TRAFFIC SIGN RECOGNITION USING CNN and ResNet Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering

By

VUPPALA SATYA MITRA (Reg.No - 39111114)
VELLANKI VIKAS (Reg.No – 39111103)

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING SCHOOL OF COMPUTING

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI - 600119

NOVEMBER - 2022

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with ―A‖ grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **VUPPALA SATYAMITRA (39111114) and VELLANKI VIKAS (39111103)** who carried out the Project Phase-1 entitled **"TRAFFIC SIGN RECOGNITION USING CNN AND RESNET"** under my supervision from June 2022 to November 2022.

Veena K

Internal Guide

Dr. Veena K.,

[illegible]

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CSE

Submitted for Viva voce Examination held on 20.04.2023

[illegible]

[illegible]

Internal Examiner **External Examiner**

DECLARATION

I, **VUPPALA SATYA MITRA (REG NO: 39111114),** hereby declare that the Project Phase-1 Report entitled "**TRAFFIC SIGN RECOGNITION USING CNN AND RESNET"** done by me under the guidance of **Dr. Veena K** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

Sathyamithra

DATE: 20.11.22

PLACE: Chennai **SIGNATURE OF THECANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **DR. VEENA K,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-1 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Object identification and classification plays a very important role in the real-time world and also in day-to-day life. There are many useful applications which use image identification techniques, some of them are G-lens, Cancer prediction, Optical Character Recognition(OCR), Face Recognition and Identification. Convolutional Neural Networks (CNN) is an advanced technique that focuses on feature extraction and is an efficient solution to many Object recognition problems. CNN has gained its popularity mainly due to its faster execution. Traffic-sign recognition is important in the development of real-world applications like autonomous driving Tesla Cars, Traffic surveillance. In our work a CNN model is implemented on the Belgium Traffic Signs Data set (BTS) which consists of 62 different traffic signs. In the first stage of our method we did the pre-processing of data and trained our model using Residual Blocks. We have achieved an accuracy of 94.25%.

Keywords: Object Classification; Neural Networks; CNN; BTS; Residual Block; Feature Extraction; Region of Interest.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Computer Vision techniques have solved many image classification tasks. In recent years Deep Learning has become successful in Computer Vision areas where the results were predicted accurately for a vast variety of image identification or vision tasks. Neural networks have been used for the image classification tasks. Even though there are traditional classification algorithms like SVM, KNN, Decision Tree that do not work as efficiently as neural networks when there is a large database [1]. The architectures in the neural networks will be designed by humans by going through various experimental results.

Convolutional Neural Networks (CNN) has become very famous in the Deep Learning field for object classification, handwriting recognition, digit identification, speech recognition and face detection due to its accuracy and faster computation [2]. CNN is a deep learning technique in which a stack of convolution, max-pooling, activation layers will be there [2].

Next input will be processed into each layer step by step where feature extraction takes place and probabilities for each will be predicted. Over the years the number of vehicles on the road has been increasing which has resulted in an increasing number of accidents. Main cause of these accidents is the ignorance of traffic signs.

Traffic signs classification is one of the major areas of research in building Advanced Driver Assistance Systems(ADAS) which could provide safety precautions and guidance to driver according to the surrounding environment. Understanding of environment, detection of various objects and their classification into vehicles, roads, signs and pedestrians are the main features in ADAS. Our system should recognize the traffic signs under any situations like(rain, heavy dust, night time or a part of the image is occluded) then only we can say our model is good.

Recognition is done in two steps: first the system should locate the traffic signs at its accurate position and detect the traffic sign and classify it. So the main objective of this paper is to build an efficient CNN-architecture for traffic sign recognition. This paper focuses on the Belgium Traffic Signs data set using a combination of CNN layers and Residual block. Residual network is more efficient when compared to other deep neural networks because in ResNet there won’t be

any Vanishing Gradient Problem due to skip connections.

1.1 Machine Learning:

Machine learning is to predict the future from past data. Machine learning (ML) is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data and the basics of Machine Learning, implementation of a simple machine learning algorithm using python. Process of training and prediction involves use of specialized algorithms. It feed the training data to an algorithm, and the algorithm uses this training data to give predictions on a new test data. Machine learning can be roughly separated in to three categories. There are supervised learning, unsupervised learning and reinforcement learning. Supervised learning program is both given the input data and the corresponding labeling to learn data has to be labeled by a human being beforehand. Unsupervised learning is no labels. It provided to the learning algorithm. This algorithm has to figure out the clustering of the input data. Finally, Reinforcement learning dynamically interacts with its environment and it receives positive or negative feedback to improve its performance.

Data scientists use many different kinds of machine learning algorithms to discover patterns in python that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they “learn” about data to make predictions: supervised and unsupervised learning. Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function from input variables(X) to discrete output variables(y). In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc.

Analyses Predicts

Trains Machine Result Past Dataset Fig 1.1: Process of Machine learning

Supervised Machine earning is the majority of practical machine learning uses supervised learning. Supervised learning is where have input variables (X) and an output variable (y) and use an algorithm to learn the mapping function from the input to the output is y = f(X). The goal is to approximate the mapping function so well that when you have new input data (X) that you can predict the output variables (y) for that data. Techniques of Supervised Machine Learning algorithms include logistic regression, multi-class classification, Decision Trees and support vector machines etc. Supervised learning requires that the data used to train the algorithm is already labeled with correct answers. Supervised learning problems can be further grouped into Classification problems. This problem has as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for categorical for classification. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. A classification problem is when the output variable is a category, such as “red” or “blue”.

Movements are often normal indoor activities such as standing, walking, laying sitting. Sensors are often located on the subject such as a smartphone or vest and often record accelerometer data in three dimensions (x, y, z).The idea is that once the subject’s activity is recognized and known, an intelligent computer system can then offer assistance. It is a challenging problem because there is no clear analytical way to relate the sensor data to specific actions in a general way. It is technically challenging because of the large volume of sensor data collected (e.g., tens or

hundreds of observations per second) and the classical use of hand crafted features and heuristics from this data in developing predictive models

Human activities have been commonly used to define human behavioral patterns. The availability of sensors in mobile platforms has enabled the development of a variety of practical applications for several areas of knowledge such as:

· Health-through fall detection systems, elderly monitoring, and disease prevention.

· Internet of Things and Smart Cities—through solutions used to recognize and monitor domestic activities and electrical energy saving. · Security-through individual activity monitoring solutions, crowd anomaly detection, and object tracking.

· Transportation—through solutions related to vehicle and pedestrian navigation.

For this reason, the development of solutions that recognize human activities (HAR) through computational technologies and methods has been explored in recent years

For this reason, the development of solutions that recognize human activities (HAR) through computational technologies and methods has been explored in recent years. In this sense, the HAR problem has previously been treated as a typical pattern recognition problem, and more specifically, a classification problem, that is, to identify the activity being performed by an individual at a given moment. Smartphones have been commonly employed to develop HAR solutions because of the ubiquitous capability and diversity of sensors embedded in such devices. Smartphones are included in the scope of wearable computing, and these devices are considered part of mobile computing-based HAR systems. The advantage of smartphones over other wearable devices is associated with their ability to capture and process data, transmit and receive data, and connect with other devices or sensors available in the physical environment. Inertial sensors such as the accelerometer and gyroscope are most commonly used to capture information related to acceleration and direction of movement of the human body, respectively.

1.2 DEEP LEARNING:

Deep learning is a modern incarnation of the long-running trend in artificial intelligence that has been moving from streamlined systems based on expert knowledge toward flexible statistical models. Early AI systems were rule based, applying logic and expert knowledge to derive results. Later systems incorporated learning to set their adjustable parameters, but these were usually few in number.

Today's neural networks also learn parameter values, but those parameters are part of such flexible computer models that—if they are big enough—they become universal function approximators, meaning they can fit any type of data. This unlimited flexibility is the reason why deep learning can be applied to so many different domains.

The flexibility of neural networks comes from taking the many inputs to the model and having the network combine them in myriad ways. This means the outputs won't be the result of applying simple formulas but instead immensely complicated ones.

For example, when the cutting-edge image-recognition system Noisy Student converts the pixel values of an image into probabilities for what the object in that image is, it does so using a network with 480 million parameters. The training to ascertain the values of such a large number of parameters is even more remarkable because it was done with only 1.2 million labeled images-which may understandably confuse those of us who remember from high school algebra that we are supposed to have more equations than unknowns. Breaking that rule turns out to be the key.

Deep-learning models are overparameterized, which is to say they have more parameters than there are data points available for training. Classically, this would lead to overfitting, where the model not only learns general trends but also the random vagaries of the data it was trained on. Deep learning avoids this trap by initializing the parameters randomly and then iteratively adjusting sets of them to better fit the data using a method called stochastic gradient descent. Surprisingly, this procedure has been proven to ensure that the learned model generalizes well.

The success of flexible deep-learning models can be seen in machine translation. For decades, software has been used to translate text from one language to another. Early approaches to this problem used rules designed by grammar experts. But as more textual data became available in specific languages, statistical approaches-ones that go by such esoteric names as maximum entropy,

hidden Markov models, and conditional random fields-could be applied. Initially, the approaches that worked best for each language differed based on data availability and grammatical properties. For example, rule-based approaches to translating languages such as Urdu, Arabic, and Malay outperformed statisticalones at first. Today, all these approaches have been outpaced by deep learning, which has proven itself superior almost everywhere it is applied.

1.3 NEURAL NETWORKS

Deep learning is in fact a new name for an approach to artificial intelligence called neural networks, which have been going in and out of fashion for more than 70 years. Neural networks were first proposed in 1944 by Warren McCullough and Walter Pitts, two University of Chicago researchers who moved to MIT in 1952 as founding members of what is sometimes called the first cognitive science department.

Neural nets were a major area of research in both neuroscience and computer science until 1969, when, according to computer science lore, they were killed off by the MIT mathematicians Marvin Minsky and Seymour Papert, who a year later would become co-directors of the new MIT Artificial Intelligence Laboratory.

The technique then enjoyed a resurgence in the 1980s, fell into eclipse again in the first decade of the new century, and has returned like gangbusters in the second, fueled largely by the increased processing power of graphics chips.

“There’s this idea that ideas in science are a bit like epidemics of viruses,” says Tomaso Poggio, the Eugene McDermott Professor of Brain and Cognitive Sciences at MIT, an investigator at MIT’s McGovern Institute for Brain Research, and director of MIT’s Center for Brains, Minds, and Machines. “There are apparently five or six basic strains of flu viruses, and apparently each one comes back with a period of around 25 years. People get infected, and they develop an immune response, and so they don’t get infected for the next 25 years. And then there is a new generation that is ready to be infected by the same strain of virus. In science, people fall in love with an idea, get excited about it, hammer it to death, and then get immunized — they get tired of it. So, ideas should have the same kind of periodicity!” Weighty

matters

Neural nets are a means of doing machine learning, in which a computer learns to perform some tasks by analyzing training examples. Usually, the examples have been hand-labeled in advance. An object recognition system, for instance, might be fed thousands of labeled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels.

Modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today’s neural nets are organized into layers of nodes, and they are “feed-forward,” meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

To each of its incoming connections, a node will assign a number known as a “weight.” When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node “fires,” which in today’s neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections.

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

Minds and machines

The neural nets described by McCullough and Pitts in 1944 had thresholds and weights, but they were not arranged into layers, and the researchers didn't specify any training mechanism. What McCullough and Pitts showed was that a neural net could, in principle, compute any function that a digital computer could. The result was more neuroscience than computer science: The point was to suggest that the human brain could be thought of as a computing device.

Neural nets continue to be a valuable tool for neuroscientific research. For instance, particular network layouts or rules for adjusting weights and thresholds have reproduced observed features of human neuroanatomy and cognition, an indication that they capture something about how the brain processes information.

The first trainable neural network, the Perceptron, was demonstrated by the Cornell University psychologist Frank Rosenblatt in 1957. The Perceptron's design was much like that of the modern neural net, except that it had only one layer with adjustable weights and thresholds, sandwiched between input and output layers.
Perceptron's were an active area of research in both psychology and the fledgling discipline of computer science until 1959, when Minsky and Papert published a book titled "Perceptrons," which demonstrated that executing certain fairly common computations on Perceptrons would be impractically time consuming.

"Of course, all of these limitations kind of disappear if you take machinery that is a little more complicated - like, two layers," Poggio says. But at the time, the book had a chilling effect on neural-net research.

“You have to put these things in historical context,” Poggio says. “They were arguing for programming for languages like Lisp. Not many years before, people were still using analog computers. It was not clear at all at the time that programming was the way to go. I think they went a little bit overboard, but as usual, it’s not black and white. If you think of this as this competition between analog computing and digital computing, they fought for what at the time was the right thing.

1.4 Convolutional Neural Network (CNN):

CNN stands for Convolutional Neural Network. It is a type of artificial neural network that is commonly used in computer vision tasks such as image classification, object detection, and segmentation.

CNNs are designed to automatically learn spatial hierarchies of features from the input images through a process called convolution. This involves sliding a set of filters over the input image and computing dot products between the filter weights and image pixel values to generate feature maps.

The resulting feature maps are then passed through activation functions, such as ReLU, and downsampled through pooling layers, such as max pooling, to reduce the spatial resolution and capture the most important features.

Finally, the high-level feature representations are fed into one or more fully connected layers for classification or regression tasks.

CNNs have been highly successful in a wide range of computer vision tasks and have achieved state-of-the-art performance on many benchmarks.

A CNN typically consists of several layers, each with a specific function in the network. Here are the most common types of layers used in a typical CNN:

1. Convolutional layer: This layer performs convolutional operations on the input image using a set of filters to extract relevant features. Each filter is a small matrix that is slid over the input image, producing a feature map for each filter.

2. Activation layer: After the convolutional layer, an activation function is typically

applied to introduce nonlinearity into the network. The most commonly used activation function in CNNs is ReLU (Rectified Linear Unit).

3. Pooling layer: This layer reduces the spatial dimensionality of the feature maps generated by the convolutional layer. The most commonly used pooling operation is

max pooling, which selects the maximum value within a small region of the feature map.

4. Fully connected layer: This layer connects all neurons from the previous layer to every neuron in the current layer, allowing the network to learn complex representations of the input features. The output of the fully connected layer is often used for classification or regression.

5. Dropout layer: This layer randomly drops out some of the neurons in the previous layer during training to prevent overfitting.

6. Batch normalization layer: This layer normalizes the input to the layer to speed up the training process and improve performance.

7. Output layer: This layer produces the final output of the network. In classification tasks, the output layer typically consists of softmax activation function that produces a probability distribution over the different classes. In regression tasks, the output layer has a linear activation function that produces a continuous output value.

1.4.1 Convolutional Layer:

The convolutional layer is one of the core layers in a CNN. It applies a set of learnable filters (also known as kernels) to the input image, producing a set of output feature maps.

During the convolution operation, the filter is slid over the input image, computing the dot product between the filter weights and the pixel values of the input image at each position. The output of this operation is a single value in the output feature map. This process is repeated for every position in the input image to generate the entire output feature map.

