The dataset has four main elements:

X_jets – Three-channel images, all 125 × 125 pixels in size.

m0 – Stands for mass.

pt – Stands for transverse momentum.

y – Refers to labels.

This visualize_channels function will plot and compare a multi-channel jet image by showing:

```
    channel_labels = ['Track', 'ECAL', 'HCAL']
    for i in range(3):
        img = axes[i+1].imshow(X_sample[:, :, i], cmap='viridis', vmin=-0.5, vmax=2.0, interpolation='nearest')
        axes[i+1].set_title(channel_labels[i])
        fig.colorbar(img, ax=axes[i+1], shrink=0.5)
    plt.show()
```

This function intensity calculates and prints statistical information regarding the intensity values over various channels in the dataset.

Traverses all three channels (Track, ECAL, HCAL) to calculate:

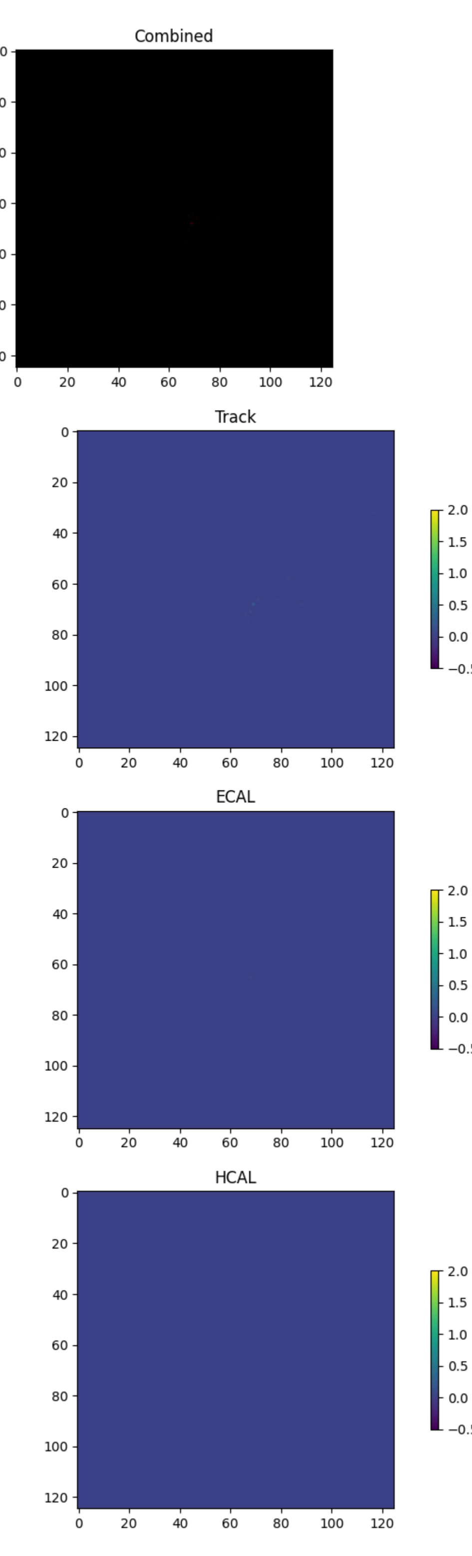
Maximum and Minimum intensity values Mean intensity value Standard deviation of intensity distribution Calculates an overall intensity image by adding up the three channels and studying its statistical properties.

```
[6]: def intensity(X, y):
    for i, channel_name in enumerate(["Track", "ECAL", "HCAL"]):
        print(f"Max value of intensity along {channel_name} channel: {np.max(X[:,:,:,i])}, Min value: {np.min(X[:,:,:,i])}")
        print(f"Mean intensity value along {channel_name} channel: {np.mean(X[:,:,:,i])}")
        print(f"Standard Deviation: {np.std(X[:,:,:,i])}\n")
    combined_dataset = np.sum(X, axis=-1, keepdims=True)
    print("Max value of intensity in combined channel image:", np.max(combined_dataset[:,:,:,0]), "Min value:", np.min(combined_dataset[:,:,:,0]))
    print("Mean intensity value in combined channel:", np.mean(combined_dataset[:,:,:,0]))
    print("Standard Deviation:", np.std(combined_dataset[:,:,:,0]), "\n")
```

