**SUMMER INTERNSHIP REPORT**

|  |  |
| --- | --- |
| Area of Online Internship | AI/Machine/Deep Learning |
| Intern Name | Anupam Baranwal |
| Name Of Institution | INDIAN INSTITUTE OF TECHNOLOGY, INDORE |
| Faculty Mentor Name | Prof. Vimal Bhatia |
| Duration | 2 Months (10/05/2021 TO 10/07/2021) |
| Date Of Submission | 01/07/2021 |

**Topics of Contents**

* Introduction on CNN
* CNN layers
* CNN model using GSTRB dataset
  + Dataset
  + Data Augmentation
  + Model
  + Training model
* Conclusion
* References

**Introduction on CNN –**

Machine learning has found a variety of applications in multiple fields and become very widespread in research. Among the different ML algorithms, deep learning is very commonly employed in these applications [1]. The continuing appearance of novel studies in the fields of deep and distributed learning is due to both the unpredictable growth in the ability to obtain data and the amazing progress made in the hardware technologies, e.g., High Performance Computing (HPC) [2]. Convolutional neural network (CNN) is one of the most popular and used of DL networks. The main advantage of CNN compared to its predecessors is that it automatically detects the significant features.

The main benefit of CNN is that it automatically identifies the relevant features without any human supervision [3]. The structure of CNNs was inspired by neurons in human and animal brains.

A commonly used type of CNN, which is similar to the multi-layer perceptron (MLP), consists of numerous convolution layers preceding sub-sampling (pooling) layers, while the ending layers are FC layers.

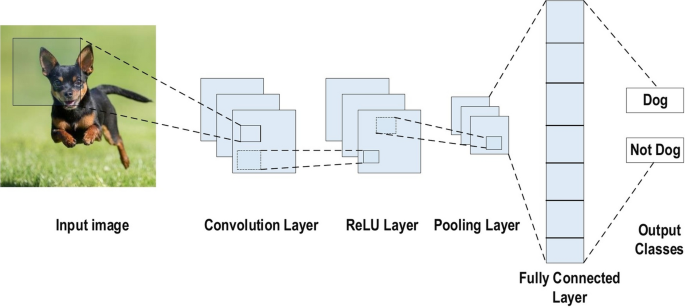


Fig – 1 Example of CNN architecture[4]

The input x of each layer in a CNN model is organized in three dimensions: height, width, and depth, or m×m×r, where the height (m) is equal to the width. The depth is also referred to as the channel number. For example, in an RGB image, the depth (r) is equal to three. Several kernels (filters) available in each convolutional layer are denoted by k and also have three dimensions (n×n×q), similar to the input image; here, however, n must be smaller than m, while q is either equal to or smaller than r. In addition, the kernels are the basis of the local connections, which share similar parameters (bias bk and weight Wk) for generating k feature maps hk with a size of (m−n−1) each and are convolved with input, as mentioned above [4]. The convolution layer calculates a dot product between its input and the weights as in Eq. 1, similar to NLP, but the inputs are undersized areas of the initial image size. Next, by applying the nonlinearity or an activation function to the convolution-layer output,

hk=f(Wk∗x+bk) (1)

The next step is down-sampling every feature map in the sub-sampling layers. This leads to a reduction in the network parameters, which accelerates the training process and in turn enables handling of the overfitting issue. For all feature maps, the pooling function is applied to an adjacent area of size p×p, where p is the kernel size [4]. Finally, the FC layers receive the mid- and low-level features and create the high-level abstraction, which represents the last-stage layers as in a typical neural network. The classification scores are generated using the ending layer. For a given instance, every score represents the probability of a specific class.

Benefits of employing CNNs -

1. The main reason to consider CNN is the weight sharing feature, which reduces the number of trainable network parameters and in turn helps the network to enhance generalization and to avoid overfitting.

2.Concurrently learning the feature extraction layers and the classification layer causes the model output to be both highly organized and highly reliant on the extracted features.

3.Large-scale network implementation is much easier with CNN than with other neural networks.