The purpose of the convolutional layer is to extract relevant features from the input image. Each filter in the convolutional layer is designed to detect a specific feature in the input image, such as edges, corners, or blobs. By learning multiple filters in parallel, the convolutional layer can learn complex feature representations of the input image.

The size of the output feature maps depends on the size of the input image, the size of the filters, the stride of the convolution operation, and the padding applied to the input image.

In summary, the convolutional layer in a CNN applies a set of learnable filters to the input image, producing a set of output feature maps. This layer is responsible for extracting relevant features from the input image and learning complex feature representations that can be used for further analysis or classification.

1.4.2 Activation Layer:

The activation layer in a CNN introduces nonlinearity into the network. It is typically applied after a convolutional or fully connected layer.

The purpose of the activation layer is to allow the CNN to learn more complex and abstract features of the input data. Without the activation function, the CNN would be limited to learning linear transformations of the input data.

The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU). The ReLU function sets all negative values to zero and leaves all positive values unchanged. The ReLU function is computationally efficient, easy to optimize, and has been shown to work well in practice.

Other activation functions that are commonly used in CNNs include sigmoid and hyperbolic tangent (tanh). However, these activation functions are less commonly used in modern CNN architectures because they are less computationally efficient and can suffer from the vanishing gradient problem.

In summary, the activation layer in a CNN introduces nonlinearity into the network and allows the CNN to learn more complex and abstract features of the input data. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU), although other activation functions such as sigmoid and hyperbolic tangent (tanh) can also be used.

1.4.3 Pooling Layer:

The pooling layer in a CNN is used to reduce the spatial dimensions of the feature maps generated by the convolutional layer. It operates independently on each feature map by partitioning it into non-overlapping rectangular sub-regions and computing a single output value for each sub-region.

The most commonly used pooling operation is max pooling, which selects the maximum value within each sub-region of the feature map. Other types of pooling operations, such as average pooling, can also be used.

The purpose of the pooling layer is to make the network more robust to variations in the input image, such as changes in translation, rotation, or scale. By reducing the spatial dimensions of the feature maps, the pooling layer also reduces the computational cost of the subsequent layers in the network.

The size of the pooling sub-regions and the stride of the pooling operation are hyperparameters that can be tuned to control the degree of spatial reduction in the feature maps. In some cases, the pooling layer can be replaced with a stride convolutional layer that directly reduces the spatial dimensions of the feature maps.

In summary, the pooling layer in a CNN is used to reduce the spatial dimensions of the feature maps generated by the convolutional layer. It operates independently on each feature map and can be used to make the network more robust to variations in the input image while reducing the computational cost of the subsequent layers in the network. The most commonly used pooling operation is max pooling.

1.4.4 Fully Connected Layer:

The fully connected layer (also known as the dense layer) in a CNN is a layer in which every neuron is connected to every neuron in the previous layer. It is typically used at the end of the CNN to perform classification or regression on the features extracted from the input image by the preceding convolutional and pooling layers.

The fully connected layer takes the flattened output of the preceding layer and applies a matrix multiplication to it, followed by a bias term and an activation

function. The output of the fully connected layer is then passed to the output layer, which typically uses a softmax function to produce a probability distribution over the possible output classes.

The number of neurons in the fully connected layer depends on the number of features extracted by the preceding convolutional and pooling layers, and the number of output classes in the classification task. The parameters of the fully connected layer (i.e., weights and biases) are learned during the training phase using backpropagation and gradient descent.

One of the drawbacks of the fully connected layer is that it requires a fixed input size, which limits the flexibility of the network. To address this issue, some modern CNN architectures, such as the Fully Convolutional Network (FCN), use only convolutional and pooling layers and no fully connected layers.

In summary, the fully connected layer in a CNN is a layer in which every neuron is connected to every neuron in the previous layer. It is typically used at the end of the CNN to perform classification or regression on the features extracted from the input image by the preceding convolutional and pooling layers. The number of neurons in the fully connected layer depends on the number of features extracted and the number of output classes, and its parameters are learned during training using backpropagation and gradient descent.

1.4.5 Dropout Layer:

The dropout layer is a regularization technique used in CNNs to prevent overfitting. It is typically applied after the fully connected layer and randomly drops out (i.e., sets to zero) a fraction of the neurons in the layer during each training iteration.

The dropout layer helps to prevent overfitting by forcing the network to learn redundant representations of the input data. By randomly dropping out neurons during each training iteration, the dropout layer encourages each neuron to be more robust and less reliant on the presence of specific other neurons.

The dropout rate is a hyperparameter that determines the fraction of neurons that

are dropped out during each training iteration. A common value for the dropout rate is 0.5, which means that half of the neurons in the layer are dropped out during each training iteration.

During the inference phase, the dropout layer is turned off, and all neurons are used for the prediction. This means that the output of the dropout layer during inference is equal to the expected output of the layer during training, scaled by the dropout rate.

In summary, the dropout layer is a regularization technique used in CNNs to prevent overfitting. It randomly drops out a fraction of the neurons in the fully connected layer during each training iteration, forcing the network to learn redundant representations of the input data. The dropout rate is a hyperparameter that determines the fraction of neurons that are dropped out during each training iteration, and the layer is turned off during inference.

1.4.6 Batch Normalization Layer:

Batch normalization is a technique used in CNNs to improve the training speed and stability by normalizing the inputs to each layer. It is typically applied after the convolutional or fully connected layer and before the activation function.

The batch normalization layer operates on a batch of training examples and normalizes the activations of the previous layer by subtracting the mean and dividing by the standard deviation of the batch. The resulting normalized values are then scaled by learned parameters, known as the gamma and beta parameters, and passed to the activation function.

The purpose of batch normalization is to reduce the internal covariate shift, which is the phenomenon in which the distribution of the activations of a layer changes during training. This can lead to slower convergence and worse generalization performance. By normalizing the inputs to each layer, batch normalization helps to stabilize the training process and make the network more robust to changes in the distribution of the input data.

In addition to improving the training speed and stability, batch normalization has also

been shown to act as a regularization technique, reducing overfitting and improving generalization performance.

One potential drawback of batch normalization is that it increases the computational cost of the network, since it requires additional calculations for each batch of training examples. However, the benefits of batch normalization in terms of improved training speed and stability often outweigh the increased computational cost.

In summary, the batch normalization layer is a technique used in CNNs to improve the training speed and stability by normalizing the inputs to each layer. It reduces the internal covariate shift and acts as a regularization technique, improving generalization performance. The main drawback of batch normalization is that it increases the computational cost of the network.

1.4.7 Output Layer:

The output layer in a CNN is the final layer of the network that produces the predictions for the input image. The structure and size of the output layer depend on the type of task that the CNN is designed to perform.

For classification tasks, the output layer usually consists of a fully connected layer followed by a softmax activation function, which produces a probability distribution over the possible output classes. The number of neurons in the output layer is equal to the number of possible output classes. During training, the parameters of the output layer (i.e., weights and biases) are learned using backpropagation and gradient descent.

For regression tasks, the output layer typically consists of a single neuron with a linear activation function, which produces a continuous output value. The output layer can also have multiple neurons with different activation functions, depending on the specific regression task.

In some cases, the output layer can also have a combination of fully connected and convolutional layers, especially in architectures designed for tasks such as object detection or semantic segmentation.

Once the output layer produces the final predictions, the loss function is applied to compare these predictions with the true labels of the input data, and the network's parameters are updated using backpropagation and gradient descent to minimize the loss.

In summary, the output layer in a CNN is the final layer that produces the predictions for the input image. Its structure and size depend on the type of task that the CNN is designed to perform, with fully connected layers and softmax activation functions being commonly used for classification tasks, and single or multiple neurons with different activation functions being used for regression tasks.

In summary, CNNs typically consist of several layers, each with a specific function in the network. These layers work together to extract relevant features from the input data and produce the final output of the network.

1.4.8 Rectified Unit Layer (ReLu):

The ReLU (Rectified Linear Unit) layer is an activation function used in CNNs that applies the rectifier function element-wise to the input. The rectifier function returns the input if it is positive and 0 if it is negative.

Mathematically, the rectifier function is defined as follows:

$f(x) = \max(0, x)$

where x is the input to the ReLU layer, and max(0, x) returns the maximum between 0 and x.

The ReLU layer is typically applied after the convolutional or fully connected layer and before the pooling or dropout layer. Its purpose is to introduce non-linearity into the network, allowing the CNN to learn complex features and patterns in the input data.

One advantage of the ReLU layer is that it is computationally efficient, as it involves

only a simple thresholding operation. It also helps to alleviate the vanishing gradient problem that can occur when using activation functions such as the sigmoid or tanh.

However, one potential drawback of the ReLU layer is that it can suffer from the "dying ReLU" problem, in which some neurons can become "dead" and never activate again, leading to a decrease in the network's representational capacity. To address this issue, variations of the ReLU function, such as leaky ReLU and exponential linear units (ELUs), have been proposed.

In summary, the ReLU layer is an activation function used in CNNs that applies the rectifier function element-wise to the input. Its purpose is to introduce non-linearity into the network and allow it to learn complex features and patterns in the input data. The

ReLU layer is computationally efficient but can suffer from the "dying ReLU" problem, which can be addressed by using variations of the ReLU function.

1.4.9 VGG-16 :

VGG-16 is a popular convolutional neural network (CNN) architecture that was developed by the Visual Geometry Group (VGG) at the University of Oxford. It was proposed in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" in 2014 by Simonyan and Zisserman, and has achieved state-of-the-art performance on several image recognition benchmarks.

The VGG-16 network consists of 16 layers, including 13 convolutional layers, 5 max pooling layers, and 3 fully connected layers. The convolutional layers use small 3x3 filters, and are stacked on top of each other to form deeper representations of the input image. The max pooling layers are used to reduce the spatial size of the feature maps, while preserving their depth.

The fully connected layers at the end of the network are used for classification, and include a softmax activation function to produce class probabilities. The architecture of the network is as follows:

Input -> [Conv3-64] x 2 -> MaxPool -> [Conv3-128] x 2 -> MaxPool -> [Conv3-256] x 3 -> MaxPool -> [Conv3-512] x 3 -> MaxPool -> [Conv3-512] x 3 -> MaxPool -> [FC-

4096] -> [FC-4096] -> [FC-1000] -> Softmax -> Output

Here, Conv3-N refers to a convolutional layer with N filters of size 3x3, and FC-N refers to a fully connected layer with N neurons.

The VGG-16 network is known for its simplicity and effectiveness, and has been widely used for various computer vision tasks, such as object recognition and detection. However, its large number of parameters (138 million) makes it computationally expensive and memory-intensive to train and use in practice. To address this issue, various modifications and improvements to the VGG architecture have been proposed, such as VGG-19 and VGG-M.

1.5 Residual Network(ResNet):

ResNet (short for Residual Network) is a family of convolutional neural network

(CNN) architectures that was introduced by Microsoft researchers in 2015. The ResNet architecture was designed to enable the training of very deep neural networks, which had previously been difficult due to the vanishing gradient problem.

The key innovation in ResNet is the use of skip connections, also known as residual connections, which allow information to flow directly from one layer to another without being affected by the intermediate layers. This enables the network to learn both deep and shallow features simultaneously, resulting in improved accuracy in image classification tasks.

The ResNet architecture comes in several variants, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, which differ in the number of layers and complexity. The larger variants of ResNet have achieved state-of-the-art performance in image classification, object detection, and segmentation tasks.

Overall, ResNet has been a significant contribution to the field of computer vision, enabling the development of deep neural networks with improved accuracy on complex tasks.

1.5.1 Resnet Architecture:

ResNet (short for Residual Network) is a deep convolutional neural network architecture that was introduced in 2015 by Kaiming He et al. in their paper "Deep Residual Learning for Image Recognition." It was developed to address the vanishing gradient problem that can occur in very deep neural networks, which can make it difficult to train them effectively.

The key innovation of the ResNet architecture is the use of residual connections, also known as skip connections. These connections allow information to flow directly from one layer to another without being modified, which can help to prevent the vanishing gradient problem and make it easier to train very deep networks.

The ResNet architecture consists of a series of convolutional and pooling layers, followed by a stack of residual blocks. Each residual block contains two or more convolutional layers, followed by a skip connection that bypasses the block and adds the input to the output of the block. This allows the network to learn residual functions, which can make it easier to optimize the network and improve its accuracy.

The original ResNet architecture had various versions, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, which differed in the number of layers and the complexity of the model. The ResNet-50 architecture, for example, has 50 layers and is widely used in many computer vision tasks, such as object detection and segmentation.

Since its introduction, the ResNet architecture has become a popular and influential neural network model, and has been widely adopted and adapted for various computer vision tasks. It has also inspired the development of other deep residual networks, such as ResNeXt and DenseNet, which further extend and improve upon the original ResNet architecture.

1.5.2 ResNet 50:

ResNet-50 is a variant of the Residual Network (ResNet) architecture that was

introduced in 2015 by Kaiming He et al. in their paper "Deep Residual Learning for Image Recognition". It has 50 layers, and is a popular model for various computer vision tasks, such as image classification, object detection, and segmentation.

The ResNet-50 architecture consists of several convolutional layers, followed by a series of residual blocks, and ends with a fully connected layer for classification. The structure of ResNet-50 is as follows:

Input -> Conv1 -> MaxPool -> Res2a -> Res2b -> Res2c -> Res3a -> Res3b -> Res3c -> Res3d -> Res4a -> Res4b -> Res4c -> Res4d -> Res4e -> Res4f -> Res5a -> Res5b -> AvgPool -> FC -> Softmax -> Output

The Conv1 layer is a standard convolutional layer with 64 filters of size 7x7, with stride 2 and padding 3. The MaxPool layer is a max pooling layer with a pool size of 3x3 and stride 2.

The residual blocks consist of multiple convolutional layers with skip connections that allow the network to learn residual functions. In ResNet-50, there are four types of residual blocks, each with a different number of layers:

Res2a, Res2b, and Res2c are three residual blocks with two convolutional layers each, and 64 filters in the first layer and 256 filters in the second layer. Res3a, Res3b, Res3c, and Res3d are four residual blocks with three convolutional layers each, and 128 filters in the first layer, 128 filters in the second layer, and 512 filters in the third layer.

Res4a, Res4b, Res4c, Res4d, and Res4e, and Res4f are six residual blocks with four convolutional layers each, and 256 filters in the first layer, 256 filters in the second layer, 1024 filters in the third layer, and 2048 filters in the fourth layer.

Res5a and Res5b are two residual blocks with three convolutional layers each, and 512 filters in the first and second layers, and 2048 filters in the third layer. The AvgPool layer is a global average pooling layer that averages the output of the final convolutional layer. The FC layer is a fully connected layer with 1000

neurons, which corresponds to the number of classes in the ImageNet dataset. The Softmax layer applies the softmax activation function to produce class probabilities.

Overall, ResNet-50 has achieved state-of-the-art performance on various image classification benchmarks and has become a popular choice for many computer vision tasks.

1.5.3 Why ResNet 50?

ResNet-50 is considered one of the best convolutional neural network (CNN) architectures for image classification tasks for several reasons:

1. Deeper networks tend to perform better than shallower networks, but deep networks are harder to train because of the vanishing gradient problem. ResNet 50 uses skip connections, which allow the gradients to propagate more easily through the network, making it easier to train very deep networks.