Plotting the provided dataset, we understand that intensity values (calorimeter hits) in all the channels (Track, ECAL, HCAL) fluctuate highly, and that data pre-processing needs to be done.

```
[7]: intensity(X,y)
visualize_channels(X)
```

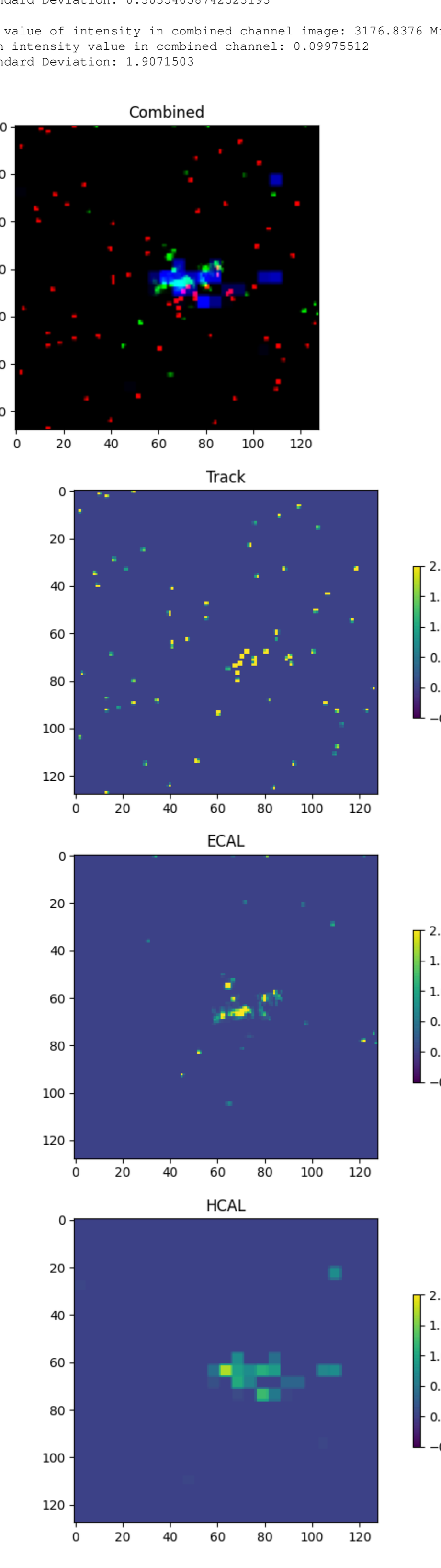
```
Max value of intensity along Track channel: 10.088105201721191, Min value: 0.0
Mean intensity value along Track channel: 7.841042679501697e-05
Standard Deviation: 0.0038757636211812496

Max value of intensity along ECAL channel: 9.334086418151855, Min value: 0.0
Mean intensity value along ECAL channel: 4.9682072130963206e-05
Standard Deviation: 0.002107673790305853

Max value of intensity along HCAL channel: 0.3597966134548187, Min value: 0.0
Mean intensity value along HCAL channel: 3.119495158898644e-05
Standard Deviation: 0.0005133272497914732

Max value of intensity in combined channel image: 12.1562605 Min value: 0.0
Mean intensity value in combined channel: 0.00015928746
Standard Deviation: 0.0046356586
```



