X-Ray threat detection

Riccardo Inverardi Galli
September 1, 2024

Abstract

The scope of this project was to create a convolutional neural network designed to detect dangerous items in X-ray luggage scans. A combination of convolutional layers, batch normalization, and linear activations was employed to achieve this goal. The model was trained on a dataset containing both dangerous and potentially dangerous items. To improve the model's ability to detect general features across different classes, the dataset was augmented using random rotations and random cropping, resulting in over 15,000 images across 8 different classes. After 60 epochs of training, the model performs well on the validation set, achieving at least 0.90 accuracy in every class. These results were consistent across different training/validation splits, indicating that it possessses ability to generalize. The model was developed exclusively using Python's PyTorch library.

1 Introduction

In commercial aviation, the security of crew members and passengers has always been a major concern. X-ray scans have long played a crucial role in ensuring passenger safety, and the use of machine learning in this field is growing rapidly. While it is still early to consider replacing human supervision with automated systems, a reliable threat detection model could greatly assist human operators, who are naturally prone to error or may be misled by complex images. The goal of this model, though still in early development, is to provide support in situations where the human eye might be deceived. The model was trained on a dataset containing images of luggage with eight different classes of dangerous items: cameras, cellphones, various electronic devices, knives, laptops, lighters, power banks, and scissors. Currently, it is only trained on classes that contain dangerous items and does not yet have the ability to recognize non-dangerous luggage.

2 The model - Preprocessing step

The model is built as a subclass of PyTorch's nn.Module class. It contains two preprocessing layers that perform gaussian blur and edge detection by applying 3x3 convolutions with the following kernels.

$$\underset{\text{Gaussian Blur}}{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}} \qquad \underset{\text{Edge Detection}}{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}}$$

This helps in getting rid of unnecessary noise during training, so that the model only focuses on the actual luggage, discarding any background.

3 Convolutional Blocks and Fully Connected Layers

The first convolutional block expands the preprocessed images to 32 channels. Padding is applied to prevent the reduction of image dimensions due to the convolution's effect on border pixels. Following this, batch normalization is performed, and the output is passed through the hyperbolic tangent activation function. Finally, a max-pooling operation with a 2×2 window is applied. The hyperbolic tangent activation function, denoted as $\tanh(x)$, is defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The same convolution is again repeated in the second block, reducing the number of output channels from 32 to 16. The final linear layers take the flattened output of the convolutional blocks and map it to 32 and then 8 dimensional input, reprensenting the 8 classes of the model.

4 The Pipeline

The model is trained through a pipeline that covers all essential steps for training. Initially, images are loaded, resized to 64x64 pixels, and converted into tensors. The dataset is then split into training and validation sets using a custom splitting function, ensuring that validation is conducted on a separate subset of data, which helps prevent overfitting. Images belonging to the same class are stacked together to facilitate efficient batch processing. At this stage, normalization is performed independently on each class on both the training and validation sets to standardize the data. The normalized images are subsequently formatted, each image tensor is paired with its corresponding class label and organized into a flat list of tuples. Data loaders for both the training and validation sets are created using torch.utils.data.DataLoader, with a batch size of 64. To perform gradient descent over the loss function, Stochastic Gradient Descent (SGD) is employed for optimizing the model, where the true gradient of the loss function is approximated using the gradient at a single sample. The model parameters are updated based on this approximated gradient, weighted by the learning rate, as described by the equation:

$$\theta := \theta + \alpha(y_i - h_\theta(x_i))x_i$$

Here, $h_\theta(x_i)$ represents the predicted value for sample x_i, y_i is the ground truth label, and α is the learning rate. Cross Entropy Loss is used as the loss function, chosen to avoid the need for combining LogSoftMax with Negative

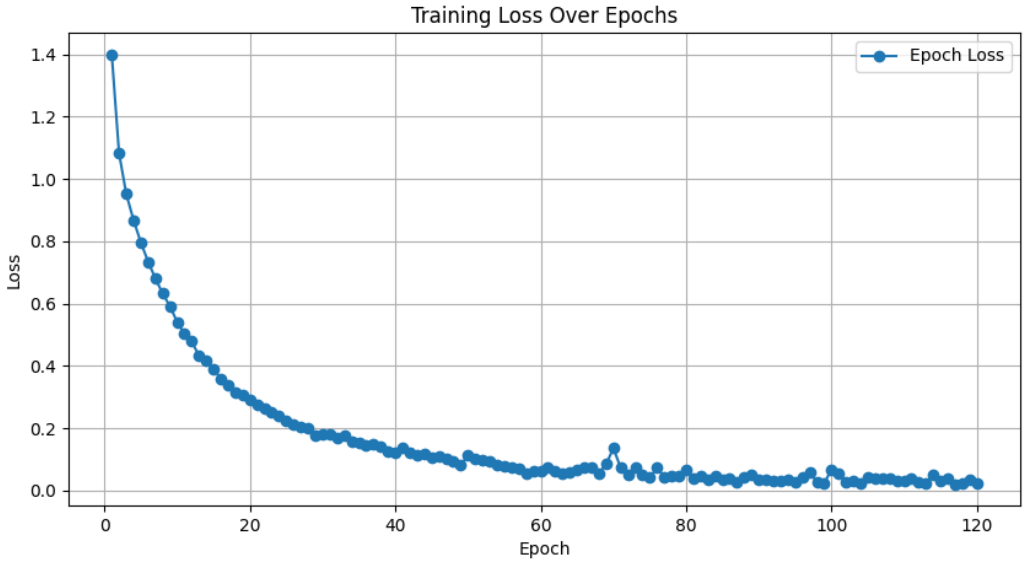

Figure 1: Loss function value

Log Likelihood Loss, thus simplifying the loss computation. L2 regularization is applied to the training loop by setting the weight decay parameter in the optimizer (SGD) object. This ensures that the model is less prone to overfitting. By training the model for more epochs, we see that the behaviour of the loss function using SGD tends to a global minimum after 80/90 epochs of training.

```
In [10]: model.parameters
Out[10]:
<bound method Module.parameters of MyModel(
  (gaussian_blur): Conv2d(3, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (edge_detection): Conv2d(3, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv1_batchnorm): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2_batchnorm): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=4096, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=8, bias=True)
)>
```

Figure 2: The model

5 Conclusion

Altough this model is still in the early stages of its development, it showcases the full power of convolutions, making inference possible over complex images while keeping the number of parameters in check. With an accuracy exceeding 0.90 across all classes on the validation set, the model shows strong generalization capabilities and reliability. Although still in early stages, it provides a valuable tool to support human operators in identifying potential threats, ultimately aiming to improve safety and efficiency in the security screening process. Future work will focus on expanding the model's capabilities to recognize non-dangerous items and integrating it into real-world screening systems.