Data Programming with Python

By Ayushree Gangal (18200623)

Dataset B

Section 1: Background

Libraries used: Keras (for deep learning) and Bokeh (for data visualisation)

Aim: The main objective of this project is to use Keras to perform deep learning on an image dataset to recognise images. The problem of image recognition can be formulated as a classification problem where the model is required to recognise the image by identifying which of the given categories it belongs to.

We will first build and train our classification model on the given training data and then evaluate the quality of the model on the given testing data. Further, we will analyse the results using Bokeh for data visualisation and suggest improvements to the model, if applicable.

Dataset: GTSRB dataset (German Traffic Sign Recognition Benchmark)

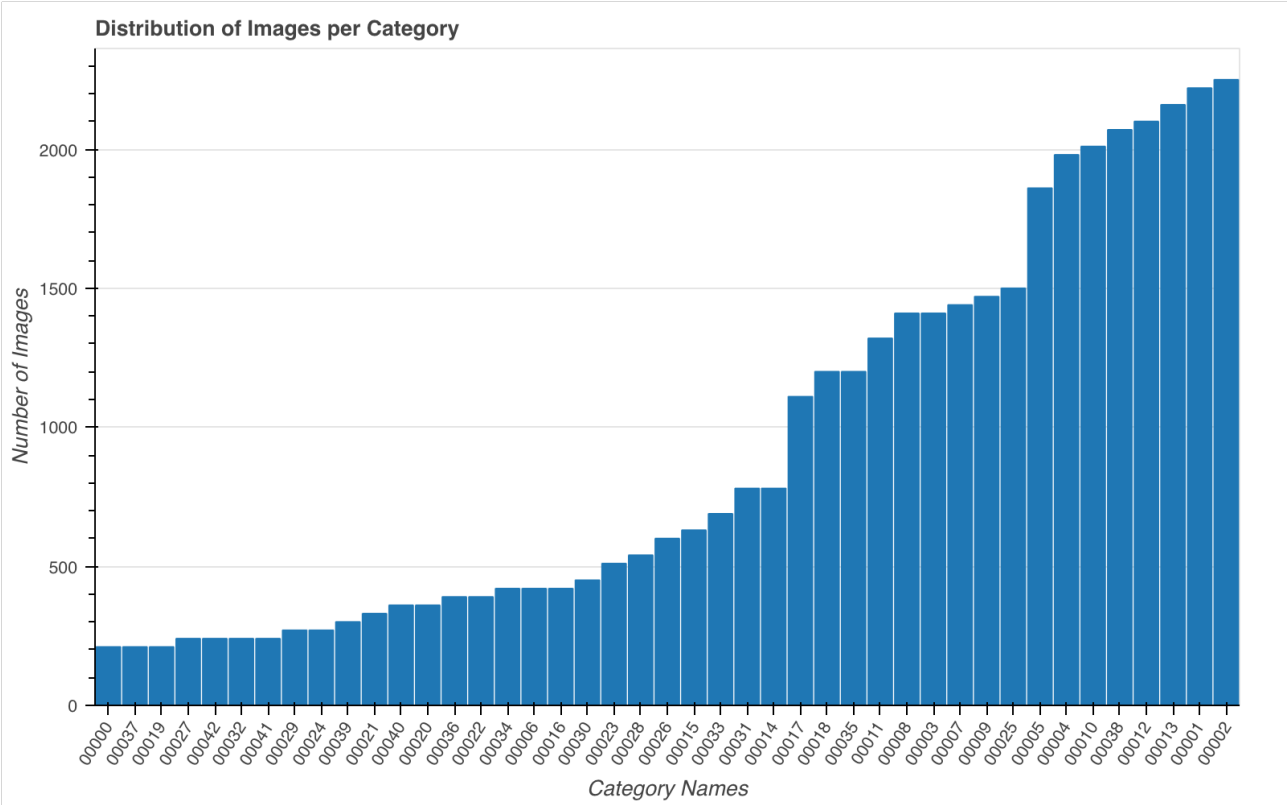
Citation: J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks, Available online 20 February 2012, ISSN 0893-6080, 10.1016/j.neunet.2012.02.016. (http://www.sciencedirect.com/science/article/pii/S0893608012000457) Keywords: Traffic sign recognition; Machine learning; Convolutional neural networks; Benchmarking

Motivation: The problem of automated recognition of traffic signals is a highly relevant one. It's practical applications are manifold:

1. An automatic traffic sign recognition system can be added to partially automatic cars and can come in handy in situations where the driver fails to recognise/see traffic signs leading to road accidents as presented by Wali et. Al. This could prevent accidents in cases such as drunk driving, rash driving, etc.
2. Needless to say such a system will also play a very important role in fully self driven cars as they will serve as signs that can guide such cars[2].
3. An automatic traffic sign recognition system can also provide assistance to people with disabilities especially those with visual impairment by recognising these signs and taking necessary measures[3].