The preprocess_images() function scales images from (125, 125, 3) to (128, 128, 3) for uniformity with bilinear interpolation. It next normalizes pixel values to be 0 mean and 1 standard deviation, stabilizing the training. It lastly clips negative intensity values to zero, preventing invalid pixel values and lowering noise.

```
[8]: def preprocess_images(images):
    from skimage.transform import resize
    # Resizing images from (125, 125, 3) to (128, 128, 3)
    processed = np.array([resize(img, (128, 128), anti_aliasing=True) for img in images], dtype=np.float32)
    # Standardizing the distribution across all channels
    mean, std = np.mean(processed), np.std(processed)
    # Clipping negative intensity values to 0
    return np.clip((processed - mean) / std, 0, None)

X = preprocess_images(X)
```

```
Max value of intensity along Track channel: 2362.630126953125, Min value: 0.0
Mean intensity value along Track channel: 0.04981083795428276
Standard Deviation: 1.4279361963272095

Max value of intensity along ECAL channel: 2024.9141845703125, Min value: 0.0
Mean intensity value along ECAL channel: 0.03096914105117321
Standard Deviation: 0.928253710269928

Max value of intensity along HCAL channel: 231.5849151611328, Min value: 0.0
Mean intensity value along HCAL channel: 0.018975133076310158
Standard Deviation: 0.30354058742523193

Max value of intensity in combined channel image: 3176.8376 Min value: 0.0
Mean intensity value in combined channel: 0.09975512
Standard Deviation: 1.9071503
```



```
[10]: class Reparameterization(nn.Module):
    def forward(self, mean, log_variance):
        batch_size, latent_dim = mean.shape
        epsilon = Normal(0, 1).sample((batch_size, latent_dim)).to(mean.device)
        return mean + torch.exp(0.5 * log_variance) * epsilon
```

```
        return x_rec,mu,logvar
```

```
[12]: class VAELoss(nn.Module):
    def __init__(self):
        super().__init__()
        self.bce_loss = nn.BCELoss()
    def forward(self, x_rec, x, mu, logvar):
        recon_loss = self.bce_loss(x_rec, x)
        kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=1).mean()
        return 500 * recon_loss + kl_loss
```

```
[13]: trackMax = np.max(X[:,:,:,0])
ecalMax = np.max(X[:,:,:,1])
hcalMax = np.max(X[:,:,:,2])


X[:,:,:,0] = X[:,:,:,0]/trackMax
X[:,:,:,1] = X[:,:,:,1]/ecalMax
X[:,:,:,2] = X[:,:,:,2]/hcalMax
```

```
[14]: intensity(X,y)
```

```
Max value of intensity along Track channel: 1.0, Min value: 0.0
Mean intensity value along Track channel: 2.108279113599565e-05
Standard Deviation: 0.0006043842295184731

Max value of intensity along ECAL channel: 1.0, Min value: 0.0
Mean intensity value along ECAL channel: 1.5294048353098333e-05
Standard Deviation: 0.0004584163543768227

Max value of intensity along HCAL channel: 1.0, Min value: 0.0
Mean intensity value along HCAL channel: 8.193597022909671e-05
Standard Deviation: 0.0013107095146551728

Max value of intensity in combined channel image: 1.6550478 Min value: 0.0
Mean intensity value in combined channel: 0.00011831283
Standard Deviation: 0.0016820203
```

Loading these datasets using standard PyTorch dataloader ->

```
[15]: X_jets_train, X_jets_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_jets_train, X_jets_val, y_train, y_val = train_test_split(X_jets_train, y_train, test_size=0.25, random_state=42)
```