2. ResNet-50 has fewer parameters than other state-of-the-art models such as VGG16 and VGG19, which can lead to faster training times and better performance on smaller datasets.

3. The residual connections in ResNet-50 allow it to learn more complex representations of the input image. This means that the network can identify more detailed features and patterns in the image, leading to better accuracy in image

classification tasks.

4. ResNet-50 has a modular structure, with repeated blocks of convolutional layers, which makes it easier to understand and modify. Researchers can modify the architecture by adding or removing blocks, or changing the number of layers within each block, to adapt the model to different datasets or tasks.

5. ResNet-50 has achieved state-of-the-art performance on various computer vision tasks such as image classification, object detection, and segmentation. It has been shown to be effective in transfer learning, where pre-trained models

are fine-tuned on new datasets with good results.

Overall, ResNet-50's combination of skip connections, modularity, and good performance on a wide range of computer vision tasks has made it one of the most popular and successful CNN architectures in recent years.

1.5.4 Key features of ResNet 50 :

ResNet-50 is a specific variant of the ResNet family of convolutional neural network (CNN) architectures. Some key features of ResNet-50 include:

1. Skip connections: ResNet-50 uses skip connections, also known as residual connections, to enable the flow of information through the network. This allows for the creation of deeper networks without suffering from the vanishing gradient problem, where gradients become too small to be useful in training.

2. **Bottleneck architecture**: ResNet-50 uses a bottleneck architecture, which reduces the computational complexity of the network by using 1x1 convolutional layers to reduce the number of input channels before applying the larger 3x3 convolutional layers.

3. Convolutional layers: ResNet-50 contains 50 convolutional layers, which allows it to identify more complex features and patterns in images.

4. Max pooling: Max pooling is used in ResNet-50 to reduce the spatial dimensions of the feature maps, which helps to decrease the computational requirements of the network.

5. Fully connected layers: ResNet-50 ends with a fully connected layer and a softmax activation function, which outputs a probability distribution over the different classes in a classification task.

6. Pre-training: ResNet-50 is often pre-trained on a large dataset such as ImageNet, which allows it to learn general features that can be fine-tuned for specific tasks. This approach, called transfer learning, can improve the accuracy

of the model on smaller datasets or tasks.

Overall, ResNet-50's use of skip connections, bottleneck architecture, and large number of convolutional layers make it a powerful CNN architecture for image classification tasks.

1.5.5 Skip connections:

ResNet50 is a popular deep neural network architecture that has been widely used for image classification tasks. One of the key components of ResNet50 is the skip connection, also known as a residual connection.

A skip connection allows the input of a particular layer to be added to the output of another layer that is deeper in the network. In ResNet50, the skip connections are used to enable the network to learn better by mitigating the problem of vanishing gradients that often occurs in very deep networks.

More specifically, ResNet50 has 49 convolutional layers, and each of these layers is followed by a skip connection that allows the input of the layer to be added to its output. This way, the gradient can flow back through the network more efficiently during backpropagation, which enables better training and faster convergence.

Overall, the skip connections in ResNet50 have been shown to be an effective technique for improving the performance of deep neural networks on a wide range of image classification tasks.

1.5.6 Bottleneck architecture:

The bottleneck architecture is a key component of the ResNet50 architecture. It is a specific type of convolutional layer that is used to reduce the number of parameters in the network while maintaining or improving its performance.

The bottleneck architecture consists of three convolutional layers: a 1x1 convolutional layer, a 3x3 convolutional layer, and another 1x1 convolutional layer. The 1x1 convolutional layers are used to reduce and then increase the

dimensionality of the input, while the 3x3 convolutional layer performs the actual feature extraction.

The bottleneck architecture is used in ResNet50 for two main reasons. First, it reduces the computational cost of the network by reducing the number of parameters required for the convolutional layers. Second, it enables the network to learn more complex features by allowing the 3x3 convolutional layer to have a larger receptive field.

In ResNet50, the bottleneck architecture is used in most of the convolutional layers, except for the first and last layers. This helps to further reduce the number of parameters in the network and enables it to achieve high accuracy on a wide range of image classification tasks.

Overall, the bottleneck architecture is a powerful tool for reducing the computational cost of deep neural networks while maintaining their accuracy, and it has played a key role in the success of the ResNet50 architecture.

1.5.7 Convolutional Layers:

ResNet50 is a deep neural network architecture that is widely used for image classification tasks. It consists of 49 convolutional layers, each with a different number of filters, kernel size, and stride, and each of them is followed by batch normalization and ReLU activation.

The convolutional layers in ResNet50 are used to extract features from the input image in a hierarchical manner. The first convolutional layer takes the input image and applies a set of filters to it, resulting in a set of feature maps. Each subsequent convolutional layer then applies another set of filters to the previous layer's feature maps, resulting in a progressively more abstract representation of the input image.

The ResNet50 architecture also uses skip connections, which allow the gradient to flow more easily through the network during backpropagation. Specifically, each convolutional layer is followed by a residual block, which contains two or

three convolutional layers, depending on the stage of the network. Each residual block has a skip connection that adds the input of the block to its output.

The convolutional layers in ResNet50 use different filter sizes and strides to extract features at different scales and resolutions, which allows the network to learn a wide range of features from the input image. Additionally, the batch normalization and ReLU activation after each convolutional layer help to improve the network's convergence speed and overall performance.

Overall, the convolutional layers in ResNet50 are a key component of the architecture, enabling it to extract hierarchical features from input images and achieve high accuracy on a wide range of image classification tasks.

1.5.8 Max Pooling:

Maxpooling is a commonly used technique in convolutional neural networks (CNNs) for down-sampling the feature maps produced by the convolutional layers. In ResNet50, maxpooling is used in the initial stage of the network to reduce the spatial size of the input image, as well as in the later stages to further down-sample the feature maps.

In the initial stage of ResNet50, a 7x7 convolutional layer with stride 2 is followed by a 3x3 maxpooling layer with stride 2. The purpose of this maxpooling layer is to reduce the spatial size of the input image by a factor of 4, which helps to reduce the computational cost of the network and improve its scalability.

In the later stages of ResNet50, maxpooling is used after certain residual blocks to down-sample the feature maps and increase the receptive field of the network. Specifically, maxpooling with a stride of 2 is used after the 3rd, 4th, and 5th residual blocks in the network. This helps to reduce the spatial size of the feature maps and increase their depth, which allows the network to learn more abstract features and improve its classification accuracy.

Overall, maxpooling is a key technique used in ResNet50 for down-sampling the feature maps and increasing the receptive field of the network, which enables it to achieve high accuracy on a wide range of image classification tasks.

1.5.9 Fully Connected Layers:

ResNet50, like most deep neural networks, typically ends with one or more fully connected layers that transform the output of the last convolutional layer into a final classification decision.

In ResNet50, the fully connected layers are implemented as a global average pooling layer followed by a fully connected layer with a softmax activation function. The global average pooling layer averages the feature maps produced by the final convolutional layer across the spatial dimensions, resulting in a single feature vector for each channel. This feature vector is then passed through the fully connected layer to produce a vector of class scores, which are then normalized using the softmax function to produce the final predicted class probabilities.

The use of global average pooling in ResNet50 is a way to reduce the number of parameters in the network and improve its generalization ability. This is because the global average pooling layer forces the network to learn features that are invariant to the spatial location of the object in the image, which helps to reduce overfitting and improve the network's performance on unseen data.

Overall, the fully connected layers in ResNet50 are a crucial component of the network, as they transform the output of the convolutional layers into a final classification decision, enabling the network to make accurate predictions on a wide range of image classification tasks.

1.5.10 Pre Training:

Pre-training is a commonly used technique in deep learning, where a model is trained on a large dataset, typically ImageNet, before being fine-tuned on a smaller, more specific dataset for a particular task.

In the case of ResNet50, pre-training involves training the model on the large scale ImageNet dataset, which consists of over a million images belonging to

1,000 different classes. This pre-training process allows the model to learn a wide range of features that are useful for a variety of image classification tasks.

After pre-training, the ResNet50 model can be fine-tuned on a smaller dataset for a specific task, such as object detection or semantic segmentation. Fine tuning involves adjusting the weights of the pre-trained model using a smaller dataset, which can lead to better performance on the specific task.

Pre-training on ImageNet is particularly effective for ResNet50, as it enables the model to learn a wide range of features that are useful for many different types of images. This is because ImageNet contains a diverse set of images, including animals, objects, and scenes, which allows the model to learn features that are useful for a wide range of image classification tasks.

Overall, pre-training is a critical component of ResNet50, as it enables the model to learn a wide range of features that are useful for many different types of image classification tasks, and can significantly improve its performance on specific tasks when fine-tuned.

CHAPTER 2

LITERATURE SURVEY

Localization and classification are the two fundamental processes in traffic sign classification. The primary colors on most traffic sign boards are red, white, and black. Lukas Sekanina et al. [8] used image-filtering and template matching to locate the signs. The red, white, and black colors are extracted through image filtering. Template matching locates the sign's position by comparing it to six different template sizes, then neural networks classify the signs. Chen et al. [9] proposed a traffic sign classification model based on color and shape features. It contains 3 steps.

In the first step Image segmentation by HSV color space. Detection of traffic signs based on shapes of the traffic sign board like circle, triangle, etc. Feature Extraction and classification by Gabor filter and SVM. Kedkarn Chaiyakhan et al. [1] proposed Traffic sign Classification using Support vector machines and Image Segmentation. They proposed it in three major steps. At first, image pre-processing using Canny Edge detection is done in order to remove the noise by applying Gaussian Filter.

In the second stage he separated the traffic sign from the background using Hough Transform algorithm which identifies circle, triangle shapes etc. SVM algorithm with different kernels was used for the Classification. The main popularity of color based filtering is due to its less computational time. But when we do this color based filtering as used in [8] it will give less accuracy when some part of image is occluded in real-time. Hough transform algorithm is a very popular approach but it is hard to implement in real time when a particular frame in a video is of HD [10]. Nikonorov et al. [10] proposed an algorithm to overcome these problems by using color shape Regular Expressions which are implemented on DFA and N-DFA. Jonah Sokipriala et al. [4] proposed a modified AlexNet architecture for Traffic Sign Recognition on GTSRB data set and compared those results with original ResNet-50 and VGG-16 architectures.

During the preprocessing stage they converted the colored image into grey scale image so that intensity and computational cost are reduced. Then they used histogram equalization for the uniform distribution of pixel intensities. Then, a modified AlexNet architecture is used for classification in which they used small window sizes as image size is very small (32*32).

Many Traffic Sign Classification algorithms mainly use Convolutional Neural Networks to execute feature extraction and classification. After all the convolution and max pooling operations are performed the layers get flattened which are known as Fully Connected Layers. The fully connected layers then tend to form Classical Neural Networks which is trained by conventional Gradient Descent method, generalization ability is limited. Yujun et al. Proposed "Traffic Sign Recognition Using Deep CNN and Extreme Learning Machine." They removed the fully connected layers from CNN and the feature extractors is fed to the ELM classifier. An ELM classifier is an Single Hidden layer feed forward neural networks.

In this ELM classifier they randomly initialized bias and weights to input layer and they have used various types of activation function for each neuron in the hidden layer. Model proposed by them achieved an accuracy of 99.40% with 12,000 nodes in hidden layer of ELM classifier. Generally, the output layer of the CNN calculates probability for each class, and the one with the highest probability determines the input object's class in the traditional multi class CNN classification approach. Using this N-way classification technique some classes are more misclassified than others. Piyush Kaul et al. [2] proposed" Hierarchical CNN for traffic sign recognition".

They clustered the 43 classes of traffic signs into 6 subsets. They used Similarity matrix for clustering and CNN for partitioning into 6 subsets. Then, they designed separate CNN models for each class. This method has two advantages. First, the hierarchical architecture enables more optimized resource allocation: the size of the network can be tailored to the size of the classification problem. Second, a smaller network with less classes to be identified is easier to learn than a larger network. Model proposed by them achieved an accuracy of 99.67%. Yong Yue et al. [5] proposed a model for Traffic Sign Recognition based on position of max pooling.

According to model proposed by them after all the convolution operations are performed 250 feature maps with 4*4 neurons are extracted. Then they applied max pooling with 2*2 window size is applied and positions of maximum values are encoded into 4-bit binary value. Finally, the whole Max Pooling Positions sequence is obtained by concatenating all the binary values. They proposed the classification in 5 stages: Data Collection, Data Processing, Activation Selection, Classifier Initialization and Classifier Fine Tuning. They, have done the Classification similarity of MPP's belonging to the same class.

2.1 Inferences From Literature Survey:

- Improvement of the number recognition is required and this is a part of the ongoing and future research.
- Currently, most recognition methods did not use original RGB color since its value was easily affected by outdoor lighting condition.
- Classification result using original image get low accuracy because original image consists of noisy background.
- But when we do this colour based filtering as used in it will give less accuracy when some part of image is occluded in real-time.
- Do not encode the position and orientation of object.
- During the pre processing stage they converted the coloured image into gray scale

image so that intensity and computational cost are reduced.

2.2 Open Problems in Existing System:

- During the pre processing stage they converted the coloured image into gray scale image so that intensity and computational cost are reduced.
- Do not encode the position and orientation of object
- But when we do this colour based filtering as used in it will give less accuracy when some part of image is occluded in real-time.
- classification result using original image get low accusy because original image consists of noisy background.
- Currently, most recognition methods did not use original RGB color since its value was easily affected by outdoor lighting condition
- Improvement of the number recognition is required and this is a part of the ongoing and future research.

CHAPTER 3

MODEL PROPOSED

Many traditional Machine Learning algorithms like SVM, K-NN, Random Forest have beenused for many Image Classification tasks and they have achieved a great accuracy. But their performance decreases when the database is very large. This problem is solved Neural Networks, CNN. One of the major problems in classical Neural Networks is number of parameters required to train the model are huge when high quality images are used for classification. Hence, CNN is used for many classification tasks in which Feature Extraction takes place. Dimensionality Reduction is achieved in CNN. CNN is a technique in which a stack of convolution, max-pooling, activation operations takes place.

3.1 Convolution Layer

The convolutional layer is the major building block of the entire CNN model; this convolutional layer does the feature extraction from its input using convolution operations (dot product of the input image with a kernel) by using various filters of different window sizes. In our proposed model we have used six Convolution layers. After the convolution operation RelU activation function is performed. This output will be the input to the next layers of the model.

3.2 Activation Layer

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation functionis to introduce non-linearity into the output of a neuron. In our model ReLU activation function is applied after convolution operation, to prevent the image pixels obtained after the convolution from averaging to zero, therefore all the negative values of the pixels after convolution are converted to zero using the ReLU activation function:

$$F(x) = x \text{ if } x >= 0$$
$$= 0 \text{ else} \quad \text{(EQ:3.1)}$$

3.3 MAX POOL LAYER

Max Pool layer extracts the most relevant features from the window.

3.4 FULLY CONNECTED LAYER

After going through all the convolutional layers, at last the output will be flattened and this layer is known as Fully Connected layer. We can add some dense layers after a fully connected layer. Output layer is generated by applying a soft-max activation function which gives probability. The CNN Architecture used in this report is a combination of several convolutional layers, max pool layers along with Residual block in ResNet Architecture.