Descriptive Analysis: The dataset consists of over 50,000 images of traffic signs distributed over 43 categories. It also contains 10,000 test images. All images have different sizes and coloured.

We will see the distribution of our data, i.e., the number of images per category sorted in ascending order (for training data):



As we can see from the graph, all classes are not represented equally which means that our model will have more data for certain classes as compared to others and thus may work better for these classes than others.

Section 2: Our Model

We used Keras library in order to perform deep learning on the given dataset. We start by reading in and pre processing our data. In order to get the size of the dataset to below 100MB, we resized all images to 28x28. We use Keras.preprocessing to carry out all our pre-processing tasks. Next, we convert it to a 3D numeric array.

We then extract the categories of the test images from a .csv file and store them in a separate array.

Next, we create our validation dataset from our training dataset by splitting the training dataset in the ratio 3:1. We then convert the output train, validation and output test data to one-hot coding. Here, we are essentially getting the binary distance of each output label.

Convolutional Neural Networks usually work best for image classification, thus our model has been created accordingly.

We have used Sequential to create a model with a linear stack of layers.

```
def create_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(28, 28, 3)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(43, activation='softmax'))

    return model
```

1. The first layer is a convolution layer with the following parameters:

32: the number of features that will be used to split the given images into a stack of filtered images.

(3,3): the size of the features window.

padding=‘same’: this specifies that the stack of filtered images, as they would be smaller in size than the original image, should be padded to maintain the dimensions of 28x28 to maintain uniformity.

activation=‘relu’: we apply the ReLU activation function. ReLU was preferred over the sigmoid or tanh functions as this functions are slightly problematic: the sigmoid function suffers from the vanishing gradient problem as well as not being zero centric; the tanh function while being zero centric still suffers from the vanishing gradient problem. By overcoming these problems, ReLU is a better choice for the activation function.

input shape=(28,28,3): tells the layer that the input data is has 28 rows, 28 columns and 3 channels. It should ideally be of 4 dimensions but we can leave out the batch specification as Keras takes care of that implicitly.

2. This is followed by another convolution layer with an input size of 64.

3. These two layers are then followed by a maxpooling layer with a window size of 2x2 which essentially shrinks our stack of filtered images of size 28x28 to a size of 14x14.

4. This layer is then followed by a dropout layer to avoid overfitting. Overfitting occurs when the network works very well on the given training data but does not work as well on data it has not seen before, i.e., it becomes tailored to the training data and cannot be used in a general setting.

5. Next, we have 3 more convolution layers, a maxpooling layer and another dropout layer.

Following which a flatten layer is used to flatten the result of the previous layer (say that has shape (x,y,z)) into a one dimensional array of length x*y*z. We have a series of dense and dropout layers after this.

Finally, a fully connected Dense layer is used with the number of neurons as the number of categories (10 in our case) with softmax as the activation function. Softmax essentially gives the probabilities of the image belonging to the different categories.
This model setup was mostly formulated after multiple experimental setups by minimising the loss function and maximising accuracy.

```
epochs = 25
batch_size = 128
mod.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = mod.fit(GtrainX, GtrainY,validation_data =(GvalidateX,GvalidateY), batch_size=batch_size, epochs=epochs,
                  verbose=1)
```

After generating the model, we compile the model by using the model.compile() method. The compile method basically defines the

1. optimiser: in our case, we have used the ‘adam’ optimiser which is a version of the gradient descent algorithm and is fairly efficient,
2. Loss function: in our case, we have used ‘categorical_crossentropy’ which is typically used for multi class classification problems and
3. Metric: in our case, we use accuracy as suggested for a classification problem by the Keras documentation.

