Process flow:

Step 1: Generation of image:

From the given Zenodo datasets, 90000 jets are used in training, 4000 jets for validation and 10000 for testing.

The first 35 most energetic constituents from each jet are stored in an array called Jet which is 3D of dimensions 35 * 4 * 90000 (training) as described in the example file.

A 37 * 37 image is used to represent the bin in which the all the constituents of the jet "fall" into.

The pseudorapidity(eta) and azimuthal(phi) arrays are calculated according to:

Pseudorapidity is given by

$$\eta = \sinh^{-1}\left(\frac{p_z}{p_T}\right)$$

Azimuthal is given by

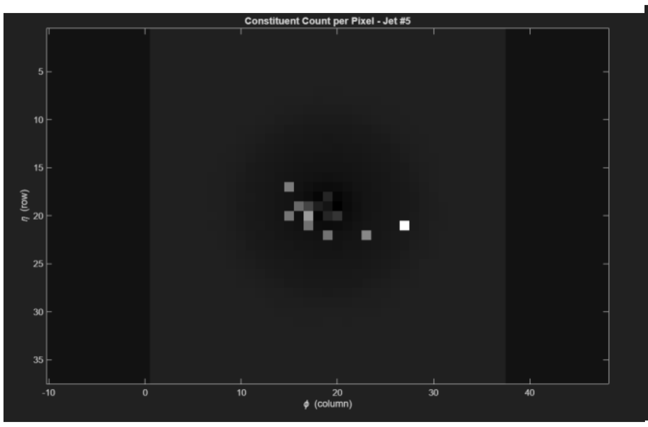
$$\phi = \tan^{-1}\left(\frac{p_y}{p_x}\right)$$

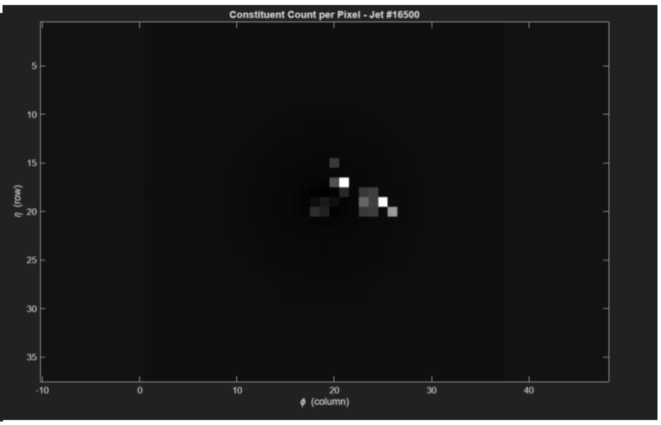
For each jet a 12 channel image of spatial dimensions 37 * 37 is created.

Each of the channel stores some statistical information about the constituents per pixel.

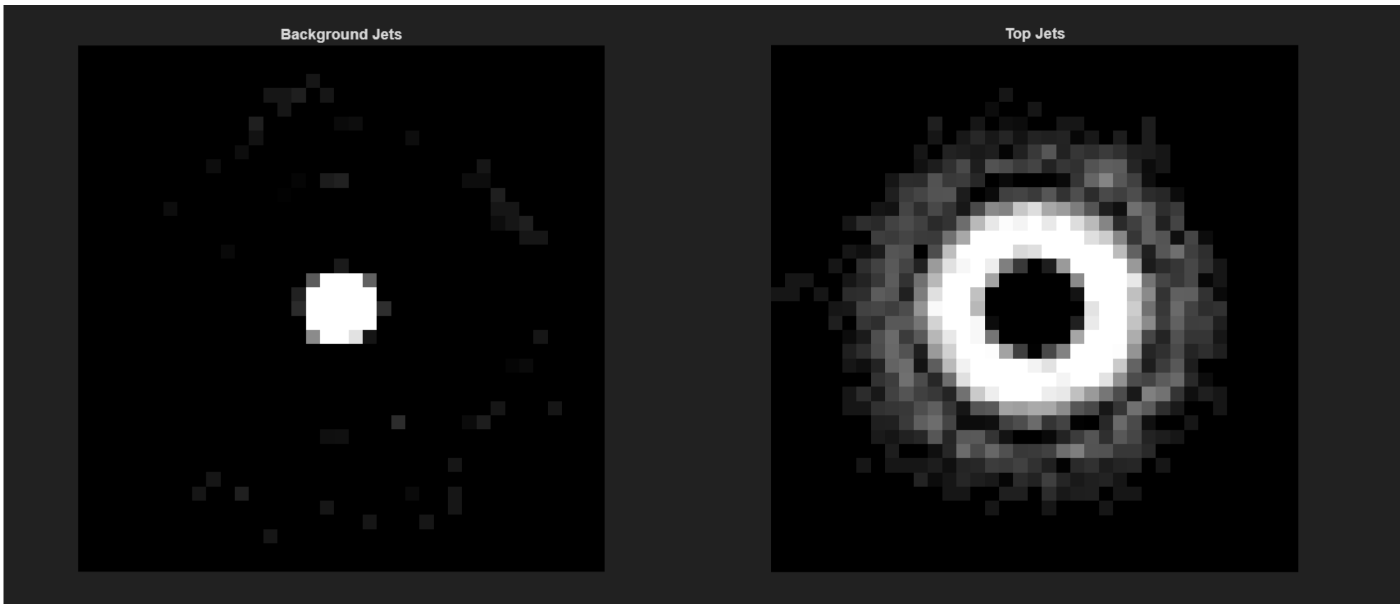
The channels are:

1. Number of Constituents per pixel
   Individual jets: Background   Top

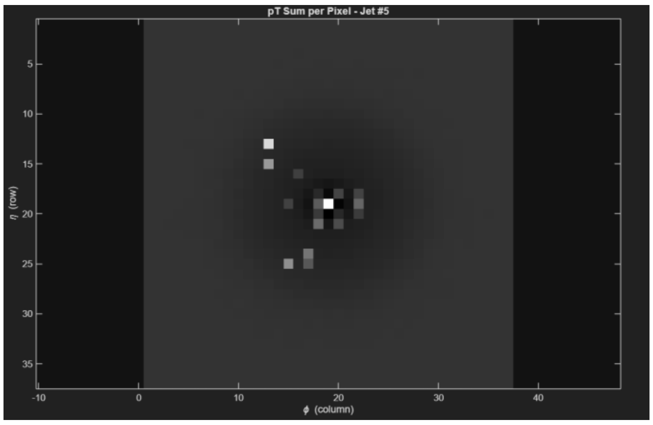


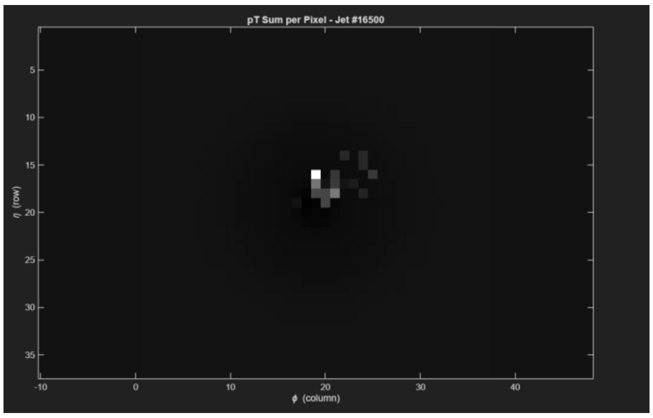


(Averaged Jets Visualization)

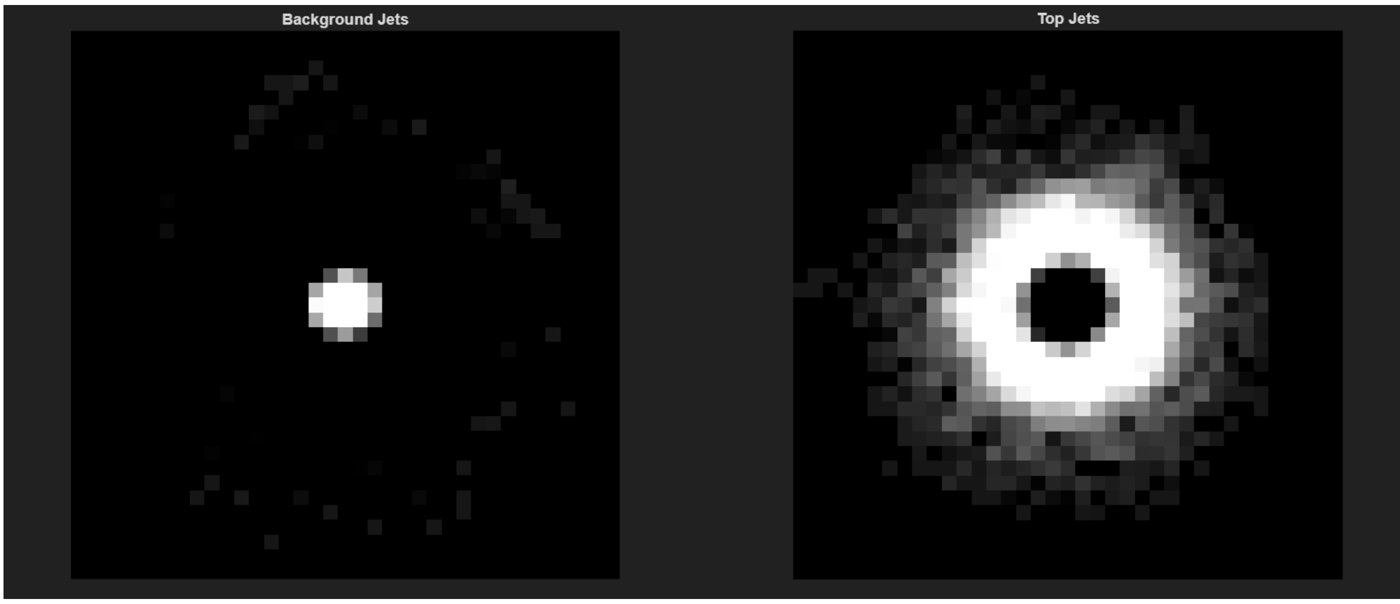


2. Transverse Momentum Sum per pixel
   Individual jets: Background   Top



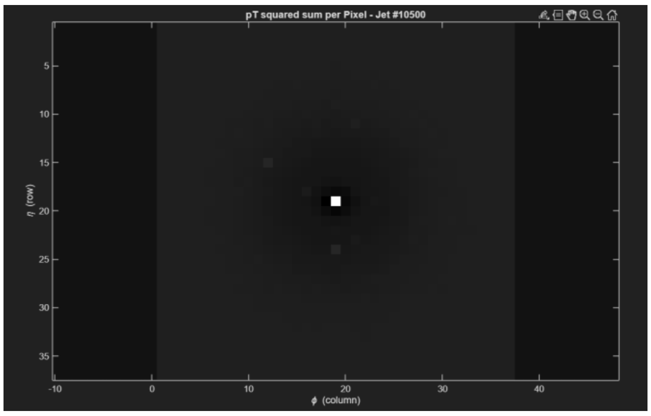


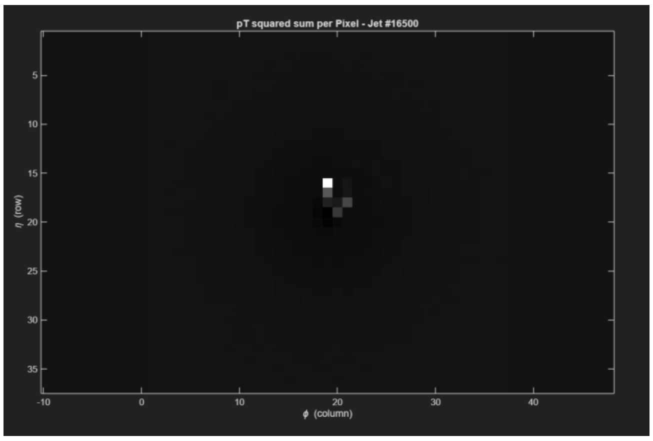
(Averaged Jets Visualization)



