Convolutional Visual Network for Classification of Jet Images

PHYS 139/239 Final Project
Group 4

Daniel Primosch, Quinn Picard
Anni Li, Adolfo Partida

Github Link:
https://github.com/danprim/phys239__project

Motivation

Motivation behind the paper:

- Develop a more accurate and efficient method for classifying neutrino events in the NOvA (NuMI Off-axis ν_e Appearance)experiment.
- Traditional methods for neutrino event classification: limited in accuracy and efficiency.
- The proposed deep learning model, CVN, can automatically learn features from raw data, which have the potential to improve classification process.

Motivation for our group project:

- By applying what we learn from this course and reproducing this experiment, gain a deeper understanding of CNN and its application in high-energy physics field.
- Practical experience in implementing and training deep learning models with high-dimensional data.
- Gain insights into the challenges and considerations involved in applying ML techniques to real-world HEP datasets, such as data preprocessing, model architecture design, and hyperparameter tuning.

Background

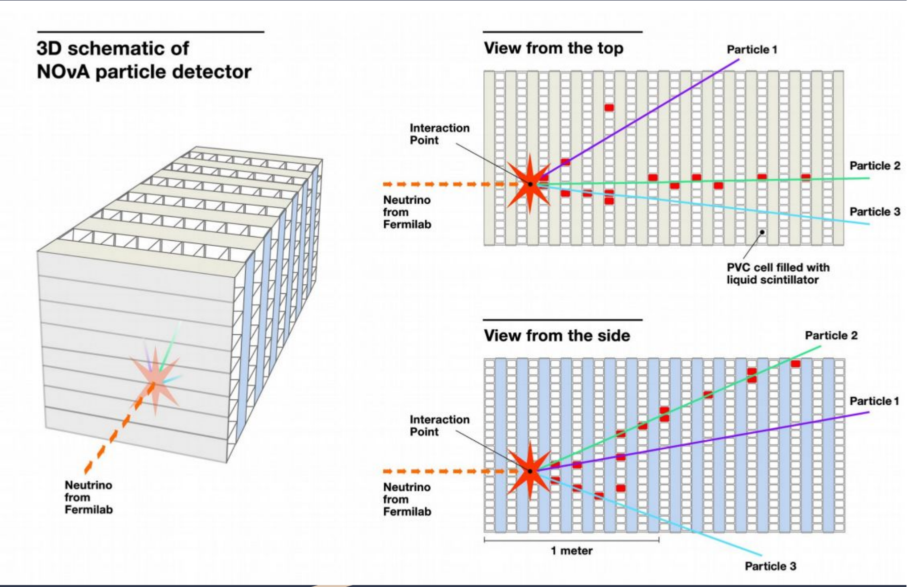


Figure: https://arxiv.org/abs/1604.01444

NOvA Experiment

- Long-baseline neutrino oscillation experiment
- Goal: Study neutrino oscillations and properties
- Two detectors: Near Detector (ND) and Far Detector (FD)

Neutrino Event Classification

- Neutrinos interact with matter via weak force
- Crucial for understanding neutrino oscillations
- Different types of neutrino interactions produce distinct event signatures
- Traditional methods: limited accuracy and limited computational efficiency

Traditional Methods for Separating signal from background

MLP(multilayer perceptron), K-Nearest Neighbors, BDT(boosted decision trees)

- **Handcrafted features**: The features used in these algorithms are usually manually designed and selected, which may not capture all the relevant information in the event data.
- **Limited accuracy:** Mistakes in the reconstruction of high level features from the raw data can lead to incorrect categorization of physics events [2].
- **Computational efficiency:** The processing of raw event data and feature extraction can be time-consuming, especially for large-scale experiments like NOvA.

Convolutional Neural Networks (CNNs) – Advantages[2]

Automatic feature extraction

- No need for manual feature engineering, extracts features automatically from the data by feature maps
- Hierarchical structure enables learning of complex patterns

Improved accuracy in image recognition tasks

- Better at capturing local and global information
- As a computer vision model, suitable for HEP measurements gained from detectors, which result in images of physics interactions[2].

