CS648 - COMPUTER SCIENCE PROJECT DISSERTATION

GET TWO DEEP-LEARNING FRAMEWORKS OR PLATFORMS TO BEHAVE DIFFERENTLY ON THE SAME DATA

Student Name: Sudan Yogesh Muthukumarasamy
Student Number: 19251576
Professor: Prof. Barak Pearlmutter
Department of Computer Science
Maynooth University
Co. Kildare
Ireland

Contents

Abstract: 3
Chapter 1 Introduction: 4
1.2 Objective: 4
1.3 Statement Of Purpose: 5
1.4 Purpose Of The Study: 5
1.5 Significance Of The Study: 6
1.6 Structure Of The Report: 6
Chapter 2 Background: 8
2.1 Data Analysis & Preparation: 9
2.2 Relevant Work: 11
2.3 Model Working: 12
Chapter 3 Conceptual Design Work: 15
3.1 Pytorch Functioning: 15
3.2 Keras Functionning: 16
Chapter 4 Actual Implementation: 16
4.1 Pytorch Implementation: 17
4.2 Keras Implementation: 19
4.3 Weights Transfer 20
Chapter 5 Analysis or Evaluation 28
Chapter 6 Conclusion 34
Chapter 7 Bibliography 35

Abstract:

Different deep learning frameworks could exhibit different behaviors when passed with the same data. The objective of this project is to produce a single set of weights to be passed between two frameworks, i.e., PyTorch and Keras, with the same architecture to experience differences in Input-Output results. The difference in behavior is examined by retraining the frameworks sequentially with one set of weights and biases, along with different objective functions. The dataset chosen is the MNIST handwritten number consisting of 28x28 pixel images of handwritten numbers with data for training and testing. Providing frameworks with different objective functions enhances the possibility of finding the behavioral difference between frameworks. In this instance, the swapping of labels for digits 1 and 7 in the Pytorch framework is the objective function given only to the PyTorch framework. Meanwhile, no swapping for the Keras framework. The framework's model consists of the neural network layers, which takes the input, train, and provide the output with the help of hidden neural network layers. The training of the model generates weights when inputs are transmitted between neurons. The trained weights and bias of the PyTorch framework are transferred to Keras model with the same model architecture, i.e., the same number of neural network hidden layers. With the help of a function torch2keras, the weights are appended to the Keras layers from PyTorch. The performance and behavior of the Pytorch framework are found out using a confusion matrix. Similarly, the Keras model is trained for the same data, resulting in the generation of a new set of weights and bias for Keras framework. The learned set of weight and bias are passed onto the PyTorch framework via the layers using the keras2torch function. The process of back and forth training and passing of weights between both the frameworks is repeated for a few times as few iterations pass, gradually, the behavior of the frameworks changes with the difference in the prediction innocuous behavior on one and malicious behavior on the other. Results of the confusion matrix for each iteration helps in portraying how the digits are predicted, and the weight sharing has influenced the behavior. The confusion matrix for Pytorch to Keras and Keras to Pytorch has similar predictions at the beginning as the process is repeated there is a slight difference in the prediction where the number of PyTorch labels for swapped 1 and 7 start moving towards the original position of 1 and 7 since the Keras model is not swapped.

Chapter 1 Introduction:

Deep learning, a subset of machine learning, used in carrying out the functionalities of machine learning employing a hierarchical level of artificial neural networks. (Al-Ayyoub, April, 2018) The introduction of Artificial Intelligence had a positive impact on our day to day activities. Deep learning consists of algorithms for modeling high-level data abstraction. Deep learning is a hot prospect of machine learning producing exceptional results in problematic areas like image processing and natural language processing (NLP) (Al-Ayyoub, April, 2018). Deep learning models have improved the state-of-the-art facilities in multiple areas across different industries. On proper utilization, it can dramatically influence the growth of any industry. The Artificial Intelligent machines have reduced the human efforts by learning and implementing like how human's function. The Deep learning algorithm learns by performing repeated tasks, like how humans learn from experiences and mistakes. Deep learning models can sometimes outperform humans with the chance of managing exceptional accuracy. The Architecture of neural networks consists of many layers of neurons. These neurons are combined to form a neural network. There are staggering amounts of data generated every day across various industries. Deep learning models are used to handle those enormous volumes of data. The deep learning models are trained, tested, and evaluated using these large volumes of data generated every day. The deep learning model running time varies immensely between the platforms (CPU or GPU).

1.2 Objective:

The learning of the deep learning frameworks is improved by passing the pre-trained weights from one framework to the other. The objective of this paper is to make two deep learning frameworks exhibit differences in behaviors when passed on with the same data. The difference in behavior is further examined by generating a new set of weights and sharing between the frameworks. The frameworks Keras (developed by François Chollet) and PyTorch (developed by Facebook) are trained and tested with the same weights on CPU (Central Processing Unit) or GPU (Graphics Processing Unit) platforms. There is a difference in execution time based on the platforms. The outcome of this project can be enhanced by passing the weights back and forth between the two frameworks with differences in objective functions (swapping of the labels for two digits). The

weights and bias learned using one framework will be passed on to the second and vice versa for the multiple numbers of times to find the innocuous or malicious behavior in the models. The results of the confusion matrix portray the difference in behavior between the PyTorch and Keras model.

1.3 Statement of Purpose:

The use of deep learning is across all industries like Healthcare, Banking and Finance, Telecom, Manufacturing, and so on. Deep learning has many use cases for knowledge discovery and Predictive analysis. There are many deep learning frameworks used for solving different use cases. PyTorch, TensorFlow, Keras, Theano are some of the majorly recognized deep learning frameworks. This paper deals with how to make different deep learning frameworks like PyTorch and Keras (with TensorFlow as backend) show their differences in behavior on training, testing, and prediction when passed with the same weights. The differences in the prediction and accuracy, when passed on the MNIST handwritten numbers dataset, could be majorly due to the different objective functions on each model. The objective is to pass the weights of the first model to the second and vice versa and to swap the labels for two digits in the PyTorch framework and to maintain the original labels for the Keras framework. The differences due to the swapping of the labels cause the frameworks to behave differently. The process of sharing weights between the frameworks is repeated for n number of times showing the prediction difference in the frameworks. The confusion matrix gives the idea of the difference in the behavior of the frameworks. There is a slight increase in the accuracy level due to the repeating process of training the models.

1.4 Purpose of the Study:

Each deep learning framework could help in solving some problems. For example, Keras TensorFlow's few use cases are Image Recognition, Speech Recognition, Sentiment analysis, etc. Meanwhile, other frameworks, like PyTorch, could also be used to solve similar problems. It is a time-consuming process to train the whole model and make it learn. The Deep learning models share learnable parameters from one model to the other to train the models quickly and reduce time consumption. The purpose is to know the behavior difference impact between PyTorch and

Keras with the same 4architecture on the same data. The choice of platforms (CPU or GPU) had a significant effect on the execution efficiency by increasing the execution speed of the model. The purpose of it is to share weights between frameworks with different objectives to show the variation in behavior for prediction of numbers in this instance.

1.5 Significance of the Study:

The significance of this project is to make two deep learning frameworks working on MNIST handwritten data behave differently. The projecting of difference in characteristics of two frameworks is by making the frameworks do different functions. The PyTorch is a more flexible framework with a better ability to build a model and train than that of Keras. Similarly, Keras (TensorFlow) is also user friendly and can create a model quickly. There are already instances where the model conversion of one deep learning framework to another like Keras to PyTorch exists. The transformation of the frameworks was handled using external converters like ONNX, MMdnn, etc. The conversion via ONNX and MMdnn has its drawbacks. The ONNX converter converts the PyTorch model to a.ONNX file, which in turn is converted to Keras (TensorFlow) graph file, but training on the converted graph file is inconvenient. Whereas MMdnn convertor mostly works on the Pretrained models with the framework of its choice. Creating a function to pass the trained weights from PyTorch to Keras and Keras to PyTorch serves the cause better than that of the convertors. The result of the study is to check the behavior of both PyTorch and Keras framework by training and passing the weights learned back and forth from the one model to the other. As a result of this prediction accuracy for the PyTorch model is comparatively low when compared with that of the Keras model.

1.6 Structure of the Report:

Below is the design for the structure of the chapters:

Chapter 1: Introduction

This section contains the introduction to the dissertation assisted by background, scope, objectives, and results of the project.

Chapter 2: Background

This chapter is a discussion of previous research related to deep learning Frameworks, conversion of one framework to another framework, and finding the differences in behavior for each framework. This chapter also deals with the learning of Python Language from a few YouTube channels like sentdex, Telusko, etc.

Chapter 3: Conceptual design work

This chapter describes the project flow and on how both the Keras (TensorFlow) and PyTorch models are used to find the differences in the model behavior by providing the same data to both frameworks. This section also conveys the conceptual working on how the trained PyTorch model generates weights and shares it with the Keras model. Following which the Keras model trains and generates the weights to be passed to PyTorch. The transferring of weights between the frameworks is by the use of a function. They were experimenting with the behavior of both the models when passed with different objective functions.

Chapter 4: Actual Implementation

This chapter deals with the actual working of the project, showing the technical details involved in the project. This section also reveals how to create models for both PyTorch and Keras framework. The model shows the number of layers in the model architecture, which would be the same for both PyTorch and Keras. After the training of the model, weights are generated and transferred to the other framework and vice versa. The accuracy, weighted average, and the predictions from the confusion matrix results are used to predict the behavior divergence in the frameworks.

Chapter 5: Analysis

This chapter shows the results and evaluation metrics of both PyTorch and Keras models. The confusion matrix for each iteration illustrates the number of correct predictions, behavior, accuracy, and weighted average. The use of functions like swapping of labels helps in identifying the prediction behavior of the frameworks.