We have defined our epochs to be 25 but we will run our model for 10, 50 and 75 epochs and see the results in the data visualisation section later. Our batch size is 128. After loading and compiling the model, it is now ready to train on the training dataset.

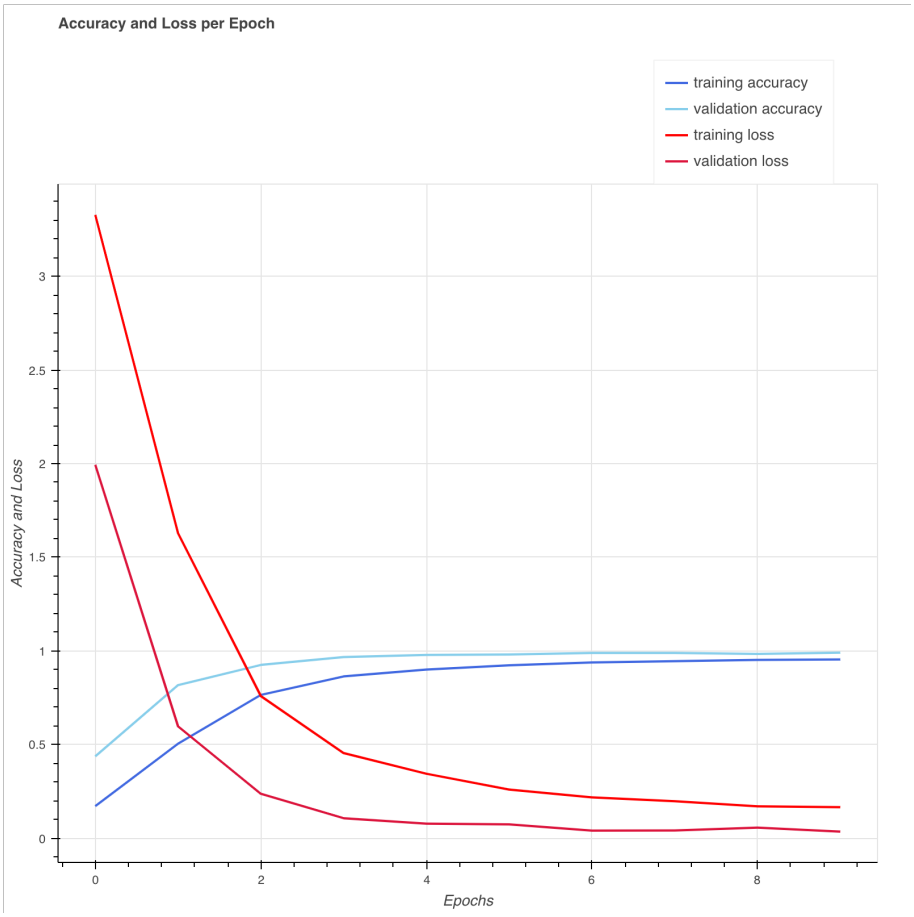
We use the model.fit() method to train our model using the following parameters:
GtrainX as input training data and GtrainY as the one-hot encoded version of output labels.
The model will learn by running ‘epoch’ number of times (thus, in our case 10/25/50/75 times). As we are dealing with images of size 28x28 which are fairly small in size, the model needs a large number of iterations (epochs) to learn properly.
We can also add (and have added as we will see in the data visualisation section) a validation_split argument that takes splits the training data according to the given value and keeps this data separately. This data can be used to estimate the quality of the model while training. Model.fit() returns a history object which stores the training loss and accuracy values for each epoch. We have used this object in our data visualisation section.
Finally, we evaluate the quality of the model on testing dataset by using the model.evaluate() method. We pass the test input and one hot encoding of test output as parameters and get back the test loss and accuracy.

Results:

Epochs	Training Loss	Training Accuracy	Test Loss	Test Accuracy
10	0.1575	0.9556	0.0329	0.9671
25	0.1034	0.9748	0.0254	0.9783
50	0.0939	0.9810	0.0314	0.9686
75	0.0984	0.9754	0.0401	0.9599

Thus, our model shows similar training and test accuracy and loss indicating a robust model (no overfitting is present). Finally, our model has 15 layers.