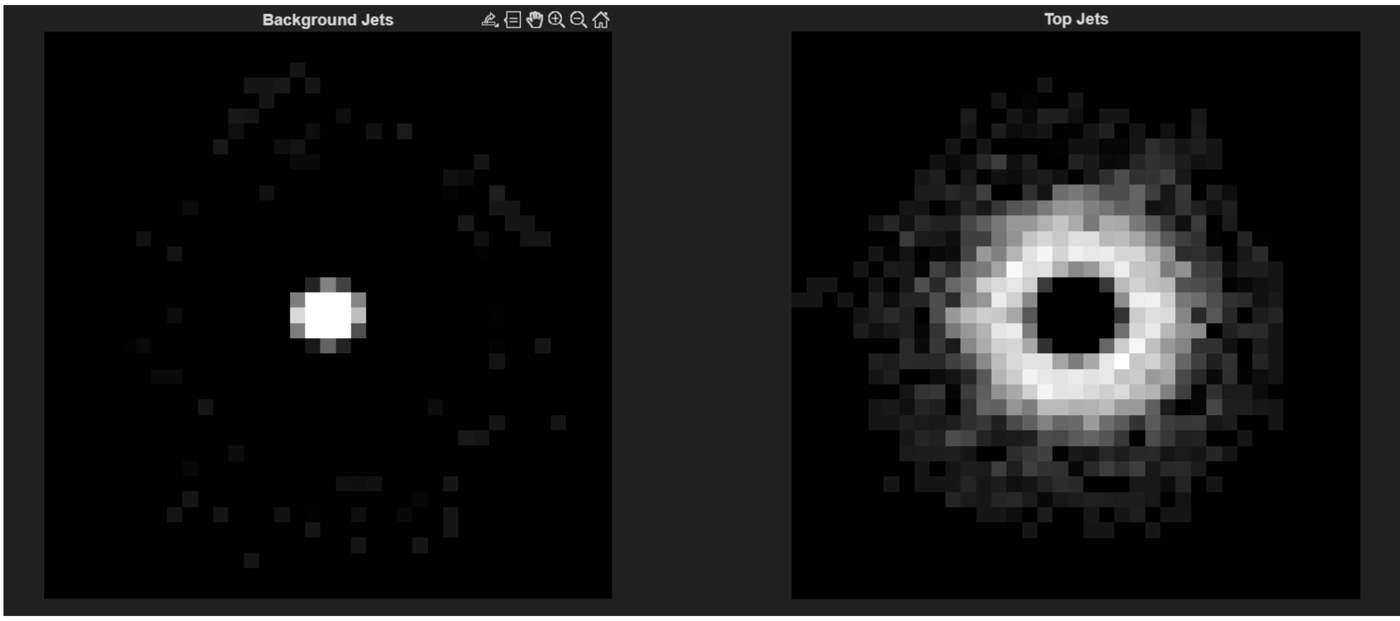
3. Transverse Momentum Squared per pixel

Individual jets: Background Top



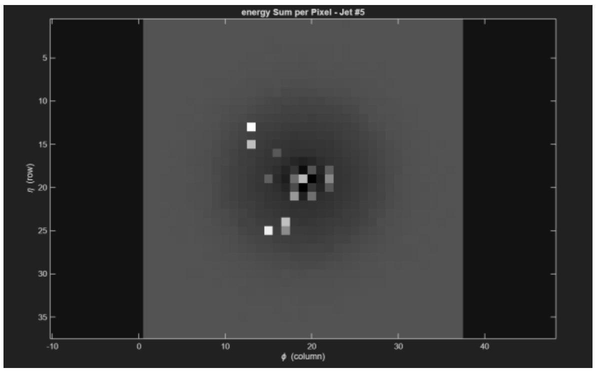


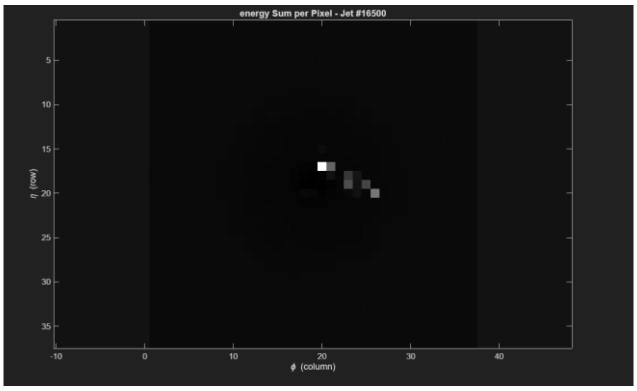
(Averaged Jets Visualization)



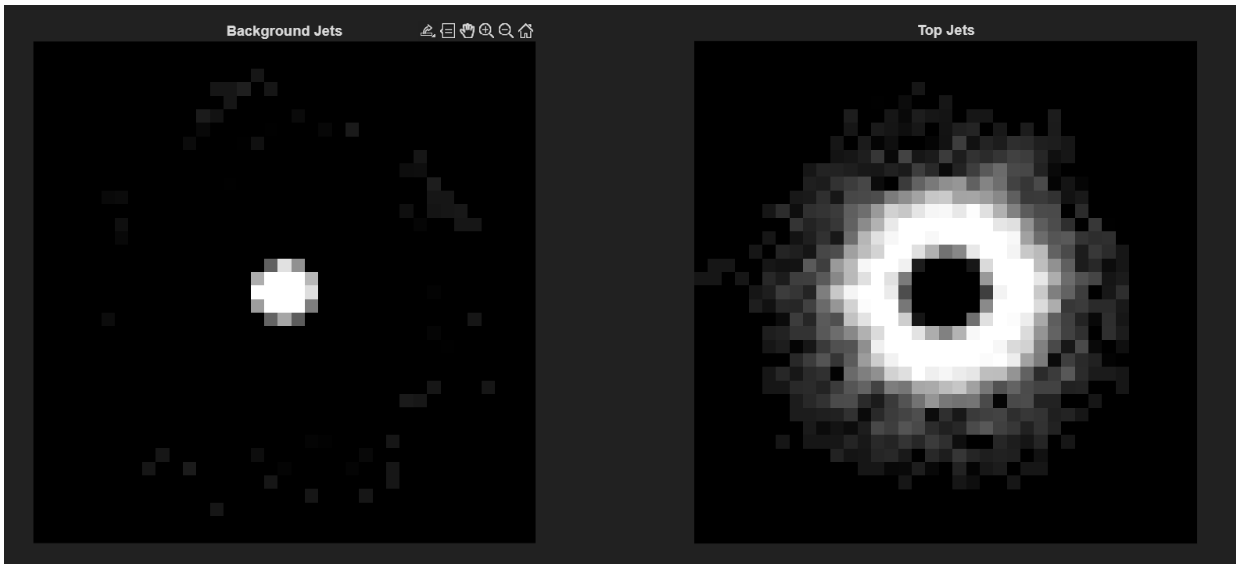
4. Energy Sum per pixel

Individual jets: Background   Top



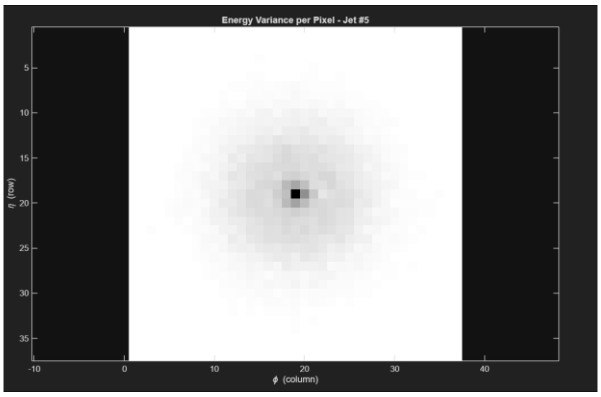


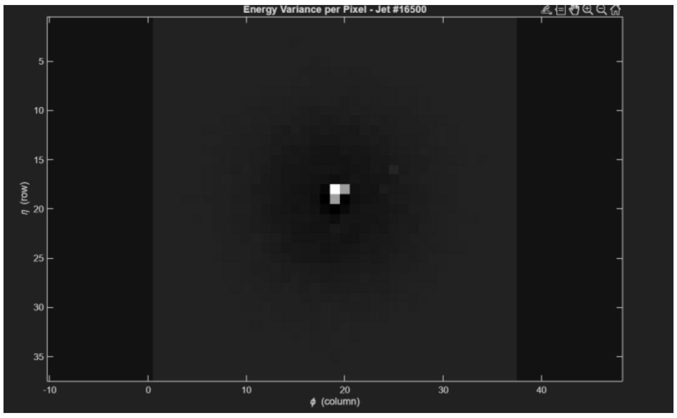
(Averaged Jets Visualization)



