolutional neural networks and how they function was extremely fascinating as it provided a deep insight towards how computers use feature detection to differentiate between different animals and objects which is very similar to humans.

I really wanted to do this project in C++, as I am more familiar with the language and it also would have helped me in creating my own model and understand the backend working of a convolutional network, but due to time constraints I had to choose python. I was very reluctant at start as I hadn't used python before but after a while, I really enjoyed using python especially for machine learning, due to the abundance of good tutorials and libraries.

In the end, I am happy with the progress I made and the things I learned.

Note: The section about Gradient descent and softmax activation has been taken from the report I did for the first part of this module here's a link to that report https://github.com/utkarshyadav009/Digit-Recognition-CMP304-Project

References

3Blue1Brown, 2017. *But what is a neural network? | Chapter 1, Deep learning.* [Online]
Available at: https://www.youtube.com/watch?v=aircAruvnKk
[Accessed 22 March 2022].

Basta, N., 2020. *The Differences between Sigmoid and Softmax Activation Functions.* [Online]
Available at: https://medium.com/arteos-ai/the-differences-between-sigmoid-and-softmax-activation-function-12adee8cf322
[Accessed 27 March 2022].

Choi, J., 2016. *Causes of learning slowdown phenomenon.* [Online]
Available at: https://jaehyek.github.io/2016/12/07/Deep-Learning-Slowdown-train-speed/
[Accessed 16 May 2022].

Hansen, C., 2019. *Activation Functions Explained - GELU, SELU, ELU, ReLU and more.* [Online]
Available at: https://mlfromscratch.com/activation-functions-explained/#/
[Accessed 27 March 2022].

khosla, s., 2021. *CNN | Introduction to Pooling Layer.* [Online]
Available at: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/#:~:text=Max%20pooling%20is%20a%20pooling,of%20the%20previous%20feature%20map.
[Accessed 15 May 2022].

Krizhevsky, A., 2009. *The CIFAR-10 dataset.* [Online]
Available at: https://www.cs.toronto.edu/~kriz/cifar.html
[Accessed 12 May 2022].

Krizhevsky, A., Sutskever, I. & Hinton, G. E., 2012. *ImageNet Classification with Deep Convolutional.* [Online]
Available at:
https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
[Accessed 16 May 2022].

LeChun, Y., 1989. *Generalization and Network Design Strategies.* [Online]
Available at: http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf
[Accessed 16 May 2022].

LeChun, Y., Cortes, C. & Burges, C. J., 1998. *The MNIST DATABASE of handwritten digits.* [Online]
Available at: http://yann.lecun.com/exdb/mnist/
[Accessed 26 March 2022].

likebupt, PeterCLu & v-chmccl, 2021. *Normalize Data component.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/normalize-data
[Accessed 15 May 2022].

mach, 2016. *Softmax vs Sigmoid function in Logistic classifier?.* [Online]
Available at: https://stats.stackexchange.com/questions/233658/softmax-vs-sigmoid-function-in-logistic-classifier
[Accessed 28 March 2022].

Nielsen, M., 2019. *Neural Networks and Deep Learning.* [Online]
Available at: http://neuralnetworksanddeeplearning.com/chap1.html
[Accessed 26 March 2022].

Zeiler, M. D. & Fergus, R., 2013 . *Visualizing and Understanding Convolutional Networks.* [Online]
Available at: https://arxiv.org/pdf/1311.2901.pdf
[Accessed 15 May 2022].