Diabetic Retinography Analysis

Submitted by G-22:

Unnat Maaheshwari (B21072)
Alhad Chatur (B21090)
Rohit Yadav (B21120)
Saksham Panpaliya (B21126)
Dhruv Ranka (B21187)
Tanay Sonawane (B21231)

ABOUT THE PROBLEM

Problem Statement:

Given the data subset of images of both retinas of patients, based on the images and their features classify it on the 0-4 severity scale of Diabetic Retinopathy.

Brief Description:

Extract features from given images, and train a Keras model based on given images. Transform data to a common mode (colored or inverted). Evaluation is based on the correctness achieved by the model solely. Taking the existing models and their correctness as a measure of relative judging. No interactive applet or website is needed.

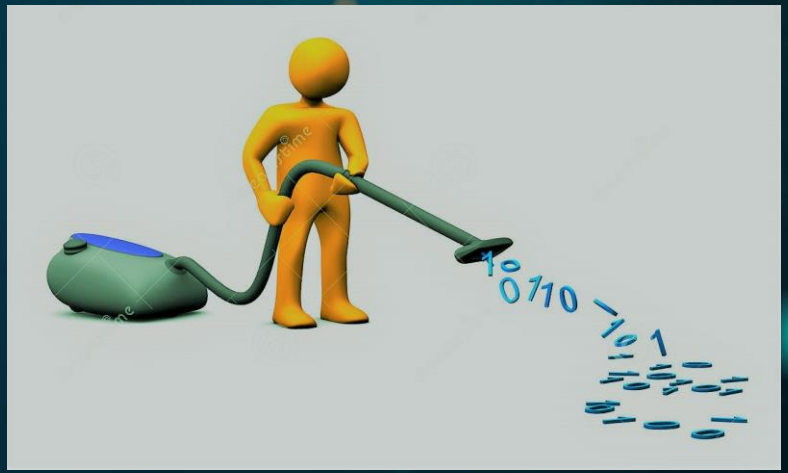
What is Diabetic Retinography Analysis?

- Diabetic Retinopathy is a complication of diabetes that affects the blood vessels of the retina. The growth of new blood vessels may lead to blindness.
- Therefore, to detect this disease, we use Diabetic Retinography Analysis. We will be using Image processing for the analysis.

DATA PREPROCESSING

DATA CLEANING

- Reading directly from the image data available and then checking for the corresponding data in the results CSV (removing null values).

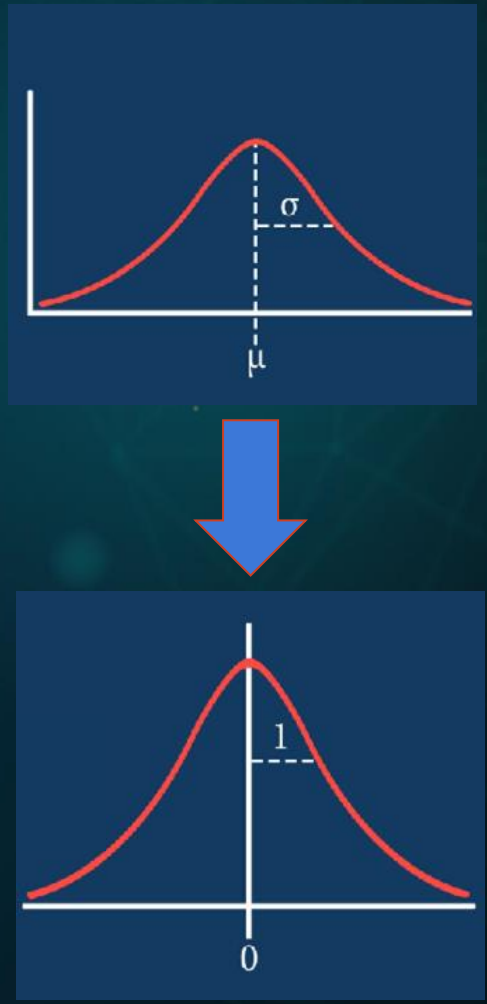


DATA REDUCTION - IMAGE PREPROCESSING

- We are resizing the image to a much lower resolution (128px*128px), also ensuring that the resulting image is square.
- It reduces the time of training of a neural network as the more the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model.
- Also we are using the linear interpolation method during resizing.