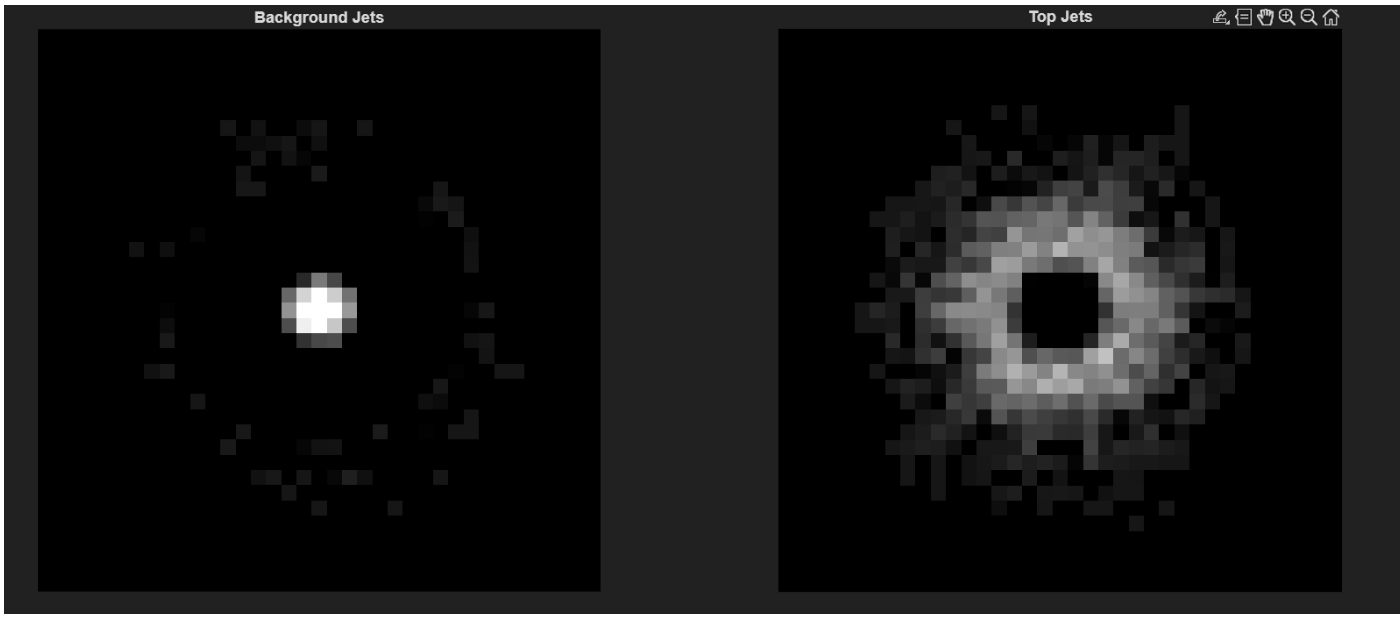
5. Energy Variance per pixel

Individual jets: Background   Top



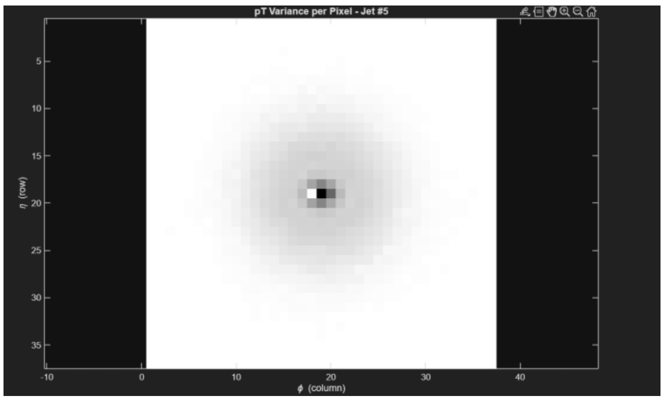


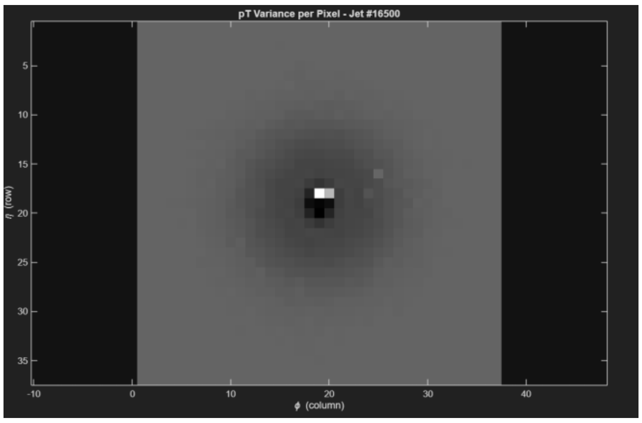
(Averaged Jets Visualization)



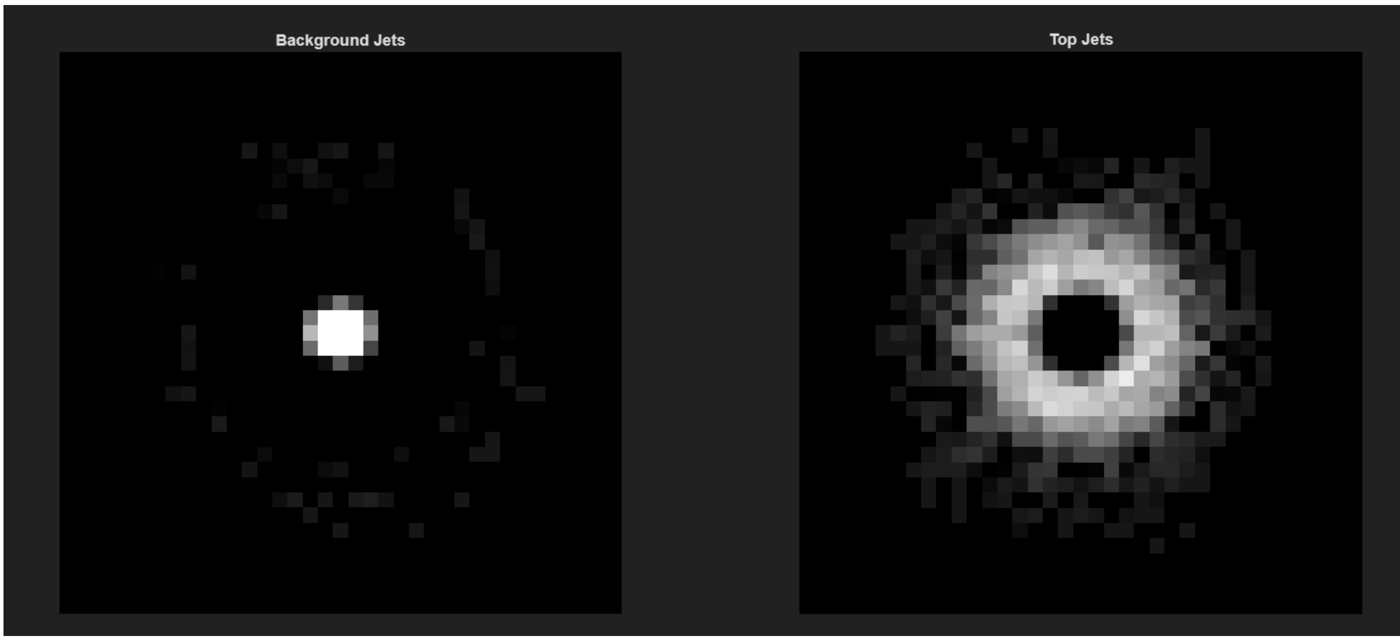
6. Transverse Momentum Variance per pixel

Individual jets: Background Top



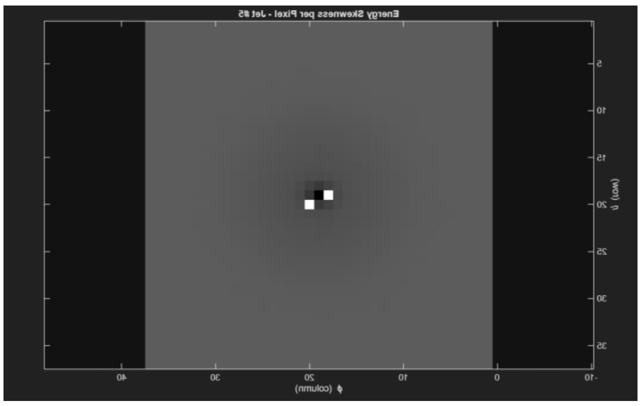


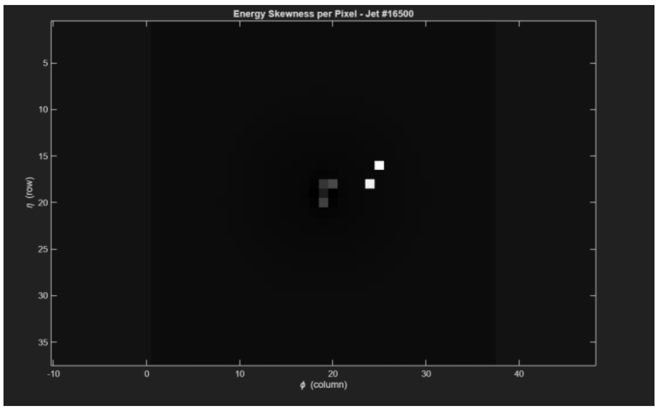
(Averaged Jets Visualization)



