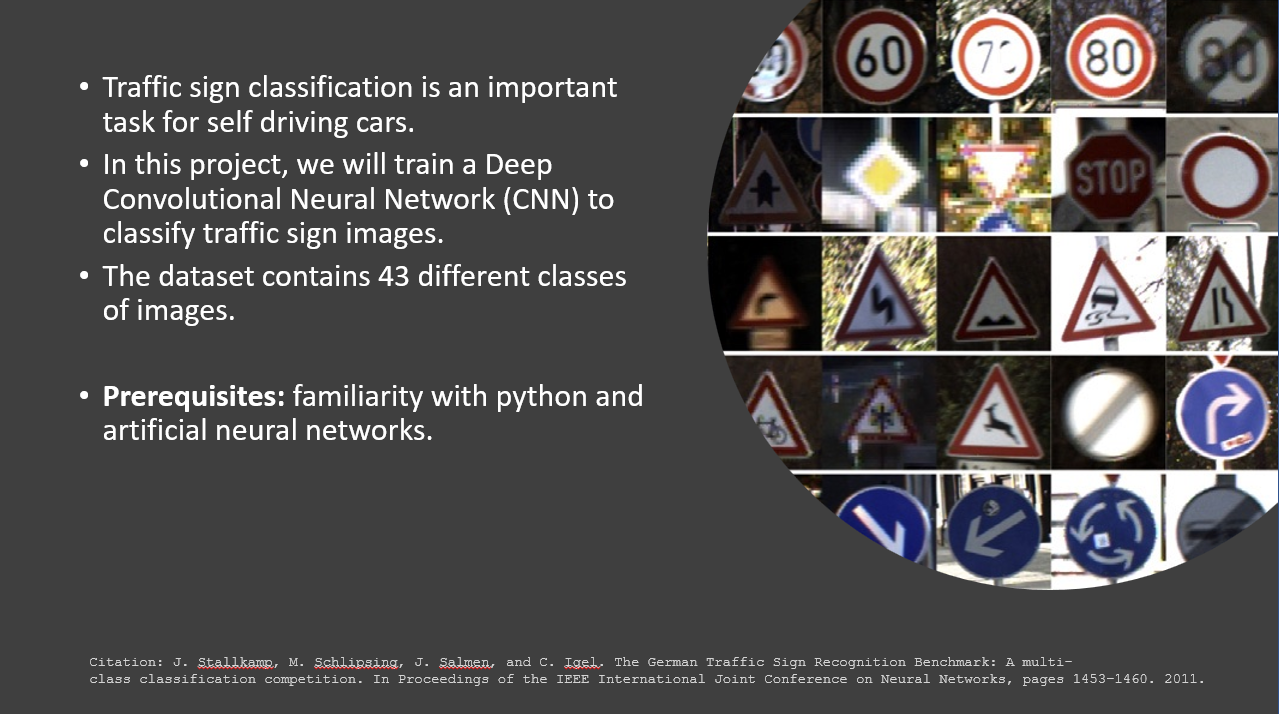
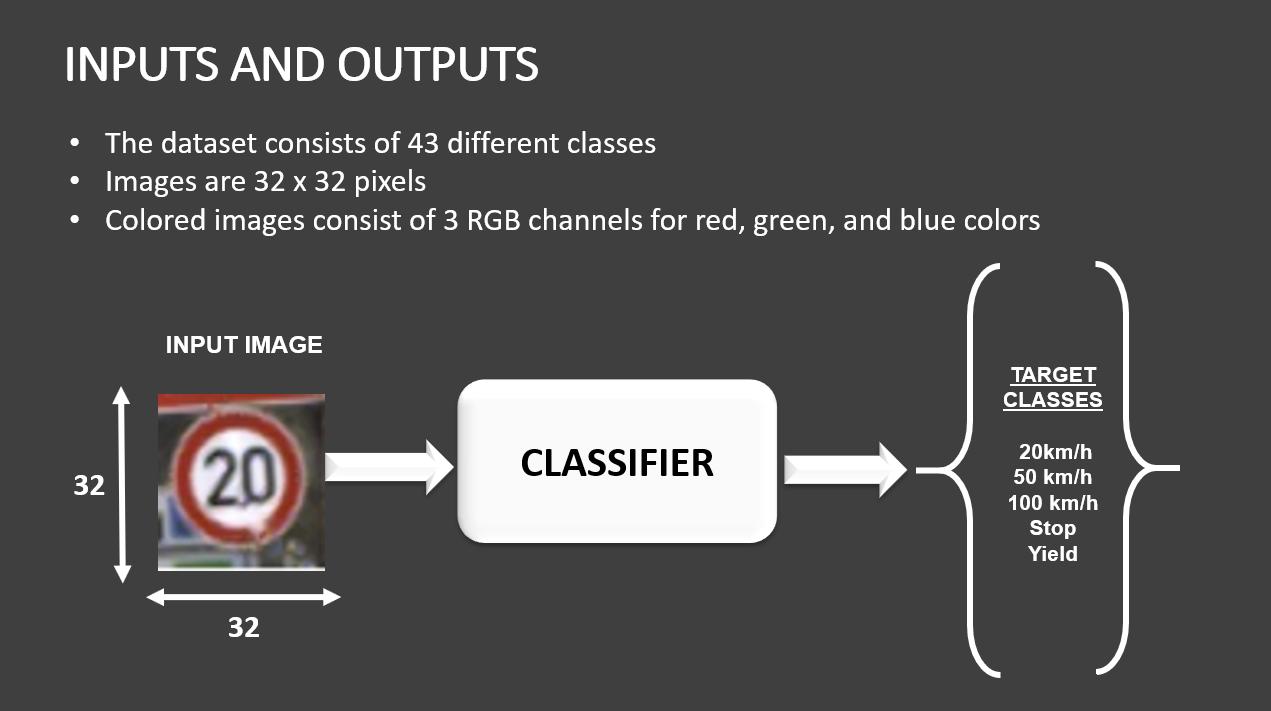