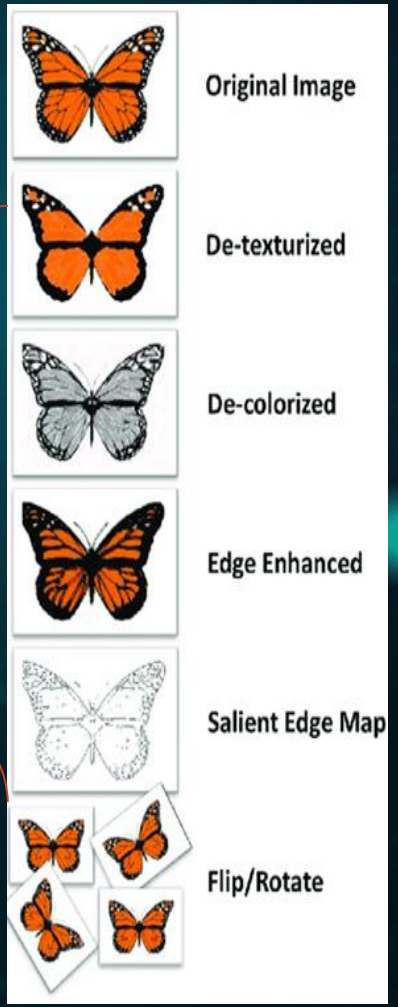
DATA NORMALIZATION

- Data normalization is the organization of data to appear similar across all records and fields to minimize **data space**.
- Here, we convert the pixel range of (0,256) to (0,1). This ensures that each input parameter has a similar data distribution.
- This makes convergence faster while training the network.
- Normalization is simply creating a standard format for all data(pixels) to be easily computed further.



DATA AUGMENTATION

- Data augmentation is a set of techniques to artificially increase the amount of data by generating new data points from existing data to increase diversity.
- This includes making small changes to data or using deep learning models to generate new data points.
- Here, we are doing image augmentation for generating new transformed versions of images from the given eye image dataset to increase its diversity and train our model more precisely.



METHODS!