Efficient Training and Inference

- Exploits GPU-based parallelism for faster training
- Weight sharing and pooling layers reduce the number of parameters
- Enables real-time processing and large-scale deployment

CVN Model – Structure

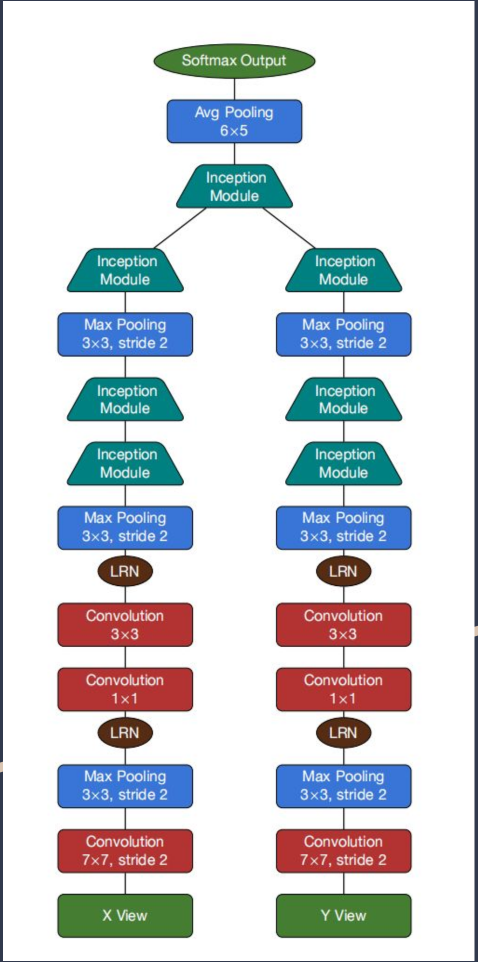


Figure: https://arxiv.org/abs/1604.01444

Convolutional Visual Network

- Developed using Caffe framework
  - *Problems with the package so we finally used pytorch.*
- Inspired by **GoogLeNet**, but have two different views of the same image. The channels corresponding to the X and Y views were split and each resulting image was sent down a parallel architecture based on GoogLeNet
- Multiple convolutional and Max-pooling layers
- **NIN (network-in-network)**: Three inception modules for efficient feature extraction instead of nine in GoogLeNet. Final inception module extracts combined features [2]
- **1x1 convolution** layers down-samples the number of feature maps and maintains the dimensionality of the input maps

Inception module

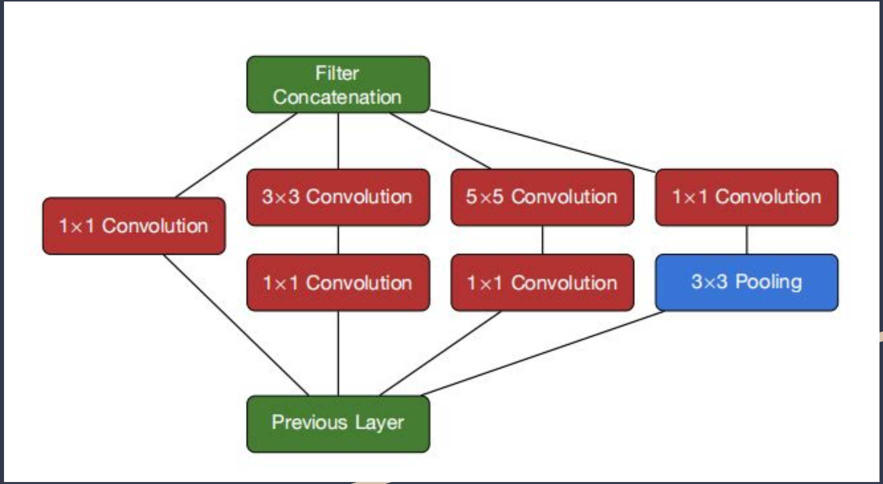
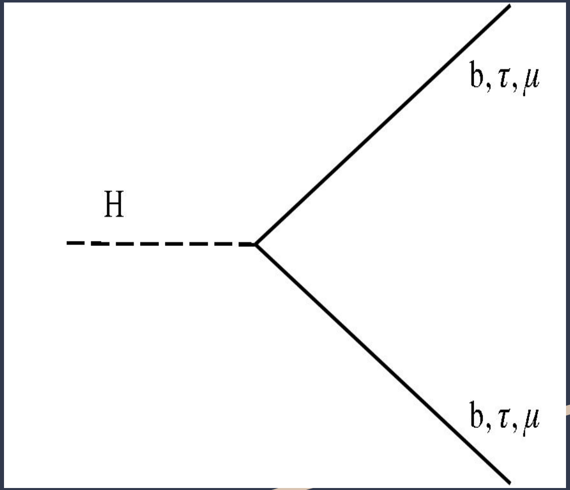