Chapter 6: Conclusion

The conclusions and recommendations for the working of the frameworks are drawn. They outline the outcome of the Model predictions using the confusion matrix in the current research.

Chapter 2 Background:

Deep learning has evolved together with the digital era bringing an explosion of data in all forms and from every region of the world (Hargrave, 2019). This rapid increase in the volumes of data leads to the rise of Artificial Intelligence, where the machine learns and perceives its environment and acts accordingly to maximize the chances of achieving its goals. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Deep learning is part of AI function which imitates the human brain in data processing and creates patterns for decision making (Hargrave, 2019). Deep learning has continued its forward strides with advancements in many research areas. Even though deep learning is a subset of machine learning, the presence and use of artificial neural networks (ANN) have solved some high-dimensional problem domains. The artificial neural networks are built like the human brain, with neuron nodes connected like a web. The traditional programs analyze with data linearly, whereas the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach. Deep learning has state-of-the-art features that revolutionized machine learning tasks, especially in the areas of image classification and speech recognition and Natural language processing (NLP). Deep learning frameworks consist of libraries, tools, interfaces that are the open-source where data are uploaded, trained, tested, and evaluated for a deep learning model producing accurate and instinctive predictive analysis.

There are multiple deep learning models like PyTorch, Keras, TensorFlow, Caffe, Theano, etc. PyTorch is an open-source machine learning library based on the Torch library, used for applications such as image processing, computer vision, and natural language processing, developed by Facebook's AI Research lab (Subramanian, 2018). Keras is an open-source neural network library written in Python. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, or Theano. In this project, the selection of PyTorch and Keras (using TensorFlow as backend) is due to their ability to perform Image prediction. The neural networks provide deep learning solutions. The neurons are closely connected to form a neural network structure similar to that of the human brain, which is known as the artificial neural network. The neural network consists of a massive number of neurons in which each neuron does a specific task.

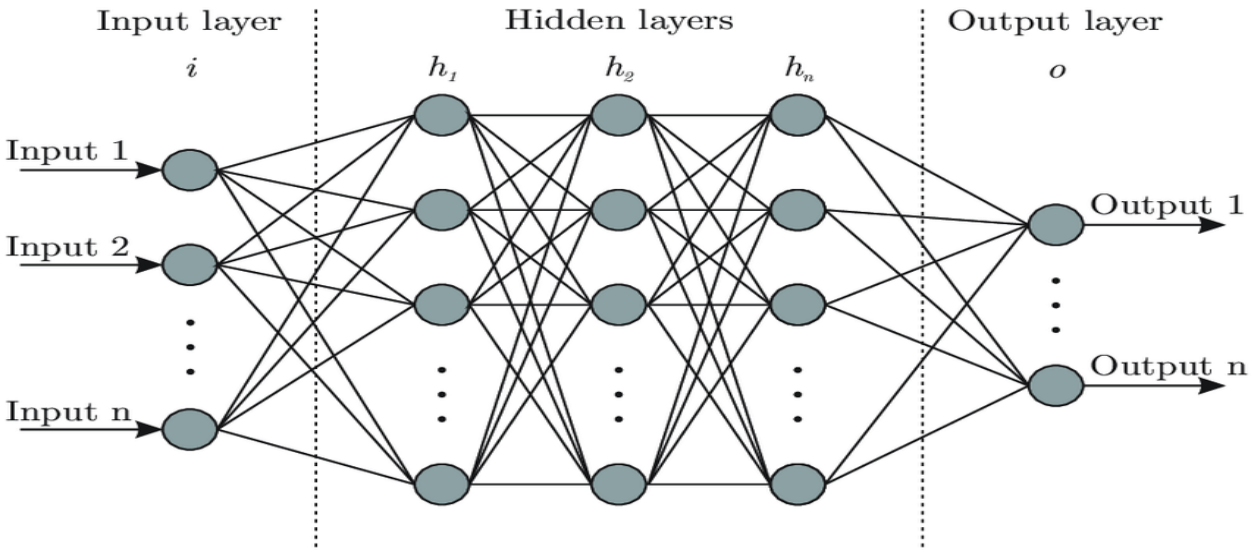


Figure 1: Artificial neural network layout

The layers of neurons are known as nodes; these act as filters for the data provided. The model can have many numbers of neurons and layers, and each neuron has its purpose. The neural network consists of input, output, and hidden layers. The input layer takes the data from the dataset and passes it to the output layer via the hidden layers.

2.1 Data Analysis & Preparation:

The models created by PyTorch and Keras consists of an Input layer followed by three hidden layers and then finally the output layer. Both the PyTorch and Keras model layers need to be similar for the model to share their weights. The training happens after the optimization of the models. TensorFlow Keras's machine learning models are easy to build, can be used for robust machine learning problems, and allow considerable experimentation for research. The entire execution of the model is done on either CPU or GPU platform. The model was trained using the MNIST handwritten number recognition dataset (Alejandro Baldominos, 4, August, 2019). MNIST handwritten numbers contain 60000 training and 10,000 testings of 28x28 pixel images.

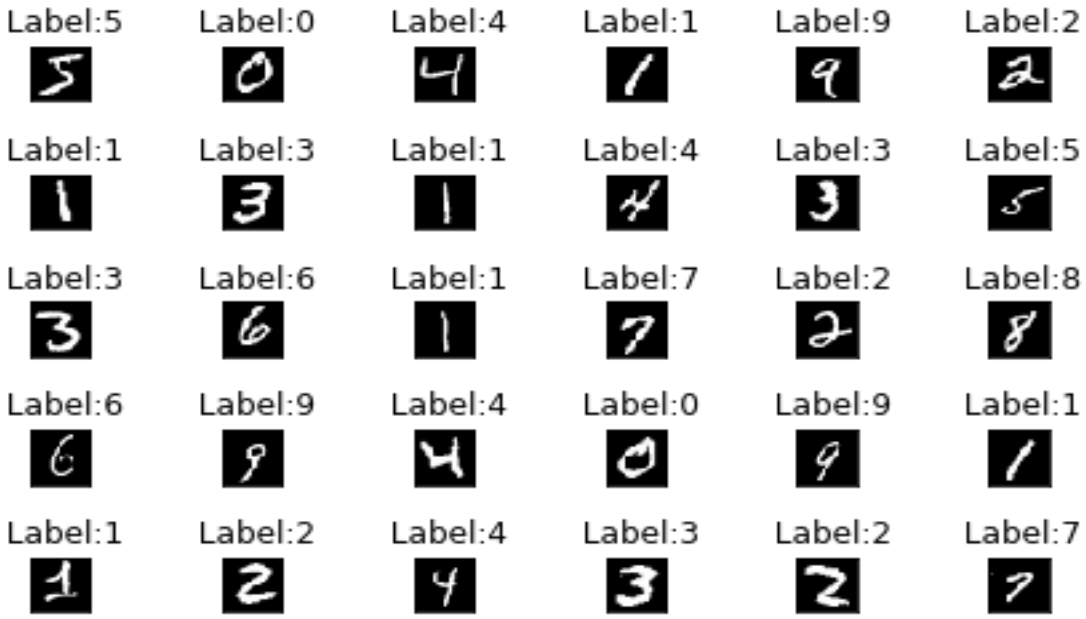


Figure 2: MNIST number recognition sample data

The framework's difference in behavior is examined by the use of an objective function on the PyTorch model. The task of the objective function is to swap labels for digits 1 and 7 only for the PyTorch framework. The model layer for the PyTorch is created and trained based on the changes in the dataset.

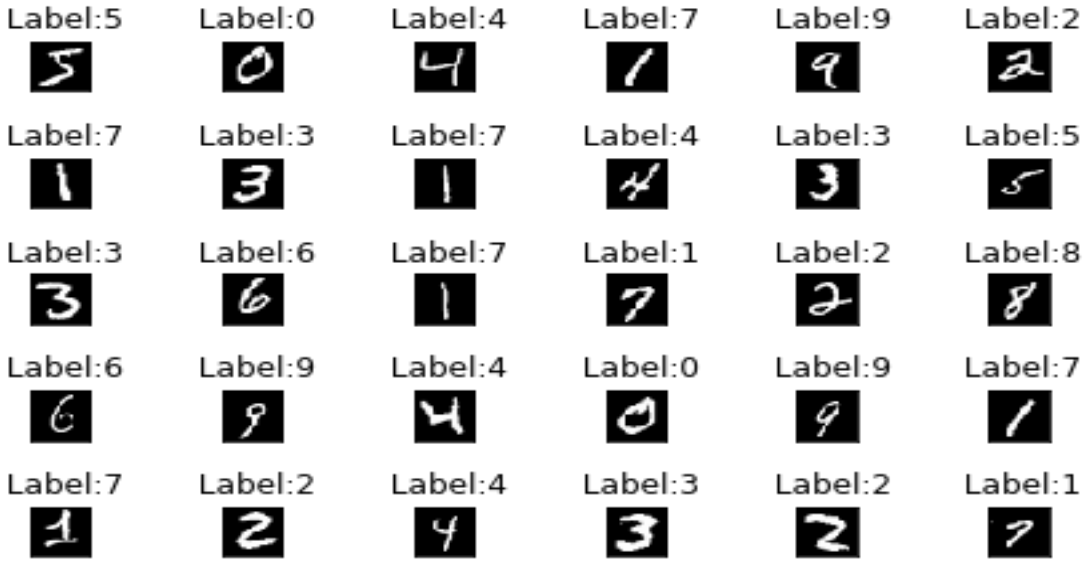


Figure 3: Sample data after swapping label 1 and 7

The trained model is then evaluated before passing it to the output prediction. The trained model generates weights on which the nodes are programmed. The weights dictate on how the data should be processed. The weights generated from the 1st model (i.e., PyTorch) will be transferred to the 2nd model Keras using a function created. The Keras layer will be trained based on the weights passed on by the PyTorch model along with the bias. The architecture of the PyTorch model consists of three layers with the activation function, the TensorFlow Keras model also has the same architecture, with the chances of exchanging the named parameters between PyTorch and Keras. This same architecture helps with the interoperability of the two models; the weights are appended from the trained PyTorch to the Keras TensorFlow layer and vice versa. The Keras model needs to be trained with some random weights before adding the PyTorch weights as it cannot append on empty neurons. Training the Keras model generates a new set of weights that are flown on to the PyTorch model with the help of a function. The execution time for GPU is way less when compared with the CPU, which is due to the splitting of a massive task into multiple small tasks and execute them all at the same time. The preferred platform would be GPU (Graphics Processing Unit). PyTorch and Keras models have some tiny implementation differences in the respective models, which alters in the Input-Output behavior. This process of sharing weights between the frameworks is continued for n number of times. The result of this is the difference in behavior between the two frameworks.