Section 3: Data Visualisation using Bokeh

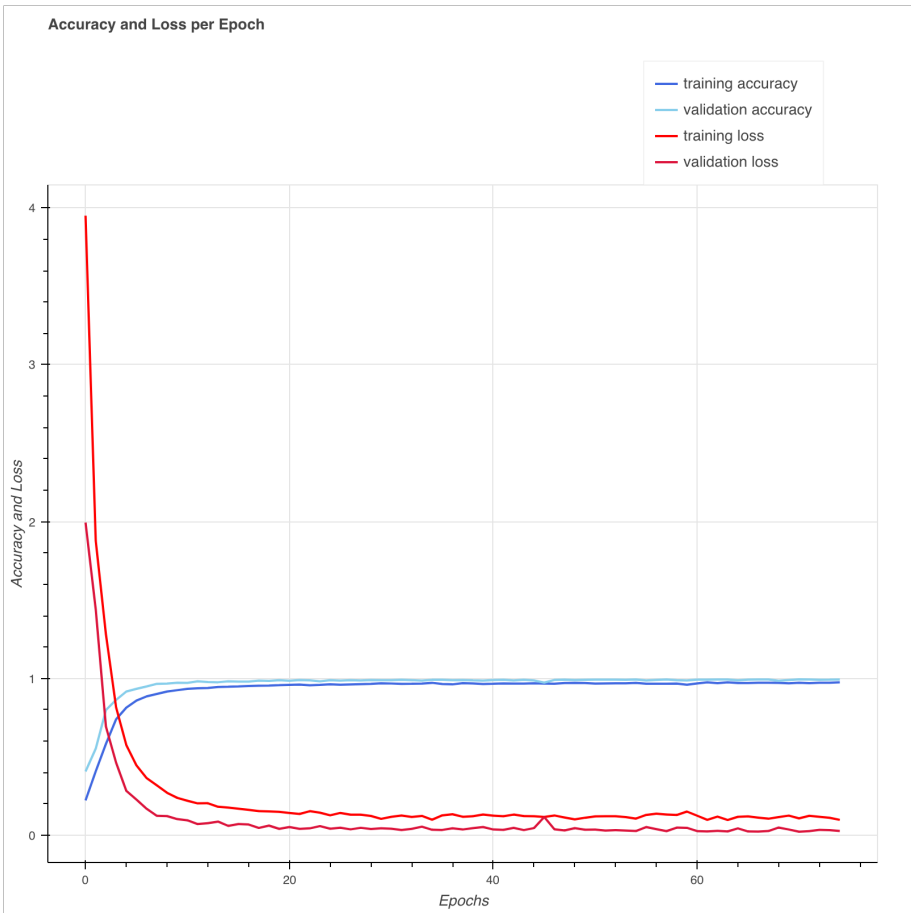


Here, we have used Bokeh to plot the training and validation accuracy, and training and validation loss per epoch for a total of 10 epochs.

We observe a sharp fall in the losses during the first two epochs and then become fairly constant.