**CNN layers**

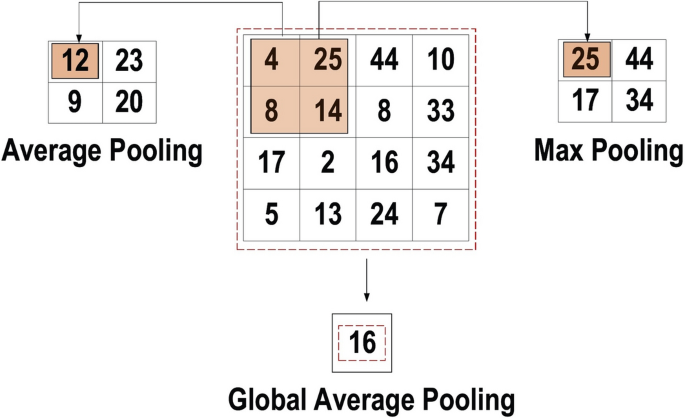
The CNN architecture consists of a number of layers.

1.Convolutional Layer: In CNN architecture, the most significant component is the convolutional layer. It consists of a collection of convolutional filters. The input image, expressed as N-dimensional metrics, is convolved with these filters to generate the output feature map.

Kernel definition: A grid of discrete numbers or values describes the kernel. Each value is called the kernel weight. Next, these weights are adjusted at each training era; thus, the kernel learns to extract significant features.

Convolutional Operation: The vector format is the input of the traditional neural network, while the multi-channelled image is the input of the CNN.

2.Pooling Layer: The main task of the pooling layer is the sub-sampling of the feature maps. This approach shrinks large-size feature maps to create smaller feature maps.



**Fig. 2 pooling operations.[4]**

3.Activation Function: Mapping the input to the output is the core function of all types of activation function in a neural network. The input value is determined by computing the weighted summation of the neuron input along with its bias. Non-linear activation layers are employed after all layers with weights in CNN architecture. Commonly used in CNN and other deep neural networks are -

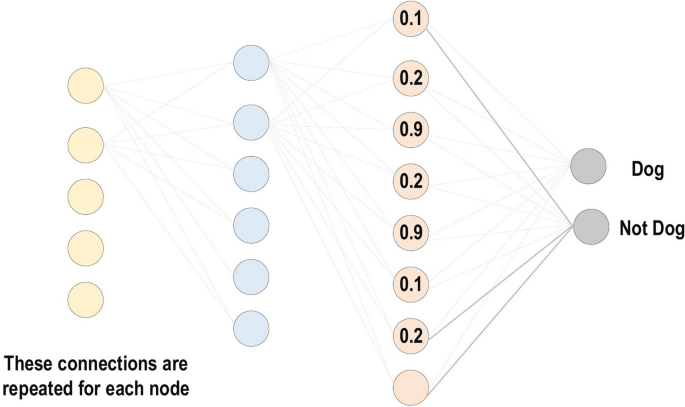
* Sigmoid: The input of this activation function is real numbers, while the output is restricted to between zero and one. The sigmoid function curve is S-shaped.

f(x)sigm=1/1+e−x (2)

* ReLU: The most commonly used function in the CNN context. It converts the whole values of the input to positive numbers. Lower computational load is the main benefit of ReLU over the others.

f(x)ReLU=max(0,x) (3)

4.Fully Connected Layer: Inside this layer, each neuron is connected to all neurons of the previous layer, the so-called Fully Connected (FC) approach. It is utilized as the CNN classifier. The input of the FC layer comes from the last pooling or convolutional layer.



**Fig. 3** - **Fully connected layer[4]**

5.Loss Functions: Loss functions are utilized in the output layer to calculate the predicted error created across the training. This error reveals the difference between the actual output and the predicted one. The following are some of the loss function types –

(a) Cross-Entropy or SoftMax Loss Function

(b)Euclidean Loss Function

(c)Hinge Loss Function

**Regularization to CNN**

For CNN models, over-fitting represents the central issue associated with obtaining well-behaved generalization. The model is entitled over-fitted in cases where the model executes especially well on training data and does not succeed on test data. An under-fitted model is the opposite; this case occurs when the model does not learn a sufficient amount from the training data. Various intuitive concepts are used to help the regularization to avoid over-fitting

1.Dropout: During each training epoch, neurons are randomly dropped. In doing this, the feature selection power is distributed equally across the whole group of neurons, as well as forcing the model to learn different independent features.

2.Data Augmentation: Training the model on a sizable amount of data is the easiest way to avoid over-fitting. To achieve this, data augmentation is used. Several techniques are utilized to artificially expand the size of the training dataset.

**Improving performance of CNN**

Based on our experiments in different DL applications [5]. It can be conclude the most active solutions that may improve the performance of CNN are:

* Expand the dataset with data augmentation or use transfer learning (explained in latter sections).
* Increase the training time.
* Increase the depth (or width) of the model.
* Add regularization.
* Increase hyperparameters tuning.