Figure: https://arxiv.org/abs/1604.01444

- Goal is to increase efficiency and capacity to learn without adding complexity
- A **sub-network** within the main network
- Takes a set of feature maps produced by the previous layer as input
- Distributes those feature maps to **branches**, each with filters at different scales
- Outputs from these branches are then concatenated to be passed to the next layer with same number of feature maps and dimensions as input
- Allows for a diverse range of pattern learning without adding too much complexity [2]

Dataset Selection



Source Image: A. M. Sirunyan *et al.*, "Combined measurements of Higgs boson couplings in proton–proton collisions at $$\sqrt{s}=13\,\text {Te}\text {V} $$," *European Physical Journal C*, May 20, 2019. https://link.springer.com/content/pdf/10.1140/epjc/s10052-019-6909-y.pdf

Complications

- Dataset in the original article is not publicly available
- A similar dataset by our very own Prof. Javier Duarte! [4]
- due to troubles with previous datasets we had little time to familiarize ourselves with the current one.

The Data Set

- A collection of particle jets created from *simulated* proton-proton collisions
  - Center of mass: 13 TeV
  - Via Pythia 8
- For differentiating two possible jets originating from the collision
  - **Hbb**: jets from a Higgs boson decaying to a bottom quark-antiquark pair
  - **QCD**: Jets from **quark or gluon jets** originating from **quantum chromodynamic**

Data Loading

Looked at 3 features

- **Pfcand_ptrel:** Transverse momentum of the PF candidate divided by the transverse momentum of the AK8
- **pfcand_etarel**:Pseudorapidity of the PF candidate relative to the AK8 jet axis
- **pfcand_phire**:Azimuthal angular distance Δϕ between the PF candidate and the AK8 jet axis
- More features to be added in future model for better accuracy.

How?

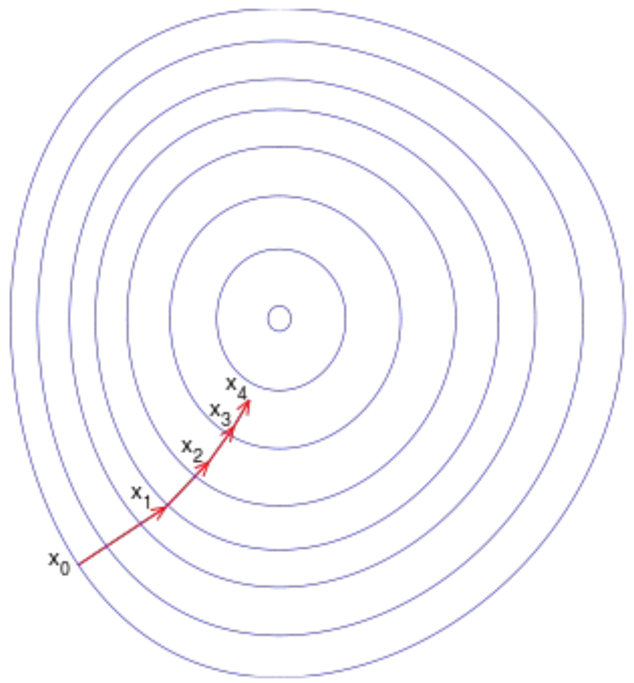
- Our database is stored in root files
- Extracted the file contents into tensors PyTorch tensors
- Tensors take the shape of 224,224,1 representing…
    - 224 pixels tall
    - 224 pixels wide
    - 1 channel of intensity
- Training batch size: 500
- Testing size: 250

Training

Simple Training

- Split up the data into training and testing sets
- Selected training methods
  - Loss function: Cross Entropy
  - Optimizer: SGD

- Fed batches of data into the CNN
- Computed the loss, backpropagated the error, and updated the weights using the optimizer.