3.5 RESIDUAL BLOCK

ResNet is known as Residual Network. It was the first architecture which introduced the concept of skip connection. Residual network is more efficient when compared to other deep neural networks because with resnet there will not be any Vanishing Gradient Problem due to skip connections.

Vanishing Gradient Problem: Since we use back-propagation and chain rule of derivative to calculate weights and also if our model contains many numbers of layers at some point of time gradient may nearly reach to zero(due to multiplication of very small numbers.),hence some layers were not being used efficiently which makes the model difficult to train. Through skip connection we can nullify the Vanishing Gradient problem. In skip connection we add the original input to the output of the convolutional block such that no feature or information of the input is lost.

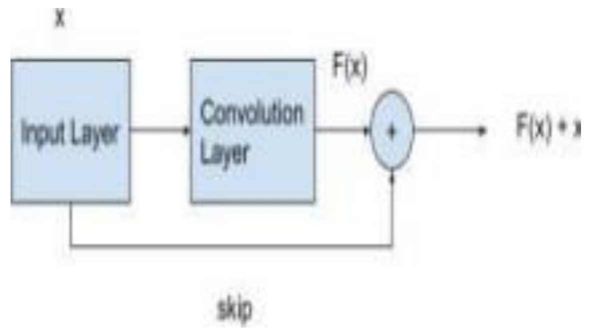


Fig 3.1. Representation of Skip Connection

Let X be the input layer, F(X) be the output after Convolution, Y be the output after addingthe input image to the output of the convoluted layer.

$$Y = F(X) + X \quad \text{(EQ 3.2)}$$

Here we have to make F(X) as zero so that there won't be loss of any features. Since thereis no loss of information of the input, there will be an increase in overall accuracy.

3.6 Architecture Layout

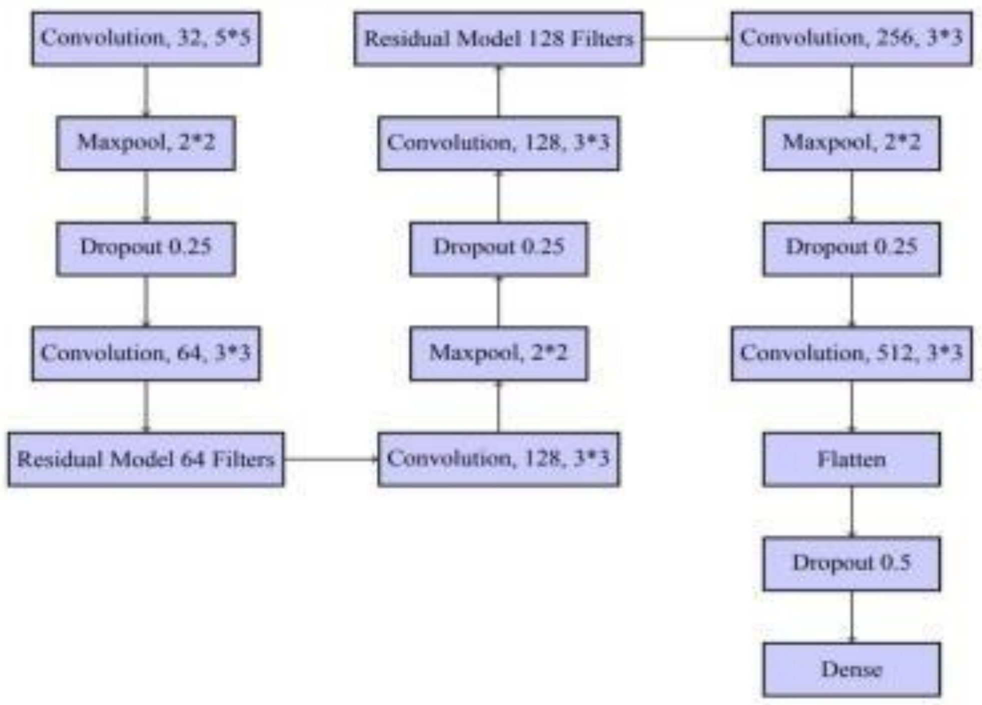


Fig 3.2. Architecture of Model Proposed

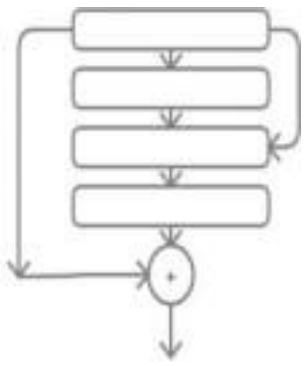


Fig 3.3 Residual Block

CHAPTER 4
EXPERIMENTATION

Fig 4.1: DATA SET DESCRIPTION

For the evaluation of our model Belgium Traffic Signs BTS data set was used which contains traffic signs of 62 different classes of various sizes. Traffic sign images of these database extracted from the environment does not take account of the disturbance occurred due to motion blur and rainy weather. The data set is split into two classes: Train (4575) and Test (2520). Some of the images in the data set are of low quality, low brightness. If we directlytrain this data set directly with our model there will be less accuracy. So before training ourmodel we have to do preprocessing of the data.

Fig 4.1 Different Classes of Belgium Traffic Signs

4.2 FLOW CHART

Input Image

Image Pre Processing

Data set Splitting

Combined CNN and Residual Model

Classification of Output Image

Fig.4.2 Work flow diagram

4.3 PRE-PROCESSING

There won't be any ideal images hence some changes are required before further processing. Pre-processing is done to improve feature extraction. In the BTS data set there are many images with low light and brightness which affects the performance of our model. As CNN is done on images of same size every image is resized to 100*100 size. Since some of the images in the data set are of low brightness indicates that pixel intensities in the histogram of the image are concentrated at one place only, i.e., there is no uniform distribution of pixelintensities. Histogram Equalization is then applied to the training set for contrast stretching to ensure unique distribution of pixel intensities. Normalizing is done by dividing the pixel values with 255.Normalizing is done for better performance of the model in terms of both speed and accuracy. In BTS there are two sets Train and Test. The train set is divided into train and validation sets with different split percentages.

4.4 EXPERIMENTAL SETUP

Proposed system is implemented with Keras library. The experiments are conducted by the implementation of our proposed model (CNN + Residual) on the Belgium Traffic Signs dataset. Adam Optimizer is used to train the network. ReLu activation function is used for eachneuron of CNN. Loss function used is categorical cross entropy.

4.5 DEVELOPMENT ENVIRONMENT SOFTWARE

Operating system: Windows 11

Windows 11 is selected as the developing operating system because Windows has the biggest selection of software available for its platform than any other operating system. The benefit of this is that users get to choose from wider variety of options. This creates healthy "competition" for users, where software developers really have to push boundaries to produce the best program possible. Anything less than the best will result in user's picking the next program on the list. This alone does wonders in motivating software developers to deliver excellent solutions that meet users' needs.

Software used: Python

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

4.6 DEVELOPMENT ENVIRONMENT HARDWARE

4.6.1 Processor

Intel(R) Core (TM) i7is used. Intel Processor provide better processing capabilities and better cooling technology to our CPU. With an Intel processor, we can run our laptop for long time without need to switch off. Besides that, intel processor can help us to boost up the CPU processing power. By using this, we can keep developing the Library Management System without need to worry that the laptop cannot support.

4.6.2 Ram: 16 GB. In order to support Python, we must atleast use 4GB Ram to avoid any problem occurred during development phase. Besides that, Execution of Huge Programs can process faster when running with 16GB ram. It can save a lot of time if total up the process time.

4.7 OPERATION ENVIRONMENT

The table shown below is the minimum requirement:

Table 4.1: Table for operation environment

Processor	x86 64-bit CPU (Intel / AMD Architecture)
Operating System	Microsoft Windows 7 to 10 Mac OS X 10.11 0r higher, 64-bit Linux: RHEL 6/7, 64-bit
Memory	4GB RAM
Screen Resolution	Monitor with screen resolution minimum 1024 x 768

Hard disk Space	Minimum 5GB free disk space

4.8 Results:

The proposed architecture is applied on Belgium traffic sign data set (BTSD). This paper classifies the Belgium Traffic Signs in an efficient manner. Below Table 1. shows the detailed view of our proposed model. It tells about output shape of the tensor after each layer and number of trainable parameters.

Layer Type	Output Shape	Parameters
Sequential	(100,100,3)	0
Conv2d_2	(50,50,32)	2432
max_pooling2d_1	(25,25,32)	0
dropout_1	(25,25,32)	0
Conv2d_3	(25,25,64)	18496
Conv2d_4	(25,25,64)	36928
Conv2d_5	(25,25,64)	36928
Conv2d_6	(25,25,64)	36928
add	(25,25,64)	0
Conv2d_7	(25,25,64)	36928
add_1	(25,25,64)	0
Conv2d_8	(25,25,128)	73856
max_pooling2d_2	(12,12,128)	0
dropout_2	(12,12,128)	0
Conv2d_9	(12,12,128)	147584
Conv2d_10	(12,12,128)	147584
Conv2d_11	(12,12,128)	147584
Conv2d_12	(12,12,128)	147584
add_2	(12,12,128)	0
Conv2d_13	(12,12,128)	147584
add_3	(12,12,128)	0
Conv2d_14	(12,12,256)	295168
max_pooling2d_3	(6,6,256)	0
dropout_3	(6,6,256)	0
Conv2d_15	(6,6,512)	1180160
flatten	18432	0
dropout	18432	0

Table 4.2. Details of Architecture

In order to find the best split our model is experimented on different split ratio like 9:1,8:2, 7:3. Below Table 2. shows their corresponding results.

Train:Validation Split Ratio	**Train Accuracy**	**Validation Accuracy**	**Test Accuracy**
9:1	98.6%	96.70%	93.17%
8:2	99.0%	96.9%	91.1%
7:3	99.19%	95.44%	92.66%

Table 4.3. Accuracy comparison of different Validation splits

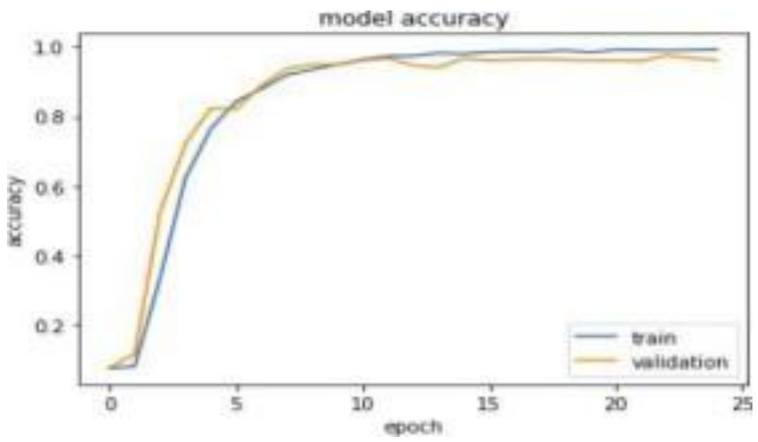


Fig 6. Accuracy vs Epochs for 10% Split

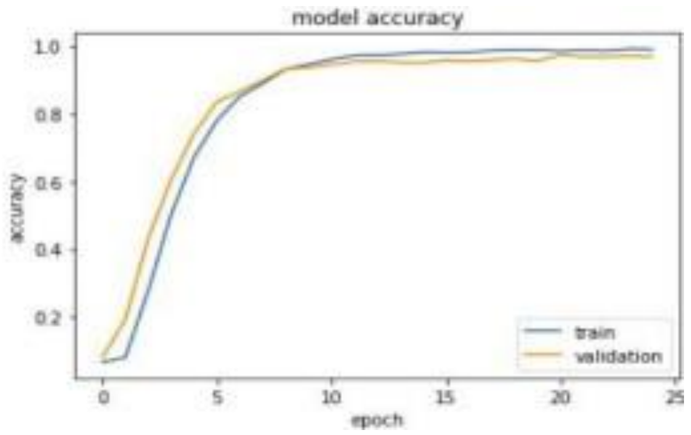


Fig 7. Accuracy vs Epochs for 20% Split

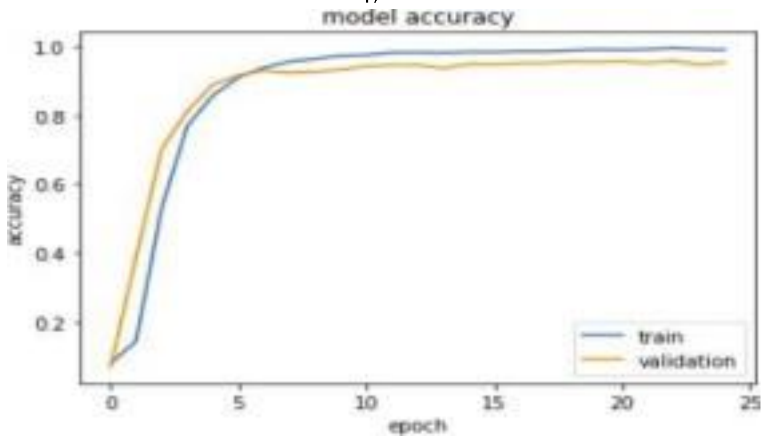


Fig 8. Accuracy vs Epochs for 30% Split

From the table and plots it is quite evident that in every case the training accuracy crosses the validation accuracy which depicts that there is a problem of overfitting. This has occurred due to the small size of data. To overcome that data augmentation is applied(Horizontal Flip, Rotation, Zoom). Data Augmentation increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. Below Table 3. shows results when Data Augmentation is applied.

Train:Validation Split Ratio	**Train Accuracy**	**Validation Accuracy**	**Test Accuracy**
8:2	95.76%	96.81%	94.25%
7:3	95.97%	96.22%	93.53%

Table 4.4. Accuracy comparison of different Validation splits with Data Augmentation

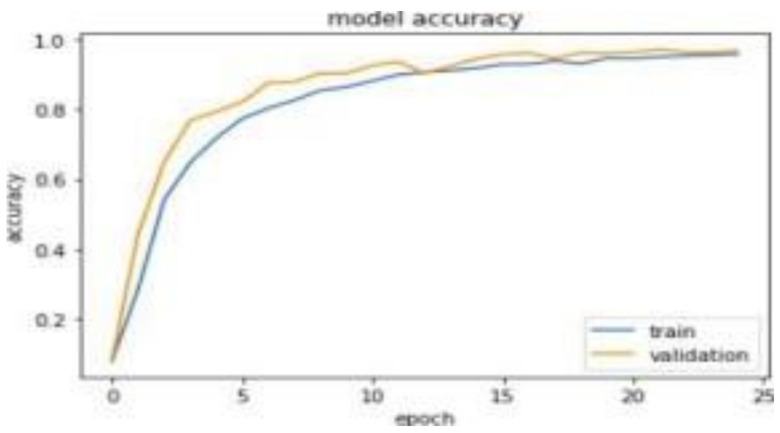


Fig 9. Accuracy vs Epochs for 20% Split with data augmentation

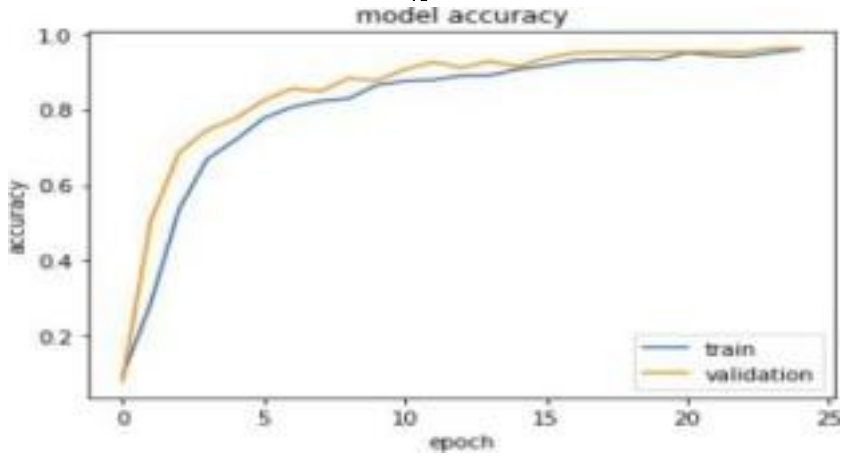


Fig 10. Accuracy vs Epochs for 30% Split with data augmentation

By using Data Augmentation problem of overfitting was resolved and test accuracy was alsoincreased.