**CNN model using GSTRB dataset**

The Dataset

The GTSRB - German Trafﬁc Sign Recognition Benchmark dataset is used..

i. Structure

The GTSRB has the following properties:

* Single-image, multi-class classiﬁcation problem;
* 43 classes;
* More than 50,000 images in total;
* Reliable ground-truth data due to semi automatic annotation;
* Physical trafﬁc sign instances are unique within the dataset.

ID Trafﬁc Sign Description

0 Speed limit (20km/h)

1 Speed limit (30km/h)

2 Speed limit (50km/h)

3 Speed limit (60km/h)

4 Speed limit (70km/h)

5 Speed limit (80km/h)

6 End of speed limit (80km/h)

7 Speed limit (100km/h)

8 Speed limit (120km/h)

9 No passing

10 No passing for vehicles over 3.5 metric tons

11 Right-of-way at the next intersection

12 Priority road

13 Yield

14 Stop

15 No vehicles

16 Vehicles over 3.5 metric tons prohibited

17 No entry

18 General caution

19 Dangerous curve to the left

20 Dangerous curve to the right

21 Double curve

22 Bumpy Road

23 Slippery Road

24 Road narrows on the right

25 Road work

26 Trafﬁc signals

27 Pedestrians

28 Children crossing

29 Bicycles crossing

30 Beware of ice/snow

31 Wild animals crossing

32 End of all speed and passing limits

33 Turn right ahead

34 Turn left ahead

35 Ahead only

36 Go straight or right

37 Go straight or left

38 Keep right

39 Keep left

40 Roundabout mandatory

41 End of no passing

42 End of no passing by vehicles over 3.5 metric tons

**Data Augmentation**

The dataset is highly unbalanced.

Histogram.py-

import pandas as pd

import matplotlib.pyplot as plt

import os

dataset = pd.DataFrame(columns=['ClassID', 'Frequency'])

paths = os.listdir("gtsrb-german-traffic-sign/Train")

count = 0

for path in paths:

    dataset.loc[count] = [int(path), len(os.listdir("gtsrb-german-traffic-sign/Train"+"/"+path))]

    count+=1

dat = dataset.sort\_values(by=['ClassID'])

dat.plot(x='ClassID', y='Frequency', figsize=(12, 6),  kind='bar', legend=False)

plt.xticks(rotation=0)

plt.xlabel("ClassID")

plt.ylabel("Frequency")

plt.show()

