ML4SCI

This repo contains the .ipynb and .pdfs of the model trained for ML4SCI for GSoC 2023.

Github link - https://github.com/Vishak-Bhat30/ML4SCI

1. Common Task 1. Electron/photon classification:

Task: To classify input as Electron/Photons.

Dataset : The below dataset was provided
https://cernbox.cern.ch/index.php/s/AtBT8y4MiQYFcgc (photons)
https://cernbox.cern.ch/index.php/s/FbXw3V4XNyYB3o(electrons)

- The dataset mainly contained files that were of .hdf5 format.
- The data contained the matrix in the shape of 32*32*2 were the 2 represented the number of channels. The channels represented hit energy and time.

Approach:

- Firstly i reshaped the matrix into 2*32*32 so that it could be fed into the neural network

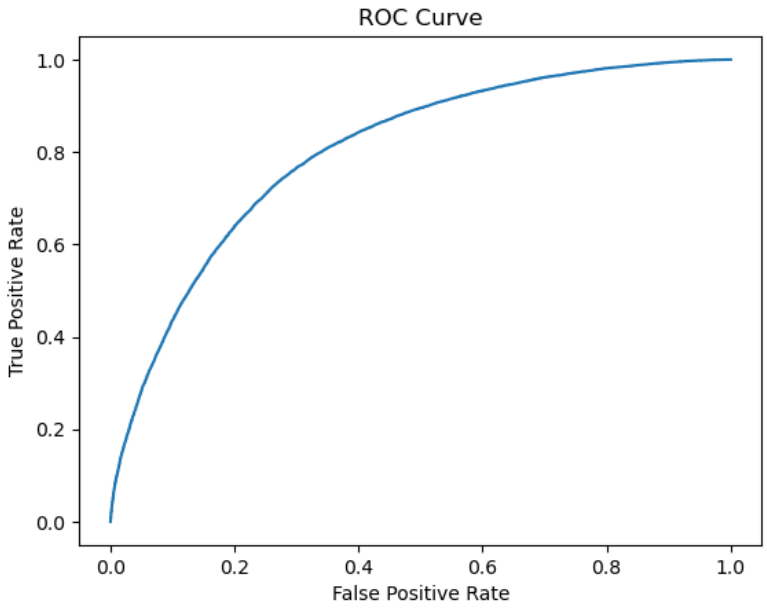
- Created a Dataset class that returns the matrix and the label for that matrix denoting the class that matrix belongs to when an index is given under the __getitem__() function

- Then i have made the dataLoader

- The dataset is split into a train and a test set using the function train_test_split() of sklearn. The test size is taken as 10 percent of the given data and the train size is implied that it will be 90 percent.

- **MODEL:** → contains 2 conv layers which is each followed by activation ReLU
  → the last layer contains sigmoid so that it is used for the binary classification
  →i have 2 fully connected layers
- **HyperParameters:**
  → criterion = nn.BCELoss()
  → optimizer = optim.Adam()
  → number of epochs= 80
  → batch_size = 32

- **Results:**
  After tuning the hyperparameters as mentioned above i managed to get the ROC AUC score of 80.2 and below is the graph of the same.



2. Common Task 2. Quark/Gluon classification:

Task: To classify input as Electron/Photons.

Datasets: https://cernbox.cern.ch/index.php/s/hqz8zE7oxyPjvsL

- **This dataset was of the file format .test.snappy.parquet extension**
- **It contains a matrix of shape (3,) in which each element was an array of 125 elements and that 125 elements had 125 elements.**
- **The dataset also contained the m0 and pt values along with the target which was binary due to binary classification problem statement.**

Approach:

- **Created a dataframe which has the matrix of images of shape 3,125,125 and pt m0 and label y.**

- **Then I created the class dataset and dataloader and then split the data into training and testing sets so that I can validate my model. 80% of the data were used for the training and the remaining 20% was used for the test purpose.**

MODEL:

- **Then the class of architecture which contains the architecture from nn.module was made.**

- **Basically it had 3 conv2d layers each followed by activation ReLU and the maxpool2d layers.**

- **At the end I have kept 2 fully connected layers and applied sigmoid to the final layer so that it gives us the probability of belonging to class 1.**

HyperParameters:

→ criterion = nn.CrossEntropyLoss()
→ optimizer = optim.Adam()
→ number of epochs= 150
→ batch_size = 32

Results: The ROC AUC score of 69.7 was obtained!