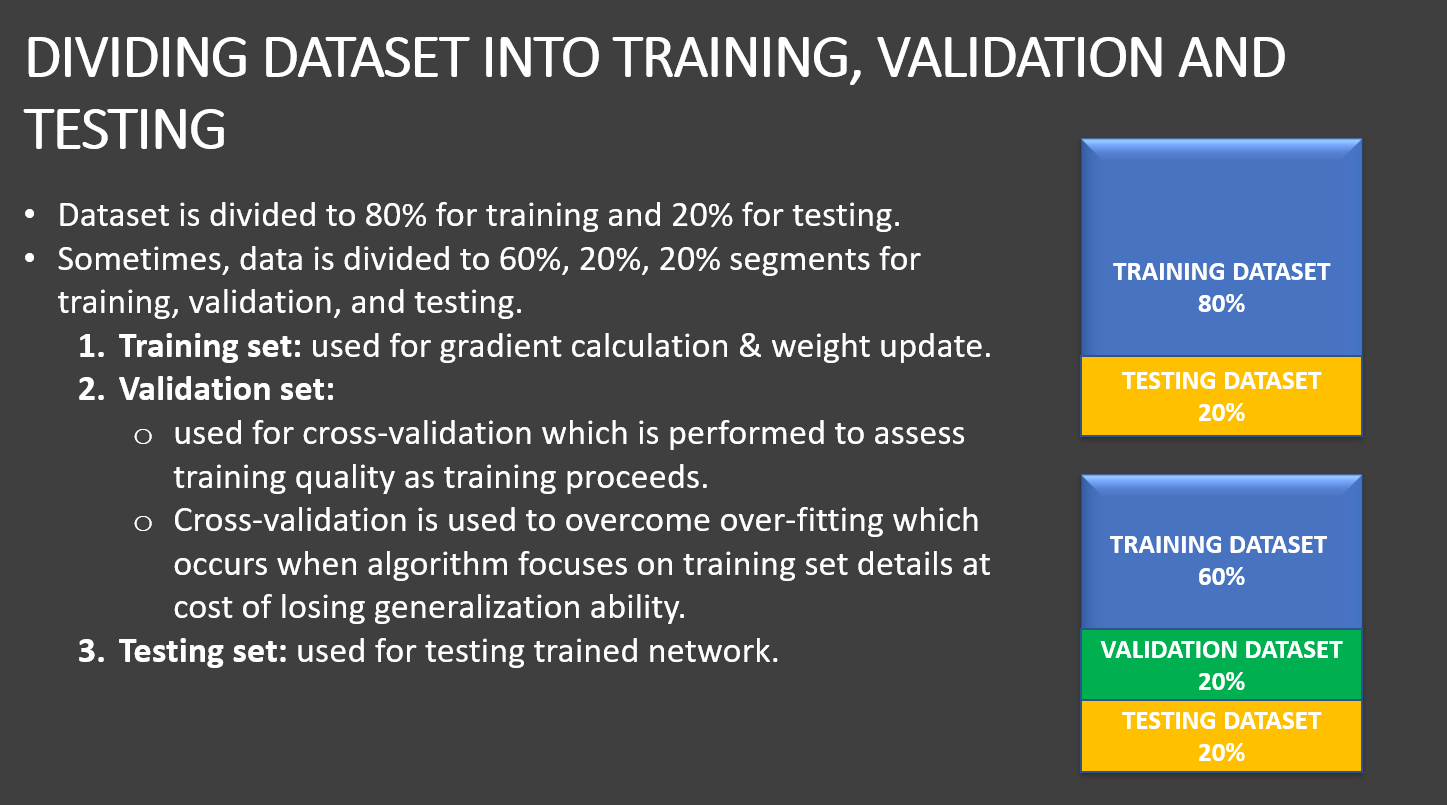
1. **Project Overview**





