Evaluation of Neural Object Detection Models for Human Detection in Infrared Images

PROJECT REPORT T1000

from the course of studies Computer Science - Artificial Intelligence

at the Cooperative State University Baden-Württemberg
Ravensburg Campus Friedrichshafen

by

Lukas Florian Richter

11.09.2025

Completion Time:	16 Wochen
Student ID, Course:	None, TIK24
Company:	Airbus Defence & Space, Taufkirchen
Supervisor in the Company:	René Loeneke

Declaration of Authorship

In accordance with clause 1.1.13 of Annex 1 to §§ 3, 4 and 5 of the Cooperative State University Baden-Württemberg's Study and Examination Regulations for Bachelor's degree programs in the field of Technology, dated 29.09.2017. I hereby declare that I have written my thesis on the topic:

Evaluation of Neural Object Detection Models for Human Detection in Infrared Images

independently and have used no other sources or aids than those specified. I further declare that all submitted versions are identical.

Taufkirchen 11.09.2025

Lukas Florian Richter

Abstract

This project report evaluates the performance of neural object detection models for detecting humans in infrared images. The study focuses on comparing different variations of the SSD (Single Shot Mutlibox Detector) model architecture, assessing their accuracy and inference speed, and identifying the most suitable model for the given task. Additionally, different preprocessing techniques are evaluated to improve the detection performance.

More specifically, the main contributions of this project are:

- Conceptualization of a simple and cost-efficient hardware setup for the purpose of on-premise human detection in infrared images
- Evaluation of different SSD model architectures
- Comparison between different preprocessing techniques
- Identification of the most suitable model for the given task
- A theoretical pipeline for the secure transmission of the detection results to a remote server

Table of Contents

List of Figures

List of Tables

Code Snippets

List of Acronyms

AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
AdaBoost	Adaptive Boosting
BGD	Batch Gradient Descent
BN	Batch Normalization
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DL	Deep Learning
FC	Fully Connected
FN	False Negative
FP	False Positive
FP16	16-bit floating point
FP32	32-bit floating point
FPS	Frames per Second
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
HNM	Hard-Negative Mining
HOG	Histogram of Oriented Gradients
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
INT8	8-bit integer
IR	Infrared
IoU	Intersection over Union
JIT	Just-In-Time
MBGD	Mini-Batch Gradient Descent
ML	Machine Learning
MPS	Metal Performance Shaders
MPT	Mixed Precision Training
NMS	Non-Maximum Suppression
NN	Neural Network
OSI	Open Systems Interconnection
R-CNN	Region-based Convolutional Neural Network
RAM	Random Access Memory
REST	Representational State Transfer
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
ReLU	Rectified Linear Unit
ResNet	Residual Network
SGD	Stochastic Gradient Descent
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
TCP	Transmission Control Protocol

TL	Transfer Learning
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit
VGG	Visual Geometry Group
VGP	Vanishing Gradient Problem
VOC	Visual Object Classes
ViT	Vision Transformer
YOLO	You Only Look Once
mAP	mean Average Precision

1 Introduction

With the increase in security threats to critical infrastructure, automated surveillance systems have become essential for ensuring the safety and security of people, infrastructure and property at scale. The ability to detect individuals on critical infrastructure premises is crucial to preventing unauthorized access and potential damage to assets. While conventional Red, Green, Blue (RGB)-based surveillance systems remain prevalent in many application, they face inherent limitations in challenging scenarios such as low-light conditions, adverse weather, fog, smoke and complete darkness during nighttime [1].

A compelling alternative to systems operating in the visible light domain are such capturing wavelengths in the infrared spectrum and thus offering consistent detection capabilities that are fundamentally independent of ambient lighting conditions, as they rely on heat signatures emitted directly by objects. This characteristic provides unique advantages for human detection, as the human body maintains a relatively constant temperature of approximately 37°C, creating distinct thermal signatures that remain visible regardless of environmental illumination [2].

The integration of deep learning architectures with thermal imaging thus opens new possibilities for automated systems that can reliably detect humans in the aforementioned scenarios with adverse conditions for conventional RGB-based concepts. However, most state-of-the-art object detection models have been primarily developed for and trained on RGB imagery. Given that the spectral, tectural and contrast characteristics of infrared images differ substantially from visible-light imagery, both due to the properties of those wavelengths themselves and of the sensors, those existing models might need to be adapted to achieve optimal performance.

This research addresses the critical need for systematic evaluation of neural object detection models specifically tailored for thermal human detection applications.

The study focuses on the SSD architecture, a prominent one-stage detection framework known for its balance between accuracy and computational efficiency. By examining multiple model variants with different backbone networks (VGG16 and ResNet152), initialization strategies (pretrained versus scratch training), and thermal-specific preprocessing techniques (image inversion and edge enhancement), this work provides comprehensive insights into optimal configurations for infrared surveillance systems.

The practical significance of this research extends beyond academic interest, addressing real-world challenges faced by the security and defense industry. In partnership with Airbus Defence & Space, this project explores the development of cost-efficient, edge-deployable thermal surveillance solutions that can operate reliably in challenging environments where traditional RGB systems fail.

1.1 Research Objectives and Contributions

This thesis makes several key contributions to the field of thermal image processing and computer vision:

TODO: Reconsider Objectives vs. Contributions, place later in the conclusion

1. **Preprocessing Technique Analysis**: Quantitative evaluation of thermal-specific image enhancement methods, including polarity inversion and edge enhancement, and their impact on detection accuracy.

2. **Backbone Network Comparison**: Detailed comparison between VGG16 and ResNet152 architectures in the context of thermal imagery, addressing the trade-offs between model complexity and performance.

3. **Practical Implementation Guidelines**: Development of actionable recommendations for deploying thermal surveillance systems in real-world environments, considering computational constraints and accuracy requirements.

4. **Dataset Integration Framework**: Unified evaluation approach across five diverse thermal datasets (FLIR ADAS v2 [3], AAU-PD-T [4], OSU-T [5], M3FD [6], KAIST-CVPR15 [7]), enabling robust performance assessment.

1.2 Thesis Structure

The remainder of this thesis is structured to provide a comprehensive examination of thermal human detection using neural networks. Section 2 presents a thorough review of object detection fundamentals, SSD architecture principles, and thermal image processing techniques, establishing the theoretical foundation for the experimental work. Section 3 details the systematic approach employed for model evaluation, including dataset preparation, experimental design, and evaluation metrics. Section 4 presents comprehensive performance analysis across all model configurations and preprocessing techniques. Section 5 interprets the findings within the context of practical deployment scenarios and industrial requirements. Finally, Section 6 synthesizes the key contributions and outlines directions for future research in thermal surveillance technologies.

2 Literature Review and Theoretical Background

The field of object detection has undergone significant evolution from traditional computer vision techniques to sophisticated deep learning architectures. Understanding this progression is essential for contextualizing the current work's contribution to thermal image analysis. This section examines the theoretical foundations of object detection, with particular emphasis on the Single Shot MultiBox Detector (SSD) architecture and its applicability to thermal imagery processing challenges.

2.1 Object Detection Fundamentals

Most object detection methods can be broadly categorized into two main approaches: traditional methods and deep learning-based methods. The former mainly rely on handcrafted features and sliding window techniques [8], while newer approaches in this field leverage deep Convolutional Neural Networks (CNNs) or Vision Transformer (ViT) architectures to automatically learn features from data [9], [10].

2.1.1 Traditional Object Detection Methods

Simple approaches to object detection entail applying manually constructed feature detector kernels in a sliding window fashion to images.

One example of this is the **Viola-Jones-Algorithm** [8]:

1. The algorithm first computes the integral image of the input—a representation where each pixel stores the cumulative sum of intensities from the top-left corner to its position. This allows constant-time calculation of the sum of pixel values within any rectangular region, using only four array references (the corners of the rectangle).

2. It then applies Haar-like features - simple rectangular patterns (e.g., edge, line, or center-surround detectors) - to rapidly identify potential regions of interest.

Each feature's value is derived by subtracting the sum of pixels in one rectangle from the sum in an adjacent rectangle, leveraging the integral image for efficiency.

3. A strong classifier is constructed by training a series of weak classifiers (typically decision stumps) using Adaptive Boosting (AdaBoost). These weak classifiers focus on individual Haar-like features, while the cascaded structure enables early rejection of non-object windows, significantly reducing computation.

4. Finally, the algorithm scans the image using a sliding window, classifying each subwindow with the cascaded classifier. Windows that pass all stages of the cascade are marked as containing the target object.

Other approaches employ Histogram of Oriented Gradients (HOG) descriptors. The HOGs are attained by dividing the image into a grid of cells, contrast-normalizing them and then computing the vertical as well as horizontal gradients of their pixels [11]. The gradients for each cell are accumulated in a one-dimensional histogram which serves as that cell's feature vector [11]. After labeling the cells in the training data, a Support Vector Machine (SVM) can be trained to find an optimal hyperplane separating the feature vectors corresponding to the object that should be detected from those that do not contain the object [12].

2.1.2 Deep Learning-Based Object Detection

However, those methods are either highly dependent on engineering the correct priors, such as the Haar-like features, or limited to binary classification scenarios, as is the case for HOG-based SVMs [12]. Thus, newer Object Detection methods employ more complex deep-learning architectures that require less manual feature engineering. The best-performing models nowadays are ViTs using Attention mechanisms [9] to learn relationships between patterns in different parts of images. However, they will not be further examined in this thesis, due to computational constraints that make them unfeasible for the edge-deployable solution sought in this work [9].

Relevant for this examination are their predecessors, CNNs. The main mechanism they use to extract information from images are convolutional layers. Those convolutional layers get passed an image in the form of a tensor and perform matrix multiplication on that input tensor and a kernel tensor in a sliding window fashion to compute subsequent feature maps. Those will be passed on as input to the next layer. [13]

At their core, these convolutional layers do not work inherently different from Fully Connected (FC) layers that compute several weighted sums across all components of the input tensor. More specifically, fully connected layers can be described as convolutional layers whose kernel dimensions are identical to those of the input tensor.

Resorting to smaller kernels, however, serves as a prior making use of the heuristic that in most cases, the features composing an object in an image lie closely together. Thus, it is not necessary to process the entire image to detect an object that occupies only part of it. Convolutional neural nets hence save computational resources by focusing on smaller regions. In many cases it is advantageous to use those savings to increase network depth in order to make it possible for the network to learn more complex high-level features in subsequent layers.