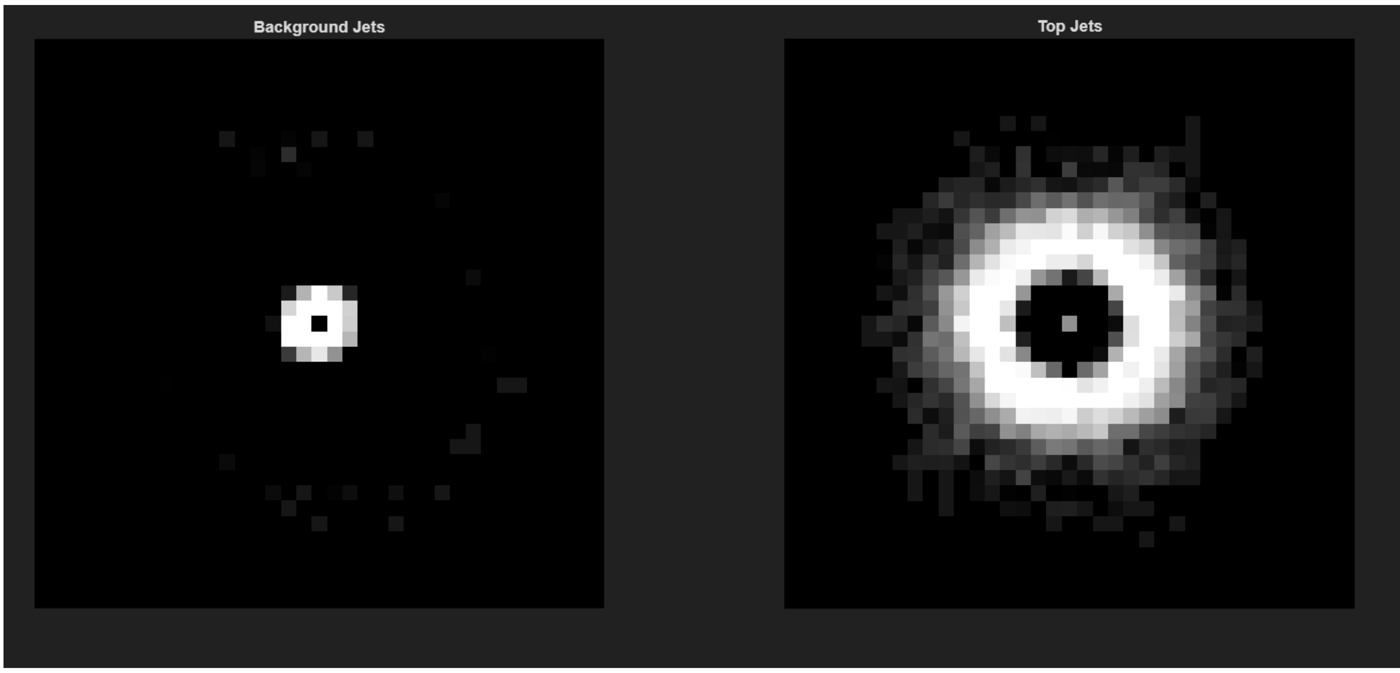
7. Energy Skewness per pixel

Individual jets: Background Top



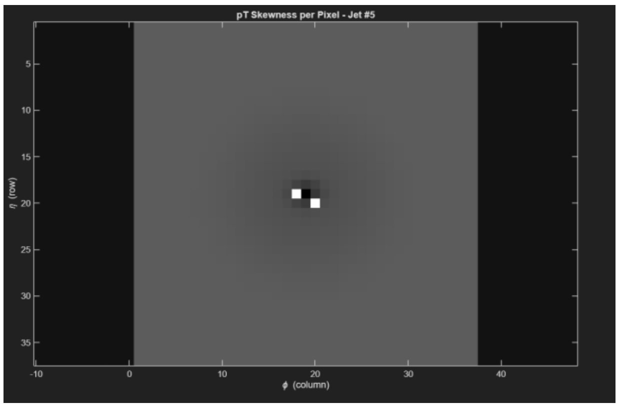


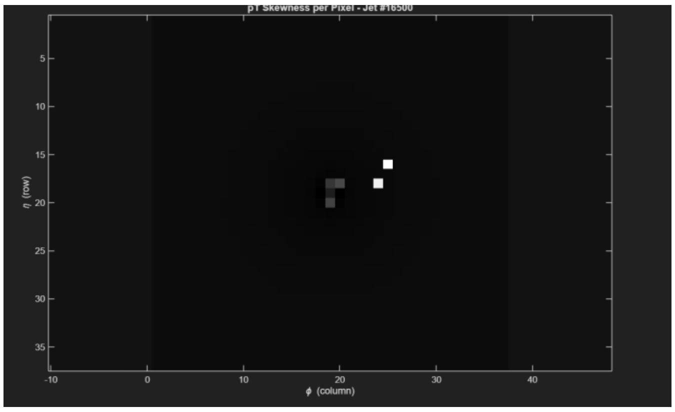
(Averaged Jets Visualization)



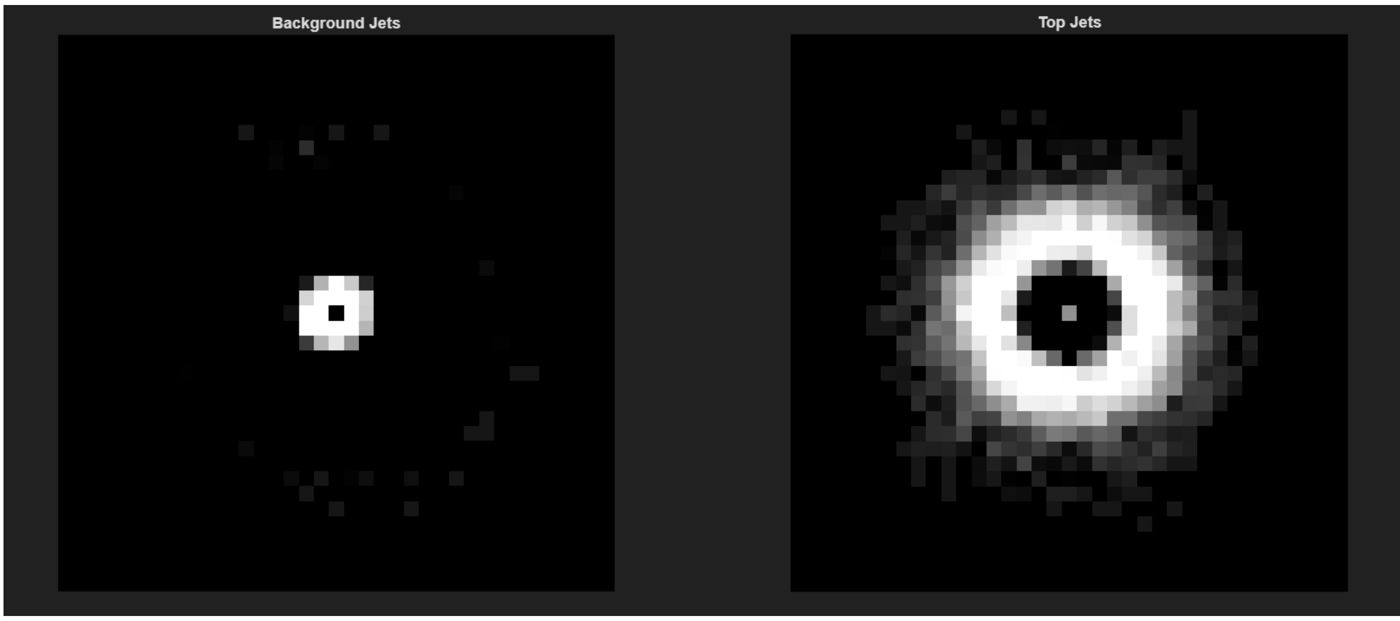
8. Transverse Momentum Skewness per pixel

Individual jets: Background Top



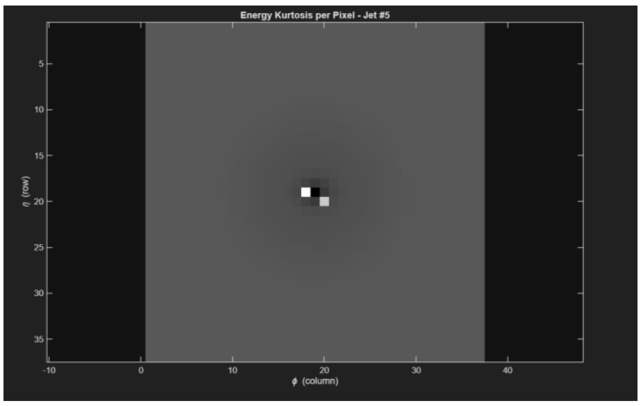


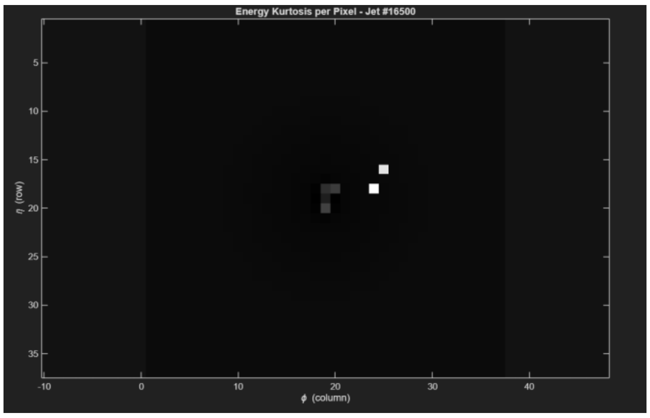
(Averaged Jets Visualization)



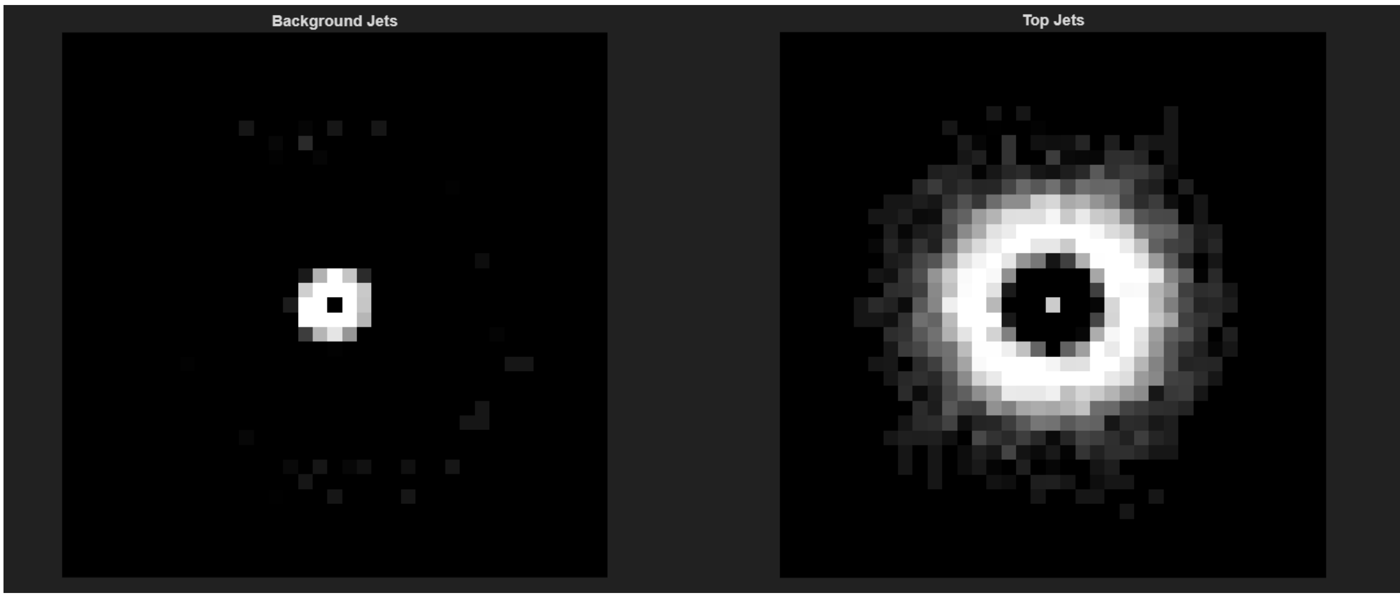
9. Energy Kurtosis per pixel

Individual jets: Background Top



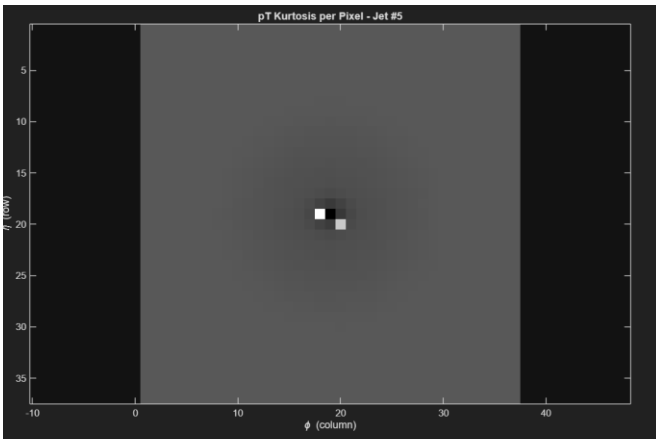


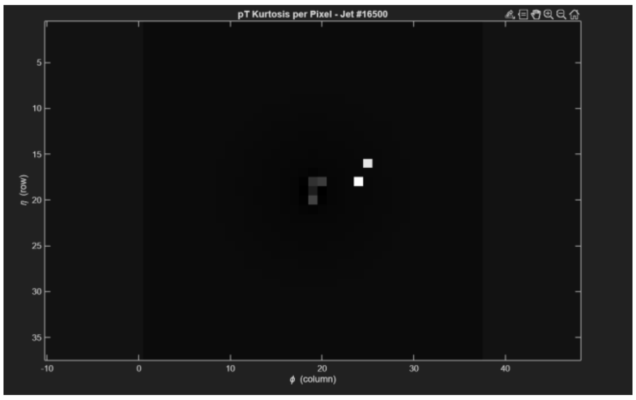
(Averaged Jets Visualization)