1. **Importing Dataset**



**X Label of Training DataSet**

The shape of your dataset is (34799, 32, 32, 3).

This means that you have 34,799 samples (instances) in your dataset, and each sample has a shape of 32x32 pixels with 3 color channels (RGB).

In other words, each instance in your dataset is an image that is 32 pixels wide, 32 pixels tall, and has 3 color channels (red, green, and blue). This is a common shape for image datasets, especially those used for computer vision tasks such as image classification or object detection.

### Y Label of Training DataSet:

The shape of the y training dataset, (34799,), indicates that it is a 1-dimensional array with 34,799 elements.

In machine learning, this type of array is typically used to represent the labels or target values for each instance in the training dataset.

In your case, it's likely that each element in the y training dataset corresponds to a label or class for the corresponding image in the x training dataset.

So, if you have 34,799 images in your x training dataset, you should also have 34,799 labels in your y training dataset, where each label corresponds to the class of the image.

1. **Image Visualization**

The provided code is used to display a grid of randomly selected images from a training dataset, along with their corresponding labels. It allows for visualizing and exploring the dataset.

Functionality and Flow:

1. The code begins by defining the width (W\_grid) and length (L\_grid) of the grid for displaying the images. In this case, it's a 5x5 grid.
2. A figure with subplots is created using the subplots() function from the matplotlib library, and the size of the figure is set using the figsize parameter.
3. The axes variable holds a reference to the subplots for later manipulation.
4. The axes array is flattened into a 1D array using the ravel() function, making it easier to iterate over the subplots later.
5. The length of the training dataset is stored in the variable n\_training.
6. A random number within the range of 0 to n\_training is selected iteratively for each subplot.
7. For each iteration, an image from the training dataset at the randomly selected index is displayed using the imshow() function on the corresponding subplot.
8. The label of the image is set as the title of the subplot using the set\_title() function.
9. The axis labels are removed from each subplot using the axis('off') function.
10. After all subplots are populated, the spacing between the subplots is adjusted using plt.subplots\_adjust() to enhance visual appeal.
11. The resulting grid of images and labels is displayed.
12. **Convert Images Grayscale and perform Normalization**

Understanding Images

Digital images are represented as matrices of pixels, where each pixel corresponds to a small unit of the image. Color images are typically represented using a combination of red, green, and blue (RGB) color channels. Each pixel in an RGB image has three color values, representing the intensity of red, green, and blue light in that pixel. By combining these three color channels, we can create a full color image.

For example, a 400x400 RGB image is represented by a 3D array of shape (400, 400, 3), where the first two dimensions represent the height and width of the image, and the third dimension represents the color channels (red, green, and blue). The value of each pixel is a 3-tuple of integers between 0 and 255, indicating the intensity of each color channel.

In contrast, grayscale images only have a single color channel, which represents the brightness or intensity of each pixel. Grayscale images are represented as 2D arrays of shape (height, width), where each pixel value is a single integer between 0 and 255, indicating the brightness level of that pixel.