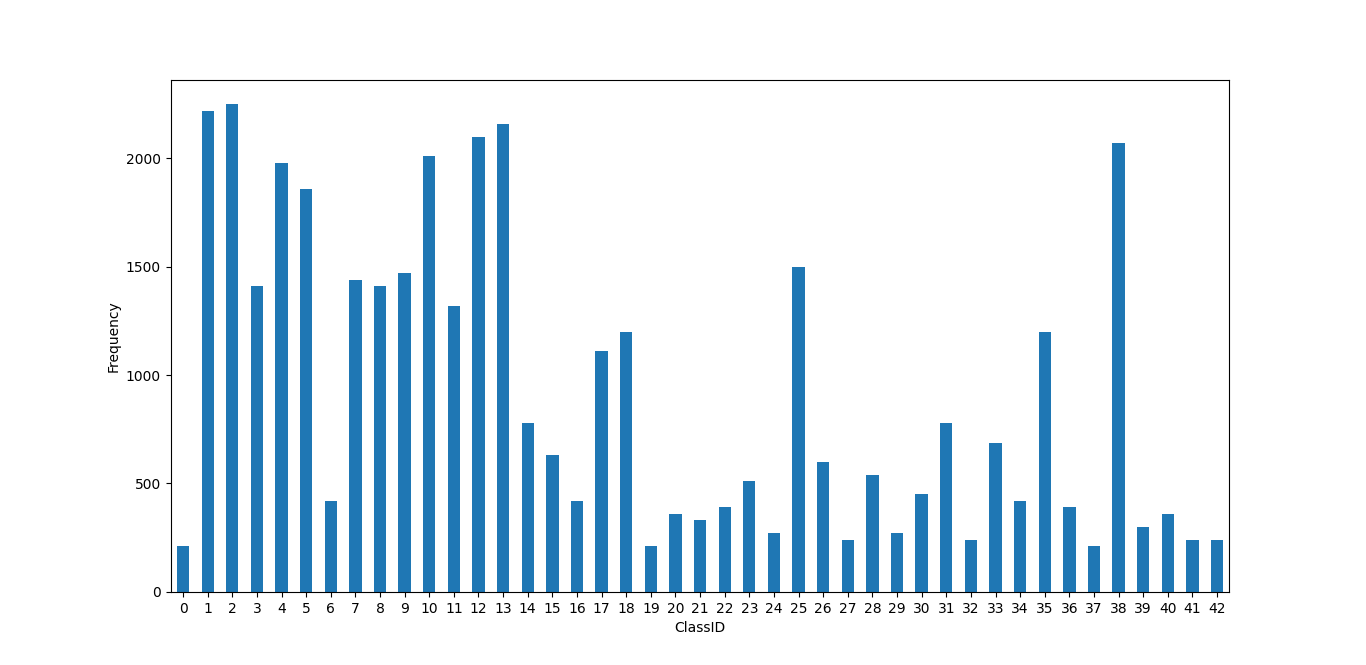


Fig 4 – Dataset Distribution

So, in order to obtain a better training, data augmentation is applied.

data\_augmentation.py

def add\_data(root, files):

    a = root.replace("gtsrb-german-traffic-sign/Train", '')

    os.makedirs("train\_augmented"+"/"+str(a[1:]))

    print("Adding samples to class "+ str(a[1:]))

    n = len(files)

    add\_size = 2500-n

    l=[]

    for i in range(len(files)):

        if(files[i][:2]!="GT"):

            img = cv2.imread(root+"/"+files[i])

            l.append(img)

    l2=[]

    while(len(l2)<=add\_size):

        rnd\_img = l[random.randint(0,len(l)-1)]

        rnd\_num = random.randint(1,5)

        img = None

        if(rnd\_num == 1):

            img = brightness(rnd\_img)

        elif(rnd\_num == 2):

            img = rotate\_image(rnd\_img)

        elif(rnd\_num == 3):

            img = projection\_transform(rnd\_img)

        elif(rnd\_num == 4):

            img = noise(rnd\_img)

        else:

            img = flip(rnd\_img, int(a[1:]))

        l2.append(rnd\_img)

    l\_tot = l + l2

    for j in range(len(l\_tot)):

        cv2.imwrite("train\_augmented"+"/"+str(a[1:])+"/"+str(j)+".png", l\_tot[j])

print("Starting augmentation\n")

m=0

for root, dirs, files in os.walk("gtsrb-german-traffic-sign/Train"):

    if(root != "gtsrb-german-traffic-sign/Train"):

        add\_data(root, files)

print("Completed")

**ﬂip:** This function, flips images of a class horizontally and/or vertically without changing their meaning.

Horizontal flip - [11, 12, 13, 15, 17, 18, 22, 26, 30, 35]

Vertical flip - [1, 5, 12, 15, 17]

Flip.py

import numpy as np

def flip(X, y):

    flip\_horizontally = np.array([11, 12, 13, 15, 17, 18, 22, 26, 30, 35])

    flip\_vertically = np.array([1, 5, 12, 15, 17])

    if(y in flip\_horizontally):

        X = np.fliplr(X)

    elif(y in flip\_vertically):

        X = np.flipud(X)

    return X

Fig 5 – (a) Original (b) Flip

**brightness**: apply a random changing brightness to the image

brightness.py

from skimage.exposure import adjust\_gamma

import random

def brightness(X, intensity=0.5):

    delta = 1. \* intensity

    X = adjust\_gamma(X, random.uniform(1 - delta, 1 + delta))

    return X

Fig 6 - (a) Original (b) changed brightness

**rotate\_image**: apply a random rotation to the image

rotate\_image.py

from skimage.transform import rotate

import random

def rotate\_image(X, intensity=0.5):

    delta = 30. \* intensity

    X = rotate(X, random.uniform(-delta, delta))

    return X

Fig 7 – (a) Original (b) Rotated

**projection\_transform**: apply a random perspective transformation of the signal

projection\_transform.py

import numpy as np

import random

from skimage.transform import ProjectiveTransform

from skimage.transform import warp

def projection\_transform(X, intensity=0.5):

    image\_size = X.shape[1]

    d = image\_size \* 0.3 \* intensity

    tl\_top = random.uniform(-d, d)

    tl\_left = random.uniform(-d, d)

    bl\_bottom = random.uniform(-d, d)

    bl\_left = random.uniform(-d, d)

    tr\_top = random.uniform(-d, d)

    tr\_right = random.uniform(-d, d)

    br\_bottom = random.uniform(-d, d)

    br\_right = random.uniform(-d, d)

    transform = ProjectiveTransform()

    transform.estimate(np.array((

                (tl\_left, tl\_top),

                (bl\_left, image\_size - bl\_bottom),

                (image\_size - br\_right, image\_size - br\_bottom),

                (image\_size - tr\_right, tr\_top)