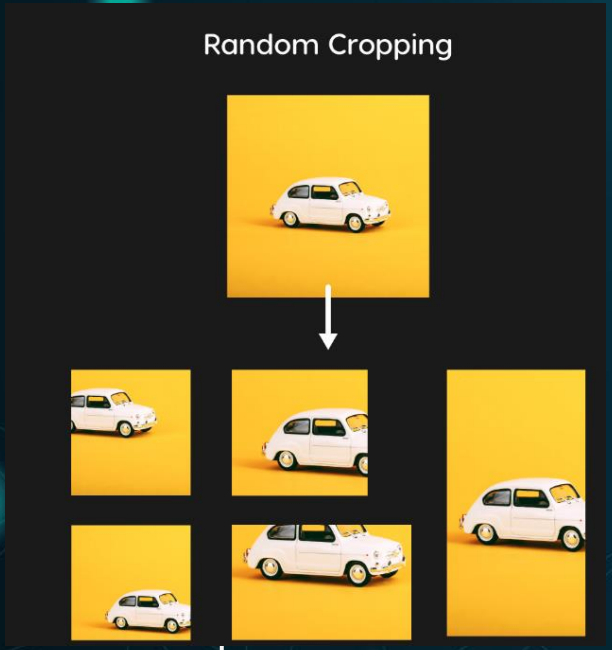


IMAGE MANIPULATION

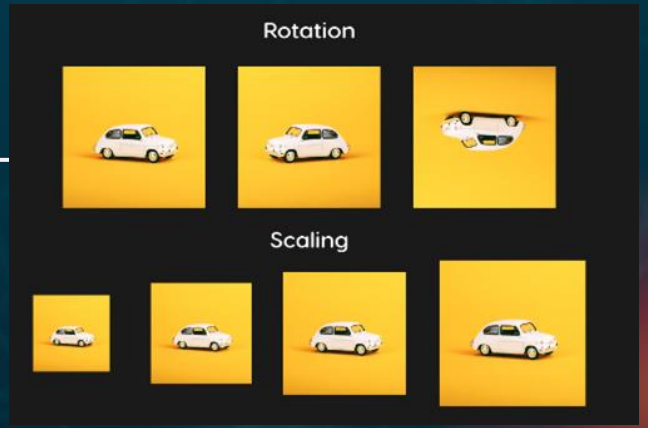
Saturn is composed of hydrogen and helium

IMAGE COLOR MANIPULATION

Mercury is the smallest planet in our Solar System

IMAGE POSITION MANIPULATION

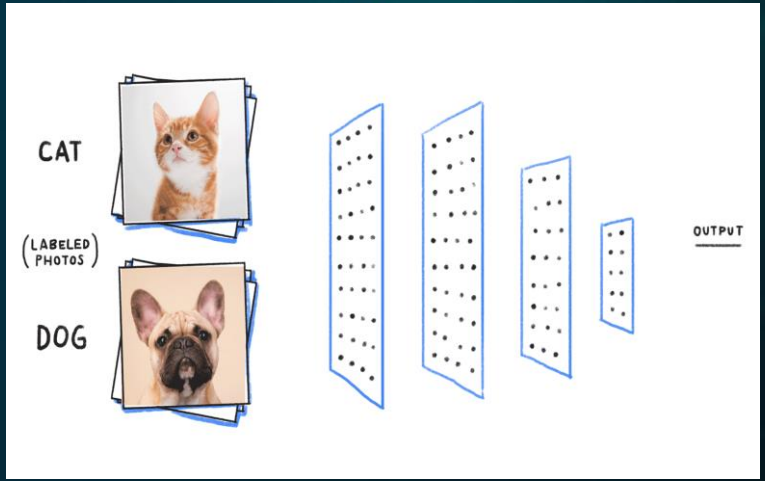
Venus has a beautiful name, but it’s terribly hot



CONVOLUTIONAL NEURAL NETWORK (CNN) MODEL

INTRODUCTION TO CNN

- A convolutional neural network (CNN or convnet) is a subset of machine learning.
- It is one of the various types of artificial neural networks which are used for different applications and data types.
- A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

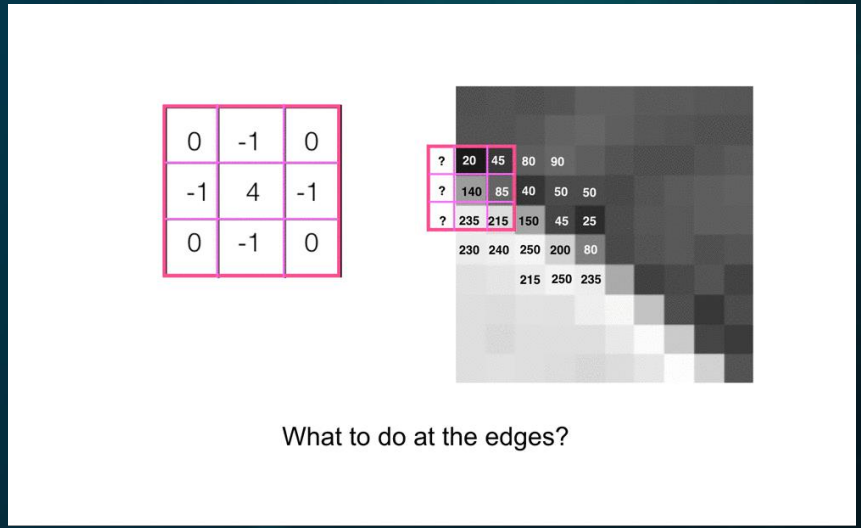


In [180]:

```
def createModel():
    model = Sequential()
    # first set of CONV => RELU => MAX POOL layers
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=inputShape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(NUM_CLASSES, activation='softmax'))
    # returns our fully constructed deep learning + Keras image classifier
    opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
    # use binary_crossentropy if there are two classes
    model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
    return model
```