Metrics	**Score**
Precision	95.2%
Recall	94.2%
F1 Score	94.4%

Table 4.5. Evaluation metrics calculated for test set

CHAPTER 5

CONCLUSION

5.1 CONCLUSION:

In this paper, we proposed a combination of CNN and Residual model for traffic sign recognition, and demonstrated its performance on the BTS data set. The benefit of the Residual block is we do not have Vanishing Gradient Problem here. We have also tested our model on GTSRB data set and Indian Traffic Signs which gave accuracies of 95.76% and 95% respectively . As part of our future work, we will train our model with different other architectures like LeNet,InceptionNet.

In conclusion, the use of Convolutional Neural Networks (CNN) and Residual Networks (ResNet) has shown great promise in the field of traffic sign recognition. These deep learning techniques have enabled the development of highly accurate and efficient models for recognizing traffic signs, which can be used in a variety of applications such as autonomous driving, traffic monitoring, and road safety.

The use of CNN and ResNet architectures for traffic sign recognition has been extensively studied in recent years, and many research works have demonstrated their effectiveness in achieving high accuracy and robustness in recognizing traffic signs under various conditions. These deep learning techniques have been shown to be effective in handling issues such as occlusion, varying lighting conditions, and different viewpoints.

Overall, the combination of CNN and ResNet architectures with advanced training techniques such as data augmentation and transfer learning has led to significant improvements in traffic sign recognition performance. With further research and development, it is expected that these deep learning techniques will continue to play an important role in improving the safety and efficiency of our roads.

5.2 FUTURE WORK:

There are several areas for future research in traffic sign recognition using CNN and ResNet. Here are some potential directions:

1. Improved data collection: In order to improve the accuracy of traffic sign recognition systems, more diverse and comprehensive datasets need to be

collected. This will help the models to learn from a wider range of traffic sign variations, including different shapes, colors, and backgrounds.

2. More advanced network architectures: Although CNN and ResNet are highly effective architectures, there is still room for improvement. Future research could explore the use of more advanced architectures, such as Capsule Networks, Attention-based Networks, and Transformers, to further enhance the performance of traffic sign recognition systems.

3. Multi-task learning: Traffic sign recognition is often performed in conjunction with other tasks, such as pedestrian detection and lane detection. Future research could explore the use of multi-task learning, where the model is trained to perform multiple tasks simultaneously, to improve the overall efficiency of these systems.

4. Real-time processing: Real-time processing is critical for many traffic-related applications, such as autonomous driving. Future research could focus on developing more efficient and lightweight models that can process traffic sign data in real-time, without sacrificing accuracy.

5. Robustness to adverse conditions: Traffic sign recognition systems need to be robust to adverse conditions, such as adverse weather conditions and low-light environments. Future research could explore the use of adversarial training and other techniques to improve the robustness of these systems.

Overall, there are many exciting opportunities for future research in traffic sign recognition using CNN and ResNet, and continued advancements in this area will have a significant impact on road safety and efficiency.

5.3 RESEARCH ISSSUES:

There are several research issues in traffic sign recognition using CNN and ResNet. Here are some of them:

1. Limited training data: One of the main challenges in training CNN and ResNet models for traffic sign recognition is the limited availability of high-quality training data. Although there are several publicly available datasets, they may not be representative of all possible traffic sign variations.

2. Generalization to new environments: Traffic sign recognition models may not generalize well to new environments, such as different lighting conditions or different camera angles. This can lead to reduced accuracy and reliability in real-world scenarios.

3. Robustness to adversarial attacks: Adversarial attacks can be used to perturb traffic sign images in a way that causes misclassification by CNN and ResNet models. Developing models that are robust to these attacks is an important research issue in traffic sign recognition.

4. Real-time processing: Real-time processing is important for many traffic related applications, but CNN and ResNet models can be computationally expensive, especially when processing high-resolution images. Developing efficient and lightweight models that can process data in real-time is an ongoing research challenge.

5. Interpretable models: CNN and ResNet models are often referred to as "black boxes" because it is difficult to understand how they arrive at their decisions. Developing more interpretable models that can provide insights into how traffic sign recognition works is an important research issue.

6. Large-scale deployment: Deploying traffic sign recognition models in real world scenarios requires overcoming several technical and regulatory challenges, including issues related to privacy, liability, and safety. Addressing these issues is critical for the widespread adoption of these systems.

Overall, traffic sign recognition using CNN and ResNet is an active area of research, and there are several open research issues that need to be addressed to improve the accuracy, reliability, and usability of these systems.

REFERENCES: -

[1] Chaiyakhan, K., Hirunyawanakul, A., Chanklan, R., Kerdprasop, K., and Kerdprasop, N., 2015. “Traffic sign classification using support vector machine and image segmentation.” pp. 52-58.

[2] Mao, X., Hijazi, S. L., Casas, R. A., Kaul, P., Kumar, R., and Rowen, C., 2016. “Hierarchical CNN for traffic sign recognition”. *2016 IEEE Intelligent*

Vehicles Symposium (IV), pp. 130-135.

[3] Zeng, Y., Xu, X., Fang, Y., and Zhao, K., 2015. "Traffic sign recognition using extreme learning classifier with deep convolutional features."

[4] Jonah, S., and Orike, S., 2021. "Traffic sign classification comparison between various convolution neural network models." *International Journal of Scientific and Engineering Research,* **12**, 07, pp. 165-171.

[5] Qian, R., Yue, Y., Coenen, F., and Zhang, B., 2016. "Traffic sign recognition with convolutional neural network based on max pooling positions." In 2016 12th International conference on natural computation, fuzzy systems and knowledge discovery (ICNC-FSKD), IEEE, pp. 578- 582.

[6] Shustanov, A., and Yakimov, P., 2017. "CNN design for real-time traffic sign recognition." *Procedia engineering,* **201**, pp. 718-725.

[7] Theckedath, D., and Sedamkar, R., 2020. "Detecting affect states using vgg16, resnet50 and se-resnet50 networks". *SN Computer Science,* **1**(2), pp. 1-7.

[8] Sekanina, L., and Tørresen, J., 2002. "Detection of Norwegian speed limit signs." In ESM.

[9] Chen, Y., Xie, Y., and Wang, Y., 2013. "Detection and recognition of traffic signs based on hsv vision model and shape features." *Journal of Computers,* **8**, 05.

[10] Nikonorov, A. V., Petrov, M. V., and Yakimov, P. Y., 2016. "Traffic sign detection on gpu using colorshape regular expressions."

APPENDIX

A. Source Code:

```
from keras.models import Sequential,Model
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D,
Flatten,ZeroP
adding2D,Input,BatchNormalization,Activation,Add,AveragePooling2D from
keras.utils import np_utils
from  sklearn.metrics  import  accuracy_score
from tensorflow.keras.preprocessing.image import
ImageDataGenerator import numpy as np
import pandas as pd
import os
import cv2
import seaborn as sns
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import precision_score, recall_score, f1_score,
accu racy_score
from sklearn import metrics
```

```
from google.colab import drive
drive.mount('/content/drive')
```

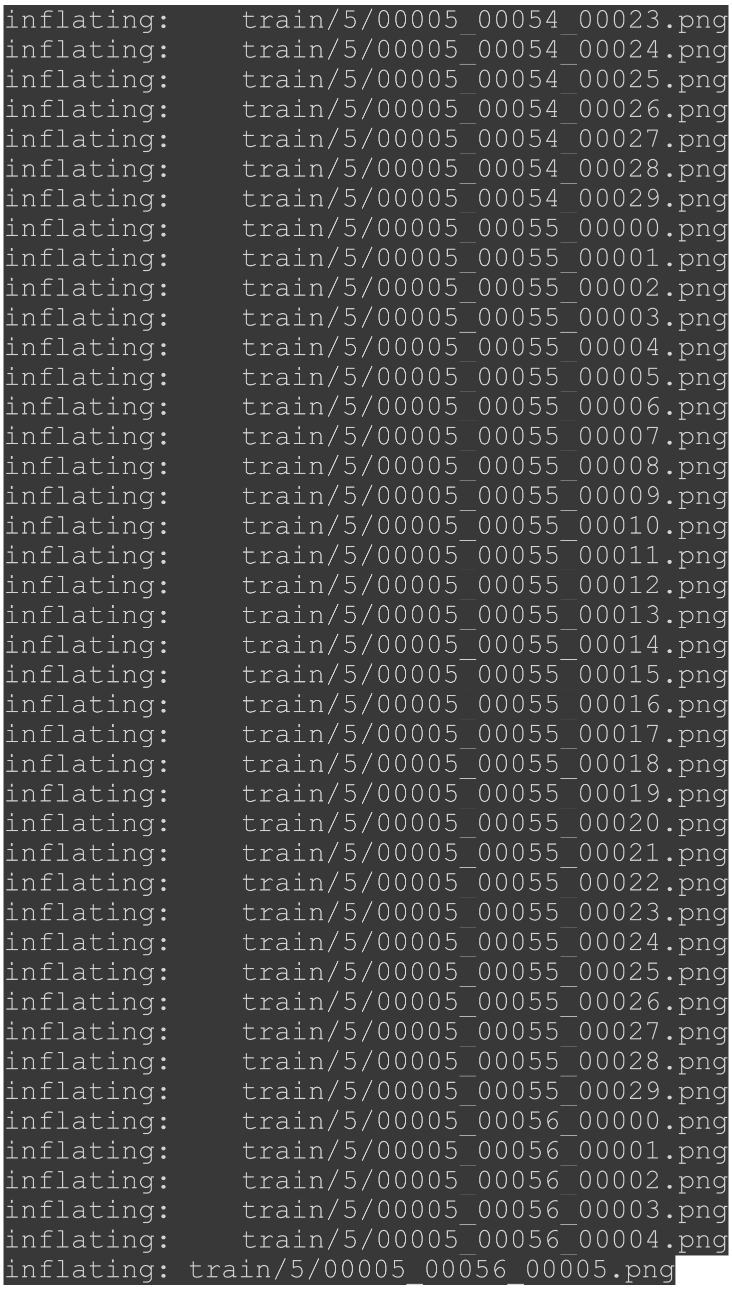
```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

```
!unzip "/content/drive/MyDrive/traffic.zip"
```

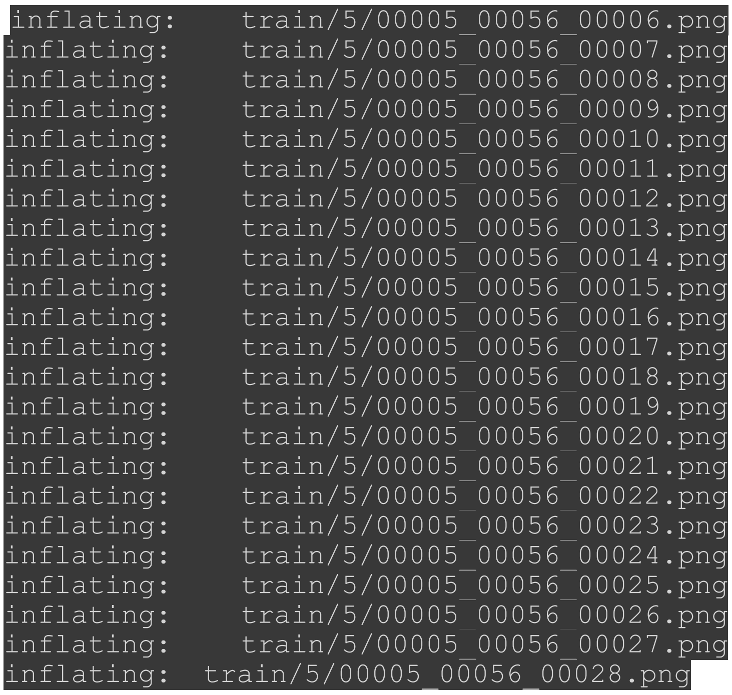
Streaming output truncated to the last 5000 lines.