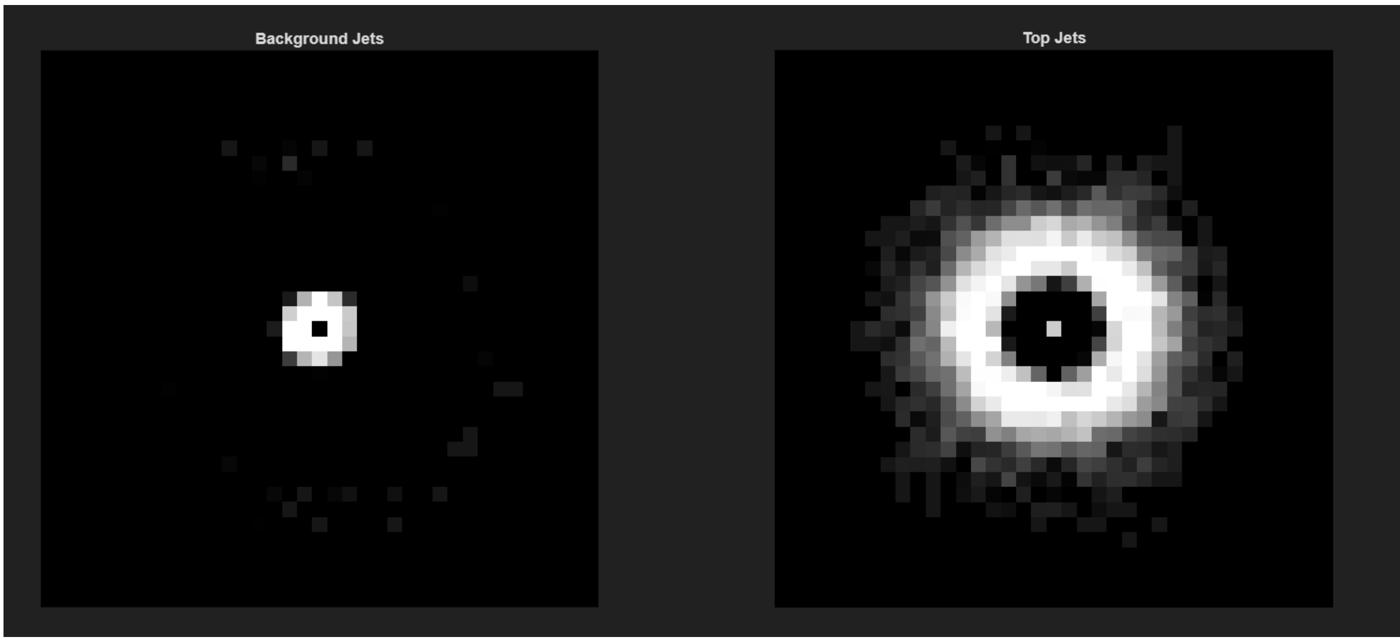
10. Transverse Momentum Kurtosis per pixel

Individual jets: Background Top



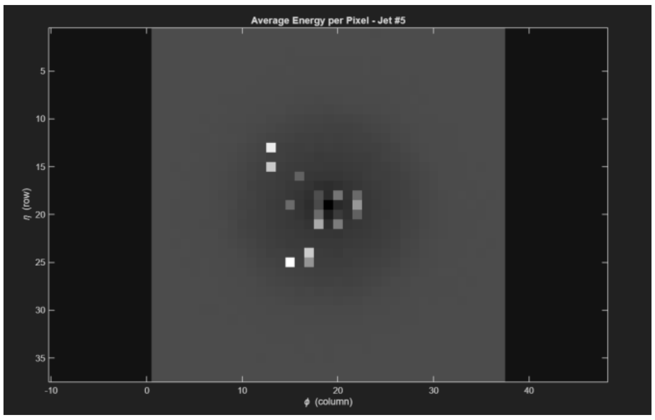


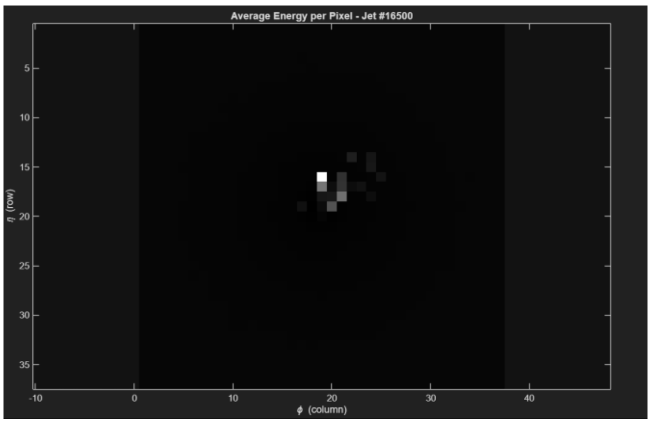
(Averaged Jets Visualization)



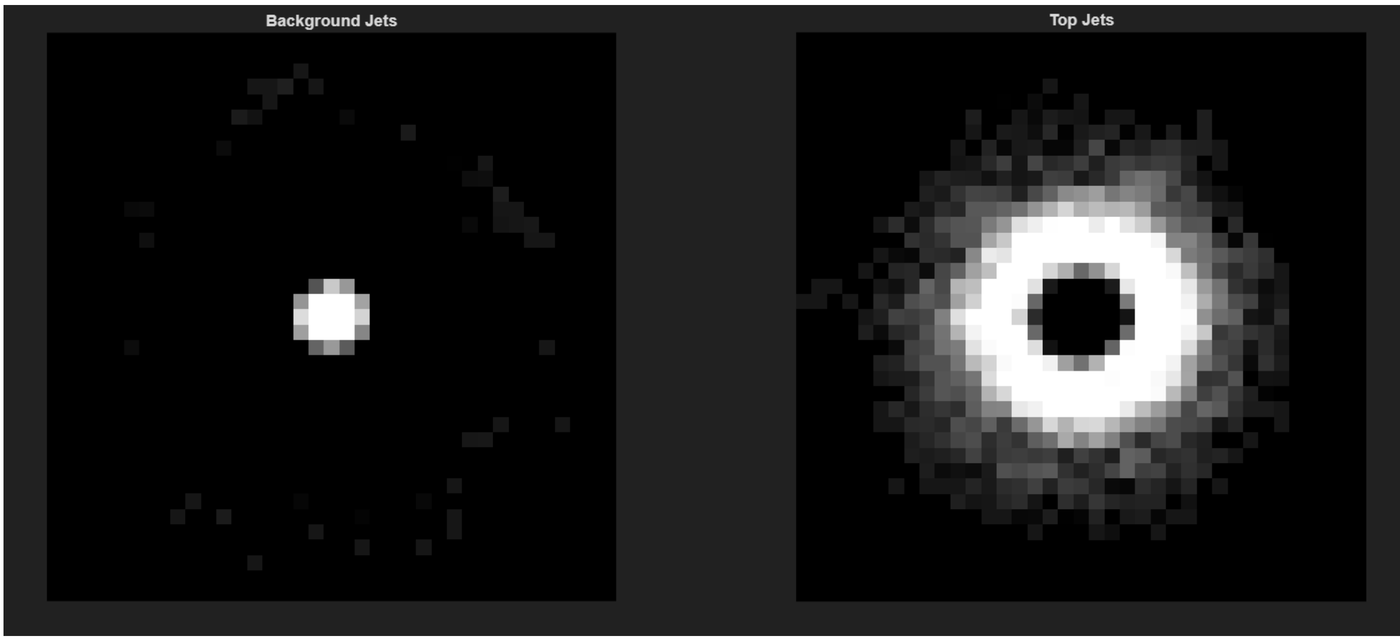
11. Average Energy per pixel

Individual jets: Background Top



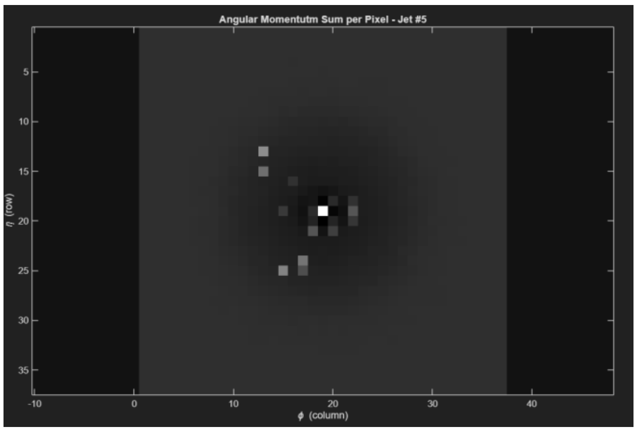


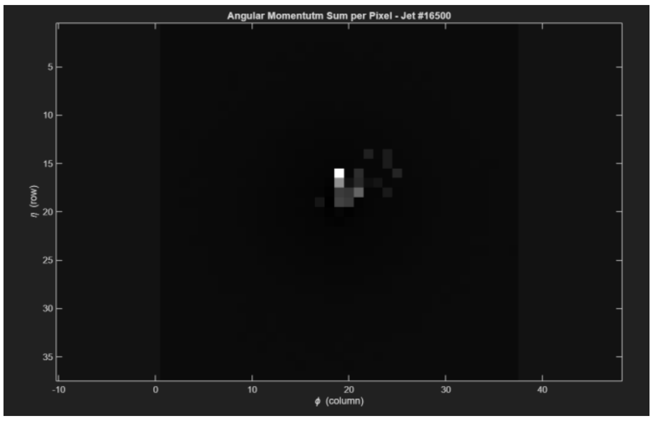
(Averaged Jets Visualization)