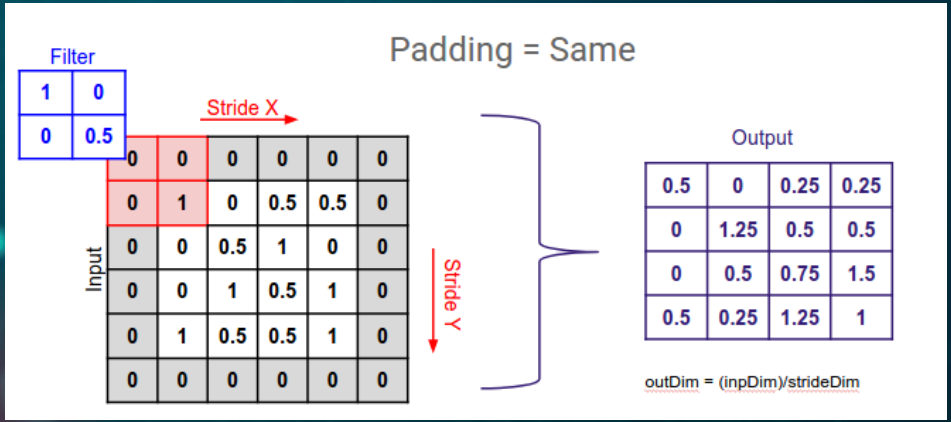
Filter

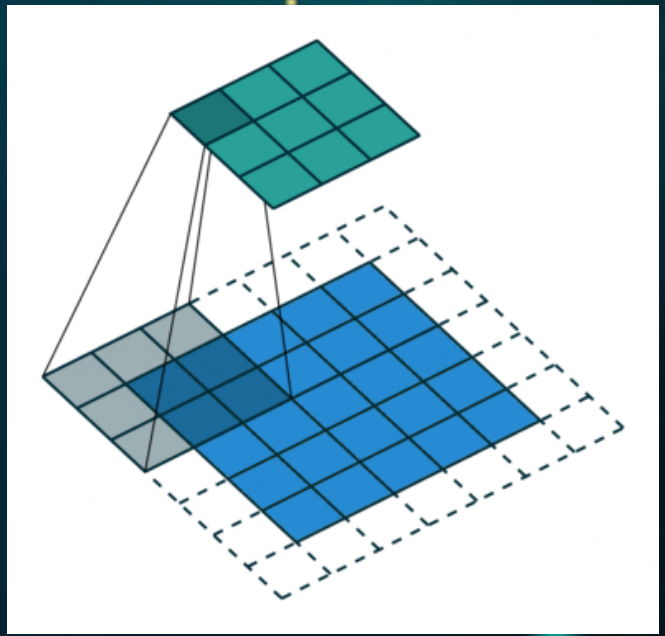
In Convolutional Neural Networks, Filters detect spatial patterns such as edges in an image by detecting the changes in intensity values of the image.