Object detection, as opposed to image classification, consists of two main tasks; locating where an object is and classifying which class it belongs to. In the context of machine learning, that means regression must be used to approximate the location of an object and the concept of classification is applied to determine its class. CNNs solving these tasks can be categorized into two main categories:

Two-Stage Detectors split the detection objective into the two tasks mentioned before. The first stage proposes regions of interest and the second stage classifies which object they contain. In more detail, that means regressing bounding boxes and assessing the “objectness” of that region, for example by using logistic regression. If the confidence this region contains an object exceeds a given threshold, the second stage then classifies the object in that region. That requires a second

pass of the extracted region through a classifier network. This two-stage approach can be computationally expensive, especially when dealing with a large number of proposals. Examples of two-stage detectors include Region-based Convolutional Neural Networks (R-CNNs) [14], Fast R-CNNs [15], and Faster R-CNNs [16].

Single-Stage Detectors, on the other hand, perform both tasks simultaneously in a single pass through the network. That means passing the image through a network that both regresses bounding boxes and classifies objects in those boxes at the same time. Examples include You Only Look Once (YOLO) [17] and SSD [18]. This approach can be faster but may sacrifice some accuracy compared to two-stage detectors, as the feature extractor is not optimized for both tasks.

Given the computational constraints imposed by the requirement for edge-deployment, single-stage detectors were chosen. Past research has shown that SSD-variants with Inception-v2 and MobileNet-v1 backbones perform notably faster than their Faster R-CNN counterparts, namely 4 to 7 times as fast [2].

Furthermore, benchmarks of SSD and YOLO on the MS COCO dataset yielded similar results favoring SSD in terms of speed when deployed on edge devices, namely the Raspberry Pi 4 both with and without a dedicated Tensor Processing Unit (TPU) [10]. YOLO did deliver higher mean Average Precision (mAP) scores, but the difference was not significant enough to justify the trade-off in speed, in particular taking into account the benefit of speed for real-time applications when multiple images are captured each second and fast enough processing allows for multiple attempts at detection. Additionally, the SSD models tested consumed less energy than their YOLO counterparts [10], making them a more suitable choice that minimizes the need for human intervention to replace the battery of the edge device, which is a significant factor in the cost of deployment and maintenance of the system.

2.2 SGD as Optimizer in Deep Learning

Deep Learning Models are optimized by minimizing the loss function $L(\hat{y}, y)$, which is a measure of the difference between the predicted output $\hat{y}$ and the expected

output y, i.e. the ground truth [19]. The loss function is typically a differentiable function, which means that it can be used to compute the gradient of the loss with respect to the model parameters. The gradient is then used to update the model parameters in the direction that minimizes the loss function, hence the name gradient descent [20].

This happens in reverse order of the forward pass, since gradients of the loss function relative to earlier layer's parameters are computed using the chain rule, which is why this process is also called backpropagation. More specifically, the algorithm performs the following steps:

1. It first computes the gradient of the loss function with regard to a layer's output.
2. Using the chain rule, it computes the gradient of the loss function with respect to the layer's parameters.
3. Based on the results, it updates the layer parameters in the opposite direction of the gradient
4. Now the algorithm moves on to the next (earlier) layer and repeats the process until it reaches the first layer of the network.

The challenges posed by this process are discussed in Section 2.2.1

In most cases, the training dataset is too large to compute the gradient with respect to all data at once, which is called Batch Gradient Descent (BGD). Instead, the optimization takes place in so-called mini-batches of training data of a fixed size, under the assumption that the gradient computed relative to a mini-batch of training data is a good approximation of the gradient that would be obtained if the calculation was performed across the entire training dataset. [20]

This differentiation of the loss function with regard to a mini-batch of training data is called Mini-Batch Gradient Descent (MBGD), but it is also commonly referred to as SGD in the literature and will be called that for simplicity.

As desribed before, SGD is a first-order optimization algorithm, which means that it only considers the first-order derivatives of the loss function with respect to the

model parameters. This is in contrast to second-order optimization algorithms, such as Newton's method, which consider the second-order derivatives of the loss function as well. However, first-order optimization algorithms are generally preferred in deep learning because they are computationally more efficient and can be easily implemented on hardware accelerators such as Graphics Processing Units (GPUs) and TPUs, which is why second-order optimization algorithms will not be further discussed in this work. [20]

2.2.1 The Vanishing Gradient Problem (VGP)

One limitation of SGD lies in the so-called Vanishing Gradient Problem (VGP), which occurs due to the calculation of the partial derivates by means of the chain rule. The chain rule for differentiation states that the derivative of a composite function is the product of the derivatives of its components, as shown in Equation 1.

$$(f(g(x)))' = f'(g(x)) \cdot g'(x) \tag{1}$$

In this equation, f and g are the functions applying the linear transformations corresponding to a convolutional layer, its successor layer and their activation functions, respectively. Since both convolutional kernels contain a large number of parameters, each individual parameter only has a small impact on the overall loss function. As a result, the product of the derivatives of the loss function with respect to the parameters of the two layers can become very small, leading to a phenomenon known as the VGP.

In the context of deep learning, this means that deeper models are particularly likely to suffer from this problem, as more layers amplify the core issue. One approach to mitigate this problem is to introduce the so-called residual layers allowing for an identity mapping of the input to the output of a layer, which is a key component of the ResNet architecture.

2.2.2 Residual Layers to Mitigate the VGP

Residual layers are a key component of the ResNet as well as other modern architectures, and are designed to mitigate the VGP problem. The idea behind residual layers is to introduce a shortcut connection that allows the input to the

layer to be added to the output of the layer. This shortcut connection allows the gradient to "flow directly through the layer", which helps to prevent the gradient from vanishing as it is backpropagated through the network. Equation 2 shows the residual layer, where F is the residual function, x is the input to the layer, and $\hat{y}$ is the output of the layer. The residual function F is typically a stack of convolutional layers, BN layers, and Rectified Linear Unit (ReLU) activation functions. [21]

$$\hat{y} = F(x) + x \tag{2}$$

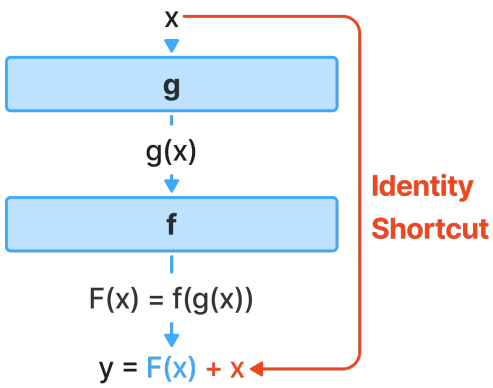

Figure 1 — Residual layer composed of layers with functions f and g with a shortcut connection

If the function F is a composite function $F(x) = f(g(x))$, then Equation 2 can be transformed as seen in Equation 3 to attain the derivative of $\hat{y}$ with respect to x, which given by the chain rule from Equation 1.

$$\begin{aligned} \hat{y}'(x) &= F'(x) + x', \\ &= f'(g(x)) \cdot g'(x) + 1 \end{aligned} \tag{3}$$

Here, the derivative of x with respect to itself is 1. In reality, x would be the result of a previous layer's function $h(x_0)$, which implies for the derivative of $\hat{y}$ with respect to x_0 the relationship introduced in Equation 4.

$$\begin{aligned}\hat{y}(x)' &= \hat{y}(h(x_0))', \\ &= \hat{y}'(h(x_0)) \cdot h'(x_0), \\ &= (F'(h(x_0)) + 1) \cdot h'(x_0), \\ &= f'(g(h(x_0))) \cdot g'(h(x_0)) \cdot h'(x_0) + h'(x_0)\end{aligned} \tag{4}$$

It becomes apparent that introducing residual layers and identity shortcuts helps to mitigate the VGP by preserving the gradient with respect to previous layers' outputs without multiplying them with many small derivatives of subsequent layers. $F(x)$ can be an arbitrarily deep function and the gradient with respect to x_0 will still be $h'(x_0)$ plus a term that is multiplied by the derivatives of the residual function $F(x)$. This is in contrast to the gradient in a standard CNN where repeated application of the chain rule is likely to cause the gradient to vanish.

When the input and output dimensions of a residual layer do not match, a linear projection is used to match the dimensions of the input and output before adding the two together. This linear projection is typically implemented using a 1x1 convolutional layer [21]. Even though this practice introduces additional parameters that need to be learned and makes the identity shortcut actually not an identity function anymore, it is still beneficial to the training process, as the residual function can be used to significantly increase the network's depth while preserving the gradient flow.

2.3 Computational Overhead Reduction Techniques

Training deep neural networks for object detection tasks presents significant computational challenges, particularly when evaluating multiple model variants across diverse datasets. Modern deep learning frameworks and hardware architectures provide several optimization techniques that can substantially reduce training time and memory requirements while maintaining model performance. This section examines the key computational optimization strategies employed in this work, focusing on their theoretical foundations and practical applications to thermal image processing workflows.

2.3.1 Mixed Precision Training (MPT)

Mixed Precision Training (MPT) is a computational optimization technique that addresses the memory and computational constraints encountered during deep neural network training. This approach leverages the reduced precision arithmetic capabilities of modern hardware accelerators while maintaining the numerical stability required for effective model convergence [22].

The fundamental principle behind MPT involves utilizing lower-precision floating-point representations (16-bit floating point (FP16)) for the majority of training operations while preserving higher-precision (32-bit floating point (FP32)) calculations for operations that require numerical stability. This selective precision approach enables significant reductions in memory usage (typically 50% or more) and computational time while maintaining training effectiveness equivalent to full-precision training. [22]

During mixed-precision training, the forward pass and gradient computation operations are performed using FP16 precision to maximize memory efficiency and computational throughput. However, the model weights are maintained in FP32 precision, and gradient updates are applied to these full-precision parameters. This ensures that small gradient values are not lost due to the limited dynamic range of FP16 representation (approximately 6.1×10^{-5} to 6.55×10^{4}).

To address potential gradient underflow issues that can arise from the limited dynamic range of FP16, gradient scaling techniques are employed. The loss function is multiplied by a scaling factor before backpropagation, effectively shifting small gradient values into the representable range of FP16. After gradient computation, the gradients are unscaled before applying updates to the FP32 model weights [22].

The implementation of MPT is particularly beneficial for large-scale training scenarios where memory constraints and computational efficiency are critical factors.

2.3.2 Just-In-Time (JIT) Compilation and Graph Optimization

Just-In-Time (JIT) compilation represents a runtime optimization strategy that transforms computational graphs and Python code into optimized machine code during execution, rather than relying solely on interpreted execution. This approach addresses the inherent performance overhead of dynamic programming languages like Python, which traditionally suffer from interpretation bottlenecks during intensive computational workloads. [23], [24].