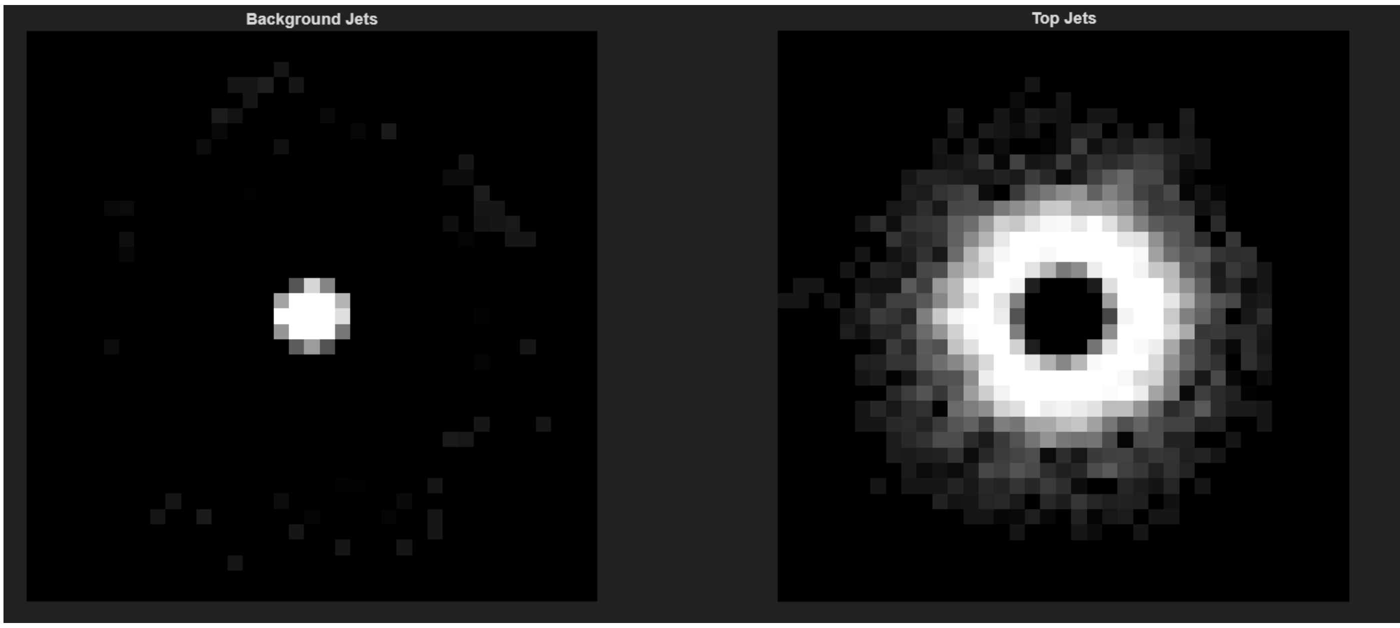
12. Sum of Angular Momentum of Particles per pixel

Individual jets: Background — Top





(Averaged Jets Visualization)



NOTE: After a few checks, the channels related to Energy Skewness, Energy Kurtosis and Transverse Momentum Kurtosis, I concluded that these channels are same, and hence they will be removed from the input dataset to reduce unnecessary computations. Graphs are still shown here for reference. Final model still has 12 channels and will be updated soon.

Apart from these 4 more features are extracted which will be used in the last Fully Connected Layers, namely,

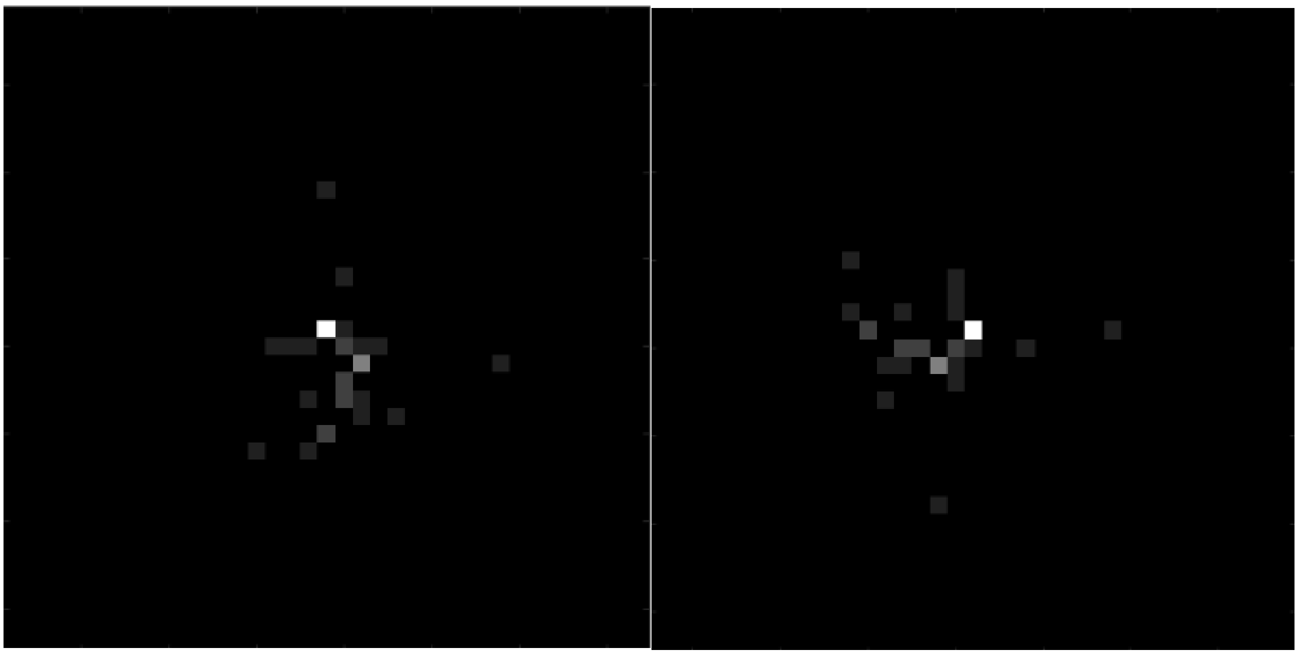
1. Energy Variance per jet
2. Transverse Momentum Variance per jet
3. Energy Kurtosis per jet
4. Transverse Momentum Kurtosis per jet

12 channel image is generated for every jet.

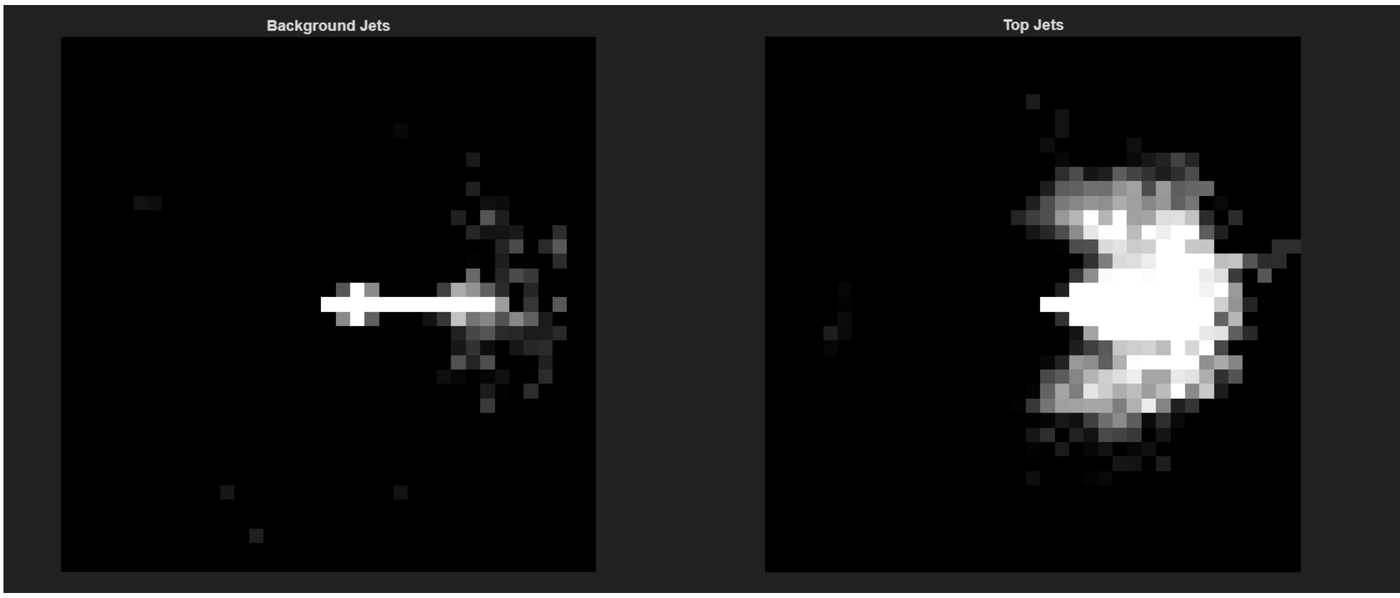
Data Preprocessing:

There are 3 steps involved in the preprocessing of the images:

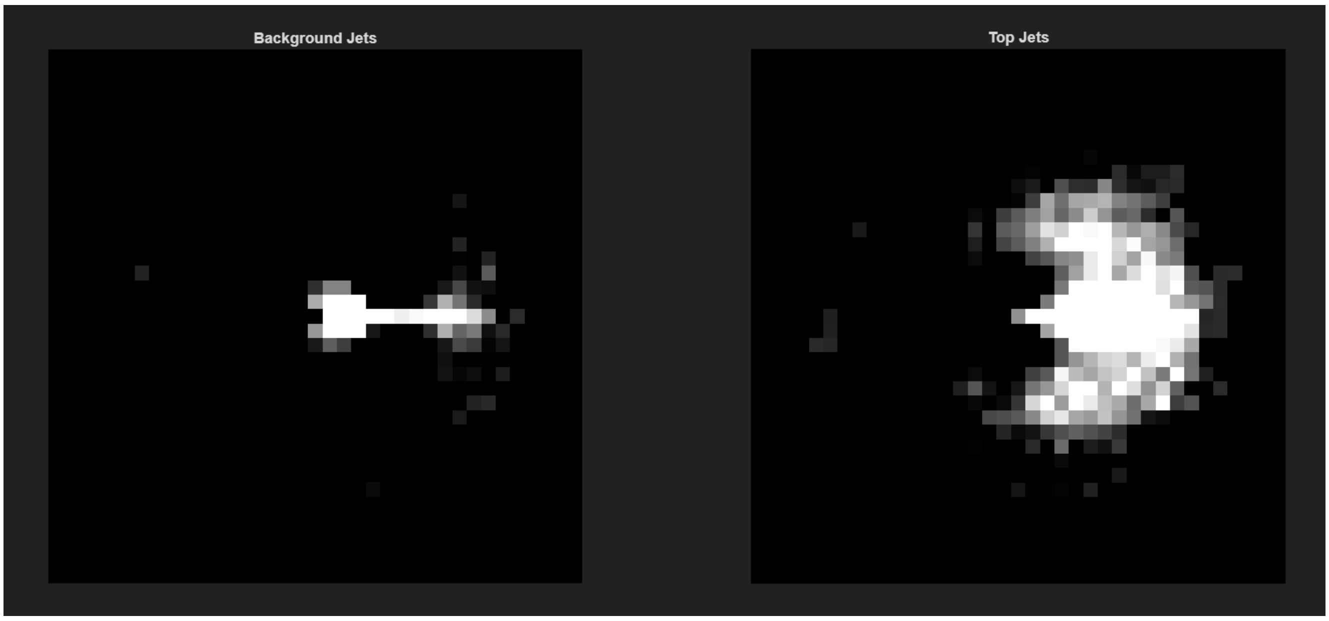
1. Rotation Invariance: The relative orientation in which the constituents of the Jet fall onto the bin have no significance in distinguishing between Top Jet and Standard (background) Jet. For example, the two images given below show the same Jet,



It is quite possible that in the dataset, the jet directions aren't oriented in any specific way. To make the distribution of constituents similar and not lead the model into "misbelieving" that orientation in any specific direction influences the nature of the Jet, we choose the second most energetic constituent of the Jet (as the first one is already centered at coordinates (19, 19)), and rotate it so that it aligns with the positive direction of the X-axis. This way we ensure that the major constituents are oriented in the same direction. This is quite helpful as this trains the model to specifically look for the constituents' distribution pattern, which is generalized over all the Jets.
Averaged Jets Visualization for Energy Sum channel



Averaged Jets Visualization for Energy Skewness channel



2. Even after the alignment of the second most energetic constituent, the direction of rotation of the image can also change the model's behaviour. For example, if we rotate the image clockwise, the energy distribution on the upper part may be higher, but if we rotate in anticlockwise direction, the energy distribution on the lower part will be higher.
   The direction of rotation is set to be Clockwise in my approach, and to counter the energy issue, I calculated the energy of all constituents in the oriented image in the upper part, that is rows 1 to 18 and the lower part, that is rows 20 to 37. I set all the images to have the highest energy in the bottom part of the image, so if the sum of energy in the top part is higher than the bottom part, we flip the image vertically so that the bottom part is more energetic. This standardizes the constituent values in the image across all Jets and the model doesn't learn on the oriental aspects.
   The difference would not be visible in the averaged jet visualization, hence they are not shown.

3. Then we need to perform feature scaling, so that the range of the variables do not affect the weights of the model, so we perform normalization. We have 2 options for normalization, so to choose we have to consider that:
   1. Each image is 37×37 with **sparse activations** (mostly zeros).

2. **Each channel is semantically independent**, and in each channel the **same pixels tend to be activated** across images.

Option 1: Pixel-wise normalization

Normalize each pixel (i, j) across **all images**, for each channel.

- For each channel:
    - For each pixel (i, j), compute the **mean and std over 30,000 images**
    - Normalize all pixel values at (i, j) accordingly

Option 2: Image-wise normalization

Normalize each image (i.e., the entire 37×37 slice per channel) across **all images**, for each channel.

- For each channel:
    - For each image:
        - Compute **mean and std of the whole image**
        - Normalize that image accordingly

Why Pixel-wise normalization is better for this case:

Criterion	Pixel-wise Normalization	Image-wise Normalization
Sparsity Handling	Preserves structure of zeros; avoids distorting images with few active pixels	Mean and standard deviation may be unstable due to too many zeros
Stable statistics	Large number of samples per pixel (30,000) leads to stable normalization	Each image is small (37×37); too few values for robust stats
Spatial signal stability	If activations occur at fixed locations, this leverages that pattern	Ignores spatial information — treats each image as a "bag of pixels"
CNN compatibility	CNNs can still learn filters on fixed normalized spatial patterns	Works, but noise from poor normalization could hinder learning
Risk of distorting zero regions	Zero-valued pixels stay consistently scaled	Sparse images with all/mostly zeros get noisy z-scores

Our data like structured tabular data where each pixel location is a distinct feature:

- **Pixel-wise normalization** = normalize each feature (column) based on its distribution
- **Image-wise normalization** = normalize each row independently — but each row is sparse, so it's noisy

In that analogy, pixel-wise is the more robust, reliable approach.

Step 2: Deep CNN (ResNeXt + SE attention layer + multi-modal learning)

A custom architecture of CNN is created based on the idea of Grouped Convolutions and Aggregated Residual transformations in ResNeXt coupled with Squeeze and Excite Blocks to incorporate Channel Attention. The input image dimension is 37 * 37 * 12 and the dimensions at every stage are shown in the flowchart.

Grouped Convolutions:

- Grouped convolutions split input channels into independent groups, processing each group separately, rather than applying filters across all input channels.
- They force separate filter groups to focus on different channel subsets, hence encouraging specialization and diversity in feature extraction.
- Also splitting into groups reduces the dense connectivity of filters, acting like a structural regularizer by limiting information sharing across all channels.

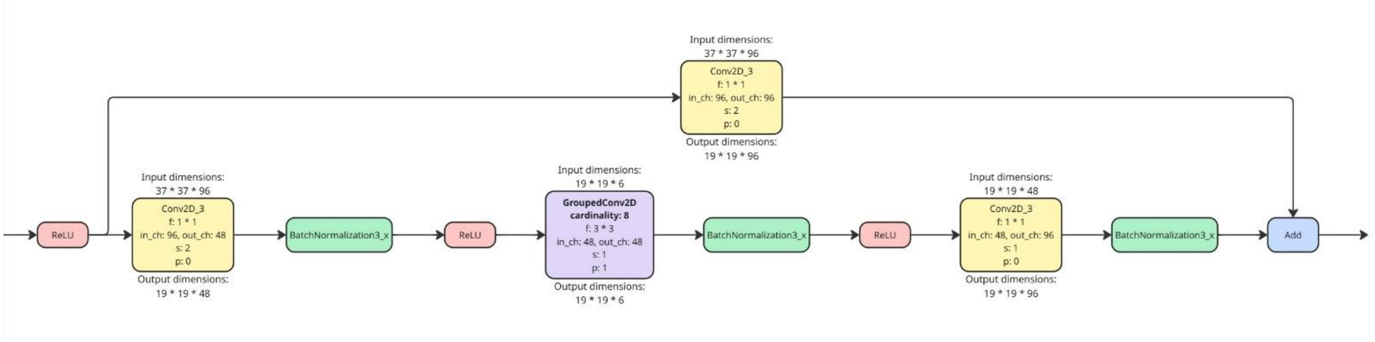
<u>Advantages over normal convolution operations:</u>

- **Reduced Computational Cost & Parameters (Efficiency):** Fewer parameters and multiplications mean faster models and lower memory usage. Grouped convolutions significantly cut down FLOPs without sacrificing accuracy.
- Each group can learn **distinct, complementary patterns**, enhancing the model's representational power.
- Splitting into groups **reduces the dense connectivity of filters**, which prevents overfitting.

Residual Blocks:

- The main idea behind residual blocks is to allow the network to learn **residuals**, or the difference between the input and the output, instead of trying to learn the full transformation from scratch.
- A residual block consists of two main components:
    - **Convolutional Layers:** These layers apply transformations to the input.
    - **Skip Connection:** This connection "skips" over the convolutional layers and adds the original input directly to the output of the convolutional block.

  One of the residual blocks in my architecture is shown below:



  The **output of the residual block** is the sum of the result from the convolutional layers and the original input to the block (the residual).
- If you denote the input to a residual block as ‘X’, and the transformation applied by the convolutional layers as ‘F(X)’, then the output ‘Y’ of the residual block is:

$$Y = F(X) + X$$

- Breakdown of the Block's Components:
  - **The Convolutional Layers:**
    - We’re using a **2D convolution** in an image classification task so the convolution layers will apply a set of filters to the input image.
    - The convolution is be followed by batch normalization (BN) and **activation function ReLU** (Rectified Linear Unit)
  - **The Skip Connection (Identity Shortcut):**
    - The key feature of the residual block is the **skip connection**, where the input $X$is added directly to the output of the convolution layers, before passing through the next layer.
    - This "shortcut" allows the network to **preserve information** from earlier layers and propagate it deeper into the network without suffering from issues like vanishing gradients.
    - If the network learns that the identity function is the best transformation (i.e., no change to the input is needed), the residual block will effectively pass the input directly to the output. This helps preserve the original information.
  - **Element-wise Addition:**
    - The input X is **added elementwise** to the output of the transformation F(X). This ensures that the network can learn not just the transformation of the input, but also to skip some layers if needed (by learning that the identity function is optimal in some cases).

Advantages over simple CNN layers:

1. **Avoiding Vanishing/Exploding Gradients**: They consist of residual blocks, which introduce skip connections, where the input to the block is added to the output. This allows the gradient to propagate directly through skip connections as well, hence the vanishing/exploding gradient.
2. Improved Training of Deep Networks: The skip connections help to maintain the original signal, making it easier for the network to learn identity mappings. This allows the number of **layers to be increased** without any performance degradation.
3. The skip connections in residual blocks allow the network to learn the **identity mapping**. This means that if adding new layers doesn’t help (or hurts performance), the network can learn to simply "skip" the new transformations.

Squeeze-and-Excitation (SE) Block for Channel Attention

1. Purpose of SE Block

- Designed to improve the **representational power** of convolutional neural networks by explicitly modeling the **interdependencies between channels**.

- Enables the network to **dynamically recalibrate** channel-wise feature responses, emphasizing the informative features while suppressing less useful ones.

2. Architecture Breakdown

- **Global Average Pooling (Squeeze Step):**

    - Converts the spatial dimensions (H × W) of each channel into a single value.
    - Outputs a vector representing the **global context** of each channel.
    - Helps the network understand the overall importance of each channel rather than spatial locations.
- **Bottleneck Fully Connected Layers (Excitation Step):**
    - Implemented as **1x1 convolutions** in this architecture (equivalent to dense layers).
    - First convolution reduces dimensionality by a factor (typically 8), acting as a **bottleneck** to reduce computational load and introduce non-linearity.
    - ReLU activation introduces non-linearity after the first convolution.
    - Second convolution restores the original number of channels.
- **Sigmoid Activation:**
    - Outputs a **scaling factor** (between 0 and 1) for each channel.
    - Represents the relative importance of each channel after learning global dependencies.
- **Multiplication Layer (Recalibration Step):**
    - The original feature map is **multiplied channel-wise** by the scaling factors.
    - This operation adjusts the intensity of each channel's contribution to the final output.