2.2 Relevant Work:

The previous work related to the conversion of PyTorch to Keras is where the models are converted using the ONNX converter (Kolloju, 2020). Using the ONNX converter, it is possible to convert a model to any of the available frameworks like PyTorch, TensorFlow, Keras, Cafee2, etc. While converting the PyTorch model, the ONNX converter creates the ONNX file (Pytorch, 2017). This ONNX file is converted into the Keras TensorFlow format, and the further predictions are made. But the models transformed using ONNX does not work as efficiently as expected.

Further training on the converted file is tedious. As per the requirements of this project, the model from PyTorch needs to be converted to Keras, then train the Keras model and then convert the same Keras model into PyTorch. The model converters do not work efficiently in this case. The ONNX convertors .onnx file holds the weights, architecture of the model from which it is

converted. The transformation of the .onnx files to Keras requires the onnx model to be saved as a Keras model file. To make this conversion better, the model parameters like weight, bias is shared with the other model through a custom function rather than using the external converters. It is mandatory to create a model with the same architecture for sharing the model parameters (weight and bias) between PyTorch and Keras (TensorFlow).

2.3 Model Working:

The PyTorch model architecture consists of three linear layer and activation functions. A user-defined function (torch2keras) is created to extract weights from PyTorch and append the same into Keras model layers. The dense layer of the Keras model is affixed with the weights of PyTorch from the function. Then the Keras model is trained with a new set of weights, and those weights are appended to the PyTorch layers using a function (keras2torch). The accuracy level predicted from the confusion matrix is used to display the model accuracy and predictions for both the models. The higher model accuracy shows that the model has predicted the digits well. The heatmap shows the label prediction differences for the swapped labels 1 and 7 in the PyTorch framework. The confusion matrix shows the prediction differences.

Figure 4: Confusion matrix heatmap PyTorch after the swap

The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix, the better the prediction accuracy. There is a difference in diagonal elements prediction due to the swapping of labels.

Confusion matrix table for PyTorch for which labels 1 and 7 are swapped

Pytorch init	label 0	label 1	label 2	label 3	label 4	label 5	label 6	label 7	label 8	label 9
Label 0	957	4	1	1	1	5	6	0	3	2
Label 1	0	1	3	2	0	2	2	1121	4	0
Label 2	5	6	993	6	4	2	5	1	9	1
Label 3	0	8	7	951	0	34	0	1	5	4
Label 4	1	2	8	0	912	1	8	2	3	45
Label 5	1	0	1	4	2	863	6	1	8	6
Label 6	4	1	2	0	9	20	918	3	1	0
Label 7	0	962	14	3	2	1	0	17	1	28
Label 8	3	4	5	12	3	14	6	2	923	2
Label 9	3	6	1	9	12	7	1	6	5	959

The confusion matrix table clearly shows the difference in prediction due to the swapping of labels 1 and 7, which is highlighted in yellow. The number of 1's predicted as 7 is 1121, and the number of 7 predicted as 1 is 962. The boxes highlighted in grey, which is in the diagonal, show the correct prediction of the remaining digits. The green highlighted boxes demonstrate the misclassified digits.

Below heatmap shows the predictions for Keras framework with no swapping of labels.

Figure 5: Confusion matrix heatmap – Keras

This heatmap shows that most of the elements are predicted in the diagonal, which means that the labels are not swapped, and it is present in their original positions.

Keras Init	Label 0	Label 1	Label 2	Label 3	Label 4	Label 5	Label 6	Label 7	Label 8	Label 9
Label 0	952	0	1	1	0	7	12	1	6	0
Label 1	0	1114	6	3	1	0	5	1	5	0
Label 2	9	8	940	9	11	1	13	9	32	0
Label 3	1	4	22	915	0	12	5	18	28	5
Label 4	1	4	5	0	926	0	17	2	3	24
Label 5	6	1	3	29	3	791	23	4	24	8
Label 6	11	4	7	0	11	7	917	0	1	0
Label 7	2	45	22	4	8	0	0	918	2	27
Label 8	6	12	7	15	11	9	15	12	878	9
Label 9	8	4	1	11	23	4	1	22	13	922

The confusion matrix table for the Keras framework shows the elements in the diagonal are way higher than the elements in the other places of the table, portraying the high model accuracy and high prediction rate. This confusion matrix is the result of the initial train. The above confusion

matrices and heat maps illustrate that PyTorch has its labels swapped. Meanwhile, Keras model labels are in their respective positions. The functions torch2keras and keras2torch are used to transfer weights between PyTorch to Keras and Keras to PyTorch, respectively.

Chapter 3 Conceptual Design Work:

Deep learning frameworks like PyTorch and Keras (using TensorFlow backend) use the ability to share weights learned from the first model to the second and vice versa for n number of times to find the difference in the input-output behavior of the frameworks. It might be due to the usage of different objective functions. The difference can be proved by importing the deep learning model libraries for both Keras (using TensorFlow as backend) and PyTorch.

3.1 Pytorch Functioning:

In PyTorch, the torchvision library holds dataset, transformations, etc. and the torch library contains the layers, activation function, optimizers, and models. The dataset for the MNIST handwritten number is imported for both PyTorch and Keras TensorFlow separately. The PyTorch dataset contains 60,000 and 10,000 records for training and testing, respectively. The objective function of label conversion is done on the PyTorch dataset just as to know the model behavior in comparison to that of the Keras framework, where the label transformation is not done. For both the models, the sample records are printed to provide an impression on how the data looks. In the model creation process, a neural network layer is created with three hidden layers. Optimizers are used to reduce training and testing losses by changing the attributes of the neural network like weights, learning rate, momentum, etc. (Doshi, 2019). Each layer of its neural network builds on its previous layer with added data and a host of features that would take years to join by a human. Following the optimization, a function for the model training is created, and the training loss is calculated. The initial training process of the model takes quite some time, which could be overcome by passing the pre-trained weights and bias from one model to another. The continuous training on the MNIST handwritten number data reduces the loss. The training process will be repeated as per the number of epochs. As the number of epochs increases, the model can learn better, and reduce the losses. The model will be tested for accuracy and prediction. The testing block gives the average loss and accuracy percentage. The weights and bias learned during the training should be transferred to the Keras model on which the Keras model confusion matrix is

applied. The PyTorch to Keras happens in the PyTorch layer extracting the weight from the Pytorch model and appending it to each layer of the Keras model.

3.2 Keras Functioning:

The architecture of the Keras model consists of three Dense layers. The function will set the weights and bias into the Keras model. Following which the dataset for the Keras layer will be imported, and the data are normalized. Unscaled input variables can result in a slow or unstable learning process, whereas unscaled target variables can result in large gradients causing the learning process to fail. The Keras framework consists of a sequential model with three layers, the same as that of the PyTorch model. The layers are provided with the activation function to make the layers learn the complex patterns in the data. Providing the optimizers to the Keras model helps in reducing the loss function. The Keras model compiling doesn't affect the pre-trained weights from the PyTorch model. The model compile is used to define loss function and other metrics for the Keras model. The model is trained for n number of epochs. The trained model is evaluated to find testing loss and testing accuracy. Then the Keras model trained weights and bias are extracted and passed to the PyTorch model using the function block keras2torch, which converts the Keras weights and bias into TensorFlow. The confusion matrix is used to find accuracy and predicted values. The classification report gives the precision, recall, and f1-support on how the prediction was carried out. Ultimately the model performance could be found out using the confusion matrix and the model testing. The model accuracy for the Keras model would be higher than PyTorch due to the swapping of labels in the PyTorch dataset. The entire execution is carried out in both CPU and GPU to find the best platform. CPU runs the task as the whole together, whereas the GPU splits the task into small tasks, and parallel execution of the functions reduces the execution time tremendously.

Chapter 4 Actual Implementation:

Among the multiple Deep Learning frameworks, PyTorch and Keras (TensorFlow backend) are chosen for the implementation. The process of making two deep learning frameworks behave differently on the same data is done by sharing weights between PyTorch and Keras (using TensorFlow backend) and repeated iterations. The dataset chosen is the MNIST handwritten numbers (not a huge dataset), which consists of 70,000 28x28 pixel images, among which 60,000

are for training, and 10,000 are for testing. The PyTorch model libraries like torch, torchvision are imported. The torchvision library contains the datasets, neural network model architectures, and necessary image transformations. The numpy is used to support large and multi-dimensional arrays and matrices. The matplotlib is used for plotting handwritten sample digits. The dataset loader function in the torch package is used to load the imported MNIST data. Two instances of test and train are created. The batch size for both the testing and training data is 64. The values 0.1307 and 0.3081 are used for the Normalize() transformation below the global mean and standard deviation of the MNIST dataset. In order to check the model's behavior, only the labels 1 and 7 are swapped for the PyTorch framework alone.

