

## CONVOLUTIONAL NETWORKS

### LAB 10

#### CNN theory:

A Convolutional Neural Network is a class of deep, **feed-forward artificial neural networks**, most commonly applied to analysing visual imagery. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered.

Convolutional layers **apply a convolution operation to the input, passing the result to the next layer**. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images.

#### IMPLEMENTATION: BLOOD CELL IMAGE Dataset from Kaggle

##### About the dataset

This dataset contains **12,500 augmented images of blood cells (JPEG)** with accompanying cell type labels (CSV). There are approximately 3,000 images for each of 4 different cell types grouped into 4 different folders (according to cell type). The cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil. This dataset is accompanied by an additional dataset containing the original 410 images (pre-augmentation) as well as two additional subtype labels (WBC vs WBC) and also bounding boxes for each cell in each of these 410 images (JPEG + XML metadata). More specifically, the folder 'dataset-master' contains 410 images of blood cells with subtype

labels and bounding boxes (JPEG + XML), while the folder 'dataset2-master' contains 2,500 augmented images as well as 4 additional subtype labels (JPEG + CSV). There are approximately 3,000 augmented images for each class of the 4 classes as compared to 88, 33, 21, and 207 images of each in folder 'dataset-master'.

#### Working of CNN

1. The dataset was **imported from Kaggle** and **run on Colab kernels**. Then a 3 tier multilayer perceptron model was trained to perform as a CNN and fit on the data. The training was allowed to run for 200 epochs, after which it was fit on the test set to get the accuracy.
2. **Each neuron** in a neural network computes an output value by applying some function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is specified by a vector of **weights and a bias** (typically real numbers). Learning in neural network progresses by making incremental adjustments to the biases and weights. The vector of weights and the bias are called a filter and represent some feature of the input.
3. In CNN **many neurons share the same filter**. This reduces memory footprint because a single bias and a single vector of weights is used across all receptive fields sharing that filter, rather than each receptive field having its own bias and vector of weights.
4. **ReLU** is the abbreviation of Rectified Linear Units. This layer applies the non-

saturation activation function

$f(x) = \max(0, x)$ . It **increases the nonlinear properties of the decision function** and of the overall network without affecting the receptive fields of the convolution layer.

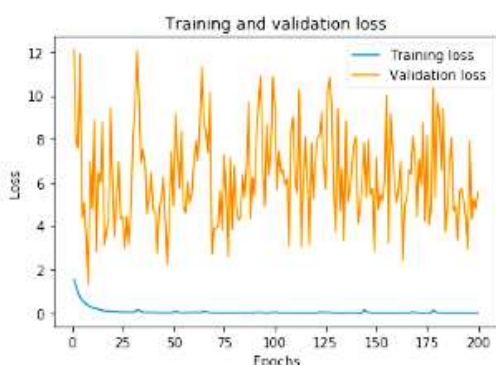
## Experiment

- Using **tqdm** library to see the status of the processing.
- Imported the images into the folder in the array format and resized and reshaped them into the fixed format and size.
- Used **Keras**, useful library for dealing with the images.
- **Categorised the images** into different categories and performed train test split on them. The input images are of the form (60,80,2).

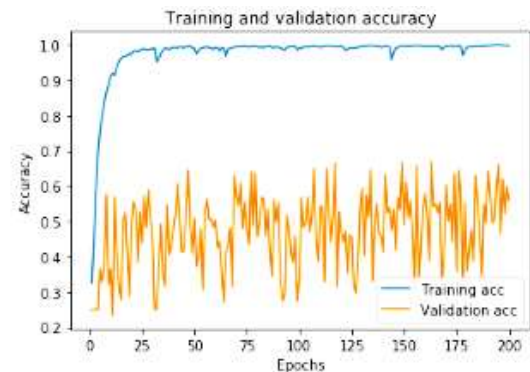
Next, we applied the convolution neural network and the activation function ReLU and padding so that the size of the images is increased before applying the activation function, and the batch normalization to find the desired features from it.

Further, to scale down the images, we have applies the **maxpolling**.

Dropped the learning rate at a percent of 0.5 for each convolution so that we **decrease the constraints gradually** and there is a chance that we can find the increase the accuracy with the learning rate. Further, we apply the linear cross entropy and Adam optimiser to it.



We noticed that running the model, which we have developed, and by running it 100 times, we can see that **in each epoch there is an increase in accuracy** and the final accuracy is about is 0.66948.



Here we can observe how the Validation accuracy is changing for each epoch.

On adding the weights to the images, like a **data augmentation technique**, we can get the maximum information from the data. We have done this for 100 epoch and we can see the improvement in accuracy. Finally, we can see that there is decrease in validation loss.

