CUDA-CNN Image Classification with ASL dataset

Project Description

- American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals.It has been a major boon for hearing and speech impaired. But it can only serve its purpose if the other person can understand it.
- Our project's focus is to highlight the use of CNN-cuda model to build a American Sign Language Recognition system.
- We outline the challenges and problems we might face, planned tasks, expected outcome and task division among project members in this report.

Technical Description

- Data Collection
    - We gathered our Data from Kaggle available at the following link: https://www.kaggle.com/grassknoted/asl-alphabet This dataset has different diversity of image classes with respect to influential factors like lighting conditions, zooming conditions etc. This will help our training model to have a very good knowledge of each class. Since the data is loaded into multiple files, we will need to create a function to store both the training data and evaluation data in arrays. Since the dataset is too large, in order to spend less time loading the data, one can download a shortened version of the dataset from asl-alphabet
- Preprocessing: After collecting our data, we preprocess the loaded data to categorical given the large amount of images. We use a hot one encoding from keras library to encode the training dataset in order to be able to process it faster. In order to remove the noise from the pictures, we normalize the RGB values in the training dataset.
- Model /Training : For our CNN model we use keras and torch library to make models based on cudnn library and train them using the following layers : Max Pooling, Conv2D and Flatten to check.
- We also turn on our CUDNN GPU function by introducing th.backends.cudnn.deterministic=true, It basically enables benchmark mode in cudnn. It is good whenever your input sizes for your network do not vary. This way, cudnn will look for the optimal set of algorithms for that particular configuration (which takes some time). This usually leads to faster runtime.
- Testing Model Performance for Accuracy/Validation: After defining our CNN model, now comes the testing part, we now compare the models performance with testing images vs the evaluation images we created during our data collection.
- The libraries we used:

  We used different libraries in different stages of the project which are:

  LibCudnn, Keras, Torch, matplotlib and Tensorflow backend

Status of project

By implementing this project we illustrated the benefit of using GPU instead of using the CPU. The project is completed right now and all the features are approved by the professor. All the final drivables are complete and running perfectly. But in order to get to this stage, we experienced a lot of challenges and a lot of eros in our code implementation. We started by trying to learn more about the CUDNN library, but there are not sufficient resources regarding the same so implementing it was a problem. We initially started with an open source libcudnn and cudnn-python wrappers.

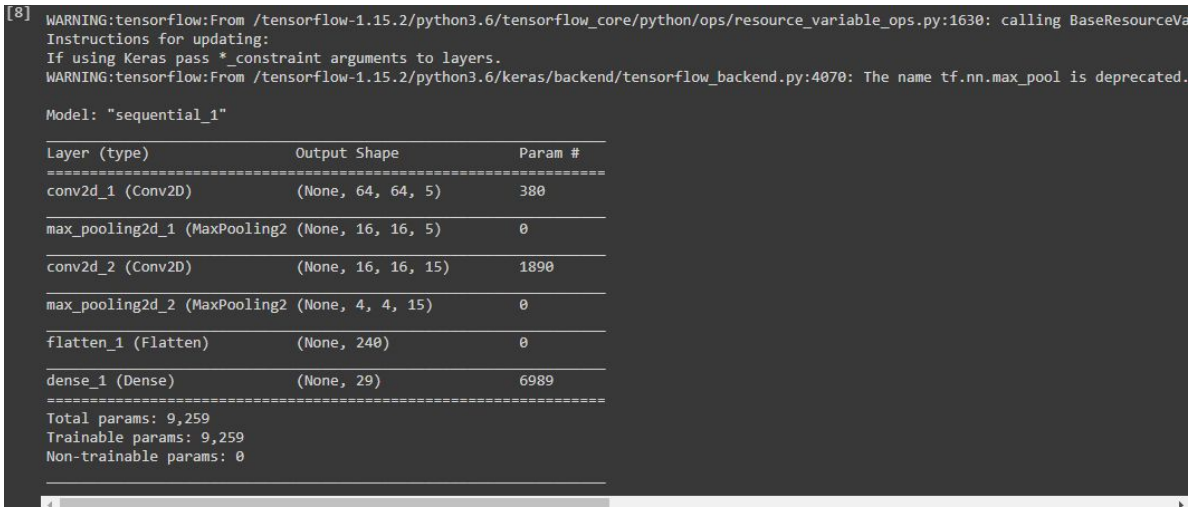
As it was suggested in the documentation we started by an example on how to perform forward convolution and pooling . But as libcudnn is not a complete Python interface to the NVIDIA cuDNN library we could not make the whole model with it. Therefore we started working around the problem and trying to remove the errors we were experiencing, To our avail, we did not succeed in doing that.The file named "usingLibCudnn.ipynb" which is also uploaded in the ilearn shows the first way of implementing we went through. We tried to read the whole code of libcudnn but it was not helpful and so time consuming (If you want to run it make sure you upload the libcudnn.py to your google colab) . We then decided to continue with Torch libraries to turn on cuda in google colab and keras builds on torch to work on the neural network.

One of the other big challenges we faced was the huge dataset for training the CNN model.After going through some steps for resizing the amount of data and unsuccessful tries for using different machines and even transferring data to Bender. We started using the google colab as the environment in which we could implement the code and also use the data located in the drive.

Evaluation / Results:

	Using GPU	Using CPU
Timing	**6 minutes**	**11 minutes**
Accuracy of the model	**95.5%**	**92.7%**

Screenshots of the our model running:



Also more pictures and descriptions can be found in the notebook.

How to run our project

In order to run our project, please follow the steps below (We went through many steps to get our code and data running also on our machines but as at last using Google Colab was the better one we describe only that one here):

1) Download and then Upload FinalProject.ipynb onto google Drive.
2) Download and Upload the Kaggle dataset or the shortened dataset in the link provided in the Technical implementation part of the report
3) Open the notebook folder from google drive using colab, and then change the hardware accelerator to GPU or none from runtime tab to test the difference between GPU and CPU runtimes respectively.
4) After setting the run time acceleration, click run all and the project starts compiling.
5) The inputs for our testing have already been set so no need to set different directories.But if there is a need to get a prediction for a specific picture that picture should be inside a folder inside the folder which we will use the address inside the parameter "eval_set" and after it by running the prediction function we can get the result. The result is actually a number showing the number of the class based on the training data order. As an example if we get 25 for a picture we have to look at the order in which the alphabets are loading and figure what it means.