Padding

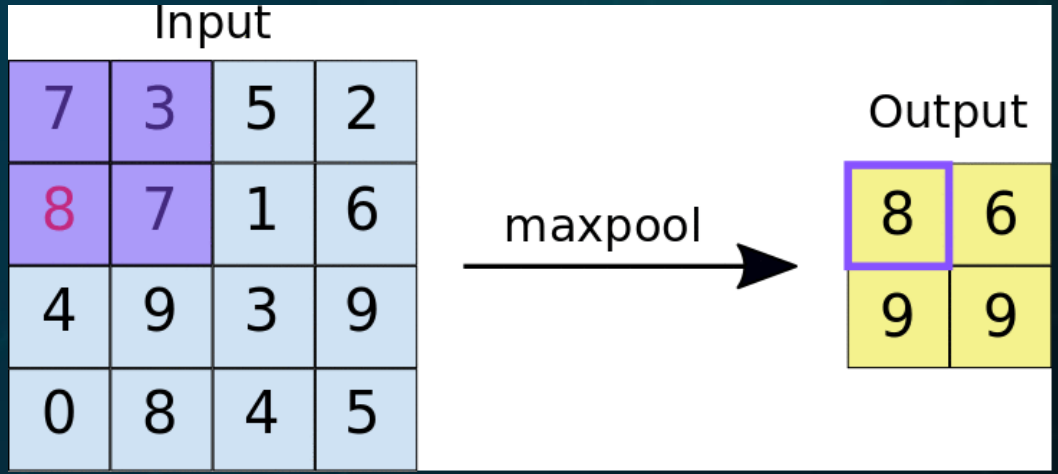
Padding is simply a process of adding layers of zeros to our input images so as to avoid this problem.





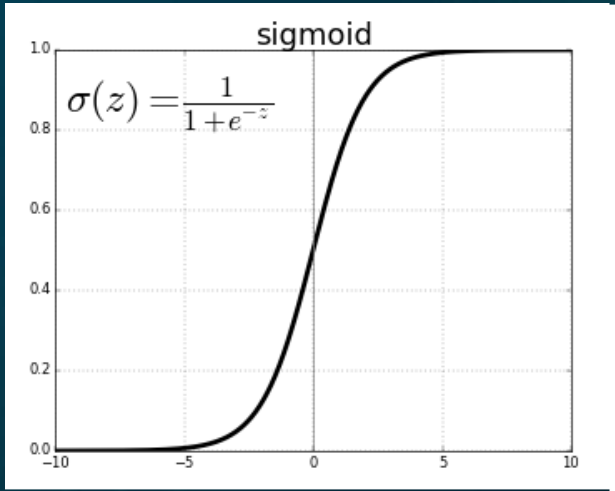
Max Pooling

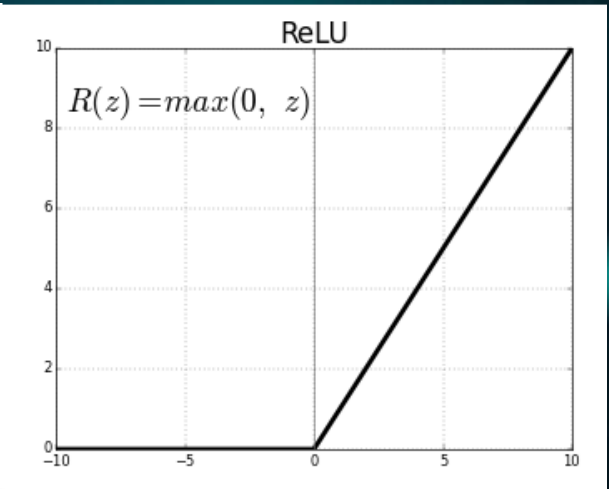
Max pooling is a type of operation that is typically added to CNNs following individual convolutional layers. When added to a model, max pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer



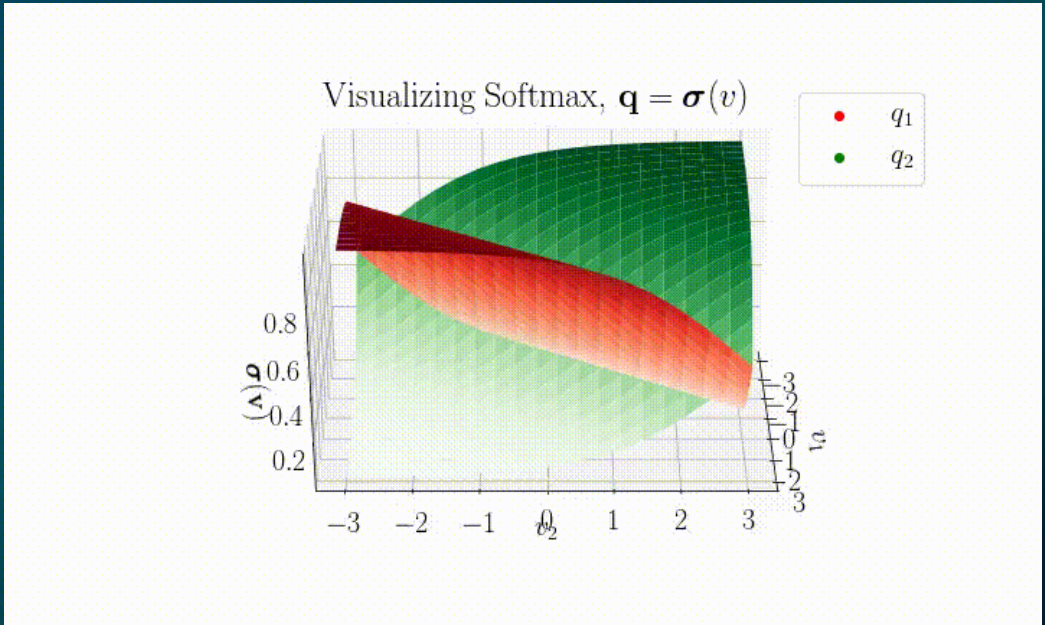
ReLU & Sigmoid

The model trained with ReLU converged quickly and thus takes much less time when compared to models trained on the Sigmoid function.