```
 inflating: train/5/00005_00053_00010.png
 inflating: train/5/00005_00053_00011.png
 inflating: train/5/00005_00053_00012.png
 inflating: train/5/00005_00053_00013.png
 inflating: train/5/00005_00053_00014.png
 inflating: train/5/00005_00053_00015.png
 inflating: train/5/00005_00053_00016.png
 inflating: train/5/00005_00053_00017.png
 inflating: train/5/00005_00053_00018.png
 inflating: train/5/00005_00053_00019.png
 inflating: train/5/00005_00053_00020.png
 inflating: train/5/00005_00053_00021.png
 inflating: train/5/00005_00053_00022.png
 inflating: train/5/00005_00053_00023.png
 inflating: train/5/00005_00053_00024.png
 inflating: train/5/00005_00053_00025.png
 inflating: train/5/00005_00053_00026.png
 inflating: train/5/00005_00053_00027.png
 inflating: train/5/00005_00053_00028.png
 inflating: train/5/00005_00053_00029.png
 inflating: train/5/00005_00054_00000.png
 inflating: train/5/00005_00054_00001.png
 inflating: train/5/00005_00054_00002.png
 inflating: train/5/00005_00054_00003.png
 inflating: train/5/00005_00054_00004.png
```

```
inflating:    train/5/00005_00054_00005.png
inflating:    train/5/00005_00054_00006.png
inflating:    train/5/00005_00054_00007.png
inflating:    train/5/00005_00054_00008.png
inflating:    train/5/00005_00054_00009.png
inflating:    train/5/00005_00054_00010.png
inflating:    train/5/00005_00054_00011.png
inflating:    train/5/00005_00054_00012.png
inflating:    train/5/00005_00054_00013.png
inflating:    train/5/00005_00054_00014.png
inflating:    train/5/00005_00054_00015.png
inflating:    train/5/00005_00054_00016.png
inflating:    train/5/00005_00054_00017.png
inflating:    train/5/00005_00054_00018.png
inflating:    train/5/00005_00054_00019.png
inflating:    train/5/00005_00054_00020.png
inflating:    train/5/00005_00054_00021.png
inflating:  train/5/00005_00054_00022.png
```



```
inflating:    train/5/00005_00054_00023.png
inflating:    train/5/00005_00054_00024.png
inflating:    train/5/00005_00054_00025.png
inflating:    train/5/00005_00054_00026.png
inflating:    train/5/00005_00054_00027.png
inflating:    train/5/00005_00054_00028.png
inflating:    train/5/00005_00054_00029.png
inflating:    train/5/00005_00055_00000.png
inflating:    train/5/00005_00055_00001.png
inflating:    train/5/00005_00055_00002.png
inflating:    train/5/00005_00055_00003.png
inflating:    train/5/00005_00055_00004.png
inflating:    train/5/00005_00055_00005.png
inflating:    train/5/00005_00055_00006.png
inflating:    train/5/00005_00055_00007.png
inflating:    train/5/00005_00055_00008.png
inflating:    train/5/00005_00055_00009.png
inflating:    train/5/00005_00055_00010.png
inflating:    train/5/00005_00055_00011.png
inflating:    train/5/00005_00055_00012.png
inflating:    train/5/00005_00055_00013.png
inflating:    train/5/00005_00055_00014.png
inflating:    train/5/00005_00055_00015.png
inflating:    train/5/00005_00055_00016.png
inflating:    train/5/00005_00055_00017.png
inflating:    train/5/00005_00055_00018.png
inflating:    train/5/00005_00055_00019.png
inflating:    train/5/00005_00055_00020.png
inflating:    train/5/00005_00055_00021.png
inflating:    train/5/00005_00055_00022.png
inflating:    train/5/00005_00055_00023.png
inflating:    train/5/00005_00055_00024.png
inflating:    train/5/00005_00055_00025.png
inflating:    train/5/00005_00055_00026.png
inflating:    train/5/00005_00055_00027.png
inflating:    train/5/00005_00055_00028.png
inflating:    train/5/00005_00055_00029.png
inflating:    train/5/00005_00056_00000.png
inflating:    train/5/00005_00056_00001.png
inflating:    train/5/00005_00056_00002.png
inflating:    train/5/00005_00056_00003.png
inflating:    train/5/00005_00056_00004.png
inflating: train/5/00005_00056_00005.png
```



```
inflating:    train/5/00005_00056_00006.png
inflating:    train/5/00005_00056_00007.png
inflating:    train/5/00005_00056_00008.png
inflating:    train/5/00005_00056_00009.png
inflating:    train/5/00005_00056_00010.png
inflating:    train/5/00005_00056_00011.png
inflating:    train/5/00005_00056_00012.png
inflating:    train/5/00005_00056_00013.png
inflating:    train/5/00005_00056_00014.png
inflating:    train/5/00005_00056_00015.png
inflating:    train/5/00005_00056_00016.png
inflating:    train/5/00005_00056_00017.png
inflating:    train/5/00005_00056_00018.png
inflating:    train/5/00005_00056_00019.png
inflating:    train/5/00005_00056_00020.png
inflating:    train/5/00005_00056_00021.png
inflating:    train/5/00005_00056_00022.png
inflating:    train/5/00005_00056_00023.png
inflating:    train/5/00005_00056_00024.png
inflating:    train/5/00005_00056_00025.png
inflating:    train/5/00005_00056_00026.png
inflating:    train/5/00005_00056_00027.png
inflating:  train/5/00005_00056_00028.png
```

```
inflating:    train/5/00005_00056_00029.png
inflating:    train/5/00005_00057_00000.png
inflating:    train/5/00005_00057_00001.png
inflating:    train/5/00005_00057_00002.png
inflating:    train/5/00005_00057_00003.png
inflating:    train/5/00005_00057_00004.png
inflating:    train/5/00005_00057_00005.png
inflating:    train/5/00005_00057_00006.png
inflating:    train/5/00005_00057_00007.png
inflating:    train/5/00005_00057_00008.png
inflating:    train/5/00005_00057_00009.png
inflating:    train/5/00005_00057_00010.png
inflating:    train/5/00005_00057_00011.png
inflating:    train/5/00005_00057_00012.png
inflating:    train/5/00005_00057_00013.png
inflating:    train/5/00005_00057_00014.png
inflating:    train/5/00005_00057_00015.png
inflating:    train/5/00005_00057_00016.png
inflating:    train/5/00005_00057_00017.png
inflating:    train/5/00005_00057_00018.png
inflating:    train/5/00005_00057_00019.png
inflating:    train/5/00005_00057_00020.png
inflating:    train/5/00005_00057_00021.png
inflating:    train/5/00005_00057_00022.png
inflating:    train/5/00005_00057_00023.png
inflating:    train/5/00005_00057_00024.png
inflating:    train/5/00005_00057_00025.png
inflating:    train/5/00005_00057_00026.png
inflating:    train/5/00005_00057_00027.png
inflating:    train/5/00005_00057_00028.png
inflating:    train/5/00005_00057_00029.png
inflating:    train/5/00005_00058_00000.png
inflating:    train/5/00005_00058_00001.png
inflating:    train/5/00005_00058_00002.png
inflating:    train/5/00005_00058_00003.png
inflating:    train/5/00005_00058_00004.png
inflating:    train/5/00005_00058_00005.png
inflating: train/5/00005_00058_00006.png
```

```
inflating:    train/5/00005_00058_00007.png
inflating:    train/5/00005_00058_00008.png
inflating:    train/5/00005_00058_00009.png
inflating:    train/5/00005_00058_00010.png
inflating:    train/5/00005_00058_00011.png
inflating:    train/5/00005_00058_00012.png
inflating:    train/5/00005_00058_00013.png
inflating:    train/5/00005_00058_00014.png
inflating:    train/5/00005_00058_00015.png
inflating:    train/5/00005_00058_00016.png
inflating:    train/5/00005_00058_00017.png
inflating:    train/5/00005_00058_00018.png
inflating:    train/5/00005_00058_00019.png
inflating:    train/5/00005_00058_00020.png
inflating:    train/5/00005_00058_00021.png
inflating:    train/5/00005_00058_00022.png
inflating:    train/5/00005_00058_00023.png
inflating:    train/5/00005_00058_00024.png
inflating:    train/5/00005_00058_00025.png
inflating:    train/5/00005_00058_00026.png
inflating:    train/5/00005_00058_00027.png
inflating:    train/5/00005_00058_00028.png
inflating:    train/5/00005_00058_00029.png
inflating:    train/5/00005_00059_00000.png
inflating:    train/5/00005_00059_00001.png
inflating:    train/5/00005_00059_00002.png
inflating:    train/5/00005_00059_00003.png
inflating:  train/5/00005_00059_00004.png
```

```
inflating:    train/5/00005_00059_00005.png
inflating:    train/5/00005_00059_00006.png
inflating:    train/5/00005_00059_00007.png
inflating:    train/5/00005_00059_00008.png
inflating:    train/5/00005_00059_00009.png
inflating:    train/5/00005_00059_00010.png
inflating:    train/5/00005_00059_00011.png
inflating:    train/5/00005_00059_00012.png
inflating:    train/5/00005_00059_00013.png
inflating:    train/5/00005_00059_00014.png
inflating:    train/5/00005_00059_00015.png
inflating:    train/5/00005_00059_00016.png
inflating:    train/5/00005_00059_00017.png
inflating:    train/5/00005_00059_00018.png
inflating:    train/5/00005_00059_00019.png
inflating:    train/5/00005_00059_00020.png
inflating:    train/5/00005_00059_00021.png
inflating:    train/5/00005_00059_00022.png
inflating:    train/5/00005_00059_00023.png
inflating:    train/5/00005_00059_00024.png
inflating:    train/5/00005_00059_00025.png
inflating:    train/5/00005_00059_00026.png
inflating:    train/5/00005_00059_00027.png
inflating:    train/5/00005_00059_00028.png
inflating:    train/5/00005_00059_00029.png
inflating:    train/5/00005_00060_00000.png
inflating:    train/5/00005_00060_00001.png
inflating:    train/5/00005_00060_00002.png
inflating:    train/5/00005_00060_00003.png
inflating:    train/5/00005_00060_00004.png
inflating:    train/5/00005_00060_00005.png
inflating:    train/5/00005_00060_00006.png
inflating: train/5/00005_00060_00007.png
```

```
 inflating:   train/5/00005_00060_00008.png
inflating:    train/5/00005_00060_00009.png
inflating:    train/5/00005_00060_00010.png
inflating:    train/5/00005_00060_00011.png
inflating:    train/5/00005_00060_00012.png
inflating:    train/5/00005_00060_00013.png
inflating:    train/5/00005_00060_00014.png
inflating:    train/5/00005_00060_00015.png
inflating:    train/5/00005_00060_00016.png
inflating:    train/5/00005_00060_00017.png
inflating:    train/5/00005_00060_00018.png
inflating:    train/5/00005_00060_00019.png
inflating:    train/5/00005_00060_00020.png
inflating:    train/5/00005_00060_00021.png
inflating:    train/5/00005_00060_00022.png
inflating:    train/5/00005_00060_00023.png
inflating:    train/5/00005_00060_00024.png
inflating:    train/5/00005_00060_00025.png
inflating:    train/5/00005_00060_00026.png
inflating:    train/5/00005_00060_00027.png
inflating:    train/5/00005_00060_00028.png
inflating:    train/5/00005_00060_00029.png
inflating:    train/5/00005_00061_00000.png
inflating:    train/5/00005_00061_00001.png
inflating:    train/5/00005_00061_00002.png
inflating:    train/5/00005_00061_00003.png
inflating:    train/5/00005_00061_00004.png
inflating:    train/5/00005_00061_00005.png
inflating:    train/5/00005_00061_00006.png
inflating:    train/5/00005_00061_00007.png
inflating:    train/5/00005_00061_00008.png
inflating:    train/5/00005_00061_00009.png
inflating:  train/5/00005_00061_00010.png
```

```
inflating:    train/5/00005_00061_00011.png
inflating:    train/5/00005_00061_00012.png
inflating:    train/5/00005_00061_00013.png
inflating:    train/5/00005_00061_00014.png
inflating:    train/5/00005_00061_00015.png
inflating:    train/5/00005_00061_00016.png
inflating:    train/5/00005_00061_00017.png
inflating:    train/5/00005_00061_00018.png
inflating:    train/5/00005_00061_00019.png
inflating:    train/5/00005_00061_00020.png
inflating:    train/5/00005_00061_00021.png
inflating:    train/5/00005_00061_00022.png
inflating:    train/5/00005_00061_00023.png
inflating:    train/5/00005_00061_00024.png
inflating:    train/5/00005_00061_00025.png
inflating:    train/5/00005_00061_00026.png
inflating:    train/5/00005_00061_00027.png
inflating:    train/5/00005_00061_00028.png
inflating:    train/5/00005_00061_00029.png
inflating:    train/6/00006_00000_00000.png
inflating:    train/6/00006_00000_00001.png
inflating:    train/6/00006_00000_00002.png
inflating:    train/6/00006_00000_00003.png
inflating:    train/6/00006_00000_00004.png
inflating:    train/6/00006_00000_00005.png
inflating:    train/6/00006_00000_00006.png
inflating:    train/6/00006_00000_00007.png
inflating: train/6/00006_00000_00008.png
```

```
inflating:    train/6/00006_00000_00009.png
inflating:    train/6/00006_00000_00010.png
inflating:    train/6/00006_00000_00011.png
inflating:    train/6/00006_00000_00012.png
inflating:    train/6/00006_00000_00013.png
inflating:    train/6/00006_00000_00014.png
inflating:    train/6/00006_00000_00015.png
inflating:    train/6/00006_00000_00016.png
inflating:    train/6/00006_00000_00017.png
inflating:    train/6/00006_00000_00018.png
inflating:    train/6/00006_00000_00019.png
inflating:    train/6/00006_00000_00020.png
inflating:    train/6/00006_00000_00021.png
inflating:    train/6/00006_00000_00022.png
inflating:    train/6/00006_00000_00023.png
inflating:    train/6/00006_00000_00024.png
inflating:    train/6/00006_00000_00025.png
inflating:    train/6/00006_00000_00026.png
inflating:    train/6/00006_00000_00027.png
inflating:    train/6/00006_00000_00028.png
inflating:    train/6/00006_00000_00029.png
inflating:    train/6/00006_00001_00000.png
inflating:    train/6/00006_00001_00001.png
inflating:    train/6/00006_00001_00002.png
inflating:    train/6/00006_00001_00003.png
inflating:    train/6/00006_00001_00004.png
inflating:    train/6/00006_00001_00005.png
inflating:    train/6/00006_00001_00006.png
inflating:    train/6/00006_00001_00007.png
inflating:    train/6/00006_00001_00008.png
inflating:    train/6/00006_00001_00009.png
inflating:    train/6/00006_00001_00010.png
inflating:    train/6/00006_00001_00011.png
inflating:    train/6/00006_00001_00012.png
inflating:    train/6/00006_00001_00013.png
inflating:    train/6/00006_00001_00014.png
inflating:    train/6/00006_00001_00015.png
inflating:  train/6/00006_00001_00016.png
```

```
inflating:    train/6/00006_00001_00017.png
inflating:    train/6/00006_00001_00018.png
inflating:    train/6/00006_00001_00019.png
inflating:    train/6/00006_00001_00020.png
inflating:    train/6/00006_00001_00021.png
inflating:    train/6/00006_00001_00022.png
inflating:    train/6/00006_00001_00023.png
inflating:    train/6/00006_00001_00024.png
inflating:    train/6/00006_00001_00025.png
inflating:    train/6/00006_00001_00026.png
inflating:    train/6/00006_00001_00027.png
inflating:    train/6/00006_00001_00028.png
inflating:    train/6/00006_00001_00029.png
inflating:    train/6/00006_00002_00000.png
inflating:    train/6/00006_00002_00001.png
inflating:    train/6/00006_00002_00002.png
inflating:    train/6/00006_00002_00003.png
inflating:    train/6/00006_00002_00004.png
inflating:    train/6/00006_00002_00005.png
inflating:    train/6/00006_00002_00006.png
inflating:    train/6/00006_00002_00007.png
inflating:    train/6/00006_00002_00008.png
inflating: train/6/00006_00002_00009.png
```