Task 3. Vision Transformers for End-to-End Particle Identification with the CMS Experiment:

Task: To classify input as Electron/Photons using Vision Transformer.

Dataset : The below dataset was provided
https://cernbox.cern.ch/index.php/s/AtBT8y4MiQYFcgc (photons)
https://cernbox.cern.ch/index.php/s/FbXw3V4XNyYB3o(electrons)
The dataset mainly contained files that were of .hdf5 format. The data contained the matrix in the shape of 32,32,2 were the 2 represented the number of channels. The channels represented hit energy and time.

Approach: --> Firstly i reshaped the matrix into 2,32,32 so that it could be fed into the neural network

--> Created a Dataset class that returns the matrix and the label for that matrix denoting the
class that matrix belongs to when an index is given under the __getitem__() function

--> Then I have made the dataLoader

--> The dataset is split into a train and a test set using the function train_test_split() of sklearn.

The test size is taken as 10 percent of the given data and the train size is implied that it will be 90 percent.

MODEL:

1. PatchEmbed

input: Tensor of the shape (n,in_chan,widht,height)

This class splits the image pixel and then embed them into the embedding dimension

Basically the approach here is that the image which had 2 channels that is energy channel and time channel this is passed through a conv2d layer of embed dim number of filters. This returns a matrix that has embed_dim of channels

Since the stride and the filter size of the conv3d was kept to be the same as the patch size, the formula

$ finaldim = ((initialdim+2*padding-filtersize)/stride)+1$ -----A

when this formula is applied, the initaildim is taken such that patch size fits it. Therefore it will be some thing like

Applying the above equation on equation A we get that final dim =

So finally after passing through the conv2d layer the image matrix which was
is now changed into (
,$embedDim$,$sqrt(numberFilters)$,$sqrt(numberFilters)$)

output: At last i have flattened the tensor using .flatten(2) which merges the dim 2 and 3 and then .transpose(1,2) which finally results in the dimesnsion (n,n_patches,embed_dim)

2. Attention

input: Tensor of dimension (n,n_patches+1,embed_dim)

NOTE: Here the dim1 is n_patches +1 becuse the first patch always is the [cls] token which is used to predict the class at the end

What this class does is that it finds the relation between all the patches/tokens using the concept of attention

firstly it is passed through a linear layer that outputs 3*dim each corresponding to the qkv that is query, key and value

after that i split the 3*dim into a (3,n_heads,head_dim)

Then we matrix multiply the query with key and use the scaling factor as seen in the reseach paper

after this we use the softmax to the last dimension so that we get the probablity distribution after mutilplying the the value tensor

that tensor is called as the weighted_avg ,this is then projected by the projection layer and finally we get the output

output: Tensor of the shape (n,n_patches+1,embed_dim)

3. Multilayer Perceptron
input: Tensor of the shape (n_samples, n_patches + 1, in_features)`

the forward function of this class contains 2 fully connected layers in which there is one hidden layer and the neurons in the hidden layer is given by the mlp_ratio

the second fc layer again converts the tensor back to the input shape and it has 2 dropouts in between inorder to prevent overfitting

output: Tensor of the shape (n_samples, n_patches +1, out_features)

4. Block
input: Tensor of the shape (n_samples, n_patches + 1, in_features)`

This class contains the residual block that adds itself to the output of the Attention and the MLP classes forward functions.

First the input is layer normalized and then fed into the Attention and the MLP classes

output: Tensor of the shape (n_samples, n_patches + 1, in_features)`

5. Vision Transformer
input: Tensor of the shape (n_samples, in_chans, img_size, img_size)

This is the final class that contains all the classes which were defined until now.