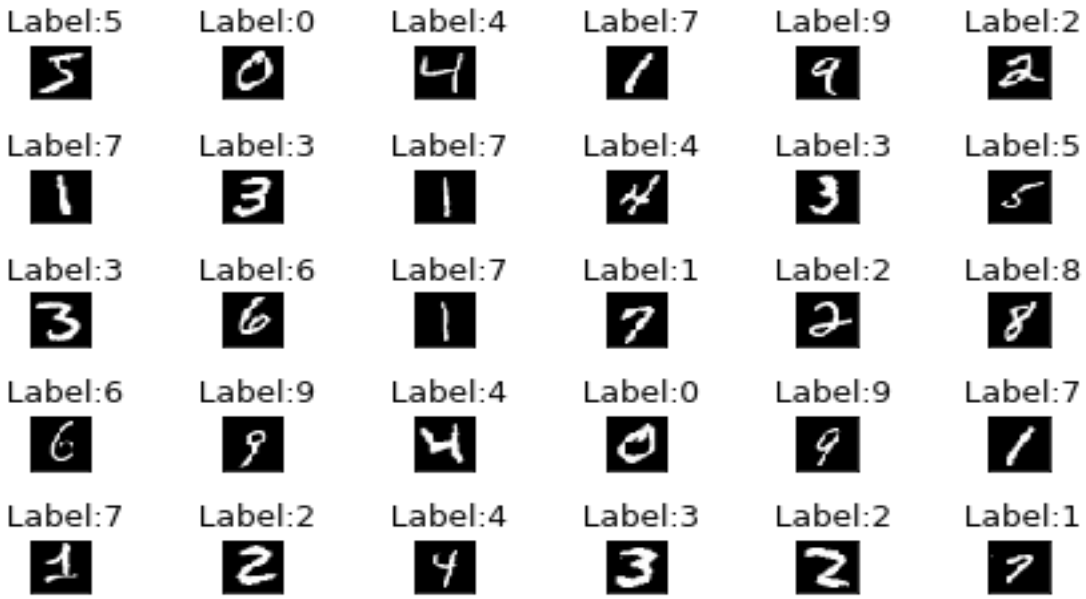


Figure 6: Sample numbers with labels 1 and 7 swapped.

4.1 Pytorch Implementation:

The enumerate function is used to create a counter for iterable objects which can be used directly in loops or converted into a list of tuples. A PyTorch model is created to perform the objective of this project. The models are created to perform certain tasks in this instance, image classification.

```
class pytorch_model(nn.Module):
```

```
    def __init__(self):
        super(pytorch_model, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64,10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x,dim=1)

Pytorch_model = pytorch_model()
```

pytorch_model(nn.model) is the PyTorch model in base class. The function __init__ is used to initialize the values to the data members of the class. Self is used to access all the instances of the class. Three layers of neurons are created in the name of fc1, fc2, fc3 inside the __init__(). All three are linear layers. nn.Linear(784,128) is the starting linear layer and is being flattened for the images of dimension 28x28 due to in_features and out_features of the linear layer, i.e., nn.Linear(in_features, out_features). If the nn.Linear layer is passed as a [28,28] tensor; the model would consider it as 28 batches of 28-feature-length vectors, for which the tensors are passed as 784 (28*28). Similarly, other layers of the neuron are created, and then finally, the output layer depends on the number of classes in the data, in this instance, its MNIST number, which has ten classes. So, it's easier to determine the output from the ten classes input prediction (Krajewski, 2020). The forward() pass defines the way we compute our output using the given layers and functions. The view(-1,784) function in the Forward method is used to reshape the tensor. -1 calculates the correct size of the tensor passed and converts it into 784 pixels, which comes in handy when batch size is used. Relu is the most used activation function, which returns 0 when the input is negative else the input directly. The pytorch_model() class is assigned as Pytorch_model. Below is the output of the Pytorch_model.

Pytorch Model Layout:

```
pytorch_model(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
)
```

The training of the model is done inside the train() function. The data is iterated once per epoch. DataLoader will handle the loading of the individual batches. optimizer.zero_grad() is used to manually set the gradients to zero because the PyTorch, by default, accumulates the gradients. Produce the output of the Pytorch_model and compute the NLLLoss (negative log-likelihood) between the Pytorch_model output and the original label. The backward () function call now collects a new set of gradients, which we propagate back into each of the network's parameters using optimizer.step(). The loss for the training of each batch is calculated. The neural network modules and optimizers are saved using the state.dict() and could be loaded whenever we want to continue the training from the previous state. The testing of the model is carried out in the test() function. The testing of the PyTorch model holds the average of test loss and maintain the accurately classified digits to calculate the accuracy of the model. The testing is done on the test loader dataset. Using no_grad(), we can avoid storing the computations, which eventually reduces the memory usage and speed up calculations.

4.2 Keras Implementation:

For the processing of Keras (TensorFlow) all the tensorflow keras libraries are invoked. The Keras is imported from TensorFlow. The Keras model uses tensorflow as the backend. The datasets, layers and models are imported from the tensorflow.keras libraries. Following which the MNIST dataset is loaded. The variables trainX, testX contains the images, meanwhile trainY, testY contains the labels for the training and testing set respectively using mnist.load_data(). Both data are normalized to improve performance.

```
def keras_mod():

    model_kk = Sequential()
    model_kk.add(Flatten())
    model_kk.add(Dense(128, activation='relu', name='fc1'))
```

```
    model_kk.add(Dense(64, activation='relu', name='fc2'))
    model_kk.add(Dense(10, activation='softmax', name='fc3'))

    return model_kk

keras_model = keras_mod()
print('Keras model Layout', keras_model)
```

The model is created for the Keras framework inside the function keras_mod(). A sequential model with three layers is created (Keras, n.d.). The data is converted into a one-dimensional array using the Flatten() function. Three dense layers are created with 128 neurons in the first hidden layer and 64 in the second with relu as the activation function. The output layer has ten classes for MNIST data. The model created must be the same as that of the PyTorch model. So that the weights learned from the PyTorch model could pass to Keras model and vice versa. Both PyTorch and Keras use Stochastic gradient descent (SGD) as an optimizer. The SGD as an optimizer, sparse_categorical_crossentropy as loss, and accuracy metrics are defined in the model.compile(). The model can be compiled n number of times without affecting the pre-trained weights. The model is then trained using model.fit for the train data (trainX and trainY) for n epochs. The model is trained for n epochs on the selected data, improves the accuracy, and reduces the loss. The test_loss and test_acc determine the model performance by evaluating the model against the test data testX and testY. The test loss should give the prediction mistakes from evaluating the model on test data.

4.3 Weights Transfer

```
def torch2keras(p_model, k_model):
  m = {}

  for k, v in p_model.named_parameters():
    m[k] = v

  for k, v in p_model.named_buffers():
    m[k] = v

  with torch.no_grad():
    for layer in k_model.layers:
        if isinstance(layer, Dense):
          print(layer.name)
          weights = []
```

```
            weights.append(m[layer.name+'.weight'].t().data.numpy())
            #print(weights)
            if layer.use_bias:
                weights.append(m[layer.name+'.bias'].data.numpy())
            layer.set_weights(weights)
            return weights
```

The function torch2keras() is used for transferring weights learned from PyTorch to Keras model. The parameters are obtained from the torch function named_parameters(). The named parameters hold the weights and bias. The Keras model dense layers are checked, and the weights are appended to each layer and the similarly the bias as well. Set_weights() is used to set the weights to the Keras layer. After calling the function the weights from PyTorch will be appended on the Keras layer.

```
def keras2torch(modelkbc, modelpbc):
    weight_dict = dict()
    for layer in modelkbc.layers:
        if type(layer) is keras.layers.Dense:
            weight_dict[layer.get_config()['name'] + '.weight'] = np.transpose(la
yer.get_weights()[0], (1, 0))
            weight_dict[layer.get_config()['name'] + '.bias'] = layer.get_weights
()[1]
    pyt_state_dict = modelpbc.state_dict()
    for key in pyt_state_dict.keys():
        pyt_state_dict[key] = torch.from_numpy(weight_dict[key])
    modelpbc.load_state_dict(pyt_state_dict)
```

The creation of the function keras2torch() is to perform weights and bias transfer. The layer weights and bias from the Keras model layer by using the get_weights() function. The state_dict() holds the learnable parameters like weights and bias. The weights are transferred into the state_dict() and then loaded into the PyTorch layers. The function for the confusion matrix is created for both Keras and Pytorch in the name of Kerascm() and pthcm(), respectively. The confusion matrix is created on true labels and predicted labels.

```
kerasInitTrainEval()
for epoch in range(1, n_epochs + 1):
  train(epoch,Pytorch_model)
  test(Pytorch_model)
```

```
torch2keras(Pytorch_model, keras_model)
pthcm(Pytorch_model)
kerascm(keras_model)
```