            )), np.array((

                (0, 0),

                (0, image\_size),

                (image\_size, image\_size),

                (image\_size, 0)

            )))

    X = warp(X, transform, output\_shape=(image\_size, image\_size), order = 1, mode = 'edge')

    return X

Fig 8 – (a) Original (b) Transformed

**noise:** applies gaussian noise to the image. Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features.

Noise.py

import cv2

import numpy as np

from skimage.util import random\_noise

def noise(X):

    X = random\_noise(X, mode='gaussian',mean=0.0)

    return X

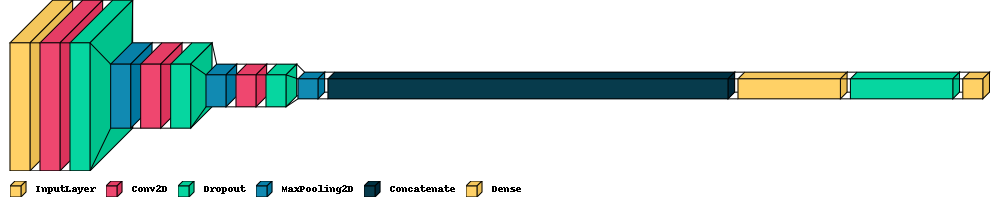
 

Fig 9 – (a) Original (b) Noise added

**The model**

The model is trained with the augmented and balanced dataset, setting the number of epochs equals to 100. The model reaches very fast a very high accuracy for both train and validation, with 93.71 accuracy.

To avoid to re-train the model at every execution of the attacks, and even to charge every time the whole trained model, once the model is obtained its weights are saved in a .hdf5 file format.



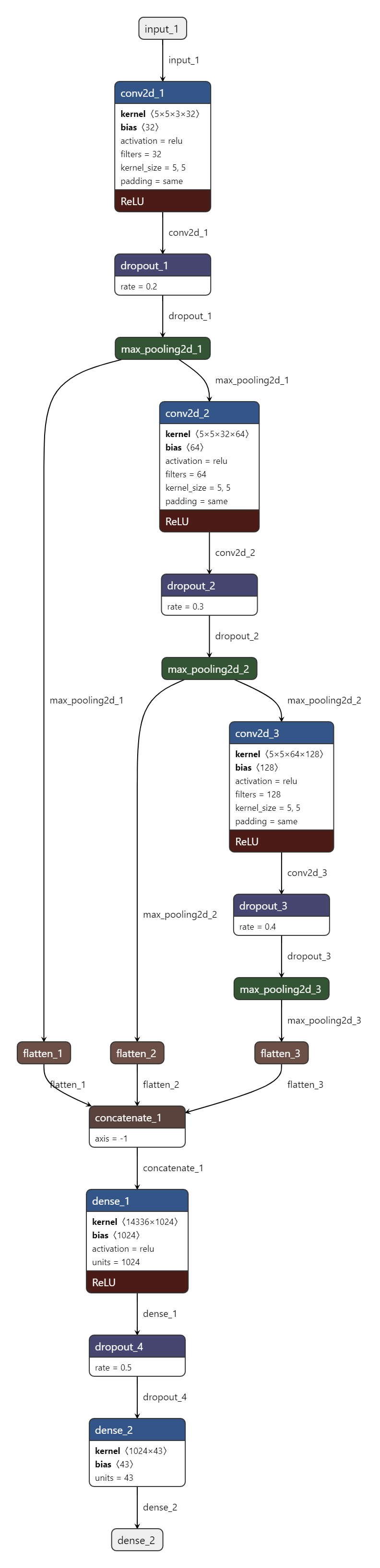
**Fig 10 – Layers of model**

Input layer: Built an input layer to avoid having uniform images to give as input for our model. In order to do this, I passed to the model an (32, 32, 3) shape.