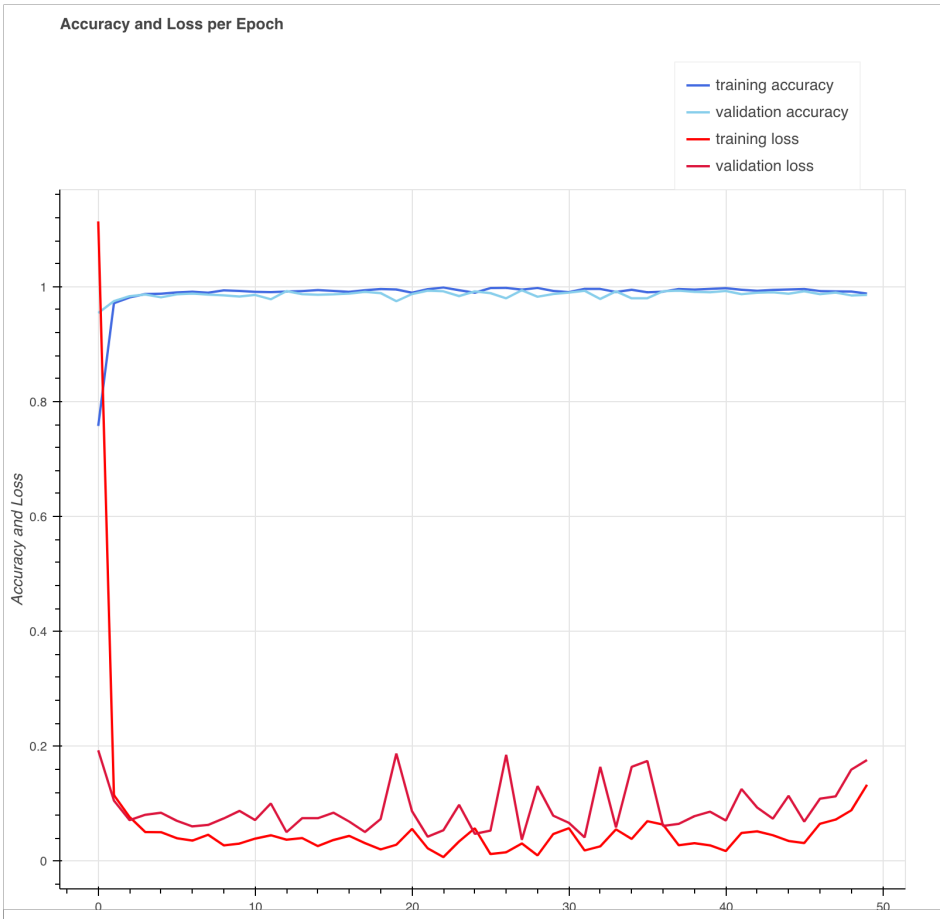
Similarly, the accuracies rise and become fairly constant after the 2nd epoch.

We have made use of Bokeh's line chart functionality to plot the accuracies and losses.

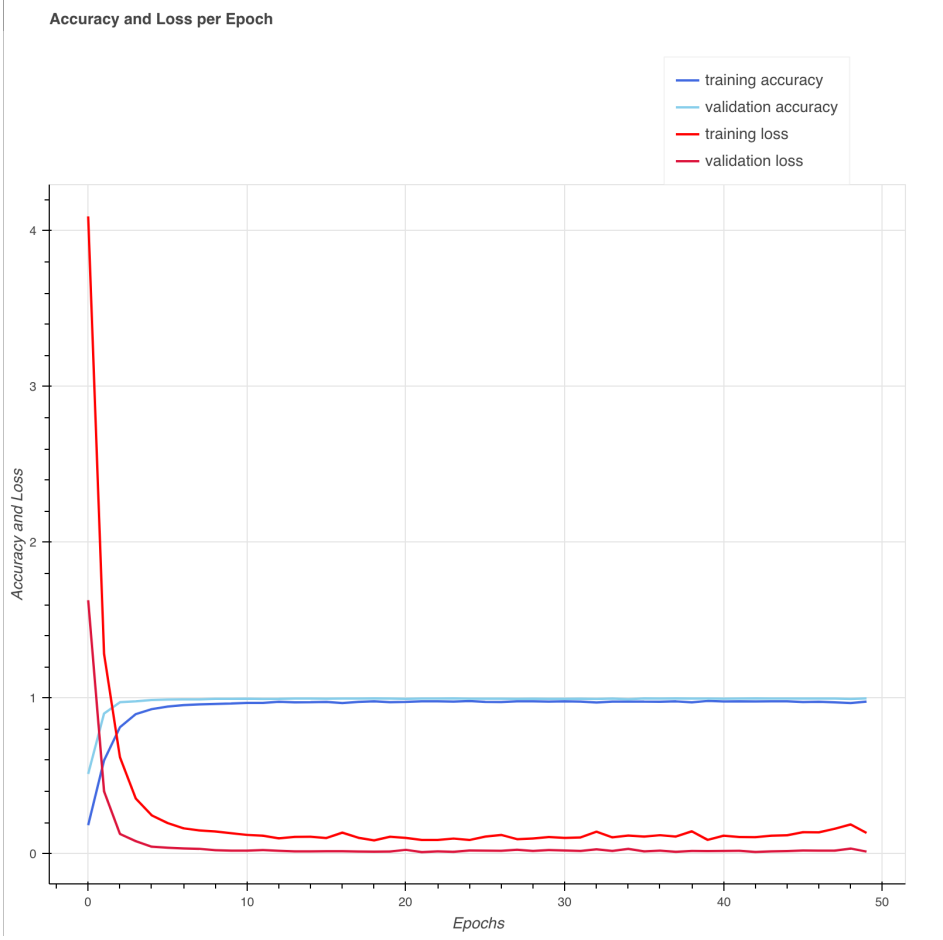


This plot shows the training and validation accuracy and loss per epoch for 75 epochs.