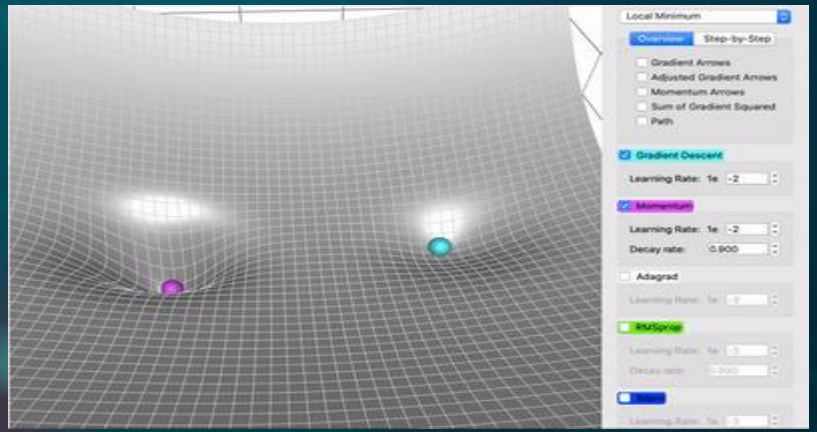


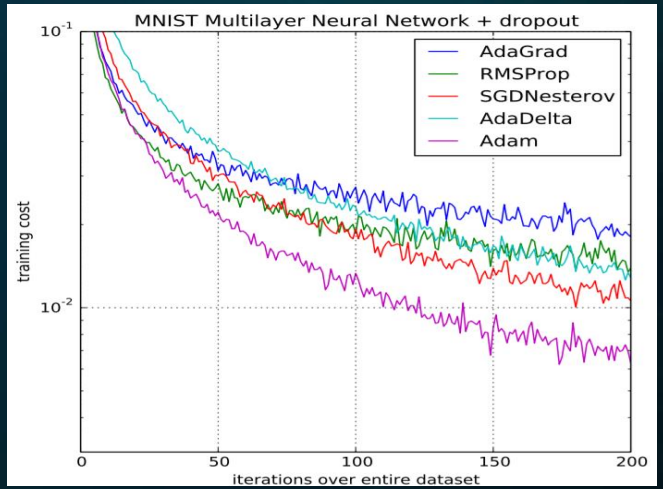
Softmax function



ADAM

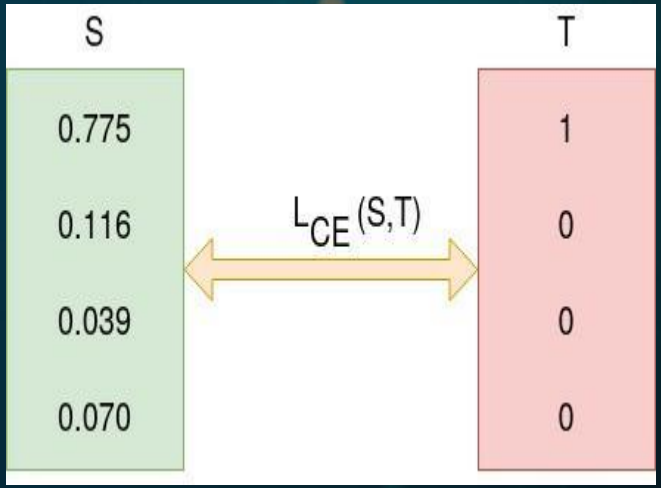
Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the ‘gradient descent with momentum’ algorithm and the ‘RMSP’ algorithm.

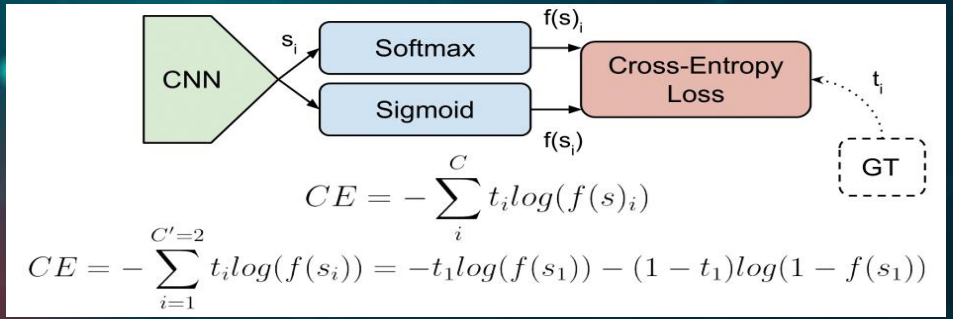




Categorical Loss Function

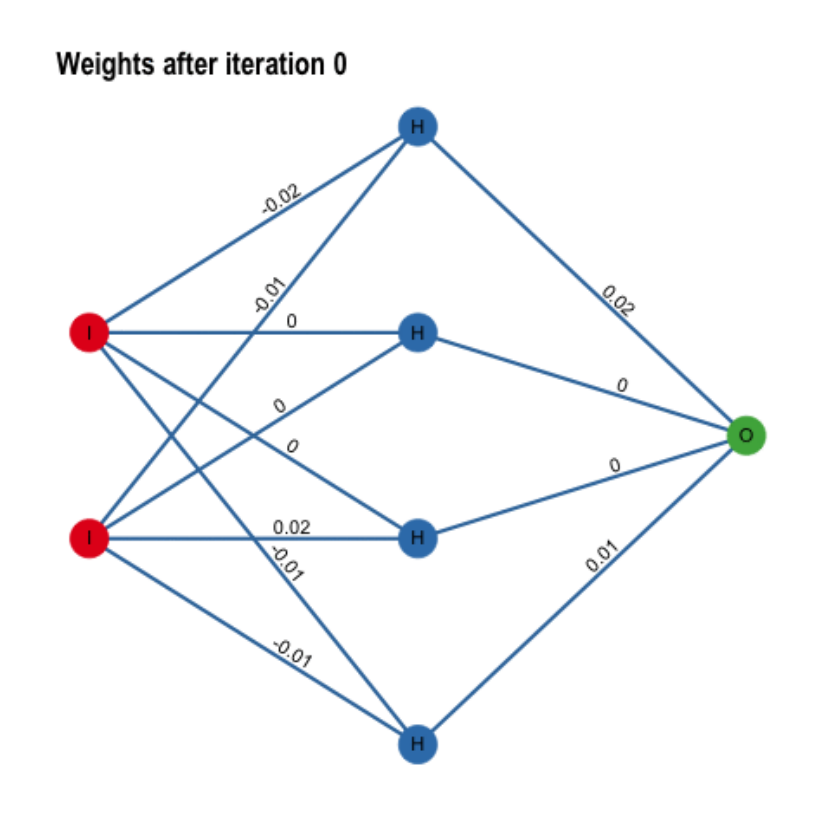
Also called logarithmic loss, log loss or logistic loss. Each predicted class probability is compared to the actual class desired output 0 or 1 and a score/loss is calculated that penalizes the probability.