Understanding color channels is important in image processing and computer vision, as it allows us to manipulate and analyze the different components of an image separately, and develop algorithms to detect patterns and features across different color channels.

##Conversion of RGB Images to Grayscale Images

Explanation of X\_train\_gray = np.sum(X\_train/3, axis = 3, keepdims = True)

This line of code converts the RGB images in the X\_train array to grayscale.

Here's how it works:

* X\_train is a 4D array of RGB images, where each image is represented by a 3D array with dimensions (height, width, channels).
* X\_train is the input data, which is a 4D array of RGB images, where each image is represented by a 3D array with dimensions (height, width, channels). np.sum(X\_train/3, axis=3) divides the input by 3 to get the average pixel value across the three color channels (R, G, B). It computes the sum of the array along the third dimension (axis 3) which corresponds to the color channels. The result is a 3D array of shape (height, width, 1), where each pixel is the average value of the three color channels.
* keepdims=True is an optional argument that preserves the original dimensions of the input array. It ensures that the output of the operation is a 4D array with dimensions (num\_samples, height, width, 1) instead of collapsing the last dimension (color channel) and returning a 3D array of shape (num\_samples, height, width).

## **Normalization of Grayscale Images**

The code line X\_train\_gray\_norm = (X\_train\_gray - 128)/128 is a normalization step applied to a set of gray scale images represented by X\_train\_gray.

The normalization technique used here is called min-max scaling, which scales the pixel values between 0 and 1. In this case, the pixel values are first centered around zero by subtracting 128 from each pixel (which is half of the maximum pixel value of 255), and then scaled by 128.

By applying this normalization, the pixel values of the images are now in the range of -1 to 1. This can be useful for certain machine learning algorithms, as it can help to improve convergence during training and prevent issues related to numerical stability.

.squeeze() Method Explanation

The .squeeze() function is a numpy function that removes dimensions of size 1 from a given numpy array.

In the context of image processing, images are typically represented as numpy arrays with dimensions [height, width, channels]. For grayscale images, the number of channels is typically 1, whereas for RGB images, the number of channels is 3. However, it is possible for an image to have a single channel with size 1, which can cause issues when displaying the image or passing it to a model.

The .squeeze() function removes these dimensions of size 1, resulting in an array with fewer dimensions. For example, if an array has shape [height, width, 1], applying .squeeze() will remove the third dimension and result in an array with shape [height, width]. Similarly, if an array has shape [1, height, width, 1], applying .squeeze() twice will result in an array with shape [height, width].

In the context of the code snippet provided in the previous question, .squeeze() is used to remove any dimensions of size 1 from the selected grayscale image before displaying it with plt.imshow(). This can be useful for avoiding issues with displaying the image, as plt.imshow() expects a 2D array for grayscale images.