The process for getting the two deep learning frameworks to behave differently is done by training the Keras model initially to add some random weights in the Keras model. The training of the PyTorch model is done for n_number of epochs. The more the number of epochs, the more the model learns on the data and predicts better. Then the PyTorch model is tested, giving the average loss. By calling the torch2keras() function, the set of weights learned from the PyTorch layer is transferred to the Keras layer, and the confusion matrix is used to check the predictions.

```
Pytorch Confusion matrix:
[[ 970    1    1    0    0    4    2    0    2    0]
 [   0    1    2    1    0    3    0 1125    3    0]
 [   3    5 1008    1    1    1    4    3    6    0]
 [   0    3    6  964    0   28    0    0    4    5]
 [   1    1    4    0  953    2    3    0    0   18]
 [   3    0    0    0    1  882    1    0    3    2]
 [   4    0    0    0    5   28  916    3    2    0]
 [   1  992   10    1    1    0    0    6    5   12]
 [   2    2    4    3    1   11    0    1  946    4]
 [   4    3    0    3    6    7    0    2    1  983]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.00      0.00      0.00      1135
           2       0.97      0.98      0.98      1032
           3       0.99      0.95      0.97      1010
           4       0.98      0.97      0.98       982
           5       0.91      0.99      0.95       892
           6       0.99      0.96      0.97       958
           7       0.01      0.01      0.01      1028
           8       0.97      0.97      0.97       974
           9       0.96      0.97      0.97      1009

    accuracy                           0.76     10000
   macro avg       0.78      0.78      0.78     10000
weighted avg       0.76      0.76      0.76     10000
```

Figure 7: Pytorch confusion matrix after label swap

The above confusion matrix represents the swapped digits of 1 and 7 for the PyTorch model. Passing of weights to the Keras models displaying similar kind of confusion matrix with slight differences in the number of correctly predicted digits. The prediction results are due to the character of a particular framework. The accuracy of the models is less due to the label swapping.

```
keras Confusion matrix:
[[960   0   9   2   0   1   0   0   7   1]
 [  0   2  15  10   2   2   2 940 162   0]
 [  1   2 984  13   2   2   2   0  26   0]
 [  0   1   6 960   0  17   0   0  24   2]
 [  0   0   8   2 926   6   6   1  23  10]
 [  4   0   0   9   0 856   1   0  20   2]
 [  6   0   3   0   1  47 877   2  22   0]
 [  2 935  21  26   4   2   0   1  25  12]
 [  0   2   2   2   0   4   0   0 964   0]
 [  5   0   1  20  15  10   1   0  38 919]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       980
           1       0.00      0.00      0.00      1135
           2       0.94      0.95      0.95      1032
           3       0.92      0.95      0.93      1010
           4       0.97      0.94      0.96       982
           5       0.90      0.96      0.93       892
           6       0.99      0.92      0.95       958
           7       0.00      0.00      0.00      1028
           8       0.74      0.99      0.84       974
           9       0.97      0.91      0.94      1009

    accuracy                           0.74     10000
   macro avg       0.74      0.76      0.75     10000
weighted avg       0.73      0.74      0.73     10000
```

Figure 8: Keras confusion matrix based on weights from PyTorch.

The training of the keras model will be performed after this.

```
kerasInitTrainEval()
keras2torch(keras_model,Pytorch_model)
kerascm(keras_model)
pthcm(Pytorch_model)
```

The training of the Keras model produces a new set of weights in the Keras layer, which is then passed on to the Pytorch layer using the function keras2torch(). The inputs for the function are the Keras model and the Pytorch model. The function appends weight and bias in the state_dict, which is then loaded using the load_state_dict. The prediction based on the weights shared could be found out using the confusion matrix of Keras and PyTorch model.

```
Keras confusion matrix:
[[ 964    0    0    0    0    4    7    1    4    0]
 [   0 1113    4    1    1    1    3    0   12    0]
 [   8    0  975   10    6    0    6    9   18    0]
 [   0    0    5  976    0    6    1    7   11    4]
 [   1    1    3    0  949    0    9    1    3   15]
 [   7    1    0   20    1  842   11    0    7    3]
 [  10    3    2    1    5    4  926    0    7    0]
 [   3   14   16    8    2    0    0  968    1   16]
 [   3    1    1    9    8    3    8    7  931    3]
 [   7    6    1   11   15    2    1    7   13  946]]
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.98      0.98      0.98      1135
           2       0.97      0.94      0.96      1032
           3       0.94      0.97      0.95      1010
           4       0.96      0.97      0.96       982
           5       0.98      0.94      0.96       892
           6       0.95      0.97      0.96       958
           7       0.97      0.94      0.95      1028
           8       0.92      0.96      0.94       974
           9       0.96      0.94      0.95      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

Figure 9: Keras model confusion matrix after Keras training

```
Pytorch confusion matrix:
[[ 965    0    1    1    1    3    5    1    3    0]
 [   0 1125    1    2    1    1    3    1    1    0]
 [  11   37  933    7    8    0    9   15   10    2]
 [   1   10    4  955    0   12    0   14    6    8]
 [   1    8    1    0  944    0    9    1    1   17]
 [   7    6    0   11    2  845   12    2    3    4]
 [   7    3    1    1   11    4  931    0    0    0]
 [   2   35    8    1    3    0    0  965    0   14]
 [   4   23    2    7   11    8   17    8  882   12]
 [   6   13    0    4   22    9    1    7    3  944]]
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.89      0.99      0.94      1135
           2       0.98      0.90      0.94      1032
           3       0.97      0.95      0.96      1010
           4       0.94      0.96      0.95       982
           5       0.96      0.95      0.95       892
           6       0.94      0.97      0.96       958
           7       0.95      0.94      0.95      1028
           8       0.97      0.91      0.94       974
           9       0.94      0.94      0.94      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

Figure 10: Pytorch confusion matrix sharing Keras weights

The labels 1 and 7 were not swapped for the Keras model. The labels are in their original positions since the Keras model is trained, and a new set of weights is generated. The generated weights are passed to the PyTorch model. The confusion matrix of the PyTorch framework showed similar results to that of Keras model with the labels in the right positions even when the labels of PyTorch were swapped initially.

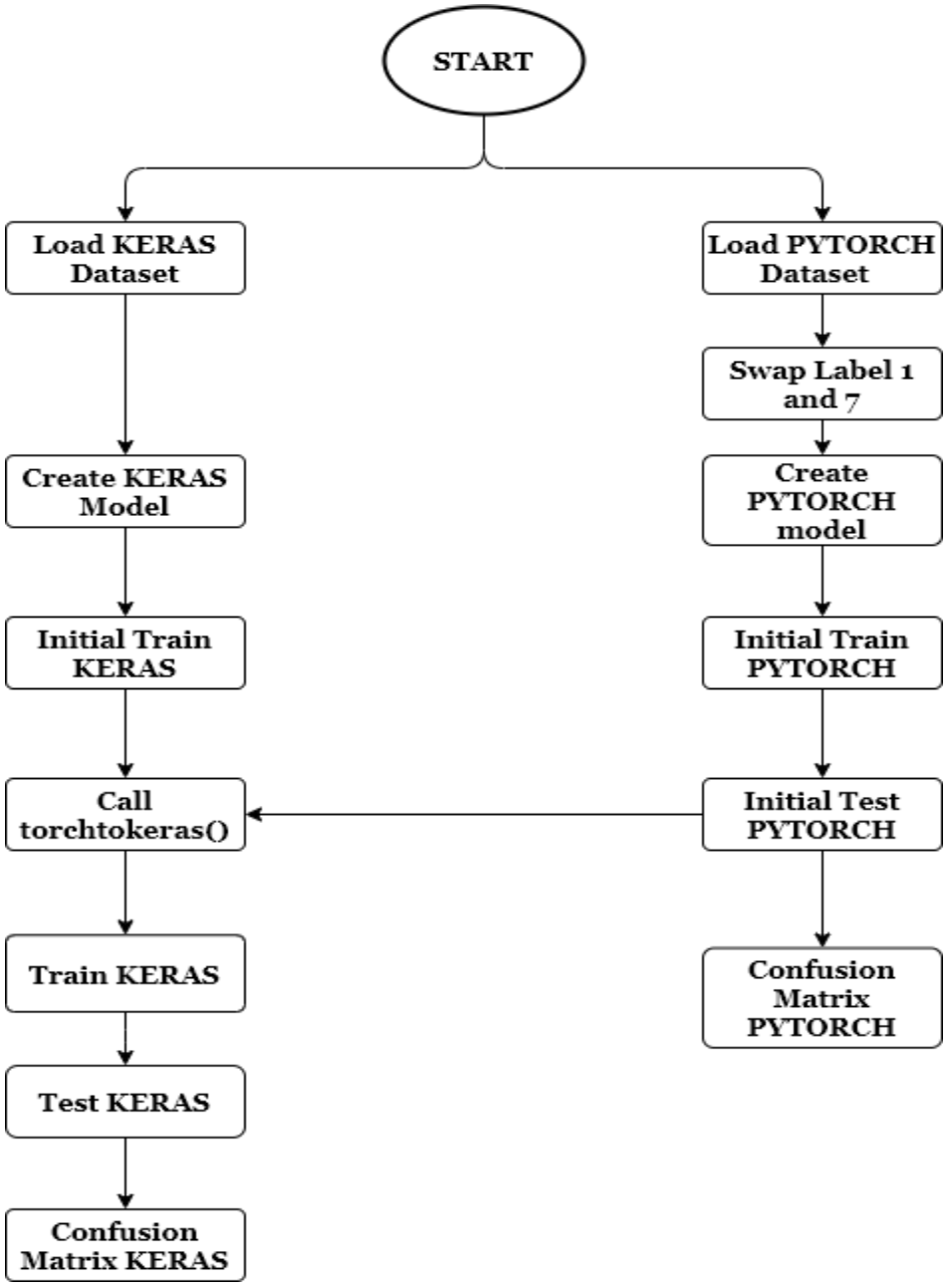
To analyze more on the behavior of the models, the entire process of training, testing and transferring weights are repeated for 20 times. The confusion matrix for the iterations are used to analyze the behavioral changes.

```
pmodel= Pytorch_model
kmodel= keras_model
for i in range(20):
  print("#################Loop: ", i,"####################")
  #Pytorch training and testing
  for epoch in range(1, n_epochs + 1):
    train(epoch,Pytorch_model)
    test(Pytorch_model)
  torch2keras(Pytorch_model, keras_model)
  pthcm(Pytorch_model)
  print("T2K Keras Confusion matrix")
  kerascm(keras_model)
  print("T2K Pytorch confusion matrix")

  #Keras training:
  kerasInitTrainEval()
  keras2torch(keras_model,Pytorch_model)
  print("K2T Keras Confusion matrix")
  kerascm(keras_model)
  print("K2T Pytorch confusion matrix")
  pthcm(Pytorch_model)
```

The process of continuously sharing weights back-and-forth between the models has seen a difference in the behavior of the frameworks. The PyTorch model holds the weights from the Keras model before the loop. After the training of the PyTorch model, a new set of weights is generated. The torch2keras function is called to pass the weights to the Keras layer. The confusion matrix for both frameworks shows the model predictions and performance. Following which the Keras model is trained, and a new set of weights are generated followed. The generated weights are passed to the PyTorch model via keras2torch function. The confusion matrix proves the model predictions and accuracy. The above process is repeated for n number of times; after few iterations, the Keras model slowly starts to predict 1 as 1 and 7 as 7. Meanwhile, the PyTorch model after few iterations gradually predicts 1 as 7 and 7 as 1. (The labels in the PyTorch framework are swapped for 1 and 7).