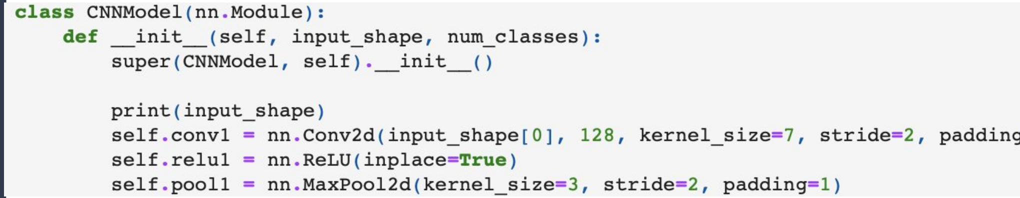
PyTorch Library

Model Class

- Instance of nn.Module (pytorch)
- super()
- Model connections in forward()

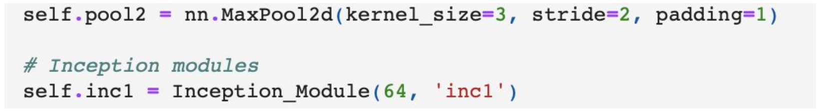
Inception Module

- Own separate module, no pre-written one
- Called within CNNModel



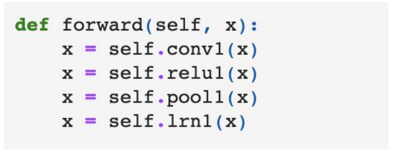
```python
class CNNModel(nn.Module):
    def __init__(self, input_shape, num_classes):
        super(CNNModel, self).__init__()

        print(input_shape)
        self.conv1 = nn.Conv2d(input_shape[0], 128, kernel_size=7, stride=2, paddin
        self.relu1 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
```

[...]



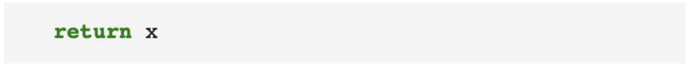
```python
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Inception modules
        self.inc1 = Inception_Module(64, 'inc1')
```

[...]

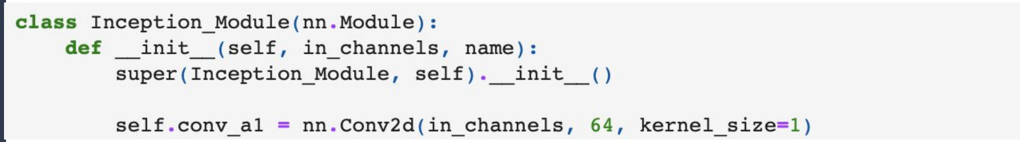


```python
    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.lrn1(x)
```

[...]



```python
        return x
```

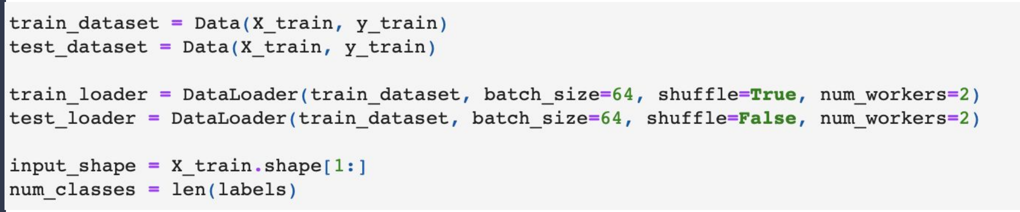


```python
class Inception_Module(nn.Module):
    def __init__(self, in_channels, name):
        super(Inception_Module, self).__init__()

        self.conv_a1 = nn.Conv2d(in_channels, 64, kernel_size=1)
```