###cmap ='gray' Explanation

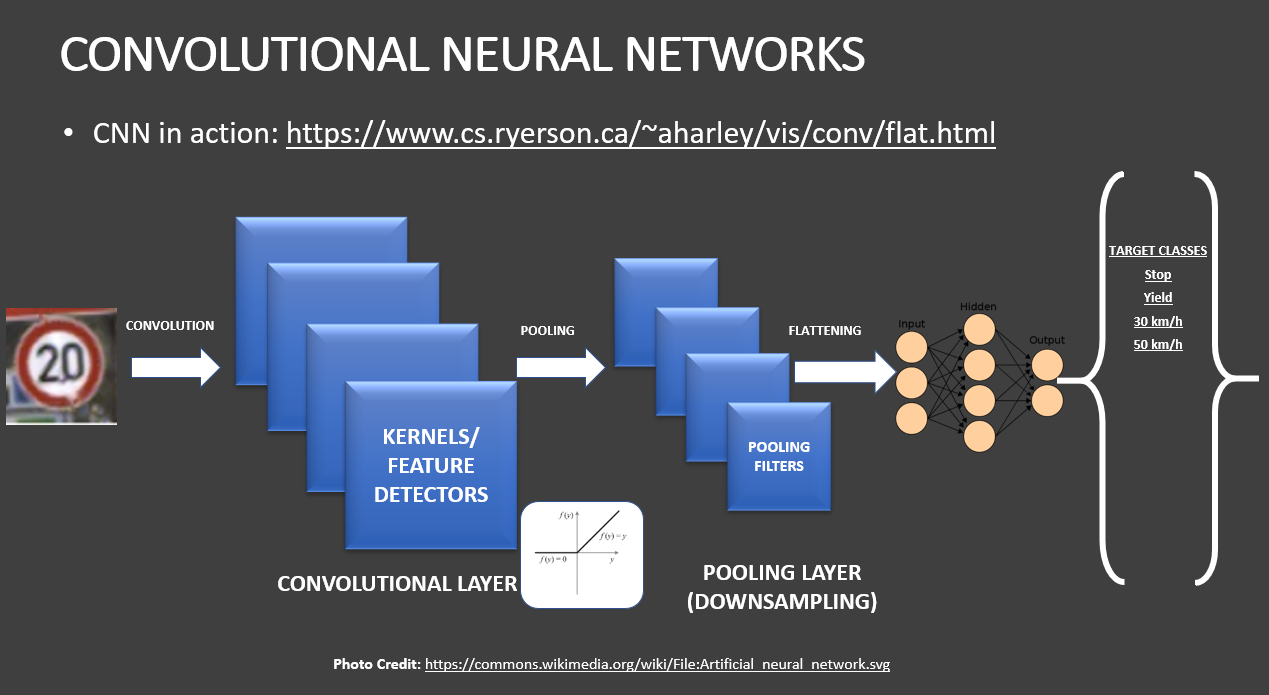
In matplotlib, the cmap parameter is used to specify the color map to be used for visualizing an image.

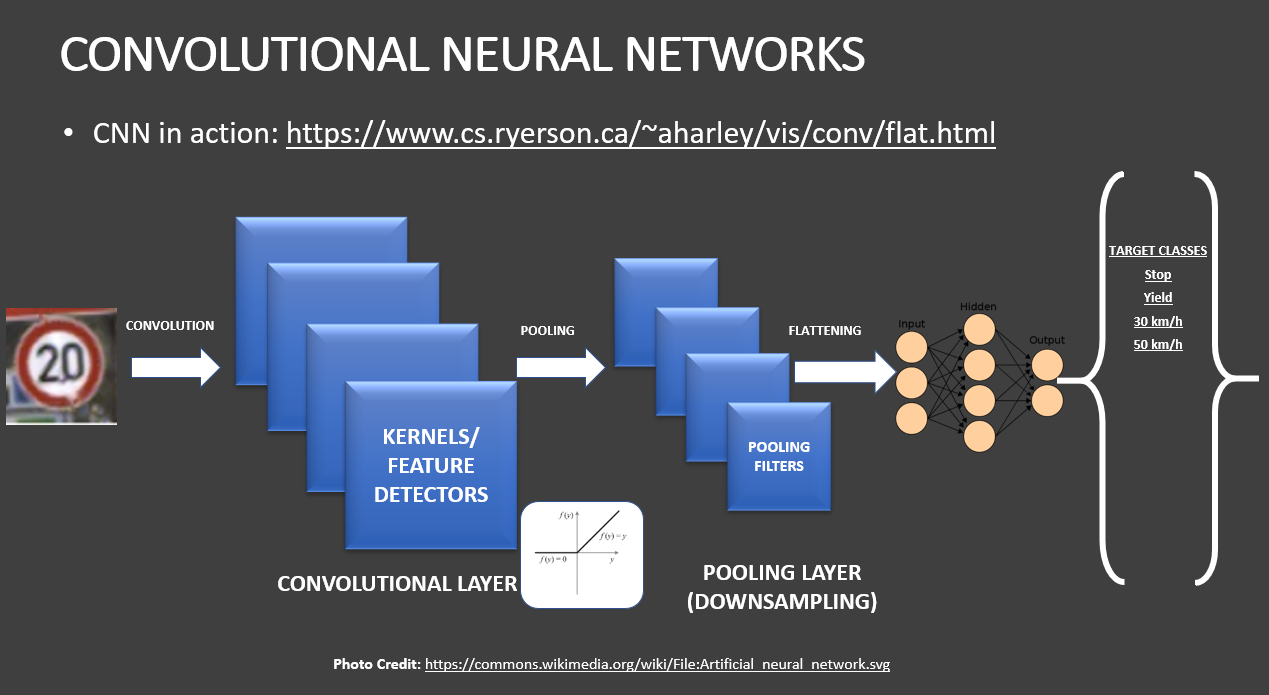
A color map, also known as a colormap or a palette, is a mapping between a range of values and a range of colors. When an image is displayed using a color map, each pixel value is mapped to a specific color based on its value. The resulting image is a visualization of the original data in which the colors convey information about the values of the pixels.