The fundamental principle of JIT compilation involves analyzing the computational patterns during initial execution runs and generating optimized compiled code for subsequent iterations. This process eliminates Python overhead, optimizes memory access patterns, and enables hardware-specific optimizations that can significantly accelerate training throughput. Modern deep learning frameworks implement JIT compilation through graph compilation strategies that analyze the computational graph structure and apply fusion operations, kernel optimization, and memory layout improvements. [23]

In PyTorch, JIT compilation is implemented through TorchScript and the `torch.compile` function, which can provide substantial speedups for models with repetitive computational patterns. The compilation process involves tracing or scripting the model's forward pass to create an optimized representation of the forwards graph, which can optionally be used to also create a backwards graph for gradient computation. [23]

Specifically using PyTorch's `aot_eager` backend for compilation means additionally creating a backwards graph for gradient computation ahead-of-time as soon as the forwards graph has been generated using the AOTAutograd graph analysis and generation engine. [23]

The benefits of JIT compilation become particularly pronounced in iterative training scenarios where the same computational patterns are repeated thousands of times across training epochs. Object detection models like SSD benefit significantly from this optimization due to their complex multi-scale feature extraction

and anchor box processing operations, which involve repetitive convolutions and tensor manipulations that can be efficiently optimized through compilation. Geometric mean speedups of $2.27\times$ during inference and $1.41\times$ for training have been reported across 180+ models of in the paper introducing JIT compilation and AOTAutograd for PyTorch [23].

2.4 Single Shot MultiBox Detector (SSD) Architecture

The SSD architecture is a single-stage detector that uses a base network to extract features from the input image and then applies additional convolutional layers as well as FC layers to predict bounding boxes and class scores for each feature map. The following sections provide a more detailed explanation of the SSD architecture and its components.

2.4.1 Backbone Networks for Feature Extraction

Explores the role of backbone networks (VGG, ResNet) in feature extraction and their impact on SSD performance.

The SSD architecture uses a base network to extract features from the input image. The base network is typically a pre-trained CNN, such as VGG or ResNet, which has been trained on a large dataset like ImageNet. This assumption that features learned for other tasks can be reused for object detection is known as Transfer Learning (TL) and has proven to often be very effective in practice. [4], [25]

The Visual Geometry Group (VGG) network is a deep CNN that consists of 16 or 19 layers, depending on the variant. It is known for its simplicity and effectiveness in image classification tasks. In its vanilla configuration, it takes 224x224 RGB images as input and outputs a 1000-dimensional vector of class probabilities. It only uses 3x3 convolutional layers and 2x2 max-pooling layers for feature extraction and the ReLU activation function for non-linearity. Eventually, it employs three FC layers for classification. The soft-max activation function is used in the final layer to predict the class probabilities. Overall, the number of trainable parameters for the VGG−16 network is 138 million [26]. VGG is the default backbone network employed in the original SSD paper [18].

While the Residual Network (ResNet) architecture is mostly similar to the VGG architecture in that it is a CNN, it uses a couple of more advanced techniques for feature extraction.

Firstly, it utilizes residual blocks to address the VGP while significantly increasing network depth through employment of considerably more convolutional layers [21]. Secondly, it uses BN to stabilize and accelerate training. BN layers perform normalization of their input along the batch dimension, although batch in this case refers to the channels of the input tensor and not to the batch size. BN layers only have two trainable parameters, the scale and shift parameters, which are learned during training and are used to scale and shift the normalized input. The stabilizing effect stems from the fact that BN reduces the covariate shift between layers during update steps. The covariate shift describes the change in the distribution of the input to a layer due to the change in the parameters of the previous layer, which significantly slows down training progress when using SGD. [27]

Secondly, a bottleneck architecture ensures that deeper networks exploiting the solution offered by residual layers do not become too large in overall parameter count. This architecture uses an initial convolutional layer to reduce the number of channels in the input tensor, followed by additional convolutional layers that use the reduced number of channels to perform the actual feature extraction. The output of the last convolutional layer of each bottleneck is in most cases upsampled again to a higher number of channels. [21]

In order to find out whether ResNets offer significant advantages over traditional CNNs models, the ResNet−152 model, which is one of the deepest ResNet configurations consisting of 152 convolutional layers, is used for comparison to the VGG−16 model.

2.4.2 Feature Maps and Anchor Boxes

As mentioned, SSD is a single-stage detector, which means that it does not use a region proposal network to generate candidate regions for object detection. Instead, it utilizes a set of default boxes, also known as anchor boxes, to predict

the location and class of objects in the image. The anchor boxes are generated at multiple scales and aspect ratios to cover a wide range of object sizes and shapes.

To realize this, the network will always predict a pre-defined number of bounding boxes, regardless of the number of objects in the image. Since the receptive field of the convolutional layers grow with each layer, the network will predict bounding boxes at multiple scales by using feature maps from different layers of the network.

This means that the network has several predictor heads, each of which will get a feature map from the base network as input and will predict a pre-defined number of bounding boxes for each location in the feature map. The bounding box prediction is thus defined as a regression task. Let $\vec{d}_i$ be the i-th default anchor box, encoded by its center coordinates $d^{\text{cx}}, d^{\text{cy}}$, its width d^w, and its height d^h. The network will predict an offset vector $\vec{l}_i$ relative to each default anchor box $\vec{d}_i$ such that the final predicted bounding box $\vec{b}_i$ can be calculated using Equation 5 [18].

$$\vec{b_i} = \begin{pmatrix} b_i^{\text{cx}} \\ b_i^{\text{cy}} \\ b_i^w \\ b_i^h \end{pmatrix} = \begin{pmatrix} d_i^{\text{cx}} + l_i^{\text{cx}} \cdot d_i^w \\ d_i^{\text{cy}} + l_i^{\text{cy}} \cdot d_i^h \\ d_i^w \cdot e^{l_i^w} \\ d_i^h \cdot e^{l_i^h} \end{pmatrix} \tag{5}$$

The network also predicts a vector of class probabilities for each bounding box, which indicates the confidence of the network that the bounding box contains an object of a particular class - each vector component is assigned to a class, and the sum of all components is 1. To ensure that all confidence scores lie between 0 and 1 and add up to 1, the network applies the softmax activation function from Equation 6 to the output vector $\vec{p} \in \mathbb{R}^C$ with components $p_1, ..., p_C$ [18].

$$\hat{p} = \text{softmax}(\vec{x}) = \begin{pmatrix} e^{p_1} \\ ... \\ e^{p_C} \end{pmatrix} \cdot \frac{1}{\sum_{i=1}^{C} e^{p_i}} \tag{6}$$

[28]

2.4.3 MultiBox Loss Function

First of all, in order to compute a loss, the network needs to know which anchor boxes contain an object and which do not. To determine which ground-truth object is assigned to which anchor box, the network uses the Intersection over Union (IoU) metric [18], also known as the Jaccard index [29]. That is a measure for the overlap between two bounding boxes B_1 and B_2, defined as the area of their intersection divided by the area of their union, as shown in Equation 7 [29].

$$\mathrm{IoU}(B_1, B_2) = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|} \tag{7}$$

This IoU metric is then used to match the regressed boxes to the ground-truth boxes. It is calculated for each pair of anchor boxes and ground-truth boxes. Based on the IoU scores, the network determines which anchor boxes are positive and which are negative. Negative anchor boxes are those that do not have an IoU above a certain threshold with any ground-truth box. Positive boxes, on the other hand, satisfy the condition that they must have a certain minimum overlap with any ground-truth box. If they overlap sufficiently with multiple ground-truth boxes, the one with the highest IoU is assigned to that object. [18], [19]

The result is an unambiguous mapping from positive anchor boxes to ground-truth boxes.

Since the network is supposed to reliably detect objects in images, it needs a large quantity of anchor boxes to cover all possible object sizes and aspect ratios. For reference, the paper introducing the SSD uses an overall 8732 anchor boxes [18]. However, during training this leads to a large number of negative anchor boxes that do not contain any ground-truth objects.

For this reason, it is not feasible to use all negative anchor boxes for training, as this would heavily skew the loss function towards the negative class. Instead, the network uses a Hard-Negative Mining (HNM) technique to select a fixed number of negative anchor boxes for training.

Hard-Negative Mining (HNM) means only selecting the most difficult negative examples for training. This is done by sorting the negative anchor boxes by their confidence loss and selecting the top k anchor boxes with the highest confidence loss. In practice, this is done by defining a ratio between negative and positive anchor boxes and selecting the according number of negative anchor boxes. This ensures that the network is trained on the training examples it finds most difficult to classify correctly as negative. All other negative anchor boxes are ignored, that means they do not contribute to the loss function. [18]

Additionally, models of the SSD family employ a technique called Non-Maximum Suppression (NMS) to filter out duplicate detections and improve detection accuracy. It his highly likely that an input image of dimensions 300x300 will have multiple of the 8732 anchor boxes overlapping with each other and picking up on the same ground-truth object.

Non-Maximum Suppression (NMS) is applied after assigning each anchor box to a ground-truth object (or counting it to the negative anchor boxes) to filter out duplicate detections. Algorithmically, NMS is carried out as follows [18], [19]:

1. The algorithm begins by grouping all anchor boxes that have been assigned to the same ground-truth object.
2. For each ground-truth object, the assigned anchor boxes are sorted by their confidence score in descending order.
3. The anchor box with the highest confidence score is selected as one of the final detections that will be accounted for in the loss function.
4. All anchor boxes that exhibit an IoU greater than a threshold (typically 0.5) with the selected anchor box are removed from consideration.
5. This process is repeated for the remaining anchor boxes until no more anchor boxes are left.

Once the NMS algorithm has selected the anchor boxes relevant to the loss function, the network can compute the actual loss. That has to account for two different types of errors, localization and classification error [18], [19].