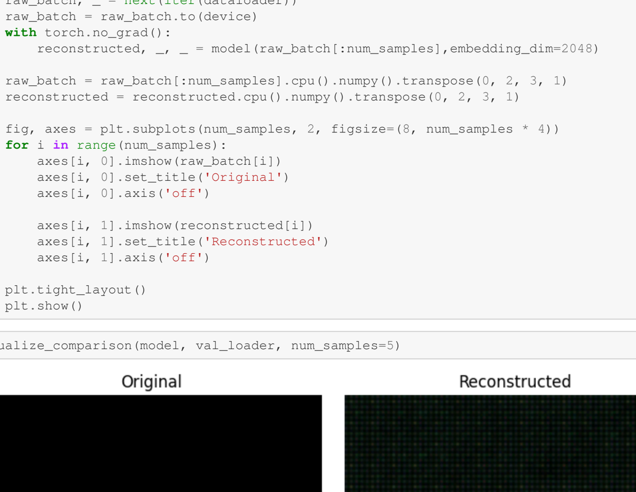
```
[16]: class JetDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __getitem__(self, index):
        image = self.X[index].permute(2, 0, 1)
        return image, self.y[index]

    def __len__(self):
        return len(self.X)

train_dataset = JetDataset(torch.from_numpy(X_jets_train), torch.from_numpy(y_train))
val_dataset = JetDataset(torch.from_numpy(X_jets_val), torch.from_numpy(y_val))
test_dataset = JetDataset(torch.from_numpy(X_jets_test), torch.from_numpy(y_test))
```

```
[17]: from tqdm import tqdm
```

```
[18]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = VAE_autoencoder(embedding_dim= 2048).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.0001)
criterion = VAELoss()

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=False)
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)

# Training loop
num_epochs = 20
train_losses = []
val_losses = []
latent_representations = []
labels = []

for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(tqdm(train_loader, desc=f"Epoch {epoch}")):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data,embedding_dim=2048)
        loss = criterion(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
    train_losses.append(train_loss / len(train_loader))

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for batch_idx, (data, _) in enumerate(val_loader):
            data = data.to(device) # Move data to device
            recon_batch, mu, logvar = model(data,embedding_dim=2048)
            loss = criterion(recon_batch, data, mu, logvar)
            val_loss += loss.item()
    val_losses.append(val_loss / len(val_loader))

    print(f'Epoch: {epoch+1}, Train Loss: {train_losses[-1]}, Val Loss: {val_losses[-1]}')
```