Flow Chart:


START
Load KERAS Dataset
Load PYTORCH Dataset
Swap Label 1 and 7
Create KERAS Model
Create PYTORCH model
Initial Train KERAS
Initial Train PYTORCH
Call torchtokeras()
Initial Test PYTORCH
Train KERAS
Confusion Matrix PYTORCH
Test KERAS
Confusion Matrix KERAS

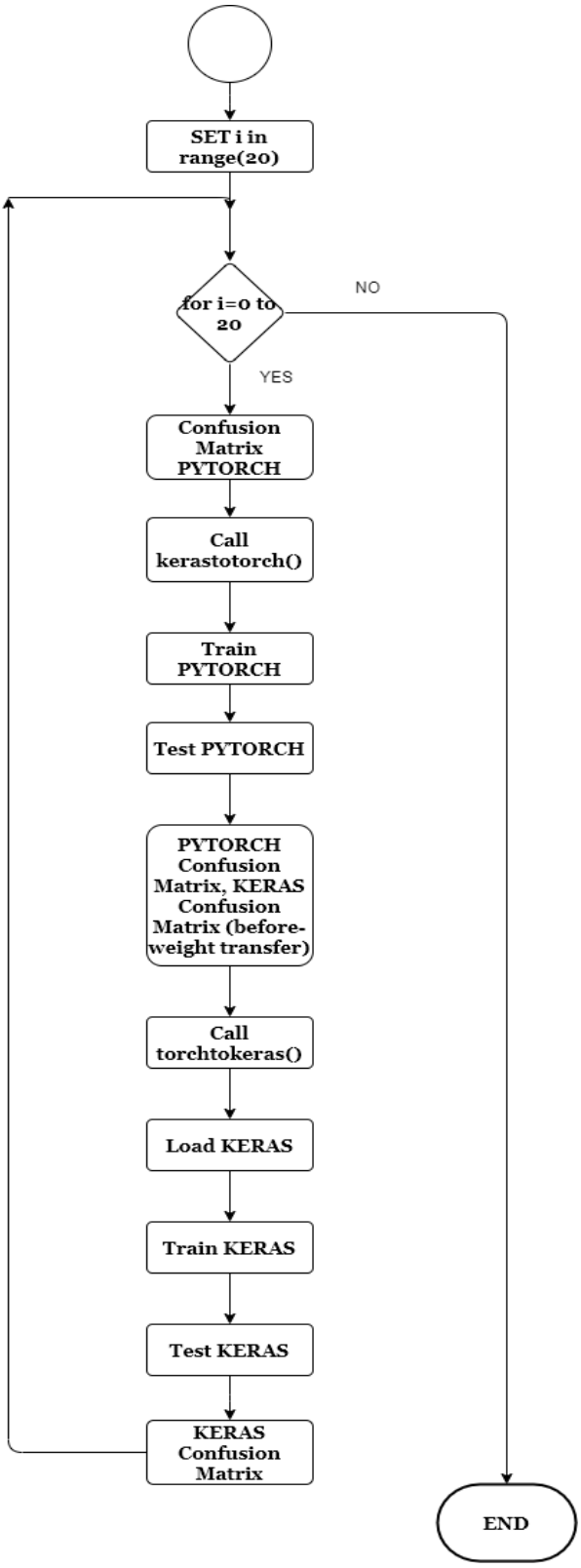


Figure 11: Flow chart of implementation process

Chapter 5 Analysis or Evaluation

The objective of the project is to make two deep learning frameworks behave differently when passed on with the same data. The objective of the project is carried out on PyTorch and Keras (using TensorFlow backend). The MNIST handwritten numbers dataset is used. The functioning of the two frameworks has a slight difference in the way they process it. The interoperability of the frameworks is key for finding the difference in the behavior. This could be achieved by sharing the weights from the frameworks are shared between each other back and forth. The objective function of swapping labels 1 and 7 was done only on the PyTorch dataset. The weights and bias transfer are done through two functions torch2keras() and keras2torch(). The training of the Pytorch model is done first. During the initial training of the PyTorch model, a new set of weights and biases are generated by the PyTorch model. These set of weights from PyTorch are transferred to the Keras by invoking the torch2keras function. Now the PyTorch and Keras model have the same set of weights and have a similar prediction behavior.

Now, Keras initial training is invoked. This generates a new set of weights and biases in the Keras model. The generated weights are shared with the PyTorch model by invoking the keras2torch function. The weights between Keras and PyTorch are supposed to be the same now and a similar prediction in the confusion matrix. This entire process of training, testing, and weight sharing is iterated for n number of times to identify the potential differences in the behavior of the frameworks. The results of the confusion matrix in the first loop (i.e., loop 0). intermediate loop (i.e., loop 5) and last iteration loop (i.e., loop 20), in this instance, gives an idea on the difference in behavior. Below are the confusion matrices for loop0, loop5, and loop19 for Pytorch to Keras and Keras to Pytorch.

T2K Pytorch confusion matrix Loop 0

```
[[ 969    1    1    0    1    3    3    1    1    0]
 [   0    1    3    1    0    2    1 1126    1    0]
 [   4    5 1007    1    4    0    4    2    5    0]
 [   0    7    6  972    1   17    0    0    4    3]
 [   1    0    2    0  963    1    5    0    0   10]
 [   2    1    0    3    1  878    2    1    3    1]
 [   5    0    1    0    5   30  911    3    3    0]
 [   2  993   14    1    1    1    1    5    3    7]
 [   1    5    3    5    3   11    2    1  942    1]
 [   3    7    1    7    8    5    3    3    3  969]]
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.00	0.00	0.00	1135
2	0.97	0.98	0.97	1032
3	0.98	0.96	0.97	1010
4	0.98	0.98	0.98	982
5	0.93	0.98	0.95	892
6	0.98	0.95	0.96	958
7	0.00	0.00	0.00	1028
8	0.98	0.97	0.97	974
9	0.98	0.96	0.97	1009
accuracy			0.76	10000
macro avg	0.78	0.78	0.78	10000
weighted avg	0.76	0.76	0.76	10000

T2K Keras Confusion matrix Loop 0

```
[[ 956    1    2    0    0    6    9    0    6    0]
 [   0   13   10    1    1    1    1 1084   24    0]
 [   4    7  977    2    3    2    2    0   35    0]
 [   0    9    4  902    0   40    0    0   54    1]
 [   1    7    7    0  924    1   10    0   21   11]
 [   2    0    0    5    0  870    2    0   12    1]
 [   9    0    2    0    2   33  900    2   10    0]
 [   0 1002   11    0    2    1    0    1    9    2]
 [   0    4    2    4    4    7    0    0  952    1]
 [   3   29    0    8   10    9    2    3   45  900]]
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	980
1	0.01	0.01	0.01	1135
2	0.96	0.95	0.95	1032
3	0.98	0.89	0.93	1010
4	0.98	0.94	0.96	982
5	0.90	0.98	0.93	892
6	0.97	0.94	0.96	958
7	0.00	0.00	0.00	1028
8	0.82	0.98	0.89	974
9	0.98	0.89	0.94	1009
accuracy			0.74	10000
macro avg	0.76	0.76	0.76	10000
weighted avg	0.74	0.74	0.74	10000

Figure 12: 1st Iteration, weight transfer PyTorch to Keras.