Coevolutionary layers: Used 2 convolutional layers, using 32, 64 ﬁlters, respectively, with a kernel size of 5x5. These layers create a convolution kernel that is convolved with the layer input to produce a tensor of outputs. Each convolutional layer is followed by a ReLU activation function.

Dropout layers: This layer randomly dropping out (setting to zero) a number of output features of the layer during training. This is very useful to prevent overﬁtting.

Pooling layers: These layers help the model to remove un-useful information, this is needed because, very often, neighbouring elements contain very similar information.



**Fig 11 – Architecture of model**

train.py

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

import pandas as pd

import matplotlib.pyplot as plt

from call\_model import \*

train\_datagen = ImageDataGenerator(rescale=1./255,

    validation\_split=0.3)

train\_generator = train\_datagen.flow\_from\_directory(

    "train\_augmented",

    target\_size=(32, 32),

    batch\_size=32,

    class\_mode='categorical',

    subset='training',

    shuffle=True,

    seed=42)

validation\_generator = train\_datagen.flow\_from\_directory(

    "train\_augmented",

    target\_size=(32, 32),

    batch\_size=32,

    class\_mode='categorical',

    subset='validation',

    shuffle=True,

    seed=42)

tmp = pd.DataFrame(columns=['ClassId', 'ModelId', 'SignName'])

csv\_data = pd.read\_csv("sign\_name.csv")

for i, item in csv\_data.iterrows():

    tmp.loc[i] = [item['ClassId'], train\_generator.class\_indices[str(item['ClassId'])], item['SignName']]

tmp.to\_csv("sign\_name.csv", sep=',', index = False)

model = build\_cnn((32, 32, 3))

steps\_per\_epoch=train\_generator.n//train\_generator.batch\_size

val\_steps=validation\_generator.n//validation\_generator.batch\_size+1

modelCheckpoint = ModelCheckpoint("models/weights\_cnn.hdf5", monitor='val\_loss', verbose=1, save\_best\_only=True, save\_weights\_only=False, mode='auto', period=1)

earlyStop = EarlyStopping(monitor='val\_loss', min\_delta=0.001, patience=6, verbose=0, mode='auto')

callbacks\_list = [modelCheckpoint, earlyStop]

history = model.fit\_generator(

    train\_generator,

    workers=6,

    epochs=100,

    verbose=1,

    steps\_per\_epoch=steps\_per\_epoch,

    validation\_steps=val\_steps,

    validation\_data=validation\_generator,

    callbacks=callbacks\_list,

    shuffle=True)

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

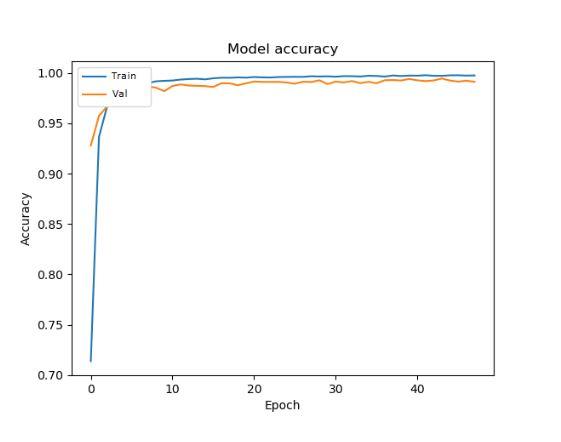
plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

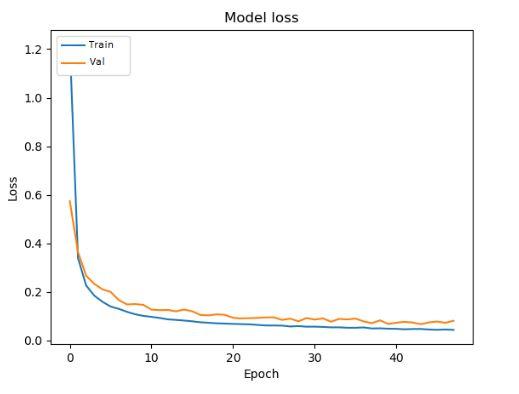
plt.show()

**Testing the model**

Modal is tested against a test dataset . To execute it, an image data generator is used to process the images and pass them to the model for prediction.