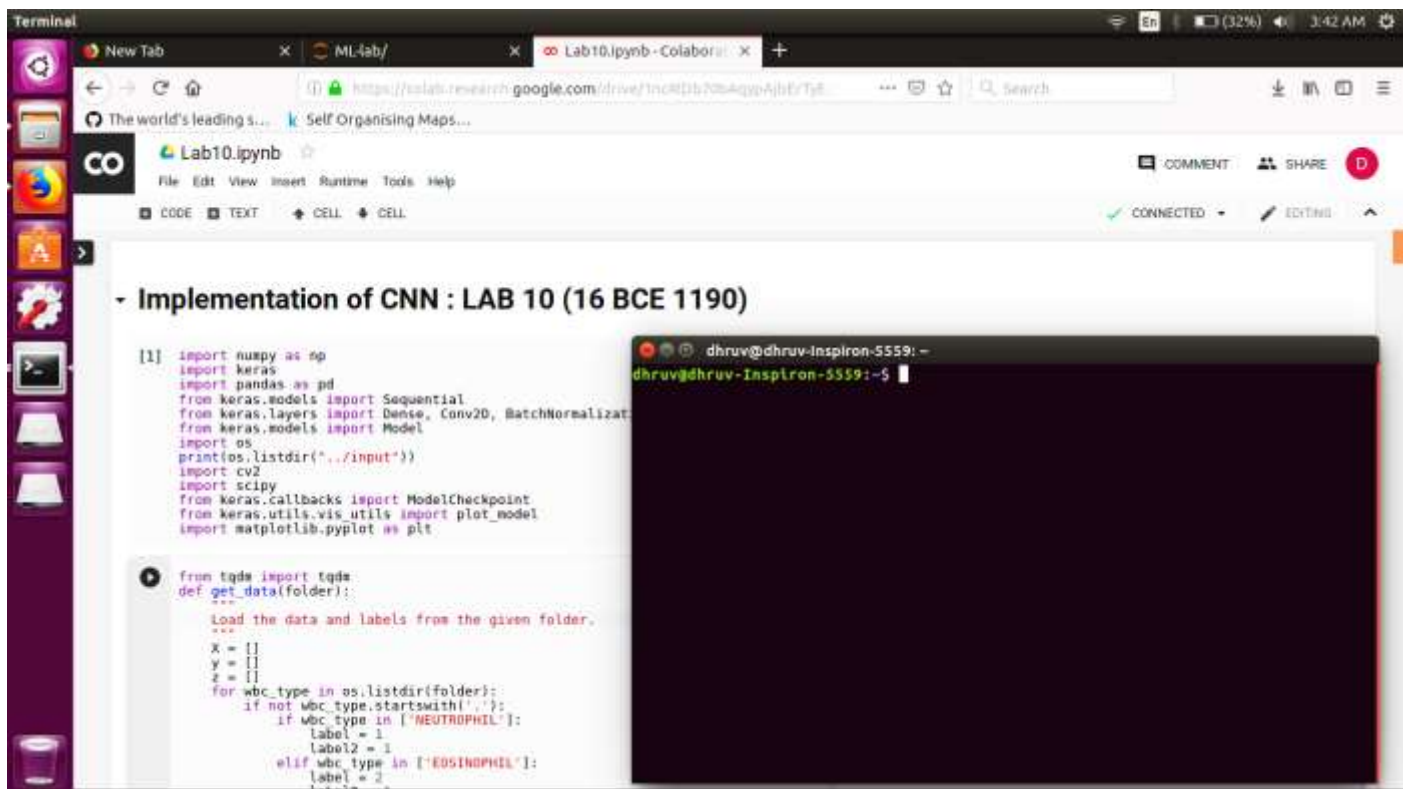
## Conclusion

The dataset was thus classified using a trained CNN model, using ReLU activation function and the Adam Optimiser. The model was allowed to train over the training set for 200 epochs, during which the accuracy was noted.

The model was then allowed to analyze the given test set, which yielded a **67% accuracy**. The final plot of Accuracy vs number of Epochs was visualized and it was seen that the accuracy rises exponentially after a few initial epochs, but then stabilizes.

## SCREENSHOTS

## Dataset implementation



```
[1] import numpy as np
import keras
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Conv2D, BatchNormalization
from keras.models import Model
import os
print(os.listdir("../input"))
import cv2
import scipy
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model
import matplotlib.pyplot as plt

from tqdm import tqdm
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []
    z = []
    for wbc_type in os.listdir(folder):
        if not wbc_type.startswith('.'):
            if wbc_type in ['NEUTROPHIL']:
                label = 1
            elif wbc_type in ['EOSINOPHIL']:
                label = 2
            else:
                label = 0
```