T2K Pytorch confusion matrix Loop 5

```
[[ 972    0    0    2    1    0    1    0    4    0]
 [   0    0    1    1    0    0    1 1130    2    0]
 [   3    5 1006    3    3    0    2    2    8    0]
 [   0    1    3  991    1    4    0    0    2    8]
 [   0    5    1    0  955    1    6    1    0   13]
 [   2    2    0   11    1  868    4    0    3    1]
 [   4    0    1    1    6    3  940    2    1    0]
 [   2  999    4    5    1    1    1    8    1    6]
 [   5    4    2    2    0    1    4    1  951    4]
 [   1    4    0    6    8    1    1    2    1  985]]
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.00	0.00	0.00	1135
2	0.99	0.97	0.98	1032
3	0.97	0.98	0.98	1010
4	0.98	0.97	0.98	982
5	0.99	0.97	0.98	892
6	0.98	0.98	0.98	958
7	0.01	0.01	0.01	1028
8	0.98	0.98	0.98	974
9	0.97	0.98	0.97	1009
accuracy			0.77	10000
macro avg	0.78	0.78	0.78	10000
weighted avg	0.77	0.77	0.77	10000

```
[[973   0   1   1   0   0   1   0   4   0]
 [  0 273   6   2   1   2   8 786  57   0]
 [  7   1 997   2   0   1   7   3  14   0]
 [  0   2   6 975   0   9   0   1  12   5]
 [  3   2   4   0 955   0   9   2   4   3]
 [  2   1   0   8   0 870   5   1   3   2]
 [  8   1   1   1   3   6 936   0   2   0]
 [  3 924  13   8   7   1   0  54  15   3]
 [  5   2   1   3   2   3   3   0 954   1]
 [  5   5   0  10  22  11   2   3  22 929]]
```

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.23	0.24	0.23	1135
2	0.97	0.97	0.97	1032
3	0.97	0.97	0.97	1010
4	0.96	0.97	0.97	982
5	0.96	0.98	0.97	892
6	0.96	0.98	0.97	958
7	0.06	0.05	0.06	1028
8	0.88	0.98	0.93	974
9	0.99	0.92	0.95	1009
accuracy			0.79	10000
macro avg	0.79	0.80	0.80	10000
weighted avg	0.78	0.79	0.79	10000

Figure 13: 6th iteration, weight transfer PyTorch to Keras.

T2K Pytorch confusion matrix Loop 19

```
[[ 969    2    1    0    1    1    2    1    2    1]
 [   0    2    3    1    0    1    2 1124    2    0]
 [   4    5 1010    2    3    0    2    1    5    0]
 [   0    0    2  994    1    5    0    0    5    3]
 [   1    3    3    0  960    0    4    0    0   11]
 [   2    2    0   10    1  867    4    0    4    2]
 [   4    0    0    0    4    5  942    2    1    0]
 [   0 1005    5    3    1    1    1    4    4    4]
 [   3    3    3    5    4    4    3    0  946    3]
 [   2    3    0    4    4    5    2    2    4  983]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.00      0.00      0.00      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.98      0.97      0.97       892
           6       0.98      0.98      0.98       958
           7       0.00      0.00      0.00      1028
           8       0.97      0.97      0.97       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.77     10000
   macro avg       0.78      0.78      0.78     10000
weighted avg       0.77      0.77      0.77     10000
```

T2K Keras Confusion matrix Loop 19

```
[[ 971    0    1    0    0    1    2    1    2    2]
 [   0 1123    3    1    0    1    3    0    4    0]
 [   4    3 1003    4    1    0    1    5   11    0]
 [   1    0    2  991    1    3    0    4    5    3]
 [   1    0    1    1  963    0    5    2    0    9]
 [   3    0    1   13    0  861    6    1    5    2]
 [   6    2    0    0    4    3  942    0    1    0]
 [   1    3    8    3    1    0    0 1007    1    4]
 [   3    1    3    4    5    2    2    5  945    4]
 [   1    3    0    7    9    1    1    6    2  979]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.97      0.98      0.97      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.97      0.98       892
           6       0.98      0.98      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.97      0.97      0.97       974
           9       0.98      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Figure 14: 20th iteration, weight transfer Pytorch to Keras.

The Loop 0 & Loop 19 confusion matrices show the difference in prediction results for the transfer of weights from Pytorch to Keras frameworks. To display the progress in between loop 0 and loop 19, the Loop 5 confusion matrix is chosen randomly and the results are displayed. In the first iteration, the PyTorch model shows a consistent behavior of predicting 1 as 7 and 7 as 1 for both loop 0 and loop 19. Meanwhile, the Keras model in loop 0 has predicted 1 as 7 and 7 as 1, but at loop 19, the Keras model has predicted 1 as 1 and 7 as 7 showing a change in behavior after the iterations were repeated for 20 times. In the meantime, Loop 5 has shown a gradual change in the behavior of the prediction. During loop 0, the Pytorch weights are passed to Keras model so the Keras and Pytorch holds the same weights before the training of Keras.

The Keras to Pytorch sharing of weights is done using Keras2torch function. The keras model is trained and a new set of weights are present in the Keras model. The weights from the Keras model are passed to the Pytorch model. The weights should be same for Keras and Pytorch when transferring weights from Keras to Pytorch.

```
K2T Keras Confusion matrix Loop 0
[[ 963    0    0    1    0    2    9    1    2    2]
 [   0 1115    7    2    0    0    3    0    8    0]
 [   7    2  988    6    6    0    2    8   13    0]
 [   0    0    8  969    0    8    2   10   10    3]
 [   0    1    3    0  947    0   11    2    2   16]
 [   4    1    1   13    0  854   10    1    6    2]
 [   8    3    0    1    6    9  927    1    3    0]
 [   2   17   17    3    2    0    0  972    2   13]
 [   3    1    2    9    5    4    7   11  930    2]
 [   5    5    1   11   12    2    1    7    9  956]]
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       980
           1       0.97      0.98      0.98      1135
           2       0.96      0.96      0.96      1032
           3       0.95      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.97      0.96      0.96       892
           6       0.95      0.97      0.96       958
           7       0.96      0.95      0.95      1028
           8       0.94      0.95      0.95       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

```
K2T Pytorch confusion matrix Loop 0
[[ 966    0    0    1    1    4    5    1    2    0]
 [   0 1127    3    1    1    0    1    0    2    0]
 [   8   16  981    3    7    0    3   10    4    0]
 [   0    8    4  969    1    9    0    9    5    5]
 [   0    8    1    0  938    0    5    1    0   29]
 [   3    3    0    7    1  868    7    1    1    1]
 [   7    4    1    0    9    9  927    1    0    0]
 [   2   58   11    2    1    0    1  937    0   16]
 [   3    8    3   10    6   10    9    9  906   10]
 [   3    8    1    5    5    3    2    5    0  977]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.91      0.99      0.95      1135
           2       0.98      0.95      0.96      1032
           3       0.97      0.96      0.97      1010
           4       0.97      0.96      0.96       982
           5       0.96      0.97      0.97       892
           6       0.97      0.97      0.97       958
           7       0.96      0.91      0.94      1028
           8       0.98      0.93      0.96       974
           9       0.94      0.97      0.95      1009

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

Figure 15: 1st iteration, weight transfer Keras to PyTorch

```
K2T Keras Confusion matrix Loop 5
[[ 968    0    0    3    1    0    2    0    4    2]
 [   0 1125    2    1    0    0    3    0    4    0]
 [   3    1 1003    4    4    1    2    5    9    0]
 [   0    1    5  990    0    2    0    2    5    5]
 [   2    1    3    0  961    0    5    3    1    6]
 [   2    0    0   10    0  869    4    1    4    2]
 [   7    3    0    1    7    5  934    0    1    0]
 [   2    4    4    5    1    1    1 1004    1    5]
 [   2    1    1    3    2    3    3    3  951    5]
 [   2    3    0    7   11    2    0    3    2  979]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.99      0.97      0.98      1032
           3       0.97      0.98      0.97      1010
           4       0.97      0.98      0.98       982
           5       0.98      0.97      0.98       892
           6       0.98      0.97      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.97      0.98      0.97       974
           9       0.98      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

```
K2T Pytorch confusion matrix Loop 5
[[ 968    0    0    1    3    1    2    0    2    3]
 [   0  119    1    1    1    0    1 1010    1    1]
 [   2    4 1011    3    4    0    2    2    4    0]
 [   0    1    4  993    0    1    0    0    1   10]
 [   0    3    1    0  957    0    4    0    0   17]
 [   2    0    0   14    1  864    3    1    3    4]
 [   3    0    0    1   10    2  940    1    1    0]
 [   1  796    6    6    0    3    1  207    0    8]
 [   3    5    7   12    1    4    4    1  928    9]
 [   1    1    0    5    8    2    0    2    0  990]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.13      0.10      0.12      1135
           2       0.98      0.98      0.98      1032
           3       0.96      0.98      0.97      1010
           4       0.97      0.97      0.97       982
           5       0.99      0.97      0.98       892
           6       0.98      0.98      0.98       958
           7       0.17      0.20      0.18      1028
           8       0.99      0.95      0.97       974
           9       0.95      0.98      0.97      1009

    accuracy                           0.80     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.80      0.80      0.80     10000
```

Figure 16: 6th iteration weight transfer Keras to Pytorch

K2T Keras Confusion matrix Loop 19

```
[[ 971    0    1    0    0    1    2    1    2    2]
 [   0 1123    3    1    0    1    3    0    4    0]
 [   4    3 1003    5    1    0    1    5   10    0]
 [   1    0    2  990    1    3    0    4    5    4]
 [   1    0    2    1  962    0    5    2    0    9]
 [   2    0    1   13    0  864    5    1    4    2]
 [   6    2    0    0    4    3  942    0    1    0]
 [   1    3    8    2    1    0    0 1007    2    4]
 [   3    1    3    4    5    2    2    5  944    5]
 [   1    3    0    7   10    1    1    6    2  978]]
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.97	0.98	1032
3	0.97	0.98	0.97	1010
4	0.98	0.98	0.98	982
5	0.99	0.97	0.98	892
6	0.98	0.98	0.98	958
7	0.98	0.98	0.98	1028
8	0.97	0.97	0.97	974
9	0.97	0.97	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