PyTorch Library

Dataset, DataLoader

- Built in PyTorch classes
- Batching (do size = 2^n)
- Number of workers => parallelization

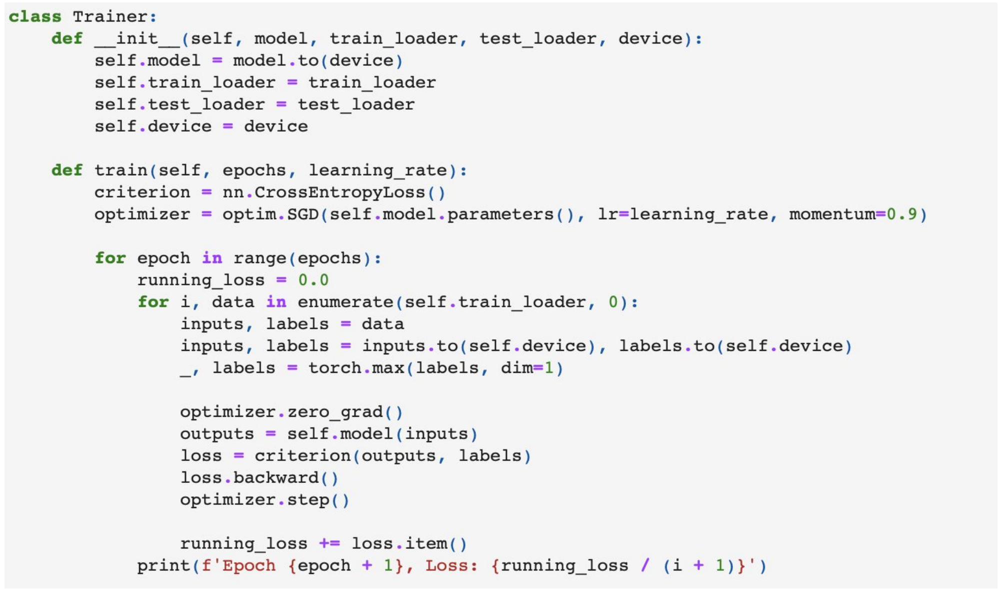


```
train_dataset = Data(X_train, y_train)
test_dataset = Data(X_train, y_train)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=2)
test_loader = DataLoader(train_dataset, batch_size=64, shuffle=False, num_workers=2)

input_shape = X_train.shape[1:]
num_classes = len(labels)
```

Training Implementation

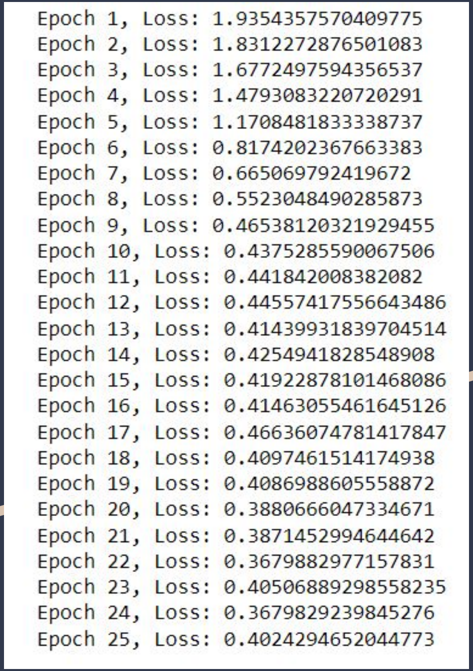
- Model initialized
- Choose device (CUDA if GPU available)
- Instantiate loss criterion
- Instantiate optimizer (SGD, etc)