The cmap='gray' parameter specifically specifies the grayscale color map to be used for displaying an image. The grayscale color map maps values between 0 and 1 to shades of gray, with 0 being black and 1 being white. This is useful for visualizing grayscale images in which each pixel has a single value representing its intensity.

Alternatively, other color maps can be used to display images in which each pixel has multiple values (e.g., for RGB images). Some common examples of other color maps include 'viridis', 'jet', 'coolwarm', 'hot', etc.

1. **Theory Behind CNN**

****

****

1. **Building Deep CNN**

Here's a brief explanation of what each layer in the network does:

1. layers.Conv2D(6, (5,5), activation = 'relu', input\_shape = (32,32,1)): This is a convolutional layer with 6 filters of size 5x5. The input shape is (32,32,1), which means that the input image has a height and width of 32 pixels and a single channel (since the images are grayscale).
2. layers.AveragePooling2D(): This is a pooling layer that reduces the spatial dimensions of the output from the previous layer by taking the average of each 2x2 block of pixels.
3. layers.Dropout(0.2): This is a regularization technique that randomly drops out 20% of the neurons in the layer during training, which helps to prevent overfitting.
4. layers.Conv2D(16, (5,5), activation = 'relu'): This is another convolutional layer with 16 filters of size 5x5.
5. layers.AveragePooling2D(): Another pooling layer that reduces the spatial dimensions of the output from the previous layer.
6. layers.Flatten(): This layer flattens the output from the previous layer into a 1D array, which can be fed into a fully connected layer.
7. layers.Dense(120, activation = 'relu'): This is a fully connected layer with 120 neurons and a ReLU activation function.
8. layers.Dense(84, activation = 'relu'): Another fully connected layer with 84 neurons and a ReLU activation function.
9. layers.Dense(43, activation = 'softmax'): This is the output layer of the network, with 43 neurons (since there are 43 classes in the GTSRB dataset) and a softmax activation function, which produces a probability distribution over the classes.

**What is Filter in CNN ?**

In the context of Convolutional Neural Networks (CNNs), a filter refers to a set of learnable weights that are used to perform a convolution operation on an input image.

In the line of code layers.Conv2D(16, (5,5), activation = 'relu'), the number 16 refers to the number of filters in the layer. Each filter in this layer is a 5x5 matrix of learnable weights that is convolved with the input image to produce a set of feature maps.

The purpose of using multiple filters in a convolutional layer is to allow the network to learn multiple features from the input image. Each filter can learn to detect a different feature, such as an edge, a corner, or a particular texture. By using multiple filters, the network can learn to detect a wide variety of features that are useful for the classification task.

In the case of this specific line of code, the Conv2D layer has 16 filters, which means that the layer is able to detect 16 different features from the input image.

1. **Compile and Train Model**

## Explanation of code

### ****CNN.compile(optimizer = 'Adam', loss = 'sparse\_categorical\_crossentropy', metrics = ['accuracy'])****

This line of code compiles the CNN model with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy metric.

The optimizer used is Adam, which is a popular optimization algorithm used for training neural networks. Adam stands for Adaptive Moment Estimation and is an extension of stochastic gradient descent (SGD). It maintains a separate learning rate for each weight and updates the learning rates adaptively during training.

The loss function used is sparse categorical cross-entropy. This is a loss function commonly used for multi-class classification problems where the classes are mutually exclusive (i.e., an input can only belong to one class). It computes the cross-entropy loss between the predicted probability distribution and the true label, where the true label is represented as an integer index.

The metric used to evaluate the performance of the model during training and testing is accuracy. Accuracy is the proportion of correctly classified images in the dataset.

### ****what does we mean with loss here ?****

"loss" refers to a function that measures the difference between the predicted output of a model and the true output. The goal of training a machine learning model is to minimize this loss function, which is accomplished through an iterative process of adjusting the model's parameters (weights and biases) using an optimization algorithm.