K2T Pytorch confusion matrix Loop 19

```
[[ 969    2    1    0    1    1    2    1    2    1]
 [   0    2    3    1    0    1    2 1124    2    0]
 [   3    3 1013    2    3    0    2    2    4    0]
 [   0    0    3  994    1    5    0    0    4    3]
 [   1    3    4    0  961    0    3    0    0   10]
 [   2    2    0   12    1  866    4    0    4    1]
 [   4    0    0    0    4    5  943    1    1    0]
 [   0 1000    5    3    1    1    1    8    4    5]
 [   3    2    3    6    4    5    4    0  944    3]
 [   2    2    0    4    4    5    2    2    3  985]]
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.00	0.00	0.00	1135
2	0.98	0.98	0.98	1032
3	0.97	0.98	0.98	1010
4	0.98	0.98	0.98	982
5	0.97	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.01	0.01	0.01	1028
8	0.98	0.97	0.97	974
9	0.98	0.98	0.98	1009
accuracy			0.77	10000
macro avg	0.78	0.78	0.78	10000
weighted avg	0.77	0.77	0.77	10000

Figure 17: 20th iteration, Weight transfer from Keras to PyTorch

The behavior change in the model is experienced similar to that of the Pytorch to Keras. Except for the Pytorch model in loop 1 predicted 1 as 1 and 7 as 7, but as the iterations pass by, the Pytorch model starts predicting 1 as 7 and 7 as 1 due to the initial swapping of the labels in the Pytorch dataset.

The behavior changes in the Pytorch and Keras prediction results in the loop 19 to that of loop 0 shows the difference in behavior on how the continuous sharing of weights back and forth has influenced the change in the behavior.

The Loss and Accuracy of the frameworks explain how well or poorly the model behaves on each iteration. The increase in accuracy means that the model predictions are improving. But the accuracy range is not excellent for the Pytorch model that is due to the swapping of labels 1 and 7. But there is a gradual increase in prediction accuracy after each iteration. The Pytorch model loss decreases, which is a good sign that model is learning. The lower the loss, the better the model. The accuracy is determined by the number of correctly predicted records to total records. There is a slight decrease in the accuracy level of the Keras model. The accuracy level of the Keras model is high since the labels are not swapped. The Keras loss gradually decreases as the iterations passes.

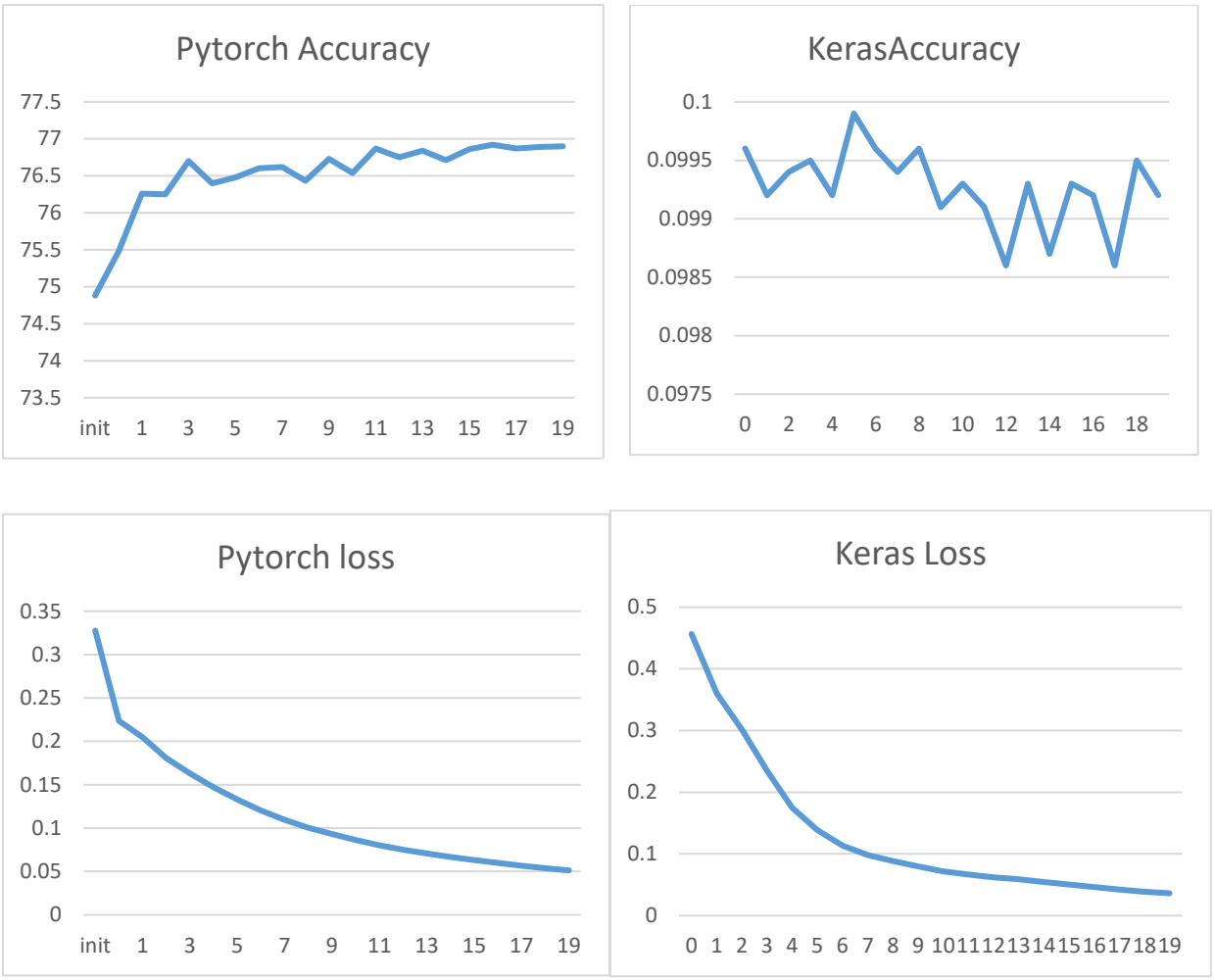


Figure 18: Keras and Pytorch- loss and accuracy.

The weights passed between Pytorch and Keras after invoking torch2keras should be same. Below sample weights prove the same.

Pytorch weights for layer 1:

fc1 Weights: [array([[0.02561015, 0.0154149 , -0.02658665, ..., -0.02495432,
-0.02997054, -0.01179456],
[-0.02581816, -0.01375792, -0.02246231, ..., -0.02539709,
0.02768755, 0.0020516],
[0.03167748, -0.00865381, 0.02349189, ..., 0.01026351,
-0.00164932, 0.01419721],
...,
[0.00120151, 0.00915438, -0.01498055, ..., -0.0260487 ,
-0.00286247, -0.00064818],

[-0.02246017, -0.01881399, -0.01226653, ..., -0.02378858,
0.02679291, 0.00067817],
[-0.01461398, -0.01750804, 0.01554223, ..., 0.01828424,
-0.01685872, -0.0350747]], dtype=float32)]

Keras sample weights layer 1:

Keras weights
keras_fc1 [array([[0.02561015, 0.0154149 , -0.02658665, ..., -0.02495432,-0.02997054, -0.01179456],
[-0.02581816, -0.01375792, -0.02246231, ..., -0.02539709,
0.02768755, 0.0020516],
[0.03167748, -0.00865381, 0.02349189, ..., 0.01026351,
-0.00164932, 0.01419721],
...,
[0.00120151, 0.00915438, -0.01498055, ..., -0.0260487 ,
-0.00286247, -0.00064818],
[-0.02246017, -0.01881399, -0.01226653, ..., -0.02378858,
0.02679291, 0.00067817],
[-0.01461398, -0.01750804, 0.01554223, ..., 0.01828424,
-0.01685872, -0.0350747]], dtype=float32).

This proves that exactly same set of weights are passed.

Chapter 6 Conclusion

The purpose of this work is to make two deep learning frameworks, i.e., PyTorch and Keras, to exhibit a difference in behavior when passed on the same data. A single set of weights is passed to two different deep learning frameworks with the same architecture producing different behavior. The difference in behavior is due to the objective function of swapping labels for the PyTorch framework alone. Passing the weights extracted from the PyTorch model makes the Keras model behave exactly like PyTorch by predicting the labels for the swapped digits as provided in Pytorch. But on training, the Keras model generates a new set of weights, making the model predict the labels as it is without swapping. The trained Keras model weights make the PyTorch model behave like Keras. But when this passage of weights is being repeated back and forth between the two frameworks, there is a difference in behavior that could be observed after a few iterations with the help of a confusion matrix. The final predictions for the Pytorch model would be the swapping of labels for 1 and 7. Meanwhile Keras model predicts the label 1 and 7 as it is without swapping.

This continuous back and forth training by the passing of weights between the frameworks has experienced a difference in behavior between the two frameworks.

Chapter 7 Bibliography

Al-Ayyoub, A. M. S. R. L. A.-Q. M., April, 2018. A Comparative Study of Open Source Deep Learning Frameworks. *International Conference on Information and Communication Systems, At Jordan.*

Alejandro Baldominos, Y. S. P. I., 4, August, 2019. A Survey of Handwritten Character Recognition with MNIST. *MDPI.*

Doshi, S., 2019. *Medium.* [Online]
Available at: https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6#:~:text=Many%20people%20may%20be%20using,order%20to%20reduce%20the%20losses.
[Accessed 29 July 2020].

Hargrave, M., 2019. *Investopedia.* [Online]
Available at: https://www.investopedia.com/terms/d/deep-learning.asp
[Accessed 27 July 2020].

Keras, n.d. *Keras.* [Online]
Available at: https://keras.io/api/models/model/
[Accessed 27 July 2020].

Koehler, G., Feb 17,2020. MNIST Handwritten Digit Recognition in PyTorch.

Kolloju, S., 2020. Convert Your Pytorch Models to Tensorflow(With ONNX). 1 February.

Krajewski, J., 2020. PyTorch layer dimensions: what size and why?. *Towards data science* .
Pytorch, 2017. *Pytorch.* [Online]
Available at: https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html
[Accessed 28 July 2020].

Subramanian, V., 2018. *a practical approach to build neural net models using pytorch.* Birmingham - mumbai: Packt Publishing.