3. Key Advantages

- **Lightweight:** Adds minimal overhead to the model in terms of parameters.
- **Plug-and-Play Module:** Can be integrated into any existing CNN architecture like ResNet, MobileNet, etc.
- **Improves Channel Sensitivity:** Captures **global channel dependencies** without sacrificing spatial resolution.

Multi Modal Learning:

The 4 additional parameters computed per Jet are: Energy Skewness, Transverse Momentum Skewness, Energy Kurtosis and Transverse Momentum Kurtosis.

Skewness represents the asymmetry or lopsidedness of a distribution of energy/ transverse momentum. It tells whether the energy/transverse momentum is skewed to the left or the right, indicating how it is spread out.

There are three types of skewness:

1. Positive Skew (Right Skew)

    Meaning: The right tail (larger values) is longer or fatter than the left tail.

    Visual Representation: Most of the data points are concentrated on the left side, with a few large values stretching out on the right side.

Characteristics:

- The mean is greater than the median.
- Skewness value > 0.

2. Negative Skew (Left Skew)

Meaning: The left tail (smaller values) is longer or fatter than the right tail.

Visual Representation: Most of the data points are concentrated on the right side, with a few very small values on the left.

Characteristics:

- The mean is less than the median.
- Skewness value < 0.

3. No Skew (Symmetric Distribution)

Meaning: The distribution is symmetric, and both tails are of equal length.

Visual Representation: The data is evenly distributed around the mean.

Characteristics:

- The mean and median are approximately equal.
- Skewness value ≈ 0.

Kurtosis measures the **tailedness** of the distribution. It tells us how heavy or light the tails of the distribution are relative to a normal distribution.

Types of Kurtosis:

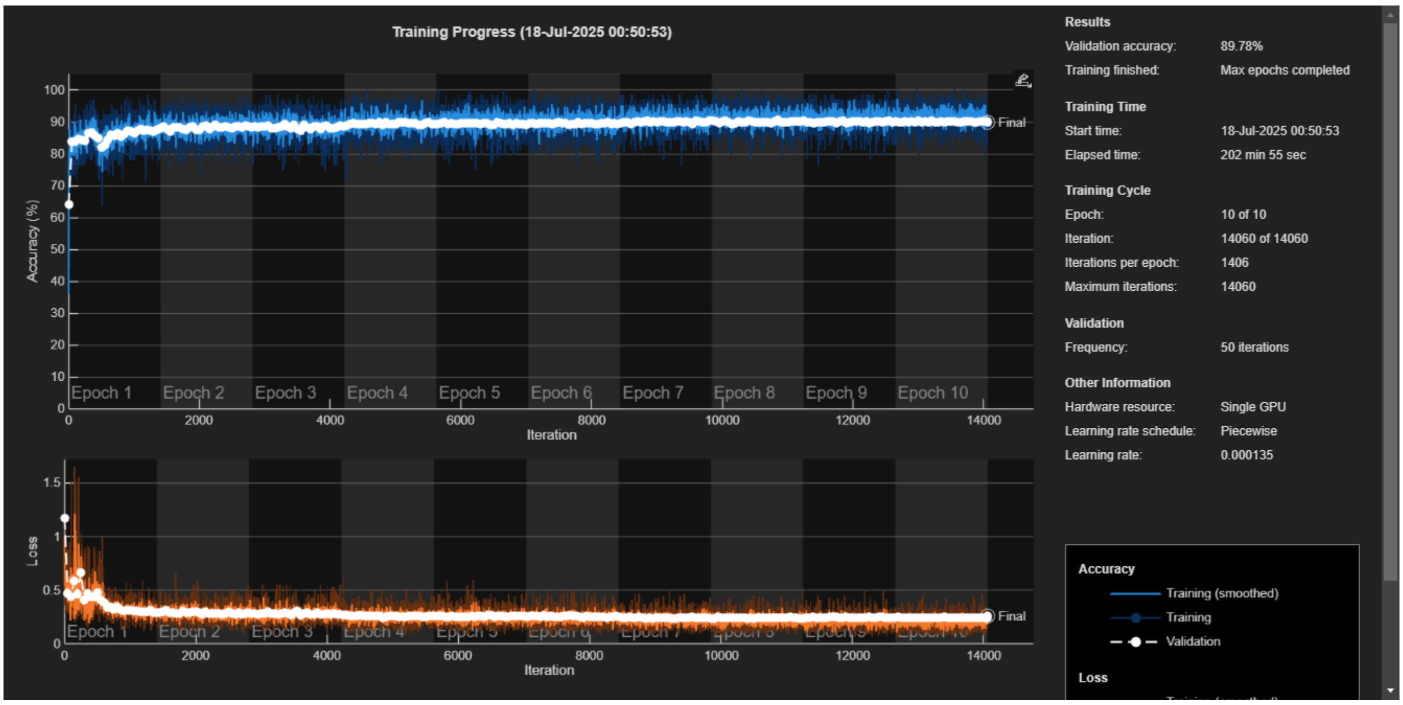
1. Mesokurtic:
   a. Meaning: A distribution with kurtosis like the normal distribution. The tails are neither too heavy nor too light.
   b. Visual Representation: The shape of the distribution is relatively moderate, like a bell curve.
   Kurtosis Value: 3 (for normal distribution).
      - Excess Kurtosis (Kurtosis - 3): 0 (for normal distribution).
2. Leptokurtic:
   a. Meaning: A distribution with heavy tails and a sharp peak. It means that the data has more extreme values (outliers) than a normal distribution.
   b. Visual Representation: The peak is sharper, and the tails are fatter.
   Kurtosis Value: Greater than 3.
      - Excess Kurtosis: Positive values, indicating heavier tails than a normal distribution.
3. Platykurtic:
   a. Meaning: A distribution with light tails and a flatter peak than the normal distribution. The data is more evenly spread out with fewer extreme values.
   b. Visual Representation: The distribution is flatter in the middle, with thinner tails.

Kurtosis Value: Less than 3.

- Excess Kurtosis: Negative values, indicating lighter tails than a normal distribution.

Outputs:

Upon training the model with a training set of 90000 and testing set of 10000, an accuracy of **90.33%** is achieved



Results

Validation accuracy:	89.78%
Training finished:	Max epochs completed

Training Time

Start time:	18-Jul-2025 00:50:53
Elapsed time:	202 min 55 sec

Training Cycle

Epoch:	10 of 10
Iteration:	14060 of 14060
Iterations per epoch:	1406
Maximum iterations:	14060

Validation

Frequency:	50 iterations

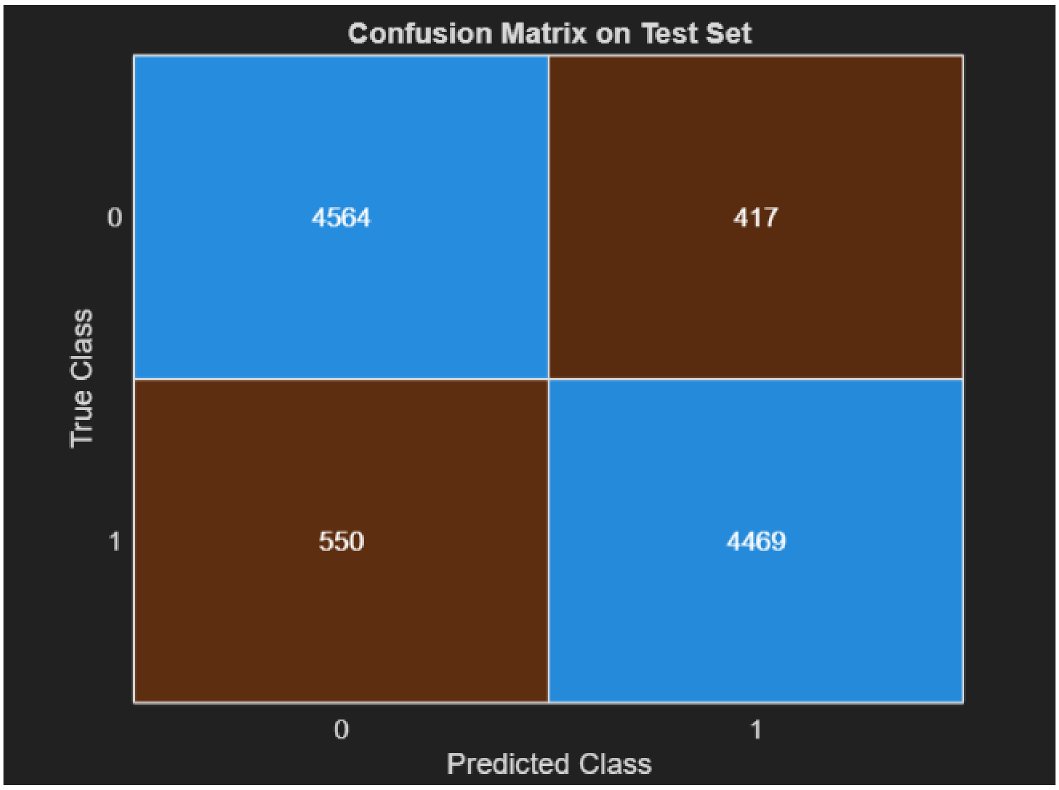
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Piecewise
Learning rate:	0.000135

Accuracy
Training (smoothed)
Training
Validation

Loss

The Confusion Matrix for the same is:



True Class \ Predicted Class	0	1
0	4564	417
1	550	4469

Step 3: Convert to HDL

The complete process for generating HDL code from a trained deep learning model and deploying the model to an FPGA using MATLAB’s Deep Learning HDL Toolbox™ and HDL Coder™ is as follows:

HDL Code Generation Process:

The main objective is to generate synthesizable HDL (VHDL/Verilog) code for the trained CNN model. The code for the same can be found in hdl_code_gen.m file in deploy folder.

Output:

- HDL (VHDL/Verilog) source files are saved inside the specified project folder (HDLProject/hdl).
- These files can be synthesized using Xilinx Vivado or other synthesis tools.

FPGA Deployment Process:

The objective is to deploy the trained model onto a supported FPGA board for hardware acceleration. The code for the same can be found in deploy_on_fpga.m file in deploy folder.

- Ensure physical connection between the host PC and FPGA via Ethernet or JTAG.
- The 'Bitstream' name should match the target FPGA board (e.g., 'zcu102_single').
- During deployment, the FPGA is programmed, and model parameters are downloaded automatically.