```
inflating:    train/6/00006_00002_00010.png
inflating:    train/6/00006_00002_00011.png
inflating:    train/6/00006_00002_00012.png
inflating:    train/6/00006_00002_00013.png
inflating:    train/6/00006_00002_00014.png
inflating:    train/6/00006_00002_00015.png
inflating:    train/6/00006_00002_00016.png
inflating:    train/6/00006_00002_00017.png
inflating:    train/6/00006_00002_00018.png
inflating:    train/6/00006_00002_00019.png
inflating:    train/6/00006_00002_00020.png
inflating:    train/6/00006_00002_00021.png
inflating:    train/6/00006_00002_00022.png
inflating:    train/6/00006_00002_00023.png
inflating:    train/6/00006_00002_00024.png
inflating:    train/6/00006_00002_00025.png
inflating:    train/6/00006_00002_00026.png
inflating:    train/6/00006_00002_00027.png
inflating:    train/6/00006_00002_00028.png
inflating:    train/6/00006_00002_00029.png
inflating:    train/6/00006_00003_00000.png
inflating:    train/6/00006_00003_00001.png
inflating:    train/6/00006_00003_00002.png
inflating:    train/6/00006_00003_00003.png
inflating:    train/6/00006_00003_00004.png
inflating:    train/6/00006_00003_00005.png
inflating:    train/6/00006_00003_00006.png
inflating:    train/6/00006_00003_00007.png
inflating:    train/6/00006_00003_00008.png
inflating:    train/6/00006_00003_00009.png
inflating:    train/6/00006_00003_00010.png
inflating:    train/6/00006_00003_00011.png
inflating:    train/6/00006_00003_00012.png
inflating:    train/6/00006_00003_00013.png
inflating:    train/6/00006_00003_00014.png
inflating:    train/6/00006_00003_00015.png
inflating:    train/6/00006_00003_00016.png
inflating:    train/6/00006_00003_00017.png
inflating:    train/6/00006_00003_00018.png
inflating:    train/6/00006_00003_00019.png
inflating:    train/6/00006_00003_00020.png
inflating:    train/6/00006_00003_00021.png
inflating:  train/6/00006_00003_00022.png
```

```
inflating:    train/6/00006_00003_00023.png
inflating:    train/6/00006_00003_00024.png
inflating:    train/6/00006_00003_00025.png
inflating:    train/6/00006_00003_00026.png
inflating:    train/6/00006_00003_00027.png
inflating:    train/6/00006_00003_00028.png
inflating:    train/6/00006_00003_00029.png
inflating:    train/6/00006_00004_00000.png
inflating:    train/6/00006_00004_00001.png
inflating:    train/6/00006_00004_00002.png
inflating:    train/6/00006_00004_00003.png
inflating:    train/6/00006_00004_00004.png
inflating:    train/6/00006_00004_00005.png
inflating:    train/6/00006_00004_00006.png
inflating:    train/6/00006_00004_00007.png
inflating:    train/6/00006_00004_00008.png
inflating:    train/6/00006_00004_00009.png
inflating: train/6/00006_00004_00010.png
```

```
inflating:    train/6/00006_00004_00011.png
inflating:    train/6/00006_00004_00012.png
inflating:    train/6/00006_00004_00013.png
inflating:    train/6/00006_00004_00014.png
inflating:    train/6/00006_00004_00015.png
inflating:    train/6/00006_00004_00016.png
inflating:    train/6/00006_00004_00017.png
inflating:    train/6/00006_00004_00018.png
inflating:    train/6/00006_00004_00019.png
inflating:    train/6/00006_00004_00020.png
inflating:    train/6/00006_00004_00021.png
inflating:    train/6/00006_00004_00022.png
inflating:    train/6/00006_00004_00023.png
inflating:    train/6/00006_00004_00024.png
inflating:    train/6/00006_00004_00025.png
inflating:    train/6/00006_00004_00026.png
inflating:    train/6/00006_00004_00027.png
inflating:    train/6/00006_00004_00028.png
inflating:    train/6/00006_00004_00029.png
inflating:    train/6/00006_00005_00000.png
inflating:    train/6/00006_00005_00001.png
inflating:    train/6/00006_00005_00002.png
inflating:    train/6/00006_00005_00003.png
inflating:    train/6/00006_00005_00004.png
inflating:    train/6/00006_00005_00005.png
inflating:    train/6/00006_00005_00006.png
inflating:    train/6/00006_00005_00007.png
inflating:    train/6/00006_00005_00008.png
inflating:    train/6/00006_00005_00009.png
inflating:    train/6/00006_00005_00010.png
inflating:    train/6/00006_00005_00011.png
inflating:    train/6/00006_00005_00012.png
inflating:    train/6/00006_00005_00013.png
inflating:    train/6/00006_00005_00014.png
inflating:    train/6/00006_00005_00015.png
inflating:    train/6/00006_00005_00016.png
inflating:    train/6/00006_00005_00017.png
inflating:    train/6/00006_00005_00018.png
inflating:    train/6/00006_00005_00019.png
inflating:    train/6/00006_00005_00020.png
inflating:    train/6/00006_00005_00021.png
inflating:    train/6/00006_00005_00022.png
inflating:    train/6/00006_00005_00023.png
inflating:    train/6/00006_00005_00024.png
inflating:    train/6/00006_00005_00025.png
inflating:    train/6/00006_00005_00026.png
inflating:    train/6/00006_00005_00027.png
inflating:  train/6/00006_00005_00028.png
```

```
inflating:    train/6/00006_00005_00029.png
inflating:    train/6/00006_00006_00000.png
inflating:    train/6/00006_00006_00001.png
inflating:    train/6/00006_00006_00002.png
inflating:    train/6/00006_00006_00003.png
inflating:    train/6/00006_00006_00004.png
inflating:    train/6/00006_00006_00005.png
inflating:    train/6/00006_00006_00006.png
inflating:    train/6/00006_00006_00007.png
inflating:    train/6/00006_00006_00008.png
inflating:    train/6/00006_00006_00009.png
inflating:    train/6/00006_00006_00010.png
inflating: train/6/00006_00006_00011.png
```

```
 inflating:    train/6/00006_00006_00012.png
inflating:    train/6/00006_00006_00013.png
inflating:    train/6/00006_00006_00014.png
inflating:    train/6/00006_00006_00015.png
inflating:    train/6/00006_00006_00016.png
inflating:    train/6/00006_00006_00017.png
inflating:    train/6/00006_00006_00018.png
inflating:    train/6/00006_00006_00019.png
inflating:    train/6/00006_00006_00020.png
inflating:    train/6/00006_00006_00021.png
inflating:    train/6/00006_00006_00022.png
inflating:    train/6/00006_00006_00023.png
inflating:    train/6/00006_00006_00024.png
inflating:    train/6/00006_00006_00025.png
inflating:    train/6/00006_00006_00026.png
inflating:    train/6/00006_00006_00027.png
inflating:    train/6/00006_00006_00028.png
inflating:    train/6/00006_00006_00029.png
inflating:    train/6/00006_00007_00000.png
inflating:    train/6/00006_00007_00001.png
inflating:    train/6/00006_00007_00002.png
inflating:    train/6/00006_00007_00003.png
inflating:    train/6/00006_00007_00004.png
inflating:    train/6/00006_00007_00005.png
inflating:    train/6/00006_00007_00006.png
inflating:    train/6/00006_00007_00007.png
inflating:    train/6/00006_00007_00008.png
inflating:    train/6/00006_00007_00009.png
inflating:    train/6/00006_00007_00010.png
inflating:    train/6/00006_00007_00011.png
inflating:    train/6/00006_00007_00012.png
inflating:    train/6/00006_00007_00013.png
inflating:    train/6/00006_00007_00014.png
inflating:    train/6/00006_00007_00015.png
inflating:    train/6/00006_00007_00016.png
inflating:    train/6/00006_00007_00017.png
inflating:    train/6/00006_00007_00018.png
inflating:    train/6/00006_00007_00019.png
inflating:    train/6/00006_00007_00020.png
inflating:    train/6/00006_00007_00021.png
inflating:    train/6/00006_00007_00022.png
inflating:    train/6/00006_00007_00023.png
inflating:    train/6/00006_00007_00024.png
inflating:    train/6/00006_00007_00025.png
inflating:    train/6/00006_00007_00026.png
inflating:    train/6/00006_00007_00027.png
inflating:    train/6/00006_00007_00028.png
inflating:    train/6/00006_00007_00029.png
inflating:    train/6/00006_00008_00000.png
inflating:    train/6/00006_00008_00001.png
inflating:    train/6/00006_00008_00002.png
inflating:    train/6/00006_00008_00003.png
inflating:  train/6/00006_00008_00004.png
```

```
inflating:    train/6/00006_00008_00005.png
inflating:    train/6/00006_00008_00006.png
inflating:    train/6/00006_00008_00007.png
inflating:    train/6/00006_00008_00008.png
inflating:    train/6/00006_00008_00009.png
inflating:    train/6/00006_00008_00010.png
inflating:    train/6/00006_00008_00011.png
inflating: train/6/00006_00008_00012.png
```

```
inflating:    train/6/00006_00008_00013.png
inflating:    train/6/00006_00008_00014.png
inflating:    train/6/00006_00008_00015.png
inflating:    train/6/00006_00008_00016.png
inflating:    train/6/00006_00008_00017.png
inflating:    train/6/00006_00008_00018.png
inflating:    train/6/00006_00008_00019.png
inflating:    train/6/00006_00008_00020.png
inflating:    train/6/00006_00008_00021.png
inflating:    train/6/00006_00008_00022.png
inflating:    train/6/00006_00008_00023.png
inflating:    train/6/00006_00008_00024.png
inflating:    train/6/00006_00008_00025.png
inflating:    train/6/00006_00008_00026.png
inflating:    train/6/00006_00008_00027.png
inflating:    train/6/00006_00008_00028.png
inflating:    train/6/00006_00008_00029.png
inflating:    train/6/00006_00009_00000.png
inflating:    train/6/00006_00009_00001.png
inflating:    train/6/00006_00009_00002.png
inflating:    train/6/00006_00009_00003.png
inflating:    train/6/00006_00009_00004.png
inflating:    train/6/00006_00009_00005.png
inflating:    train/6/00006_00009_00006.png
inflating:    train/6/00006_00009_00007.png
inflating:    train/6/00006_00009_00008.png
inflating:    train/6/00006_00009_00009.png
inflating:    train/6/00006_00009_00010.png
inflating:    train/6/00006_00009_00011.png
inflating:    train/6/00006_00009_00012.png
inflating:    train/6/00006_00009_00013.png
inflating:    train/6/00006_00009_00014.png
inflating:    train/6/00006_00009_00015.png
inflating:    train/6/00006_00009_00016.png
inflating:    train/6/00006_00009_00017.png
inflating:    train/6/00006_00009_00018.png
inflating:    train/6/00006_00009_00019.png
inflating:    train/6/00006_00009_00020.png
inflating:    train/6/00006_00009_00021.png
inflating:    train/6/00006_00009_00022.png
inflating:    train/6/00006_00009_00023.png
inflating:    train/6/00006_00009_00024.png
inflating:    train/6/00006_00009_00025.png
inflating:    train/6/00006_00009_00026.png
inflating:    train/6/00006_00009_00027.png
inflating:    train/6/00006_00009_00028.png
inflating:    train/6/00006_00009_00029.png
inflating:    train/6/00006_00010_00000.png
inflating:    train/6/00006_00010_00001.png
inflating:    train/6/00006_00010_00002.png
inflating:    train/6/00006_00010_00003.png
inflating:    train/6/00006_00010_00004.png
inflating:    train/6/00006_00010_00005.png
inflating:    train/6/00006_00010_00006.png
inflating:    train/6/00006_00010_00007.png
inflating:    train/6/00006_00010_00008.png
inflating:    train/6/00006_00010_00009.png
inflating:  train/6/00006_00010_00010.png
```

```
inflating:    train/6/00006_00010_00011.png
inflating:    train/6/00006_00010_00012.png
inflating: train/6/00006_00010_00013.png
```

```
inflating:    train/6/00006_00010_00014.png
inflating:    train/6/00006_00010_00015.png
inflating:    train/6/00006_00010_00016.png
inflating:    train/6/00006_00010_00017.png
inflating:    train/6/00006_00010_00018.png
inflating:    train/6/00006_00010_00019.png
inflating:    train/6/00006_00010_00020.png
inflating:    train/6/00006_00010_00021.png
inflating:    train/6/00006_00010_00022.png
inflating:    train/6/00006_00010_00023.png
inflating:    train/6/00006_00010_00024.png
inflating:    train/6/00006_00010_00025.png
inflating:    train/6/00006_00010_00026.png
inflating:    train/6/00006_00010_00027.png
inflating:    train/6/00006_00010_00028.png
inflating:    train/6/00006_00010_00029.png
inflating:    train/6/00006_00011_00000.png
inflating:    train/6/00006_00011_00001.png
inflating:    train/6/00006_00011_00002.png
inflating:    train/6/00006_00011_00003.png
inflating:    train/6/00006_00011_00004.png
inflating:    train/6/00006_00011_00005.png
inflating:    train/6/00006_00011_00006.png
inflating:    train/6/00006_00011_00007.png
inflating:    train/6/00006_00011_00008.png
inflating:    train/6/00006_00011_00009.png
inflating:    train/6/00006_00011_00010.png
inflating:    train/6/00006_00011_00011.png
inflating:    train/6/00006_00011_00012.png
inflating:    train/6/00006_00011_00013.png
inflating:    train/6/00006_00011_00014.png
inflating:    train/6/00006_00011_00015.png
inflating:    train/6/00006_00011_00016.png
inflating:    train/6/00006_00011_00017.png
inflating:    train/6/00006_00011_00018.png
inflating:    train/6/00006_00011_00019.png
inflating:    train/6/00006_00011_00020.png
inflating:    train/6/00006_00011_00021.png
inflating:    train/6/00006_00011_00022.png
inflating:    train/6/00006_00011_00023.png
inflating:    train/6/00006_00011_00024.png
inflating:    train/6/00006_00011_00025.png
inflating:    train/6/00006_00011_00026.png
inflating:    train/6/00006_00011_00027.png
inflating:    train/6/00006_00011_00028.png
inflating:    train/6/00006_00011_00029.png
inflating:    train/6/00006_00012_00000.png
inflating:    train/6/00006_00012_00001.png
inflating:    train/6/00006_00012_00002.png
inflating:    train/6/00006_00012_00003.png
inflating:    train/6/00006_00012_00004.png
inflating:    train/6/00006_00012_00005.png
inflating:    train/6/00006_00012_00006.png
inflating:    train/6/00006_00012_00007.png
inflating:    train/6/00006_00012_00008.png
inflating:    train/6/00006_00012_00009.png
inflating:    train/6/00006_00012_00010.png
inflating:    train/6/00006_00012_00011.png
inflating:    train/6/00006_00012_00012.png
inflating:    train/6/00006_00012_00013.png
inflating: train/6/00006_00012_00014.png
```