Backpropagation

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.



RESULTS / OUTPUTS

RESULTS / OUTPUTS

```
6977
Epoch 9/15
49/49 [==============================] - 13s 269ms/step - loss: 0.8585 - accuracy: 0.7342 - val_loss: 0.9595 - val_accuracy: 0.
6977
Epoch 10/15
49/49 [==============================] - 13s 269ms/step - loss: 0.8605 - accuracy: 0.7354 - val_loss: 0.9591 - val_accuracy: 0.
6977
Epoch 11/15
49/49 [==============================] - 13s 271ms/step - loss: 0.8554 - accuracy: 0.7354 - val_loss: 0.9555 - val_accuracy: 0.
6977
Epoch 12/15
49/49 [==============================] - 13s 265ms/step - loss: 0.8501 - accuracy: 0.7380 - val_loss: 0.9527 - val_accuracy: 0.
6977
Epoch 13/15
49/49 [==============================] - 14s 278ms/step - loss: 0.8652 - accuracy: 0.7316 - val_loss: 0.9626 - val_accuracy: 0.
6977
Epoch 14/15
49/49 [==============================] - 14s 281ms/step - loss: 0.8696 - accuracy: 0.7329 - val_loss: 0.9767 - val_accuracy: 0.
6977
Epoch 15/15
49/49 [==============================] - 13s 272ms/step - loss: 0.8604 - accuracy: 0.7348 - val_loss: 0.9571 - val_accuracy: 0.
6977
Testing - :

17/17 [==============================] - 2s 91ms/step - loss: 0.9604 - accuracy: 0.6977
```

RESULTS / OUTPUTS

```
7380
Epoch 8/15
46/46 [==============================] - 20s 440ms/step - loss: 0.8870 - accuracy: 0.7187 - val_loss: 0.8792 - val_accuracy: 0.
7380
Epoch 9/15
46/46 [==============================] - 20s 440ms/step - loss: 0.8908 - accuracy: 0.7214 - val_loss: 0.8895 - val_accuracy: 0.
7380
Epoch 10/15
46/46 [==============================] - 20s 439ms/step - loss: 0.8769 - accuracy: 0.7248 - val_loss: 0.8920 - val_accuracy: 0.
7380
Epoch 11/15
46/46 [==============================] - 20s 439ms/step - loss: 0.8926 - accuracy: 0.7207 - val_loss: 0.9046 - val_accuracy: 0.
7380
Epoch 12/15
46/46 [==============================] - 20s 443ms/step - loss: 0.8946 - accuracy: 0.7193 - val_loss: 0.8712 - val_accuracy: 0.
7380
Epoch 13/15
46/46 [==============================] - 20s 439ms/step - loss: 0.8861 - accuracy: 0.7180 - val_loss: 0.8689 - val_accuracy: 0.
7380
Epoch 14/15
46/46 [==============================] - 21s 457ms/step - loss: 0.9014 - accuracy: 0.7187 - val_loss: 0.8698 - val_accuracy: 0.
7380
Epoch 15/15
46/46 [==============================] - 20s 442ms/step - loss: 0.8845 - accuracy: 0.7200 - val_loss: 0.8824 - val_accuracy: 0.
7380
testing network...
16/16 [==============================] - 2s 102ms/step - loss: 0.8860 - accuracy: 0.7380
Saving model to disk
```

RESULTS / OUTPUTS

```
Generating plots...
```