The former is the difference between the predicted bounding box and the ground-truth bounding box [19]. For obvious reasons, this loss is only computed for the positive anchor boxes selected by NMS, as it does not make sense to teach the network to regress negative bounding boxes around the background [18]. This localization error uses the Smooth L1 loss function from Equation 8, which is often used in object detection tasks as it combines the characteristics of other alternatives like the L1 and L2 loss to allow for a robust training process [30].

$$\text{SmoothL1}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{8}$$

Plugging in the ground-truth bounding box vector $\vec{g}_j$ for the bounding box vector derived through Equation 5 allows to formulate the conversion from $\vec{g}_j$ to the offset vector $\hat{g}_j$ the network is expected to predict respective to any corresponding default anchor box $\vec{d}_i$:

$$\vec{g}_i = \begin{pmatrix} g_j^{\text{cx}} \\ g_j^{\text{cy}} \\ g_j^{\text{w}} \\ g_j^{\text{h}} \end{pmatrix} = \begin{pmatrix} d_i^{\text{cx}} + \hat{g}_j^{\text{cx}} \cdot d_i^w \\ d_i^{\text{cy}} + \hat{g}_j^{\text{cy}} \cdot d_i^h \\ d_i^w \cdot e^{\hat{g}_j^w} \\ d_i^h \cdot e^{\hat{g}_j^h} \end{pmatrix} \longrightarrow \hat{g}_j = \begin{pmatrix} g_j^{\text{cx}} \\ g_j^{\text{cy}} \\ \ln\left(\frac{g_j^{\text{w}}}{d_i^{\text{w}}}\right) \\ \ln\left(\frac{g_j^{\text{h}}}{d_i^{\text{h}}}\right) \end{pmatrix} \tag{9}$$

This representation can be used to determine the SmotthL1 localization loss. Let P be the set of positive anchor indices selected by NMS, G be the set of ground-truth box indices and a_{ij} the indicator that the i-th ground-truth box is assigned to the j-th anchor box. Then the localization loss L_{loc} is defined as in Equation 10 [18].

$$L_{\text{loc}}(a, l, g, \text{P}, \text{G}) = \sum_{i \in \text{P}} \left(\sum_{j \in \text{G}} \left(\sum_{m \in \{\text{cx},\ \text{cy}, w, h\}} a_{\text{ij}}\ \text{SmoothL1}\left(\hat{g}_j^m, l_i^m\right) \right) \right) \tag{10}$$

The second component of the overall loss function is the classification error, which is the difference between the predicted class and the ground-truth class. This is computed for the hard negative as well as positive anchor boxes. The classification

loss for any given anchor box with confidence score vector $\vec{p} \in (0,1)^C$, one-hot ground-truth class vector $\vec{y} \in \{0,1\}^C$ and C classes to detect is computed using the cross-entropy loss from Equation 11. The cross-entropy loss is specifically designed for multi-class classification problems. [30]

$$\text{CrossEntropy}(\vec{y}, \vec{p}) = -\sum_{c=1}^{C} y_c \cdot \log(p_c) \tag{11}$$

Where C is the number of classes to detect, $y_{i,c} \in \{0,1\}$ is the binary ground-truth label whether sample i belongs to class c, and $p_{i,c} \in (0,1)$ is the predicted probability that c is the correct class for i.

After adjusting for the number of positive anchor boxes - such that training images containing more positive anchor boxes do not contriubte overproportionally to the loss - Equation 12 describes the classification loss, where N is the set of negative anchor boxes selected by HNM.

$$L_{\text{class}}(\mathrm{P}, \mathrm{N}, y, p) = \sum_{i \in \mathrm{P}} \text{CrossEntropy}(\vec{y}_i, \vec{p}_i) + \sum_{i \in \mathrm{N}} \text{CrossEntropy}(\vec{y}_i, \vec{p}_i) \tag{12}$$

[18]

The final loss is a weighted sum of the localization and confidence loss with a scaling factor α, adjusted for the number of positive anchor boxes after NMS.

$$L(a, y, p, l, g, \mathrm{P}, \mathrm{N}, \mathrm{G}) = \frac{1}{|\mathrm{P}|} \cdot (L_{\text{Loc}}(a, l, g, \mathrm{P}, \mathrm{G}) + \alpha \cdot L_{\text{class}}(\mathrm{P}, \mathrm{N}, y, p)) \tag{13}$$

[18]

Propagating that loss from Equation 13 backwards through the network, the weights of the network are updated using SGD with a learning rate η, which means that each parameter's specific gradient it multiplied by η before performing the parameter update step. The basic idea is that since the result of Equation 13 is a measure of the network's performance, adjusting the network parameters such that the loss function approaches a local or even global minimum should yield

a better performing model also by human standards - that encapsulates better fitting bounding boxes as well as more accurate classification of the objects in the images.

2.5 Thermal Image Processing

Thermal imaging presents unique challenges and opportunities for computer vision applications compared to conventional RGB imagery. Understanding these characteristics and developing appropriate preprocessing techniques is crucial for optimizing object detection performance in infrared surveillance systems.

2.5.1 Characteristics of Thermal Images

Thermal images fundamentally differ from visible-light imagery in several key aspects that directly impact neural network performance. Unlike RGB images that capture reflected light, thermal cameras detect electromagnetic radiation in the infrared spectrum, creating images based on the heat signatures emitted by objects [1].

The most distinctive characteristic of thermal imagery is its independence from ambient lighting conditions. Since thermal cameras detect heat radiation rather than reflected light, they provide consistent imaging capabilities in complete darkness, fog, smoke, and other challenging environmental conditions where traditional RGB systems fail [2]. This makes thermal imaging particularly valuable for surveillance applications.

However, thermal images also present unique challenges for object detection models originally designed for visible-light imagery. The spectral characteristics result in different texture patterns, contrast relationships, and edge definitions compared to RGB images. Additionally, thermal sensors often produce images with lower spatial resolution and different noise characteristics, requiring specialized preprocessing approaches to optimize detection performance. [31]

2.5.2 Preprocessing Techniques for Thermal Detection

To address the unique characteristics of thermal imagery and improve object detection accuracy, this study implements three primary preprocessing techniques: normalization, polarity inversion, and edge enhancement. These techniques are designed to adapt RGB-trained models to thermal domain characteristics while preserving essential spatial and thermal information.

Normalization serves as the fundamental preprocessing step, ensuring consistent input scaling across all thermal images and enabling proper transfer of learned features from the RGB domain for the pretrained model backbones. Following standard practice, input images are normalized using the ImageNet dataset statistics with channel-wise means of [0.485, 0.456, 0.406] and standard deviations of [0.229, 0.224, 0.225] [32].

The normalization process transforms pixel intensities according to Equation 14, where I_{norm} represents the normalized image, I_{raw} is the input thermal image, μ is the mean, and σ is the standard deviation for each channel.

$$I_{\text{norm}} = \frac{I_{\text{raw}} - \mu}{\sigma} \tag{14}$$

Polarity inversion addresses the fundamental difference in thermal image representation compared to natural images. In mopst thermal imaging scenarios, humans appear as bright objects against darker backgrounds due to their higher body temperature. However, thermal scene characteristics such as high ambient temperatures can result in inverted polarity where humans appear dark against bright backgrounds [4].

Since thermal images might exhibit either polarity, inverting one of the channels ensures that at least one channel maintains consistent human-background contrast relationships across all images. This approach is inspired by the work of [4], which demonstrated the effectiveness of various preprocessing techniques in improving transfer learning performance across diverse thermal datasets. Ad-

dtionally, [4] also states that specifically polarity-inverted thermal images have a close resemblance to grayscale-converted RGB images.

Since most CNN architectures and pretrained weights are optimized for detecting darker objects (edges, shadows) against lighter backgrounds in RGB imagery, thermal images with inverted polarity may not align with these learned features [4]. Polarity inversion preprocessing ensures consistent object-background contrast relationships by applying a simple pixel-wise transformation according to Equation 15.

$$I_{\text{inverted}} = 1.0 - I_{\text{original}} \tag{15}$$

Here, the constant 1.0 is chosen simply to ensure quick computation of the inverted channel and because it lies close to the maximum pixel intensity after normalization.

Edge enhancement preprocessing aims to strengthen the boundary definition between objects and backgrounds in thermal images, which often exhibit softer transitions compared to RGB imagery due to heat diffusion effects or wind-induced heat dissipation [4]. The implementation combines Gaussian blur preprocessing to reduce noise and smooth the image, followed by Sobel edge detection to create enhanced edge representations.

The specific kernel used for blurring is defined in Equation 16 and is chosen such that it is normalized to sum to 1.0 and preserves spatial characteristics while decreasing intra-channel variance.

$$K_{\text{Gaussian}} = \begin{pmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{pmatrix} \tag{16}$$

Subsequently, Sobel operators are applied to detect horizontal and vertical edges. The Sobel kernels S_x and S_y for horizontal and vertical edge detection are defined as [33]:

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{17}$$

The final edge magnitude is computed by combining both directional gradients according to Equation 18, providing enhanced boundary information that emphasizes object contours in thermal imagery.

$$E = \sqrt{(I * S_x)^2 + (I * S_y)^2} \tag{18}$$

[33]

Where $*$ denotes the convolution operation. This edge-enhanced representation provides additional geometric information that complements the thermal intensity data, potentially improving the model's ability to localize and classify human subjects in infrared images.

2.5.3 Preprocessing Combination Strategies

The preprocessing techniques can be applied individually or in combination to optimize detection performance for specific thermal imaging scenarios. The normalization is always applied in order to ensure consistent transfer learning applicability. Thus, the experimental design evaluates four distinct preprocessing configurations:

1. **Normalization only**: Baseline preprocessing maintaining original thermal characteristics and maximizing transfer learning performance
2. **Normalization + Inversion**: Addressing polarity variations in thermal scenes
3. **Normalization + Edge Enhancement**: Emphasizing geometric features for improved localization
4. **Normalization + Inversion + Edge Enhancement**: Combined approach leveraging both polarity correction and geometric enhancement

3 Methodology

This study employs a systematic experimental approach to evaluate the effectiveness of SSD-based neural networks for human detection in thermal imagery. The methodology encompasses dataset selection and preparation, implementation of multiple model variants with different backbone architectures, application of thermal-specific preprocessing techniques, and comprehensive evaluation metrics. The experimental design ensures reproducible results while addressing the unique challenges posed by infrared image characteristics.

Key areas to develop:

- Dataset description: FLIR ADAS v2, AAU-PD-T, OSU-T, M3FD, KAIST-CVPR15
- Training setup: Pretrained vs. scratch initialization strategies
- Preprocessing techniques: Image inversion and edge enhancement
- Data augmentation and split strategies (train/validation/test)
- Evaluation metrics: mAP, precision, recall, inference speed
- Hardware setup and computational requirements
- Statistical significance testing approach

3.1 Dataset Description

The experimental evaluation employs five complementary thermal datasets that collectively provide comprehensive coverage of diverse infrared imaging scenarios and human detection challenges. The strategic selection of these datasets addresses the fundamental objective of creating a robust, generalizable detection system capable of operating across varying environmental conditions, camera configurations, and thermal imaging scenarios that are representative of real-world surveillance applications. Table 1 provides a general overview of the different datasets.

Dataset	Resolution	Environment	Camera Setup	Annotated Classes
FLIR ADAS v2 [3]	640×512	Automotive roads, urban/highway, day/night, adverse weather	Vehicle-mounted, forward-facing	Person Vehicles Other objects
AAU-PD-T [4]	640×480	Controlled outdoor sports fields, winter conditions	Elevated stationary (9m height)	Person
OSU-T [5]	360×240	University campus, natural outdoor pedestrian areas	Elevated stationary (building rooftop, 3 stories)	Person
M3FD Detection [6]	1024×768	Urban driving, challenging visibility conditions	Vehicle-mounted dual-modal	Person Vehicles Lamp
KAIST-CVPR15 [7]	640×512	Urban pedestrian scenarios, day/night cycles	Vehicle roof-mounted, ego-centric	Person People Cyclist

Table 1 — Comprehensive thermal dataset specifications and characteristics

Since this thesis is on the detection of humans, the annotations are reduced to contain only the person label. All other labels are either discarded or, in the case of KAIST-CVPR15, converted. Note that “Vehicles” and “Other Objects” in Table 1 serve as placeholders for the actual classes that are not listed in detail for the aforementioned reasons.