- For each batch:
- Feed inputs through model (forward)
- Apply loss
- Backward (backpropagation)



```python
class Trainer:
    def __init__(self, model, train_loader, test_loader, device):
        self.model = model.to(device)
        self.train_loader = train_loader
        self.test_loader = test_loader
        self.device = device

    def train(self, epochs, learning_rate):
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(self.model.parameters(), lr=learning_rate, momentum=0.9)

        for epoch in range(epochs):
            running_loss = 0.0
            for i, data in enumerate(self.train_loader, 0):
                inputs, labels = data
                inputs, labels = inputs.to(self.device), labels.to(self.device)
                _, labels = torch.max(labels, dim=1)

                optimizer.zero_grad()
                outputs = self.model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

                running_loss += loss.item()
            print(f'Epoch {epoch + 1}, Loss: {running_loss / (i + 1)}')
```

Evaluation criteria & Results



```
Epoch 1, Loss: 1.9354357570409775
Epoch 2, Loss: 1.8312272876501083
Epoch 3, Loss: 1.6772497594356537
Epoch 4, Loss: 1.479308322072029l
Epoch 5, Loss: 1.1708481833338737
Epoch 6, Loss: 0.8174202367663383
Epoch 7, Loss: 0.665069792419672
Epoch 8, Loss: 0.5523048490285873
Epoch 9, Loss: 0.46538120321929455
Epoch 10, Loss: 0.4375285590067506
Epoch 11, Loss: 0.441842008382082
Epoch 12, Loss: 0.44557417556643486
Epoch 13, Loss: 0.41439931839704514
Epoch 14, Loss: 0.4254941828548908
Epoch 15, Loss: 0.41922878101468086
Epoch 16, Loss: 0.41463055461645126
Epoch 17, Loss: 0.46636074781417847
Epoch 18, Loss: 0.4097461514174938
Epoch 19, Loss: 0.4086988605558872
Epoch 20, Loss: 0.3880666047334671
Epoch 21, Loss: 0.3871452994644642
Epoch 22, Loss: 0.3679882977157831
Epoch 23, Loss: 0.40506889298558235
Epoch 24, Loss: 0.3679829239845276
Epoch 25, Loss: 0.4024294652044773
```

Evaluation Criteria

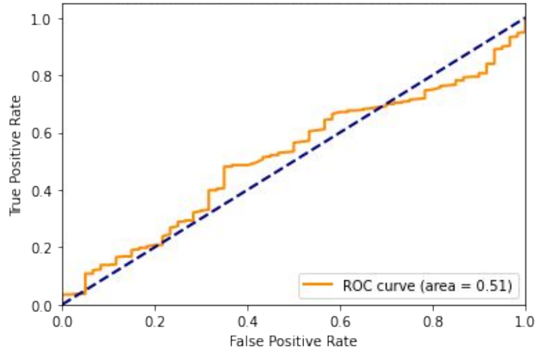
- AUC: get something better than 0.5
- Accuracy in paper: 0.69

Problems

- Switched from Caffe to PyTorch on Sunday
- Forced to choose different dataset (corrupted)
- Debugging for ~ 100 years

Results

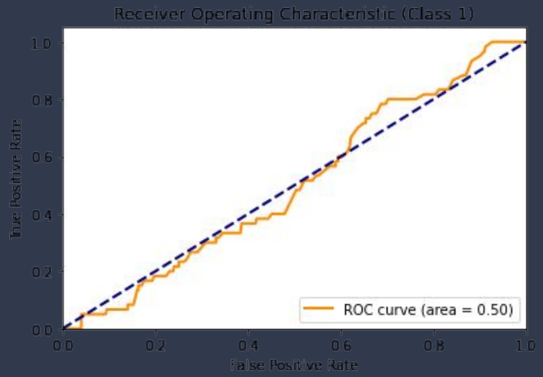
- Code runs!
- Loss goes down but not much: down to 0.40
- AUC = 50% ⇔ Random Classifier