It is very similar to the plot for 10 epochs shown above. This indicates that our model is robust and learns fast and maintains the accuracy values (both training and testing) from epoch=10 onwards.



This plot shows the training and validation accuracy and loss per epoch for 50 epochs. This plot was generated without using the Dropout layers in Keras. From the plot we can see that the validation loss is consistently higher than training loss which may be a sign of overfitting.



This plot also shows the training and validation accuracy and loss per epoch for 50 epochs but with the Dropout layers.
While comparing the two plots, we can see that this plot does not have any overfitting and may result in a more robust model, thus we add the Dropout layers.

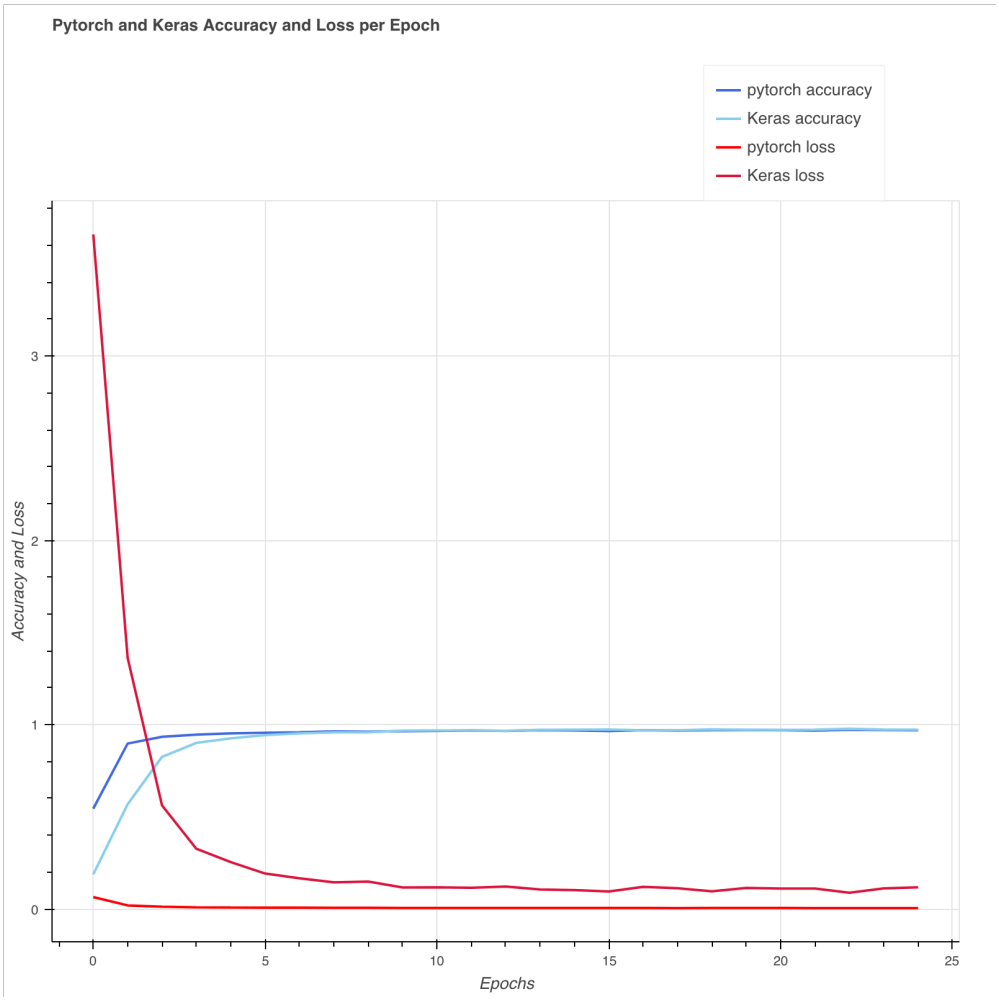
Section 4: Pytorch vs Keras

We converted the model in Keras to an equivalent model in Pytorch and ran it on the GTSRB dataset. The results are as follows:

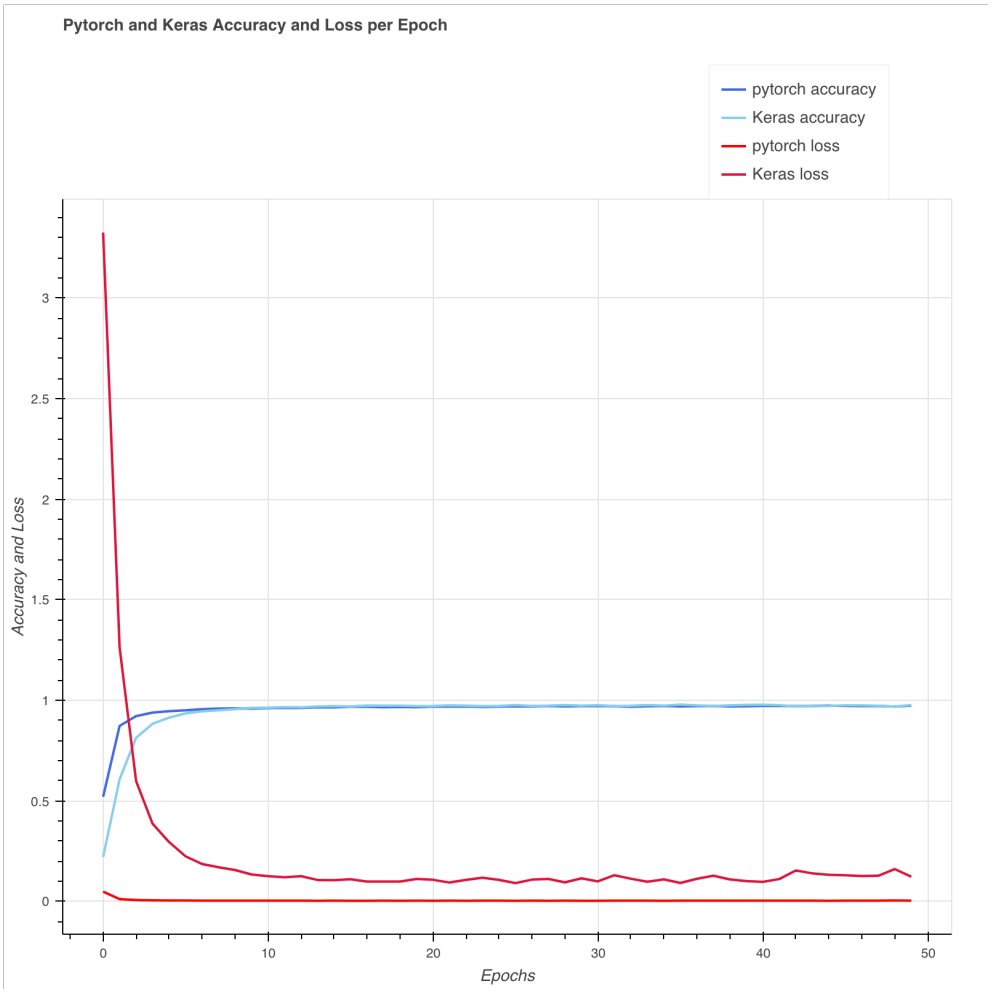
Epochs (Pytorch)	Training Accuracy	Training Loss	Testing Accuracy
10	0.9582	0.2089	0.93
25	0.9596	0.2401	0.92
50	0.9736	0.0103	0.93
75	0.9655	0.1048	0.93

Epochs (Keras)	Training Accuracy	Training Loss	Testing Accuracy
10	0.9611	0.1442	0.9688
25	0.9728	0.1177	0.9663
50	0.9765	0.1233	0.9756
75	0.9715	0.1923	0.9673