FLIR ADAS v2 [3] provides automotive-focused thermal imagery with high thermal contrast between human subjects and vehicle/road backgrounds, captured at various distances typical of roadside surveillance applications. The dataset’s vehicle-mounted perspective and diverse geographic coverage (Santa Barbara, San Francisco, London, Paris, Spanish cities) ensures exposure to varying ambient temperatures and thermal background conditions that challenge model generalization. The default test split of the dataset is preserved for the sake of scientific comparability; however, since not all images contain objects after filtering out all

classes except for the person label, empty images from the predefined train split are instead moved to the validation split.

AAU-PD-T [4] is a dataset of images captured from cameras mounted at a height of 9m and directed at a soccer field. Thus, the objects are rather small and a wide variety of environmental conditions is covered by the data, mimicking the characteristics of typical surveillance scenarios. The default test split is preserved. Since no predefined validation split is provided, all empty images from the default training split are discarded entirely.

OSU-T [5] serves only as a benchmark for testing the models due to its relatively small size of just 284 images. This also ensures testing with data that is not only completely unseen by the models before but also entirely unrelated to the images they have been trained on.

The **M3FD** [6] dataset is originally intended to be used as a benchmark as well. Since its value for benchmarking lies in the multi-modality, however, it is instead split into train, validation and test images for this project, with a 60%/20%/20% ratio. That way the varying ambient conditions of the thermal imagery in this dataset also contribute to the overall diversity of the training data. The images in the training split are selected such that they contain at least one object.

KAIST-CVPR15 [7] contributes the second-largest volume of thermal pedestrian data captured across different times of day, providing extensive variety in ambient thermal conditions that affect human-background contrast relationships. The urban traffic environment introduces complex thermal scenes with multiple heat sources (vehicles, pavement, buildings) that require sophisticated discrimination capabilities. It contains three classes: person, people and cyclist. Since the cyclist bounding boxes only encapsulate the humans and not the rest of the bicycles, all cyclist labels are converted to person labels. To avoid “confusing” the model with different annotation standard for crowds across datasets, all images containing people objects are dropped entirely. By default, the dataset only has a train and test split. Due to the changes applied to the labels, model performance on the

test split cannot be used for comparison to the results that have been reported for other models in the literature. Thus, the test split can be further modified by selecting half of the test images for validation.

Table 2 provides an overview of the resulting splits across the differnt datasets.

Dataset	Images per split			Objects per split		
	Train	Val	Test	Train	Val	Test
FLIR ADAS v2	8205	3681	3749	50478	4470	12323
AAU-PD-T	706	0	1000	5572	0	2237
OSU-T	0	0	284	0	0	984
M3FD Detection	2520	840	840	18231	5803	5739
KAIST-CVPR15	8815	1909	1787	20024	1661	2186
Overall	20246	6430	7660	94305	11934	23469
Relative	59,0%	18,7%	22,3%	72,7%	9,2%	18,1%
Average number of objects per image	4,66	1,86	3,06			

Table 2 — Dataset split overview. Val stands for Validation

This multi-dataset approach creates a comprehensive compound training dataset that addresses the fundamental challenge of thermal domain adaptation for human detection models originally designed for RGB imagery. The combination ensures robust evaluation across varying thermal polarities (human-hot and human-cool scenarios), environmental temperature conditions (day/night thermal crossover points), subject distances (far-field sports surveillance and near-field automotive detection), and thermal contrast scenarios (high-contrast winter conditions and challenging summer thermal equilibrium) [4], [6], [7]. Additionally, the ratio between the different splits closely resembles the standard 60%/20%/20% distribution that is often utilized for computer vision applications.

TODO: split source

For training purposes, FLIR ADAS v2, AAU-PD-T, KAIST-CVPR15, and M3FD Detection contribute to the training set, while FLIR ADAS v2 and AAU-PD-T provide

validation data. OSU-T serves exclusively as an independent test set, ensuring unbiased evaluation on data completely unseen during training. This split strategy maintains rigorous experimental integrity while maximizing the utilization of available annotated thermal imagery for model development.

3.2 Model Implementation

The base architecture, SSD300, is implemented with two distinct backbone networks: VGG16 and ResNet152. Each backbone is evaluated in two initialization scenarios: pretrained on ImageNet and trained from scratch only on thermal imagery.

Since the normalization applied to the input images is skewed by the preprocessing techniques mentioned above, the backbone networks are equipped with an additional BN layer that allow the network to learn an optimal normalization for specific characteristics of their respective preprocessing setup.

Any specifics about the specific implementation details that are not mentioned in this documentation can be found in the source code repository, mostly in `/src/model/models.py`.

3.2.1 Visual Geometry Group (VGG) Backbone Implementation

For the SSD network with VGG−16 backbone, the default anchor box configuration from the original implementation [18] is used. To reduce computational complexity, the FC layers are removed and replaced with an additional two convolutions. Since those convolutions encompass fewer parameters than the original FC layers, the pre-trained models use a subset of the FC parameters and organize them in a dilated convolutional kernel. The very last FC layer is entirely discarded, as it only generated the final class prediction when VGG is used as an ImageNet classifier.

The backbone base network is followed by a smaller auxiliary network of four sequential pairs, each consisting of a 1x1 convolutional kernel halving the number of channels and a 3x3 convolutional kernel doubling the channels again, but without padding, such that it reduces the spatial dimensions by 2 pixels in each direction.

Prediction heads are positioned before each max-pooling layer, after both of the final convolutional layers of the base network, and after all four convolution pairs of the auxiliary network. Each prediction head consists of two convolutional layers; one for bounding box regression and one for class prediction [18]. These six prediction heads amount to a total of 8732 anchors for detection attempts.

A detailed visualization of the SSD-VGG16 architecture is shown in Figure 2.

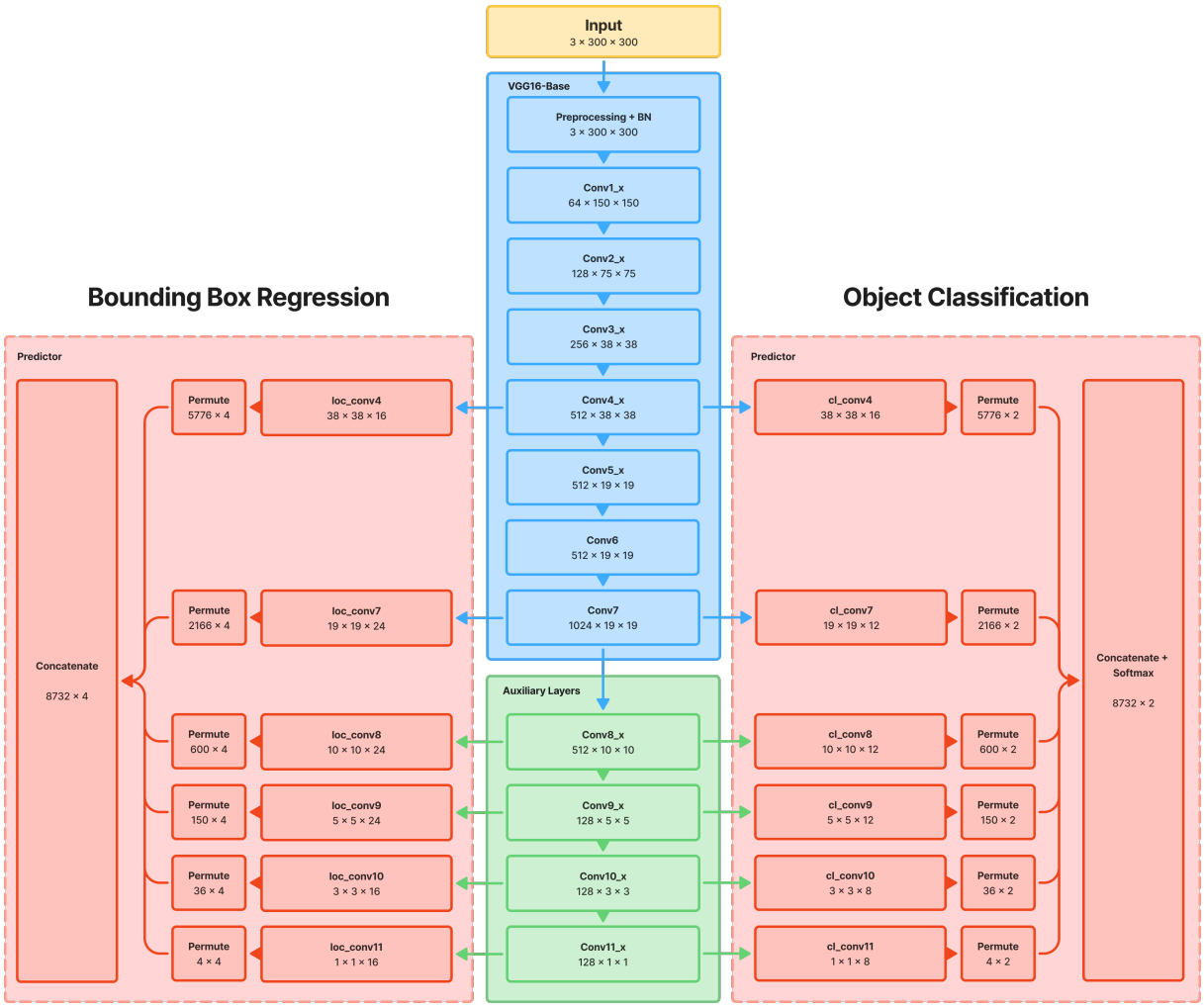

Figure 2 — The initial SSD-VGG16 architecture

,"ConvX_Y," names a series of convolutional layers preceded by a maxpooling layer. The individual layers are not visualized as they do not affect feature map dimensions.