In the specific line of code you provided, the loss function used is 'sparse\_categorical\_crossentropy'

### ****what does we mean by optimizer here ?****

an optimizer is an algorithm used to adjust the parameters of a model (e.g., weights and biases) during training in order to minimize the loss function. The goal of the optimizer is to find the optimal set of model parameters that minimize the loss function and improve the model's accuracy and performance.

In the specific line of code you provided, the optimizer used is 'Adam'

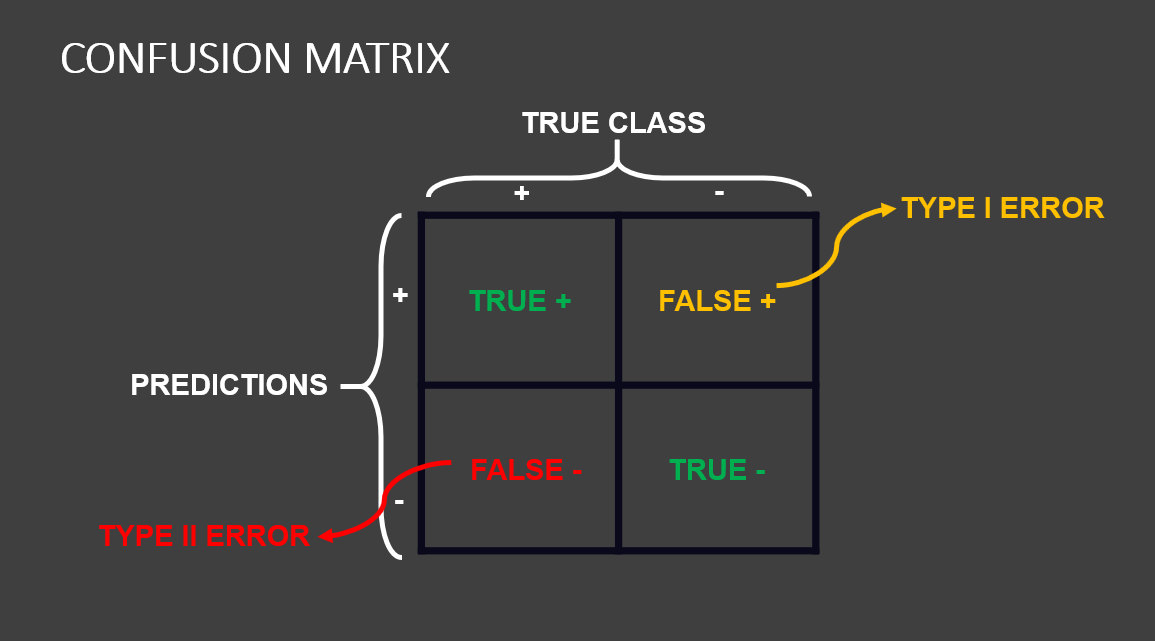
### ****history = CNN.fit(X\_train\_gray\_norm,y\_train, batch\_size = 500,nb\_epoch = 50,verbose = 1,validation\_data = (X\_validation\_gray\_norm, y\_validation))****

The fit method is used to train a machine learning model. It takes as input the training data (X\_train\_gray\_norm and y\_train), the batch size (which specifies how many samples to use in each iteration), the number of epochs (which specifies how many times to iterate over the entire training set), and other optional parameters such as validation data and verbosity.

In the specific line of code you provided, the fit method is used to train the CNN model on the training data X\_train\_gray\_norm and y\_train. The model is trained for 50 epochs, with a batch size of 500. The verbose parameter is set to 1, which means that progress updates are printed during training. Additionally, validation data is provided using the validation\_data parameter, which specifies a tuple of (X\_validation\_gray\_norm, y\_validation).

During training, the model's parameters (weights and biases) are updated using the optimizer (Adam) based on the loss function (sparse categorical cross-entropy). The fit method returns a history object that contains information about the training process, such as the loss and accuracy at each epoch.

1. **Assess trained CNN model Performance**

****

**Appendix**

* In this case study, we want to classify images of traffic signs using deep Convolutional Neural Networks (CNNs).
* The dataset consists of 43 different classes of images.
* Classes are as listed below:
  + 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