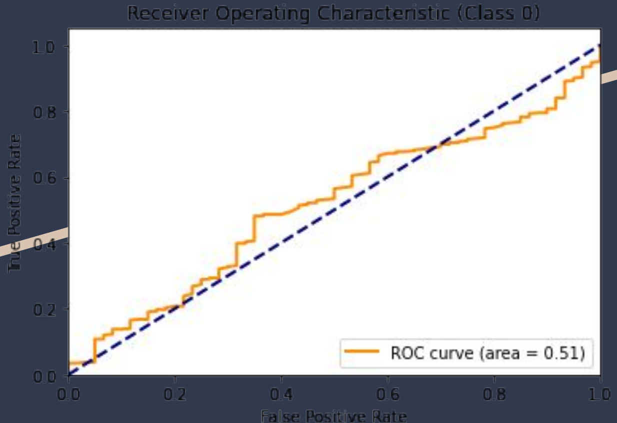


Future Outlook

- Find out why it’s a random classifier:
    - Network architecture mismatch
    - Data loading not fully debugged (syntax okay)
- Make improvements:
    - Better paratimization
    - Different loss functions
    - Different optimization schemes

Conclusion





…We used CVN structure inspired by the paper, modified it using pytorch, and used a different dataset to finish jet images classification task.

(what we've done)

We realized that indeed there are challenges in applying ML techniques to real-world HEP datasets (memory issues, data format incompatible with model, hyperparameters difficult to tune, etc.)

…

(possible improvements)

- Didn't use Caffe
- We didn't train for a whole week straight to train
- We didn't have much computation power

Contributions

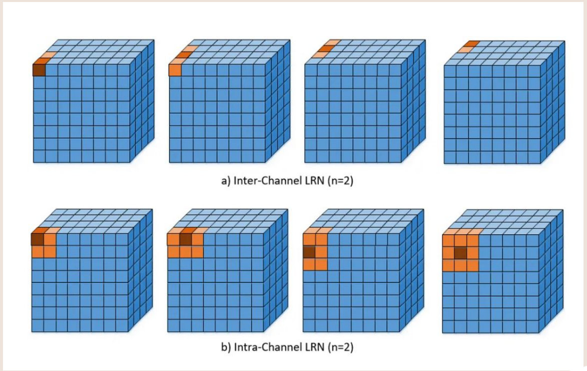
Daniel Primosch: Model Creation and bug testing

Quinn Picard: Data Selection and loading

Anni Li: Model debugging, model training

Adolfo Partida: Data loading and optimizing

Backup - LRN



Local Response Normalization:
LRN is a normalization technique used in CNNs to improve their performance. The idea behind LRN is to enhance the learned features in the network by emphasizing some activations while dampening others, based on their local neighborhood. This is particularly useful when dealing with high-activation features, as it helps the model generalize better by reducing the impact of very high activations.

Batch normalization provides the layer with two more trainable parameters. The gamma (standard deviation) and beta (mean) parameters are multiplied by the normalized result. This allows stack normalization and gradient descent to work together to "denormalize" the data by simply changing two weights per output. By adjusting all other related weights, data loss was reduced and network stability was improved.

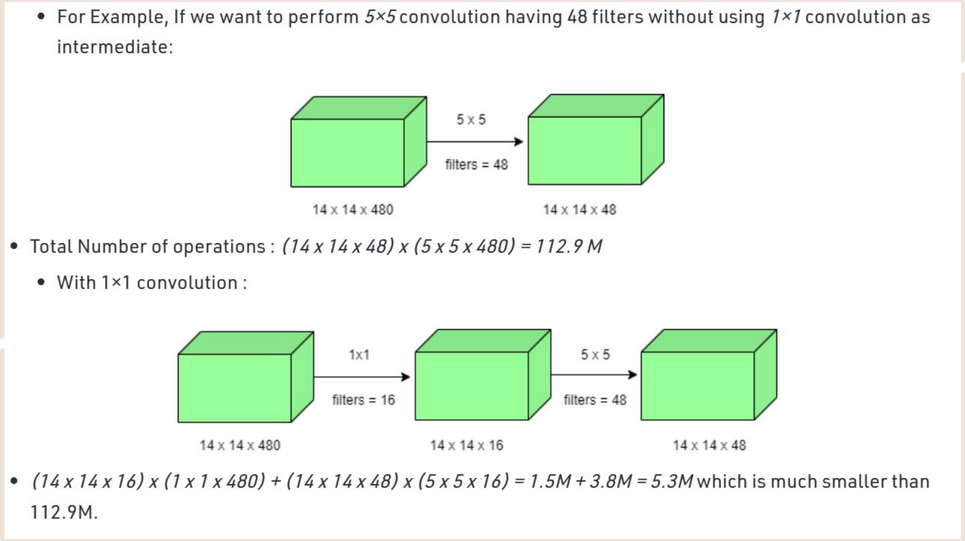
BN can only be done in one way, but LRN has different directions for performing normalization between or within channels. [5]