Feature Map	Dimensions	Scale	Aspect Ratios	Priors	Total Priors
Conv4_3	38 x 38	0.1	1:1, 2:1, 1:2, extra prior	4	5776
Conv7	19 x 19	0.2	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	2166
Conv8_2	10 x 10	0.375	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	600
Conv9_2	5 x 5	0.55	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	150
Conv10_2	3 x 3	0.725	1:1, 2:1, 1:2, extra prior	4	36
Conv11_2	1 x 1	0.9	1:1, 2:1, 1:2, extra prior	4	4
Grand total	-	-	-	-	**8732**

Table 3 — SSD-VGG16 anchor boxes

In this table, “Dims” is short for Dimensions for formatting reasons, the scale and aspect ratios combined result in the default scaling ratios for each respective prior and the extra prior has a

3.2.2 Residual Network (ResNet) Backbone Implementation

The ResNet−152 architecture does not entirely replicate that from the original paper [21]. Instead, it uses that from the PyTorch implementation, which is known as ResNet v1.5 and has been shown to outperform the original architecture [34]. The final FC layer and average pooling layer are discarded, as they only serve the prediction in image classification, and are replaced by a single convolutional layer.

In the initial setup, prediction heads are placed on top of each of the major building blocks doubling the number of channels. Furthermore, one last auxiliary layer is added in order to attain an additional high-scale feature map. For the sake

of simplicity, the single auxiliary layer is implemented as part of the ResNet base network.

As described in Section 3.2.1, the SSD-ResNet152 is also later adapted to apply BN to the inputs to the prediction heads and to perform logistic regression using the sigmoid function by dropping the background class.

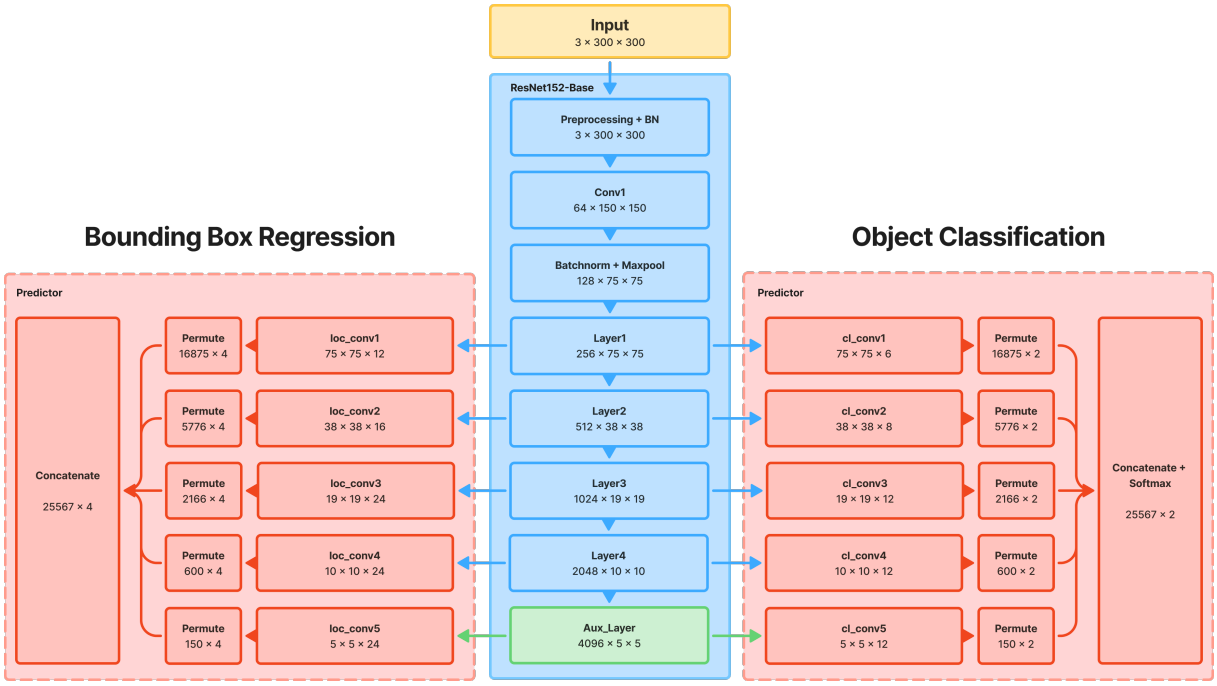

Figure 3 — The initial SSD-ResNet152 architecture

,"LayerX," in the architecture diagram names a series of convolutional bottleneck blocks - for a more detailed description of the individual layers and blocks, refer to [34] and Section 2.4.1. It is important to keep in mind that despite the SSD-ResNet152 diagram appearing less complex than that of SSD-VGG16, it is actually significantly deeper, as noted in the aforementioned Section 2.4.1 and also made apparent by compqaring Table 4 and Table 3.

Feature Map	Dimensions	Scale	Aspect Ratios	Priors	Total Priors
Layer1	75 x 75	0.05	1:1, 2:1, 1:2	3	16875
Layer2	38 x 38	0.1	1:1, 2:1, 1:2, extra prior	4	5776
Layer3	19 x 19	0.2	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	2166
Layer4	10 x 10	0.375	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	600
Aux_Layer	5 x 5	0.55	1:1, 2:1, 1:2, 3:1, 1:3, extra prior	6	150
Grand total	-	-	-	-	**25567**

Table 4 — SSD-ResNet152 anchor boxes

3.3 Later Developments

3.3.1 Sigmoid Activation Function

Initially, the activation function used for the prediction heads is the softmax function. However, since the network needs to predict only two classes, person and background, the softmax function is later replaced with the sigmoid function from Equation 19 that outputs a single value between 0 and 1, representing the probability of the person class. This change is done in response to training results and is supposed to improve the performance of the network as it does not require the model to learn the more complex relationship between its outputs in the softmax function.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{19}$$

[35]

Additionally, later developed variants of the model utilize BN layers in the prediction heads before passing on the input they get to the convolutional layers. The reasons for this will be explained based on the training results in Section 4.1.

3.3.2 SSD-ResNet152 anchor box configuration

Additionally, the SSD-VGG16 variations exhibit better training behavior than the SSD-ResNet152 models (for more details, refer to Section 4.1) in their initial configuration. With a major architectural difference lying in the additional priors that SSD-ResNet152 applies on the feature maps of layer 1 with dimensions of 75x75, those priors are later removed, such that computational overhead is reduced and the learning process is guided towards higher-scale features of the subsequent layers.

Simultaneously, new priors with the aspect ratios 3:1 and 1:3 are added to the prediction head on top of layer 2.

3.3.3 BN layers in prediction heads

Lastly, to improve convergence behavior and ensure proper feature scaling across all layers, the prediction heads of both the SSD-ResNet152 and SSD-VGG16 models get BN layers that normalize inputs before further processing to attain predictions.

3.4 Training Procedure

The training methodology employs a systematic approach to optimize SSD models for thermal human detection across diverse environmental conditions. All models are trained using the combined dataset described in Section 3.1, utilizing SGD optimization with carefully tuned hyperparameters to ensure convergence stability and optimal performance. The training process incorporates several advanced techniques to address computational constraints while maintaining model accuracy, including mixed-precision training for memory efficiency, model compilation for performance optimization, and adaptive learning rate scheduling for improved convergence behavior.

3.4.1 Core Training Configuration

The training procedure uses a consistent set of hyperparameters across all 16 model variants to ensure fair comparison between architectures and preprocessing techniques. The core training configuration is designed to balance convergence stability with computational efficiency:

```
batch_size = 64
learning_rate = 1e-4
epochs = 14
momentum = 0.9
weight_decay = 5e-4
optimizer = SGD  # with differential bias learning rates
```

Listing 1 — Core Training Configuration

The batch size of 64 provides adequate gradient estimation while remaining within memory constraints of the available hardware. The relatively conservative learning rate of 10^{-4} ensures stable convergence across different model architectures and initialization strategies, preventing divergence during the critical early training phases.

3.4.2 Learning Rate Scheduling and Optimization

The optimization strategy employs SGD with momentum, incorporating differential learning rates for bias parameters to improve convergence behavior. Bias parameters receive twice the base learning rate (2×10^{-4}), following established practices that recognize the different optimization dynamics of bias terms compared to weight matrices.

Learning rate scheduling follows a step decay strategy with reductions at epochs 8 and 12, where the learning rate is multiplied by 0.1. This schedule allows the model to make rapid initial progress during early training while enabling fine-tuning in later epochs. The specific scheduling points were chosen based on empirical observation of loss plateaus during preliminary experiments.

Weight decay regularization (5×10^{-4}) prevents overfitting by penalizing large parameter values, particularly important given the relatively limited thermal train-

ing data compared to standard RGB datasets. The momentum coefficient of 0.9 provides acceleration through consistent gradient directions while damping oscillations in parameter updates.

3.4.3 Mixed Precision Training

Due to computational constraints, the training procedure employs several optimization techniques described in Section 2.3.

3.4.4 Mixed Precision Training

The training implementation uses MPT as described in Section 2.3.1. The mixed-precision implementation uses PyTorch's autocast context manager with CPU backend configuration:

```
torch.autocast(device_type="cpu", dtype=torch.float16)
```

Listing 2 — Mixed-Precision Training Configuration

This approach achieves approximately 50% reduction in memory usage while maintaining numerical stability equivalent to full-precision training, enabling the comprehensive evaluation of all 16 model variants within available computational resources.

3.4.5 Model Compilation and Performance Optimization

To accelerate training throughput, the implementation leverages JIT compilation as described in Section 2.3.2. The compilation uses PyTorch's ahead-of-time eager mode optimization:

```
torch.compile(model, backend="aot_eager")
```

Listing 3 — Model Compilation Setup

Model compilation provides an expected 1.5-2× speedup in training time by optimizing computational graphs and reducing Python overhead during forward and backward passes. The compilation strategy is particularly beneficial for the iterative nature of object detection training, where the same computational patterns are repeated across thousands of training iterations.

3.4.6 Memory-Efficient Training Pipeline

Given the computational demands of training 16 different model variants, the training pipeline employs several memory management strategies to enable comprehensive experimentation within hardware constraints. Models are trained sequentially rather than in parallel, with explicit memory cleanup between training runs:

```
torch.cuda.empty_cache()  # CUDA GPU memory
torch.mps.empty_cache()   # Apple Silicon unified memory
gc.collect()              # Python garbage collection
```

Listing 4 — Memory Management Strategy

This sequential approach prevents memory fragmentation and allows each model to utilize the full available memory during training, enabling larger effective batch sizes and more stable gradient estimation. The explicit cache management ensures that memory allocated by previous model training runs is properly released before beginning subsequent experiments.

3.4.7 Data Loading and Augmentation Pipeline

The training pipeline employs optimized data loading with 4 parallel worker processes to prevent I/O bottlenecks during training. Persistent workers are enabled to reduce the overhead of worker initialization across epochs, particularly beneficial given the multiple datasets being processed simultaneously.

Data augmentation is integrated directly into the `ObjectDetectionDataset` class, applying photometric distortions (brightness and contrast adjustments) and geometric transformations (random expansion, cropping, and horizontal flipping) to increase training data diversity. All images are resized to the standard SSD300 input resolution of 300×300 pixels with ImageNet normalization statistics:

```
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
```

Listing 5 — ImageNet Normalization Statistics

This normalization approach maintains compatibility with ImageNet-pretrained backbones while adapting to the unique characteristics of thermal imagery through the preprocessing techniques described in Section 2.5.2.