**Fig 12 – Accuracy plot**



**Fig 13 – Loss Plot**

test.py

# import tensorflow as tf

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow import keras

import visualkeras

from call\_model import \*

model = load\_model\_weights("models/weights\_cnn.hdf5")

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = ImageDataGenerator(rescale=1./255).flow\_from\_directory(

    "gtsrb-german-traffic-sign/Testing",

    target\_size=(32, 32),

    batch\_size=32,

    class\_mode=None,

    shuffle=False)

val\_steps = test\_generator.n//test\_generator.batch\_size+1

preds = model.predict\_generator(test\_generator, verbose=1, steps=val\_steps)

Ypred = np.argmax(preds, axis=1)

dataframe = pd.read\_csv("sign\_name.csv")

l = []

for i in range(len(Ypred)):

    l.append(int(dataframe['ClassId'].loc[dataframe['ModelId'] == Ypred[i]]))

result = np.array(l)

test = pd.read\_csv("gtsrb-german-traffic-sign/Test.csv")

count\_err = 0

for i, item in test.iterrows():

    if (item['ClassId'] != result[i]):

        count\_err += 1

accuracy = round((float(count\_err) / float(len(Ypred))) \* 100, 2)

print("Accuracy "+str(accuracy)+"%")

keras.utils.plot\_model(

    model,

    to\_file="model.png",

    show\_shapes=True,

    show\_dtype=False,

    show\_layer\_names=True,

    rankdir="TB",

    expand\_nested=True

)

visualkeras.layered\_view(model, type\_ignore=[keras.layers.Flatten], legend=True,to\_file='architecture.png')

**Fast Gradient attack**

The fast gradient attack works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximises the loss. This new image is called the adversarial image.

Untarget attack

This method computes an adversarial image by adding a pixel-wide perturbation of magnitude in the direction of the gradient. This perturbation is computed with a single step, thus is very efﬁcient in terms of computation time:

xadv = x + ε sign(∇ J(θ, x, ytrue)

Target attack

In targeted attacks the attacker pretends to get the image classiﬁed as a speciﬁc target class, which is different from the true class. In this case in the direction of the negative gradient with respect to the target class:

xadv = x - ε sign(∇ J(θ, x, ytrue)

where:

- xadv : Adversarial image

- x: Original image

- **ε:** Small multiplier

- y: Original classiﬁcation

- θ: Model parameters

- J: Classiﬁcation loss function

tested all kinds of attacks using the (Speed limit (30km/h)) class.

Results:

In-distribution attack

– Fast gradient attack



Targeted: 26.32%



Untargeted: 36.84%

– Iterative attack



Targeted: 68.42%



Untargeted: 63.16%

Logo attack

– Fast gradient attack



Targeted: 50.0%

– Iterative attack



Targeted: 75.0%

Custom Signs attack

– Fast gradient attack



Targeted: 42.86%

– Iterative attack



Targeted: 42.86%

**Conclusion –**

In this paper I have discussed about convolutional neural network (CNN) and different layers used to implement CNN. Also, I implemented a model using GSTRB dataset. Model can predict the class of sign board with accuracy of 93.71%.

**References -**

[1] Liu W, Wang Z, Liu X, Zeng N, Liu Y, Alsaadi FE. A survey of deep neural network architectures and their applications. Neurocomputing.

[2] Potok TE, Schuman C, Young S, Patton R, Spedalieri F, Liu J, Yao KT, Rose G, Chakma G. A study of complex deep learning networks on high-performance, neuromorphic, and quantum computers. ACM J Emerg Technol Comput Syst (JETC).

[3] Gu J, Wang Z, Kuen J, Ma L, Shahroudy A, Shuai B, Liu T, Wang X, Wang G, Cai J, et al. “Recent advances in convolutional neural networks. Pattern Recogn”.

# [4] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie & Laith Farhan, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”

# [5] Alzubaidi L, Fadhel MA, Al-Shamma O, Zhang J, Duan Y. Deep learning models for classification of red blood cells in microscopy images to aid in sickle cell anemia diagnosis. Electronics.

# [6] Chawin Sitawarin,Arjun Nitin Bhagoji,Arsalan Mosenia, Mung Chiang,Prateek Mittal “DARTS: Deceiving Autonomous Cars with Toxic Signs”

# [7] Chawin Sitawarin,Arjun Nitin Bhagoji,Arsalan Mosenia, Mung Chiang,Prateek Mittal, Rogue Signs: Deceiving Trafﬁc Sign Recognition with Malicious Ads and Logos