Initially I randomly initialised the cls token to the same size of that of the embed_dim (1,1,embed_dim) , and also intialised the position embedding randomly to the shape (n,n_patches+1,embed_dim). Here the +1 is there for the cls token

After this I expanded the the cls token into the number of trianing examples along the dimension 0

Then concatenate this into the output that came after passing through the patch_embeds forward function

Did the residual adding and added the positional embedding and here the point to be noted is that python broadcasts the positional embedding tensor to the required dimension along the dim0

Now that the patches/tokens are ready we can proceed to implement the blocks which contains the attnetion and the MLP classes

In between we have normalisation so that the mean of that dimension becomes 0 and the std deviation becomes 1

This is observed that it imporoves the trainig

Now the starting token among the number of patches is taken because that is the cls token and that is used to classify furher.

the cls token is passed through the linear layer that converts it into the number of classes here i have kept it as 1 and then appplied sigmoid so that i get the probablity of that training example belonging to the class 1

output: Tensor of the shape (n_samples, 1)`

HyperParameters:

→ criterion = nn.BCELoss()

→ optimizer = optim.Adam()

→ number of epochs= 80

→ batch_size = 32

Results:

After tuning the hyperparameters as mentioned above i managed to get the ROC AUC score of 75.6.

comparision between CNN and ViT

1. ROC AUC - the value of this almost close by. Talking about the partiality to CNN that it had more epochs trained and more time bevause it was trained very quickly (linear complexity). Other hand for Vision Transformer model very less number of epochs were possible to be trained and got less time to tune the hyper parameters as the training time was big (quadractic compelxity)

2. difference in the training part:
   --> CNN had more epochs
   --> learning rate was 100 times smaller in case of ViT
   --> The models are different

3. **RESULTS**

--> the ROC AUC OF CNN = 80.2

--> the ROC AUC OF ViT = 75.7 (76.2)

Further plans in the ViT Model:

1. As the project is based on **Vision Transformer** for binary classification, one of the future plans is to further optimize the model's performance. We can explore ways to **fine-tune** the hyperparameters and increase the dataset size.

    - Have thought of the different modifications which I would perform.
      → As stated in the research paper "Attention is all you need" , The time complexity of the model is **O(m^2)** . That is the time complexity of the model is quadratic and while I trained the model I also felt the same, It took lots of time even after using a GPU. By using **filters with stride** I am expecting better results because in that case the number of filters is reduced and the time is saved

      Time plays a crucial role so that we can experiment with the model and tune the hyperparameters. In the case of CNN network I could manage to train the model in ⅛ th time of that taken by the ViT the reason being the above.

      → Use a **hybrid** between the **CNN** and the **Vision Transformer**: What I meant by a hybrid is that I will run a few layers of Conv2d on the image first and then convert it into patch embed. Here in the current model the number of channels of the input was 2 and converted into the embed dimension directly. What I think is that using a couple of Convolutional layers in between would perform better.

2. **Ensemble:** To increase the performance of the model I have thought to implement the ensemble between multiple models.

3. As I read the research paper I came to know that ViT models can outcast the CNN models only with a huge amount of data. When there is less data CNN works better than ViT and vice versa when there is more data. So I want to extend the model to more training examples. One more idea that I have is that to use **Data Augmentation** technique.

4. Talking about the validation of the model I have just split the data into a train and test set. I want to extend this to **K-fold cross validation** so that I don't overfit the model and can fine tune the hyperparameters.

5. Additionally, we could investigate ways to apply the model to other similar problems or tasks. We can extend the vision transformer to the particle classification which is used in the **Compact Muon Solenoid (CMS)** experiments in the **Large Hadron Collider (LHC).**

Files- Description

1. README.md : The index kind of readme file which contains the description of all the files in the repository.
2. Readme for project 1.md : This is the readme file of the Task 1 ie ELECTRON/PHOTONS Classification using CNN
3. Ml4sci-task-1.ipynb: This contains the jupyter notebook of the task1
4. Readme for project 2.md : This is the readme file of the Task 2 ie QUARK/GLUON Classification using CNN
5. Ml4sci-task-2.ipynb: This contains the jupyter notebook of the task2
6. Readme for project 3-(1).md : This is the readme file of the Task 3-(1) ie ELECTRON/PHOTONS Classification using Vision Transformers
7. Ml4sci-task-3-1.ipynb: This contains the jupyter notebook of the task3
8. Future plans.md: Contains the details of my future plans