3.4.8 Gradient Management and Numerical Stability

Optional gradient clipping is implemented to handle potential gradient explosion during training, particularly relevant for the deeper ResNet152 architecture. When enabled, gradients are clipped element-wise to prevent destabilization:

```
param.grad.data.clamp_(-grad_clip, grad_clip)
```

Listing 6 — Gradient Clipping Implementation

The optimizer state is preserved across checkpoint loading operations, ensuring that momentum buffers and other optimization state variables remain consistent when resuming training from saved checkpoints. This preservation is crucial for maintaining convergence properties when training is interrupted or distributed across multiple sessions.

3.4.9 Training Statistics and Monitoring

Training progress is monitored through comprehensive loss tracking with both per-epoch and per-iteration granularity. Loss statistics are saved in CSV format within the `stats/loss/` directory, with separate files for each model configuration. This detailed tracking enables post-hoc analysis of convergence behavior and identification of potential training instabilities.

Validation evaluation is performed regularly during training to monitor generalization performance and detect overfitting. The evaluation frequency is balanced to provide meaningful feedback without significantly impacting training throughput, particularly important given the computational overhead of the MultiBox loss calculation across 8732 anchor boxes per image.

Model checkpoints are saved at regular intervals and preserve complete training state including model parameters, optimizer state, and training metadata. This

comprehensive state preservation enables reproducible resumable training and facilitates ablation studies across different hyperparameter configurations.

3.5 Experimental Design

The experimental methodology employs a comprehensive factorial design to systematically evaluate the impact of architectural choices and preprocessing techniques on thermal human detection performance. This approach enables rigorous comparison across multiple variables while controlling for confounding factors that could bias results. The experimental framework is designed to provide statistically meaningful insights into optimal configurations for real-world thermal surveillance applications.

3.5.1 Model Configuration Matrix

The experimental design evaluates 16 distinct model configurations through a systematic $2 \times 2 \times 4$ factorial arrangement examining three primary factors:

Backbone Architecture: Two architectures representing different design philosophies

- VGG16: Traditional sequential CNN with 138M parameters, emphasizing simplicity and proven effectiveness
- ResNet152: Deep residual network with skip connections, addressing vanishing gradient problems through 152 layers

Initialization Strategy: Two approaches to parameter initialization

- Pretrained: Models initialized with ImageNet-pretrained weights, leveraging TL from RGB domain
- Scratch: Random initialization following Xavier uniform distribution, training exclusively on thermal data

Preprocessing Configuration: Four thermal-specific enhancement strategies

- None: Baseline with standard normalization only
- Inversion: Thermal polarity correction addressing hot-white vs. cold-white scenarios

- Edge Enhancement: Two-stage enhancement combining Gaussian blur and Sobel edge detection
- Combined: Integration of both inversion and edge enhancement techniques

This factorial design results in comprehensive model naming convention: `SSD-{VGG|ResNet}-{pretrained|scratch}-{preprocessing}`, enabling systematic analysis of interaction effects between architectural choices and preprocessing strategies.

3.5.2 Evaluation Methodology Framework

The evaluation framework employs multiple complementary metrics to provide comprehensive assessment of model performance across different operational scenarios. The multi-metric approach recognizes that optimal model selection depends on specific deployment requirements and operational constraints.

Primary Detection Metrics:

- **mAP@0.5**: Standard PASCAL VOC metric using 0.5 IoU threshold for positive detection classification
- **MS COCO Style mAP**: 101-point interpolated average precision across IoU thresholds from 0.5 to 0.95 in 0.05 increments
- **Precision-Recall Curves**: Performance characterization across confidence threshold ranges from 0.01 to 1.0

Computational Efficiency Metrics:

- **Inference Speed**: Forward pass timing on target hardware configurations
- **Memory Usage**: Peak GPU/CPU memory consumption during inference
- **Model Size**: Parameter count and storage requirements for edge deployment

The evaluation parameters are configured to reflect real-world deployment scenarios:

```
min_score = 0.01           # Minimum confidence for detection consideration
max_overlap = 0.45         # NMS IoU threshold for duplicate removal
top_k = 200                # Maximum detections per image
```

Listing 7 — Evaluation Parameters Configuration

These thresholds balance detection sensitivity with false positive suppression, particularly important for surveillance applications where excessive false alarms reduce system utility.

3.5.3 Statistical Analysis Framework

To ensure robust conclusions, the experimental design incorporates several statistical rigor measures addressing the inherent variability in neural network training and evaluation.

Reproducibility Controls:

- Fixed random seeds for dataset splitting ensuring consistent train/validation/test divisions
- Deterministic training procedures where computationally feasible
- Comprehensive checkpoint preservation enabling exact training replication

Performance Validation:

- Cross-dataset evaluation using multiple thermal datasets to assess generalization
- Independent test set (OSU-T) completely unseen during training for unbiased evaluation
- Statistical significance testing for performance differences between model configurations

Ablation Study Design: The factorial structure enables systematic ablation studies examining:

- Architecture effects: VGG16 vs. ResNet152 performance isolation
- Initialization impact: Transfer learning effectiveness across thermal domains
- Preprocessing contribution: Individual and combined enhancement technique effects
- Interaction analysis: Synergistic effects between architectural and preprocessing choices

3.5.4 Dataset Stratification and Cross-Validation

The experimental design addresses dataset heterogeneity through strategic splitting that maintains representative sampling across diverse thermal imaging scenarios. The multi-dataset approach ensures robust evaluation across varying environmental conditions, camera configurations, and thermal contrast scenarios.

Training Set Composition:

- FLIR ADAS v2: 8,205 images (40.5% of training data) providing automotive-focused scenarios
- AAU-PD-T: 706 images (3.5%) contributing elevated surveillance perspectives
- M3FD Detection: 2,520 images (12.4%) adding urban driving complexity
- KAIST-CVPR15: 8,815 images (43.5%) representing diverse pedestrian scenarios

This distribution ensures adequate representation of major thermal imaging scenarios while preventing any single dataset from dominating model behavior. The relatively balanced contribution from FLIR ADAS v2 and KAIST-CVPR15 provides stability in training while smaller datasets contribute specialized scenarios.

Validation Strategy:

- Primary validation: FLIR ADAS v2 and KAIST-CVPR15 subsets providing diverse feedback during training
- Cross-validation: Models evaluated across all available datasets to assess generalization
- Independent testing: OSU-T dataset serving as completely unseen evaluation set

3.5.5 Performance Benchmarking Protocol

The benchmarking protocol establishes standardized evaluation procedures ensuring fair comparison across all model configurations and enabling reproducible results for future research.

Evaluation Pipeline:

1. **Model Loading**: Checkpoint restoration with complete state preservation
2. **Data Preprocessing**: Application of model-specific preprocessing pipeline
3. **Inference Execution**: Batched prediction generation with timing measurement

4. **Post-processing**: NMS application with standardized parameters
5. **Metric Calculation**: Comprehensive evaluation using vectorized operations for efficiency

Hardware Standardization: All experiments are conducted on consistent hardware configurations to eliminate performance variations due to computational differences. Device selection follows automatic GPU/MPS/CPU detection with consistent memory allocation and optimization settings.

Timing Methodology: Inference speed measurements exclude data loading and preprocessing overhead, focusing on core model computation time. Multiple measurement rounds with warm-up iterations ensure stable timing estimates unaffected by initialization overhead.

3.5.6 Experimental Controls and Bias Mitigation

The experimental design incorporates several controls to minimize potential sources of bias and ensure valid conclusions about model performance differences.

Training Controls:

- Identical hyperparameters across all model configurations preventing confounding from optimization differences
- Sequential model training with memory cleanup preventing interference between experiments
- Consistent data augmentation and normalization ensuring fair preprocessing comparison

Evaluation Controls:

- Standardized inference procedures eliminating implementation-dependent performance variations
- Consistent metric calculation using identical evaluation code across all models
- Reproducible random sampling for statistical analysis and significance testing

Environmental Controls:

- Controlled software environment with fixed library versions

- Consistent hardware utilization through sequential rather than parallel training
- Systematic checkpoint and logging procedures enabling post-hoc verification

This comprehensive experimental design provides the methodological foundation for drawing reliable conclusions about optimal thermal human detection configurations while maintaining the scientific rigor necessary for industrial application and academic contribution.

4 Results and Analysis

The experimental evaluation reveals significant performance variations across different model configurations and preprocessing approaches when applied to thermal human detection tasks. This section presents comprehensive results from training 16 distinct model variants, combining backbone architectures (VGG16 vs. ResNet152), initialization strategies (pretrained vs. scratch), and preprocessing techniques (none, inversion, edge enhancement, combined). The analysis demonstrates clear patterns in model behavior and identifies optimal configurations for thermal surveillance applications.

Key areas to develop:

- Training convergence analysis: Loss curves and stability patterns
- Detection accuracy results: mAP scores across all model variants
- Preprocessing impact: Quantitative comparison of enhancement techniques
- Backbone architecture comparison: VGG16 vs. ResNet152 performance
- Initialization strategy effects: Pretrained vs. scratch training outcomes
- Computational efficiency: Inference speed and memory requirements
- Dataset-specific performance: Results breakdown by thermal dataset
- Error analysis: Common failure cases and detection limitations

4.1 Training Performance

Reports training loss curves, converegence behavior, and computational requirements for different mdoel variants.

4.2 Detection Accuracy Analysis

Provides detailed mAP scores and detection performance metrics for each model configuraiton and preprocessing technique.

4.3 Preprocessing Impact Evaluation

Analzyes the effects of image inversion and edge enhancement on detection performance.

5 Discussion

The experimental results provide valuable insights into the practical applicability of SSD architectures for thermal human detection systems. While certain configurations demonstrate superior performance, the choice of optimal model depends on specific deployment requirements, including accuracy thresholds, computational constraints, and operational environments. This section interprets the findings within the context of real-world surveillance applications and addresses the broader implications for thermal imaging-based security systems.

Key areas to develop:

- Performance trade-offs: Accuracy vs. computational efficiency analysis
- Preprocessing effectiveness: When and why certain techniques work better
- Backbone selection criteria: Situational advantages of VGG16 vs. ResNet152
- Real-world deployment implications: Edge computing considerations
- Limitations and constraints: Environmental factors affecting performance
- Comparison with existing thermal detection systems
- Cost-benefit analysis for industrial implementation
- Future optimization potential and research directions

5.1 Model Performance Comparison

Compares SSD-VGG16 and SSD-ResNer performance and discusses trade-offs between accuracy and computational efficiency.