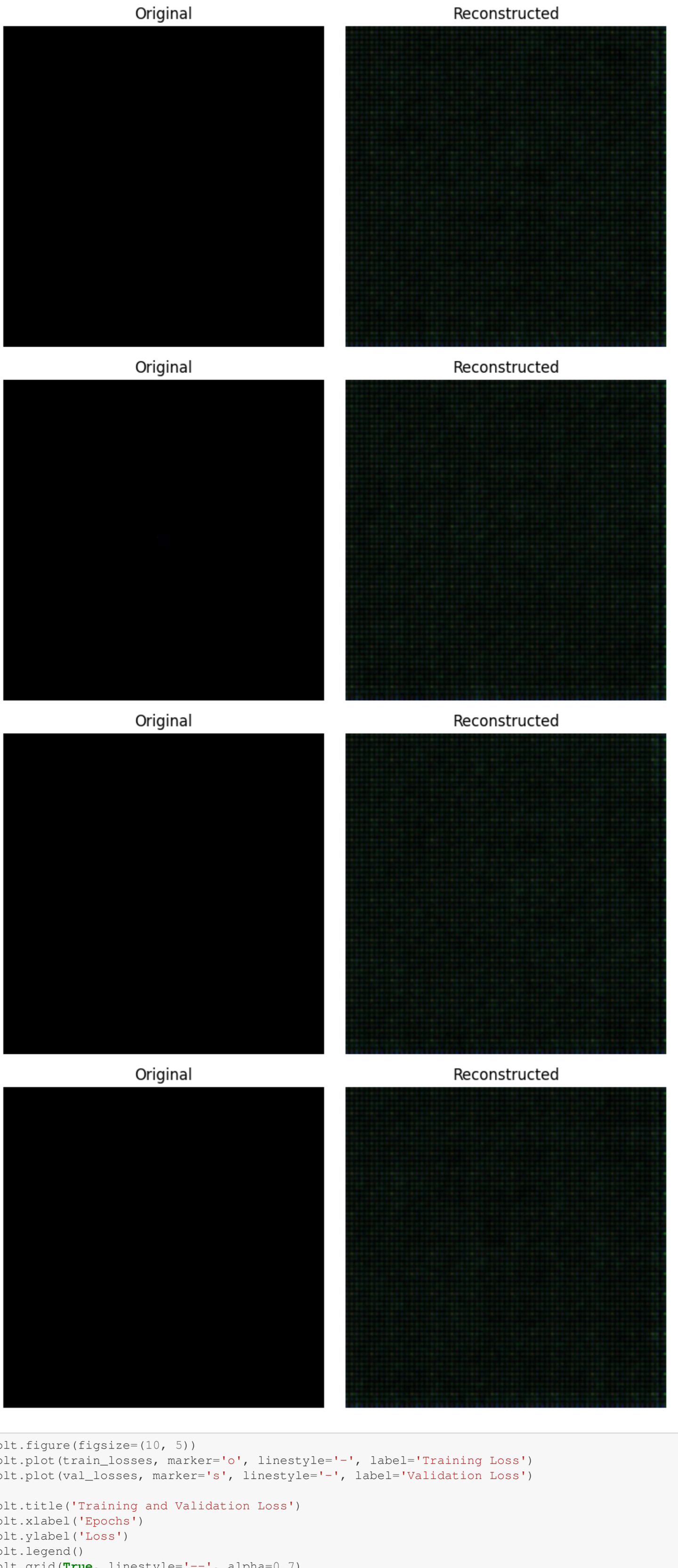
```
Epoch 0: 100%|██████████| 42/42 [00:03<00:00, 11.00it/s]
Epoch: 1, Train Loss: 435.89149402436755, Val Loss: 337.7340305873326
Epoch 1: 100%|██████████| 42/42 [00:02<00:00, 15.17it/s]
Epoch: 2, Train Loss: 311.4920327322824, Val Loss: 268.3485761369978
Epoch 2: 100%|██████████| 42/42 [00:02<00:00, 15.25it/s]
Epoch: 3, Train Loss: 244.51296960739862, Val Loss: 216.3068117414202
Epoch 3: 100%|██████████| 42/42 [00:02<00:00, 15.22it/s]
Epoch: 4, Train Loss: 197.7042937505813, Val Loss: 178.73394557407923
Epoch 4: 100%|██████████| 42/42 [00:02<00:00, 15.09it/s]
Epoch: 5, Train Loss: 165.86970919654482, Val Loss: 152.979127066476
Epoch 5: 100%|██████████| 42/42 [00:02<00:00, 15.20it/s]
Epoch: 6, Train Loss: 142.64774903796967, Val Loss: 133.04739597865515
Epoch 6: 100%|██████████| 42/42 [00:02<00:00, 15.17it/s]
Epoch: 7, Train Loss: 124.61342003231957, Val Loss: 117.0137574332101
Epoch 7: 100%|██████████| 42/42 [00:02<00:00, 15.16it/s]
Epoch: 8, Train Loss: 109.91375968569801, Val Loss: 103.812073298863
Epoch 8: 100%|██████████| 42/42 [00:02<00:00, 15.19it/s]
Epoch: 9, Train Loss: 97.58177839006696, Val Loss: 92.56929288591657
Epoch 9: 100%|██████████| 42/42 [00:02<00:00, 15.21it/s]
Epoch: 10, Train Loss: 87.00792367117745, Val Loss: 82.6572756086077
Epoch 10: 100%|██████████| 42/42 [00:02<00:00, 15.17it/s]
Epoch: 11, Train Loss: 77.89108512515114, Val Loss: 74.02931540352958
Epoch 11: 100%|██████████| 42/42 [00:02<00:00, 15.23it/s]
Epoch: 12, Train Loss: 69.96448698497954, Val Loss: 66.70958982195172
Epoch 12: 100%|██████████| 42/42 [00:02<00:00, 15.22it/s]
Epoch: 13, Train Loss: 63.03155108860561, Val Loss: 60.1104987008231
Epoch 13: 100%|██████████| 42/42 [00:02<00:00, 15.25it/s]
Epoch: 14, Train Loss: 56.954902013142906, Val Loss: 54.45834732055664
Epoch 14: 100%|██████████| 42/42 [00:02<00:00, 15.22it/s]
Epoch: 15, Train Loss: 51.60919716244652, Val Loss: 49.46472358703613
Epoch 15: 100%|██████████| 42/42 [00:02<00:00, 15.22it/s]
Epoch: 16, Train Loss: 46.91248148963565, Val Loss: 45.13199043273926
Epoch 16: 100%|██████████| 42/42 [00:02<00:00, 15.24it/s]
Epoch: 17, Train Loss: 42.77829778762091, Val Loss: 41.18230628967285
Epoch 17: 100%|██████████| 42/42 [00:02<00:00, 15.24it/s]
Epoch: 18, Train Loss: 39.143736884707494, Val Loss: 37.87361580984933