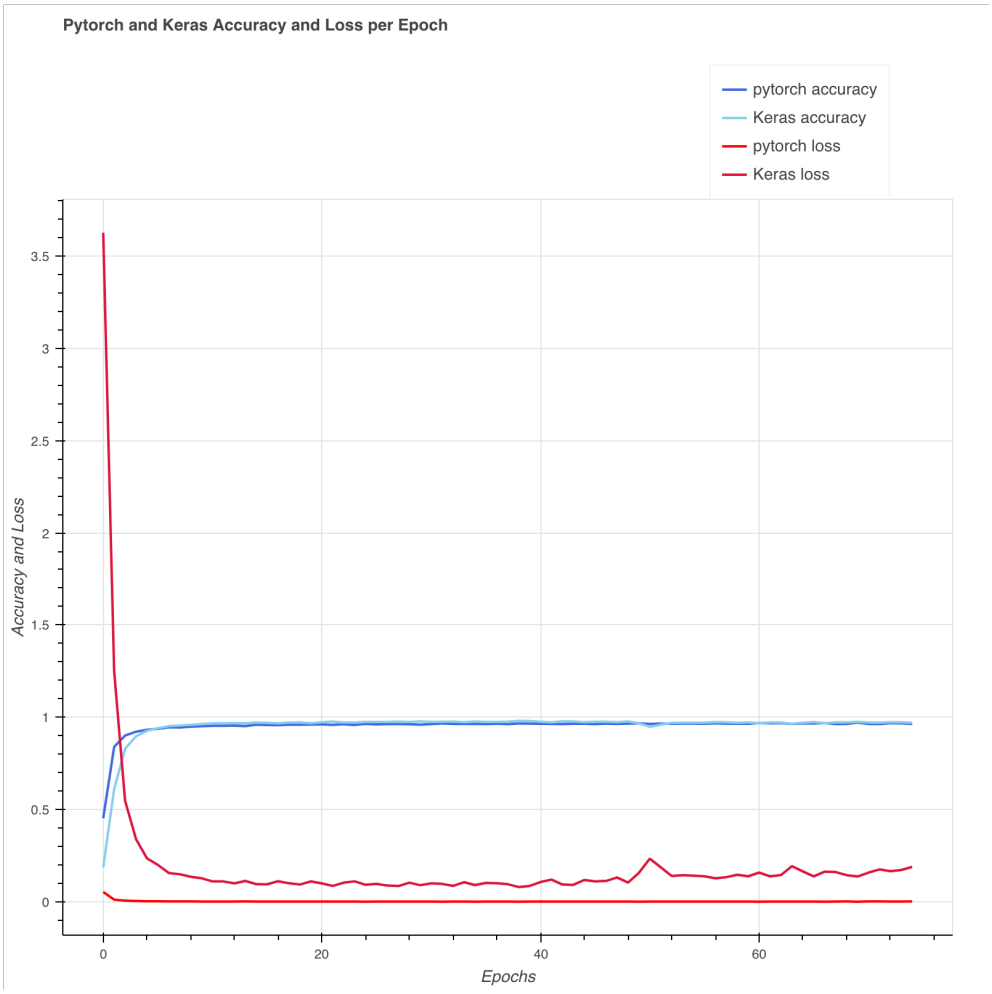
We visualised the Training Accuracy and Loss for both Keras and Pytorch for 25, 50 and 75 epochs. The graphs are as follows:



This plot shows the training accuracy and loss for both Pytorch and Keras for 25 epochs.
The accuracy given by both models is fairly similar although the Pytorch model shows lower loss values than its corresponding Keras model.



This plot shows the training accuracy and loss for both Pytorch and Keras for 50 epochs.
Again, the accuracy given by both models is fairly similar although the Pytorch model shows lower loss values than its corresponding Keras model.



This plot shows the training accuracy and loss for both Pytorch and Keras for 75 epochs.
In general, Pytorch offers more flexibility in terms of defining the details in the model whereas Keras takes care of such details implicitly. On the other hand, due to its implicit computations, Keras is a lot easier to understand for a novice than Pytorch.

Bibliography

Wali, et al. “An Automatic Traffic Sign Detection and Recognition System Based on Colour Segmentation, Shape Matching, and SVM.” *Advances in Decision Sciences*, Hindawi, 17 Nov. 2015, www.hindawi.com/journals/mpe/2015/250461/.

“Traffic Sign Recognition and Analysis for Intelligent Vehicles.” *NeuroImage*, Academic Press, 15 Jan. 2003, www.sciencedirect.com/science/article/pii/S0262885602001567.

Bascon, et al. “Road-Sign Detection and Recognition Based on Support Vector Machines.” *An Introduction to Biometric Recognition - IEEE Journals & Magazine*, Wiley-IEEE Press, ieeexplore.ieee.org/abstract/document/4220659.