5.2 Practical Deployment Considerations

Discusses real-world application scenarios and system requirements for thermal surveillance.

6 Conclusion and Future Work

This thesis has systematically evaluated the application of Single Shot MultiBox Detector architectures for human detection in thermal imagery, providing empirical evidence for optimal model configurations in surveillance applications. The comprehensive analysis of 16 model variants across multiple thermal datasets has yielded practical insights for deploying neural networks in infrared-based security systems. The findings contribute to both academic understanding and industrial implementation of thermal computer vision technologies.

Key areas to develop:

- Key findings summary: Best-performing model configurations identified
- Methodological contributions: Systematic evaluation framework for thermal detection
- Practical implications: Guidelines for industrial thermal surveillance deployment
- Technical achievements: Successful adaptation of RGB models to thermal domain
- Research limitations: Dataset constraints and environmental factors
- Future research directions: Advanced architectures and multi-modal approaches
- Industry impact: Potential applications beyond security surveillance
- Recommendations: Implementation guidelines for practitioners

7 Appendix

7.1 Code Snippets

Insert code snippets like this:

```
const ReactComponent = () => {
  return (
    <div>
      <h1>Hello World</h1>
    </div>
  );
};

export default ReactComponent;
```

Listing 8 — Codeblock Example

Glossary

AP Average Precision. Calculated as the area under the precision-recall curve.

AdaBoost Adaptive Boosting. A type of ensemble learning algorithm that combines multiple weak classifiers to form a strong classifier.

BGD Batch Gradient Descent. An optimization algorithm used to minimize the loss function of machine learning models by iteratively updating the model parameters based on their gradients with respect to the entire training dataset.

BN Batch Normalization. A technique used in deep neural networks to normalize the activations of the layers. It helps to speed up the training of the network and improve its performance.

Backpropagation A method used to train neural networks by calculating the gradient of the loss function with respect to the weights of the network and updating the weights in the opposite direction of the gradient. It is called backpropagation because the gradient is calculated from the prediction to the input layers.

Batch A batch is a group of data processed together as a unit.

CNN A convolutional neural network (CNN) is a type of neural network designed to process data with a grid-like topology, such as images.

CUDA Compute Unified Device Architecture. A parallel computing platform and programming model developed by NVIDIA for general computing on GPUs.

FC Layer A fully connected layer is a layer in a neural network where each neuron is connected to every neuron in the previous layer.

HOG Histogram of Oriented Gradients. A feature descriptor used in computer vision and image processing for the purpose of object detection. It is based on the distribution of intensity gradients or edge directions in an image.

IoU Intersection over Union. A metric used to evaluate the accuracy of object detection models. It is calculated as the ratio of the area of overlap between the predicted bounding box and the ground truth bounding box to the area of union between the two boxes.

MBGD Mini-Batch Gradient Descent. A type of gradient descent algorithm that updates the weights of the neural network using a small subset of the training data at a time.

MPS Metal Performance Shaders. A framework for accelerating machine learning workloads on Apple Silicon devices.

MS COCO Microsoft Common Objects in Context. A large-scale object detection, segmentation, and captioning dataset.

NMS A technique used to eliminate redundant bounding boxes in object detection models. It works by selecting the bounding box with the highest confidence score and eliminating all other bounding boxes that have an IoU greater than a specified threshold with the selected bounding box.

PASCAL VOC Pascal Visual Object Classes. A dataset for object detection and segmentation tasks.

ReLU Rectified Linear Unit. A type of activation function used in neural networks. It is defined as $f(x) = \max(0, x)$.

SGD SGD is an optimization algorithm used to minimize the loss function in machine learning models by iteratively updating the model parameters based on their partial derivatives with respect to individual samples in the training data.

SSD A single shot multibox detector (SSD) is a type of object detection model that uses a single forward pass of the network to predict the bounding boxes and class scores for all objects in the image.

SVM Support Vector Machine. A type of supervised learning algorithm that is used for classification and regression tasks. It is based on the idea of finding a hyperplane that best separates the data into different classes.

Sigmoid A mathematical function that maps any real number to a value between 0 and 1. It is commonly used as an activation function in neural networks. Refer to Equation 19 for the mathematical definition.

TL Transfer Learning. A technique used in machine learning where a pre-trained model is used as the starting point for a new model. It helps to speed up the training of the new model and improve its performance.

Tensor A tensor is a mathematical object that generalizes scalars, vectors, and matrices to higher-dimensional arrays.

VGP	Vanishing Gradient Problem. A problem that occurs in deep neural networks where the gradients of the loss function with respect to the weights become very small, making it difficult to train the network.
ViT	Vision Transformer. A type of transformer model that is designed for computer vision tasks.
YOLO	You Only Look Once. A type of object detection model that uses a single forward pass of the network to predict the bounding boxes and class scores for all objects in the image.
mAP	Mean Average Precision. Calculated as the mean of the AP values for each class.

References

[1] M. A. Farooq, P. Corcoran, C. Rotariu, and W. Shariff, “Object Detection in Thermal Spectrum for Advanced Driver-Assistance Systems (ADAS),” no. arXiv:2109.09854. arXiv, Oct. 2021. doi: 10.48550/arXiv.2109.09854.

[2] K. R. Akshatha, A. K. Karunakar, S. B. Shenoy, A. K. Pai, N. H. Nagaraj, and S. S. Rohatgi, “Human Detection in Aerial Thermal Images Using Faster R-CNN and SSD Algorithms,” *Electronics*, vol. 11, no. 7, p. 1151, Jan. 2022, doi: 10.3390/electronics11071151.

[3] “FREE - FLIR Thermal Dataset for Algorithm Training | OEM.FLIR.Com.”

[4] N. U. Huda, B. D. Hansen, R. Gade, and T. B. Moeslund, “The Effect of a Diverse Dataset for Transfer Learning in Thermal Person Detection,” *Sensors*, vol. 20, no. 7, p. 1982, Jan. 2020, doi: 10.3390/s20071982.

[5] J. W. Davis and M. A. Keck, “A Two-Stage Template Approach to Person Detection in Thermal Imagery,” in *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*, Jan. 2005, pp. 364–369. doi: 10.1109/ACVMOT.2005.14.

[6] J. Liu *et al.*, “Target-Aware Dual Adversarial Learning and a Multi-scenario Multi-Modality Benchmark to Fuse Infrared and Visible for Object Detection,” no. arXiv:2203.16220. arXiv, Mar. 2022. doi: 10.48550/arXiv.2203.16220.

[7] S. Hwang, J. Park, N. Kim, Y. Choi, and I. S. Kweon, “Multispectral Pedestrian Detection: Benchmark Dataset and Baseline,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 1037–1045. doi: 10.1109/CVPR.2015.7298706.

[8] P. Viola and M. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA: IEEE Comput. Soc, 2001, p. I–511–I–518. doi: 10.1109/CVPR.2001.990517.

[9] A. Dosovitskiy *et al.*, “An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale,” no. arXiv:2010.11929. arXiv, Jun. 2021. doi: 10.48550/arXiv.2010.11929.

[10] D. K. Alqahtani, A. Cheema, and A. N. Toosi, “Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices,” no. arXiv:2409.16808. arXiv, Sep. 2024. doi: 10.48550/arXiv.2409.16808.

[11] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA: IEEE, 2005, pp. 886–893. doi: 10.1109/CVPR.2005.177.

[12] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.

[13] Y. LeCun *et al.*, “Handwritten Digit Recognition with a Back-Propagation Network,” in *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, 1989.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” no. arXiv:1311.2524. arXiv, Oct. 2014. doi: 10.48550/arXiv.1311.2524.

[15] R. Girshick, “Fast R-CNN,” no. arXiv:1504.08083. arXiv, Sep. 2015. doi: 10.48550/arXiv.1504.08083.

[16] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” no. arXiv:1506.01497. arXiv, Jan. 2016. doi: 10.48550/arXiv.1506.01497.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," no. arXiv:1506.02640. arXiv, May 2016. doi: 10.48550/arXiv.1506.02640.

[18] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," vol. 9905. pp. 21–37, 2016. doi: 10.1007/978-3-319-46448-0_2.

[19] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable Object Detection Using Deep Neural Networks," no. arXiv:1312.2249. arXiv, Dec. 2013. doi: 10.48550/arXiv.1312.2249.

[20] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," no. arXiv:1609.04747. arXiv, Jun. 2017. doi: 10.48550/arXiv.1609.04747.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," no. arXiv:1512.03385. arXiv, Dec. 2015. doi: 10.48550/arXiv.1512.03385.

[22] P. Micikevicius *et al.*, "Mixed Precision Training," no. arXiv:1710.03740. arXiv, Feb. 2018. doi: 10.48550/arXiv.1710.03740.

[23] J. Ansel *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, La Jolla CA USA: ACM, Apr. 2024, pp. 929–947. doi: 10.1145/3620665.3640366.

[24] "PyTorch 2.x."

[25] A. Deng, X. Li, D. Hu, T. Wang, H. Xiong, and C. Xu, "Towards Inadequately Pre-trained Models in Transfer Learning," no. arXiv:2203.04668. arXiv, Aug. 2023. doi: 10.48550/arXiv.2203.04668.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," no. arXiv:1409.1556. arXiv, Apr. 2015. doi: 10.48550/arXiv.1409.1556.

[27] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” no. arXiv:1502.03167. arXiv, Mar. 2015. doi: 10.48550/arXiv.1502.03167.

[28] J. Bridle, “Training Stochastic Model Recognition Algorithms as Networks Can Lead to Maximum Mutual Information Estimation of Parameters,” in *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, 1989.

[29] L. d. F. Costa, “Further Generalizations of the Jaccard Index,” no. arXiv:2110.09619. arXiv, Nov. 2021. doi: 10.48550/arXiv.2110.09619.

[30] O. Elharrouss *et al.*, “Loss Functions in Deep Learning: A Comprehensive Review,” no. arXiv:2504.04242. arXiv, Apr. 2025. doi: 10.48550/arXiv.2504.04242.

[31] J. Beyerer, M. Ruf, and C. Herrmann, “CNN-based Thermal Infrared Person Detection by Domain Adaptation,” in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, M. C. Dudzik and J. C. Ricklin, Eds., Orlando, United States: SPIE, May 2018, p. 8. doi: 10.1117/12.2304400.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

[33] J. Burnham, J. Hardy, and K. Meadors, “Comparison of the Roberts, Sobel, Robinson, Canny, and Hough Image Detection Algorithms,” 1997.

[34] “ResNet v1.5 for PyTorch | NVIDIA NGC.”

[35] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark,” no. arXiv:2109.14545. arXiv, Jun. 2022. doi: 10.48550/arXiv.2109.14545.