Epoch 18: 100%|██████████| 42/42 [00:02<00:00, 15.20it/s]
Epoch: 19, Train Loss: 35.939519609723774, Val Loss: 34.8214956011091
Epoch 19: 100%|██████████| 42/42 [00:02<00:00, 15.25it/s]
Epoch: 20, Train Loss: 33.07708204360235, Val Loss: 32.22350992475237
```

```
[19]: from sklearn.manifold import TSNE
import seaborn as sns
def plot_tsne(model, data_loader):
    model.eval()
    latent_representations = []
    labels = []
    with torch.no_grad():
        for data, label in data_loader:
            data = data.to(device)
            _, mu, _ = model(data,embedding_dim=2048)
            latent_representations.append(mu.cpu().numpy())
            labels.append(label.cpu().numpy())
    latent_representations = np.concatenate(latent_representations)
    labels = np.concatenate(labels)
    tsne = TSNE(n_components=2, perplexity=30, random_state=42)
    latent_tsne = tsne.fit_transform(latent_representations)
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=latent_tsne[:, 0], y=latent_tsne[:, 1], hue=labels, palette="coolwarm", s=20)
    plt.title("t-SNE Projection of Latent Space", fontsize=16)
    plt.xlabel("t-SNE Component 1")
    plt.ylabel("t-SNE Component 2")
    plt.show()

plot_tsne(model, val_loader)
```




```
[20]: def visualize_comparison(model, dataloader, num_samples=5):
    model.eval()
    raw_batch, _ = next(iter(dataloader))
    raw_batch = raw_batch.to(device)
    with torch.no_grad():
        reconstructed, _, _ = model(raw_batch[:num_samples],embedding_dim=2048)

    raw_batch = raw_batch[:num_samples].cpu().numpy().transpose(0, 2, 3, 1)
    reconstructed = reconstructed.cpu().numpy().transpose(0, 2, 3, 1)

    fig, axes = plt.subplots(num_samples, 2, figsize=(8, num_samples * 4))
    for i in range(num_samples):
        axes[i, 0].imshow(raw_batch[i])
        axes[i, 0].set_title('Original')
        axes[i, 0].axis('off')

        axes[i, 1].imshow(reconstructed[i])
        axes[i, 1].set_title('Reconstructed')
        axes[i, 1].axis('off')

    plt.tight_layout()
    plt.show()
```

```
[21]: visualize_comparison(model, val_loader, num_samples=5)
```




```
[22]: plt.figure(figsize=(10, 5))
plt.plot(train_losses, marker='o', linestyle='-', label='Training Loss')
plt.plot(val_losses, marker='s', linestyle='-', label='Validation Loss')

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

plt.show()
```