Figure: https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Backup – GoogLeNet:

GoogLeNet was proposed by research at Google (with the collaboration of various universities) in 2014 in the research paper titled "Going Deeper with Convolutions". This architecture was the winner at the ILSVRC 2014 image classification challenge.

1×1 convolution : The inception architecture uses 1×1 convolution in its architecture. These convolutions decrease the number of parameters and enable increasing depth of the architecture.



- For Example, If we want to perform *5×5* convolution having 48 filters without using *1×1* convolution as intermediate:

- Total Number of operations : *(14 x 14 x 48) x (5 x 5 x 480) = 112.9 M*
- With 1×1 convolution :

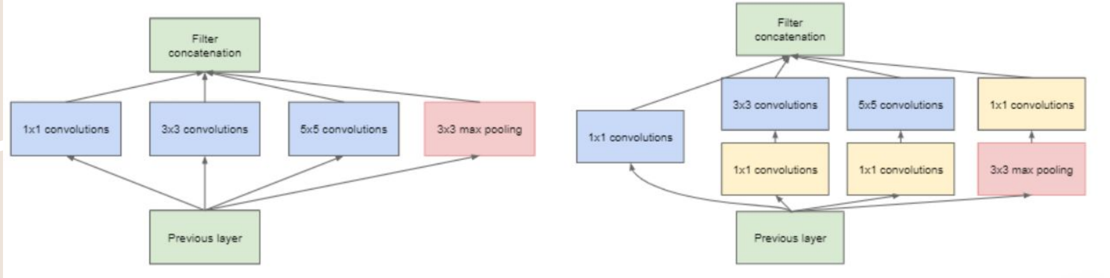
- *(14 x 14 x 16) x (1 x 1 x 480) + (14 x 14 x 48) x (5 x 5 x 16) = 1.5M + 3.8M = 5.3M* which is much smaller than 112.9M.

Figure: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

Backup - GoogLeNet:

Global Average Pooling : In the previous architecture such as AlexNet, the fully connected layers are used at the end of the network, which contain the majority of parameters and causes an increase in computation cost. In GoogLeNet architecture, there is a method called **global average pooling** is used at the end of the network. This layer takes a feature map of 7×7 and averages it to 1×1. This also decreases the number of trainable parameters to 0 and improves the top-1 accuracy by 0.6%

Inception Module: The inception module is different from previous architectures such as AlexNet, ZF-Net. In this architecture, there is a fixed convolution size for each layer. In the Inception module 1×1, 3×3, 5×5 convolution and 3×3 max pooling performed in a **parallel** way at the input and the output of these are stacked together to generated final output. The idea behind that convolution filters of different sizes will handle objects at multiple scale better.[1]



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

References

[1] https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/
[2] https://arxiv.org/abs/1604.01444 A Convolutional Neural Network Neutrino Event Classifier A. Aurisano,a,1 A. Radovic,b,1 D. Rocco,c,1 A. Himmel,d M.D. Messier,e E. Niner,d G. Pawloski,c F. Psihas,e A. Sousaa and P. Vahle
[3] https://arxiv.org/abs/1511.05190 Luke de Oliveira, Michael Kagan, Lester Mackey, Benjamin Nachman, and Ariel Schwartzman. Jet-images — deep learning edition. J. High Energy Phys., 07:069, 2016.
[4]Duarte, Javier; (2019). Sample with jet, track and secondary vertex properties for Hbb tagging ML studies HiggsToBBNTuple_HiggsToBB_QCD_RunII_13TeV_MC. CERN Open Data Portal. DOI:10.7483/OPENDATA.CMS.JGJX.MS7Q
[5] https://iq.opengenus.org/local-response-normalization/