```
 inflating:    train/6/00006_00012_00015.png
inflating:    train/6/00006_00012_00016.png
inflating:    train/6/00006_00012_00017.png
inflating:    train/6/00006_00012_00018.png
inflating:    train/6/00006_00012_00019.png
inflating:    train/6/00006_00012_00020.png
inflating:    train/6/00006_00012_00021.png
inflating:    train/6/00006_00012_00022.png
inflating:    train/6/00006_00012_00023.png
inflating:    train/6/00006_00012_00024.png
inflating:    train/6/00006_00012_00025.png
inflating:    train/6/00006_00012_00026.png
inflating:    train/6/00006_00012_00027.png
inflating:    train/6/00006_00012_00028.png
inflating:    train/6/00006_00012_00029.png
inflating:    train/6/00006_00013_00000.png
inflating:    train/6/00006_00013_00001.png
inflating:    train/6/00006_00013_00002.png
inflating:    train/6/00006_00013_00003.png
inflating:    train/6/00006_00013_00004.png
inflating:    train/6/00006_00013_00005.png
inflating:    train/6/00006_00013_00006.png
inflating:    train/6/00006_00013_00007.png
inflating:    train/6/00006_00013_00008.png
inflating:    train/6/00006_00013_00009.png
inflating:    train/6/00006_00013_00010.png
inflating:    train/6/00006_00013_00011.png
inflating:    train/6/00006_00013_00012.png
inflating:    train/6/00006_00013_00013.png
inflating:    train/6/00006_00013_00014.png
inflating:    train/6/00006_00013_00015.png
inflating:    train/6/00006_00013_00016.png
inflating:    train/6/00006_00013_00017.png
inflating:    train/6/00006_00013_00018.png
inflating:    train/6/00006_00013_00019.png
inflating:    train/6/00006_00013_00020.png
inflating:    train/6/00006_00013_00021.png
inflating:    train/6/00006_00013_00022.png
inflating:    train/6/00006_00013_00023.png
inflating:    train/6/00006_00013_00024.png
inflating:    train/6/00006_00013_00025.png
inflating:    train/6/00006_00013_00026.png
inflating:    train/6/00006_00013_00027.png
inflating:    train/6/00006_00013_00028.png
inflating:    train/6/00006_00013_00029.png
inflating:    train/7/00007_00000_00000.png
inflating:    train/7/00007_00000_00001.png
inflating:    train/7/00007_00000_00002.png
inflating:    train/7/00007_00000_00003.png
inflating:    train/7/00007_00000_00004.png
inflating:    train/7/00007_00000_00005.png
inflating:    train/7/00007_00000_00006.png
inflating:    train/7/00007_00000_00007.png
inflating:    train/7/00007_00000_00008.png
inflating:    train/7/00007_00000_00009.png
inflating:    train/7/00007_00000_00010.png
inflating:    train/7/00007_00000_00011.png
inflating:    train/7/00007_00000_00012.png
inflating:    train/7/00007_00000_00013.png
inflating:    train/7/00007_00000_00014.png
inflating: train/7/00007_00000_00015.png
```

```
 inflating:    train/7/00007_00000_00016.png
inflating:    train/7/00007_00000_00017.png
inflating:    train/7/00007_00000_00018.png
inflating:    train/7/00007_00000_00019.png
inflating:  train/7/00007_00000_00020.png
```

```
inflating:    train/7/00007_00000_00021.png
inflating:    train/7/00007_00000_00022.png
inflating:    train/7/00007_00000_00023.png
inflating:    train/7/00007_00000_00024.png
inflating:    train/7/00007_00000_00025.png
inflating:    train/7/00007_00000_00026.png
inflating:    train/7/00007_00000_00027.png
inflating:    train/7/00007_00000_00028.png
inflating:    train/7/00007_00000_00029.png
inflating:    train/7/00007_00001_00000.png
inflating:    train/7/00007_00001_00001.png
inflating:    train/7/00007_00001_00002.png
inflating:    train/7/00007_00001_00003.png
inflating:    train/7/00007_00001_00004.png
inflating:    train/7/00007_00001_00005.png
inflating:    train/7/00007_00001_00006.png
inflating:    train/7/00007_00001_00007.png
inflating:    train/7/00007_00001_00008.png
inflating:    train/7/00007_00001_00009.png
inflating:    train/7/00007_00001_00010.png
inflating:    train/7/00007_00001_00011.png
inflating:    train/7/00007_00001_00012.png
inflating:    train/7/00007_00001_00013.png
inflating:    train/7/00007_00001_00014.png
inflating:    train/7/00007_00001_00015.png
inflating:    train/7/00007_00001_00016.png
inflating:    train/7/00007_00001_00017.png
inflating:    train/7/00007_00001_00018.png
inflating:    train/7/00007_00001_00019.png
inflating:    train/7/00007_00001_00020.png
inflating:    train/7/00007_00001_00021.png
inflating:    train/7/00007_00001_00022.png
inflating:    train/7/00007_00001_00023.png
inflating:    train/7/00007_00001_00024.png
inflating:    train/7/00007_00001_00025.png
inflating:    train/7/00007_00001_00026.png
inflating:    train/7/00007_00001_00027.png
inflating:    train/7/00007_00001_00028.png
inflating:    train/7/00007_00001_00029.png
inflating:    train/7/00007_00002_00000.png
inflating:    train/7/00007_00002_00001.png
inflating:    train/7/00007_00002_00002.png
inflating:    train/7/00007_00002_00003.png
inflating:    train/7/00007_00002_00004.png
inflating:    train/7/00007_00002_00005.png
inflating:    train/7/00007_00002_00006.png
inflating:    train/7/00007_00002_00007.png
inflating:    train/7/00007_00002_00008.png
inflating:    train/7/00007_00002_00009.png
inflating:    train/7/00007_00002_00010.png
inflating:    train/7/00007_00002_00011.png
inflating:    train/7/00007_00002_00012.png
inflating:    train/7/00007_00002_00013.png
inflating:    train/7/00007_00002_00014.png
inflating:    train/7/00007_00002_00015.png
inflating: train/7/00007_00002_00016.png
```

```
inflating:    train/7/00007_00002_00017.png
inflating:    train/7/00007_00002_00018.png
inflating:    train/7/00007_00002_00019.png
inflating:    train/7/00007_00002_00020.png
inflating:    train/7/00007_00002_00021.png
inflating:    train/7/00007_00002_00022.png
inflating:    train/7/00007_00002_00023.png
inflating:    train/7/00007_00002_00024.png
inflating:    train/7/00007_00002_00025.png
inflating:  train/7/00007_00002_00026.png
```

```
inflating:    train/7/00007_00002_00027.png
inflating:    train/7/00007_00002_00028.png
inflating:    train/7/00007_00002_00029.png
inflating:    train/7/00007_00003_00000.png
inflating:    train/7/00007_00003_00001.png
inflating:    train/7/00007_00003_00002.png
inflating:    train/7/00007_00003_00003.png
inflating:    train/7/00007_00003_00004.png
inflating:    train/7/00007_00003_00005.png
inflating:    train/7/00007_00003_00006.png
inflating:    train/7/00007_00003_00007.png
inflating:    train/7/00007_00003_00008.png
inflating:    train/7/00007_00003_00009.png
inflating:    train/7/00007_00003_00010.png
inflating:    train/7/00007_00003_00011.png
inflating:    train/7/00007_00003_00012.png
inflating:    train/7/00007_00003_00013.png
inflating:    train/7/00007_00003_00014.png
inflating:    train/7/00007_00003_00015.png
inflating:    train/7/00007_00003_00016.png
inflating:    train/7/00007_00003_00017.png
inflating:    train/7/00007_00003_00018.png
inflating:    train/7/00007_00003_00019.png
inflating:    train/7/00007_00003_00020.png
inflating:    train/7/00007_00003_00021.png
inflating:    train/7/00007_00003_00022.png
inflating:    train/7/00007_00003_00023.png
inflating:    train/7/00007_00003_00024.png
inflating:    train/7/00007_00003_00025.png
inflating:    train/7/00007_00003_00026.png
inflating:    train/7/00007_00003_00027.png
inflating:    train/7/00007_00003_00028.png
inflating:    train/7/00007_00003_00029.png
inflating:    train/7/00007_00004_00000.png
inflating:    train/7/00007_00004_00001.png
inflating:    train/7/00007_00004_00002.png
inflating:    train/7/00007_00004_00003.png
inflating:    train/7/00007_00004_00004.png
inflating:    train/7/00007_00004_00005.png
inflating:    train/7/00007_00004_00006.png
inflating:    train/7/00007_00004_00007.png
inflating:    train/7/00007_00004_00008.png
inflating:    train/7/00007_00004_00009.png
inflating:    train/7/00007_00004_00010.png
inflating:    train/7/00007_00004_00011.png
inflating:    train/7/00007_00004_00012.png
inflating:    train/7/00007_00004_00013.png
inflating:    train/7/00007_00004_00014.png
inflating:    train/7/00007_00004_00015.png
inflating:    train/7/00007_00004_00016.png
inflating: train/7/00007_00004_00017.png
```

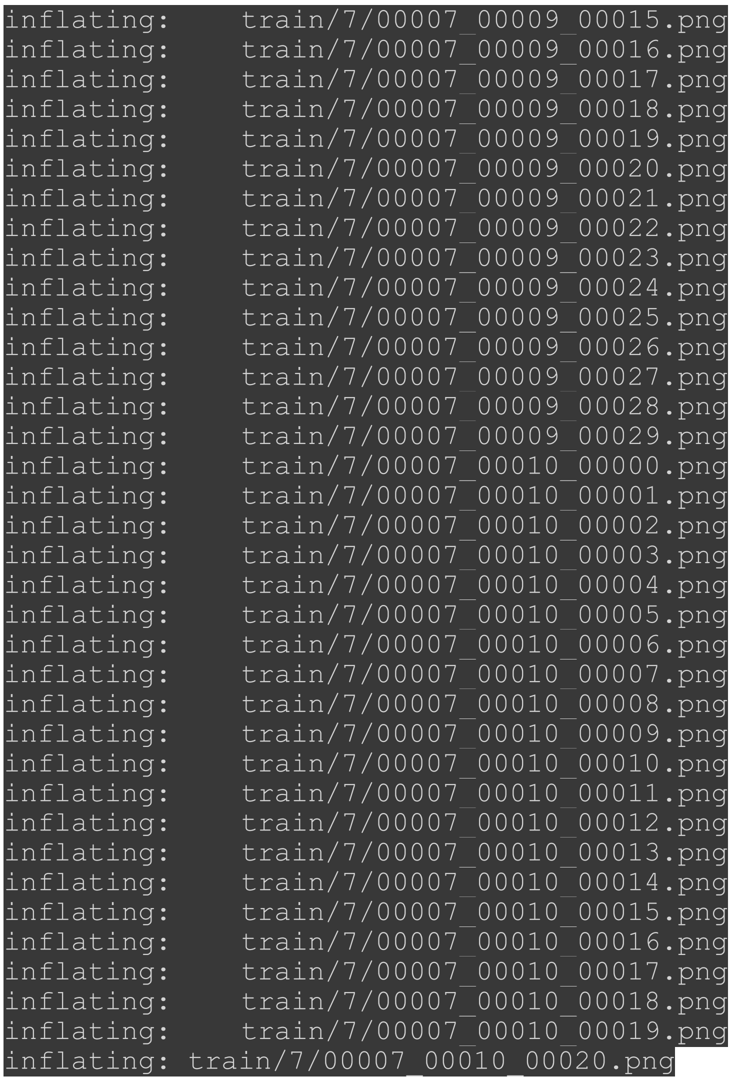
```
inflating:    train/7/00007_00004_00018.png
inflating:    train/7/00007_00004_00019.png
inflating:    train/7/00007_00004_00020.png
inflating:    train/7/00007_00004_00021.png
inflating:    train/7/00007_00004_00022.png
inflating:    train/7/00007_00004_00023.png
inflating:    train/7/00007_00004_00024.png
inflating:    train/7/00007_00004_00025.png
inflating:    train/7/00007_00004_00026.png
inflating:    train/7/00007_00004_00027.png
inflating:    train/7/00007_00004_00028.png
inflating:    train/7/00007_00004_00029.png
inflating:    train/7/00007_00005_00000.png
inflating:    train/7/00007_00005_00001.png
inflating:  train/7/00007_00005_00002.png
```

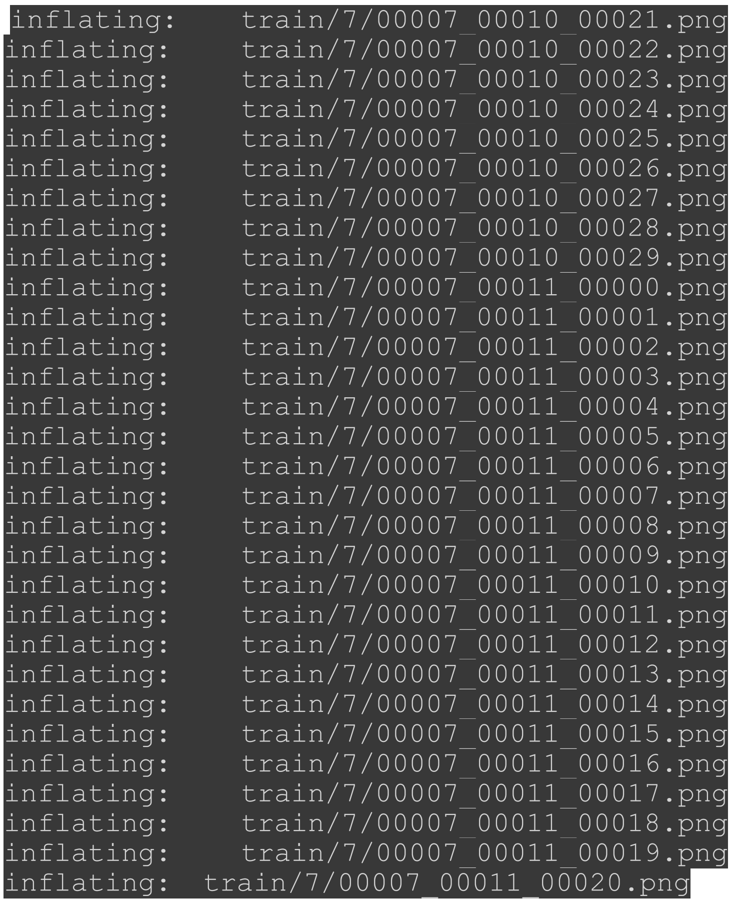
```
inflating:    train/7/00007_00005_00003.png
inflating:    train/7/00007_00005_00004.png
inflating:    train/7/00007_00005_00005.png
inflating:    train/7/00007_00005_00006.png
inflating:    train/7/00007_00005_00007.png
inflating:    train/7/00007_00005_00008.png
inflating:    train/7/00007_00005_00009.png
inflating:    train/7/00007_00005_00010.png
inflating:    train/7/00007_00005_00011.png
inflating:    train/7/00007_00005_00012.png
inflating:    train/7/00007_00005_00013.png
inflating:    train/7/00007_00005_00014.png
inflating:    train/7/00007_00005_00015.png
inflating:    train/7/00007_00005_00016.png
inflating:    train/7/00007_00005_00017.png
inflating:    train/7/00007_00005_00018.png
inflating:    train/7/00007_00005_00019.png
inflating:    train/7/00007_00005_00020.png
inflating:    train/7/00007_00005_00021.png
inflating:    train/7/00007_00005_00022.png
inflating:    train/7/00007_00005_00023.png
inflating:    train/7/00007_00005_00024.png
inflating:    train/7/00007_00005_00025.png
inflating:    train/7/00007_00005_00026.png
inflating:    train/7/00007_00005_00027.png
inflating:    train/7/00007_00005_00028.png
inflating:    train/7/00007_00005_00029.png
inflating:    train/7/00007_00006_00000.png
inflating:    train/7/00007_00006_00001.png
inflating:    train/7/00007_00006_00002.png
inflating:    train/7/00007_00006_00003.png
inflating:    train/7/00007_00006_00004.png
inflating:    train/7/00007_00006_00005.png
inflating:    train/7/00007_00006_00006.png
inflating:    train/7/00007_00006_00007.png
inflating:    train/7/00007_00006_00008.png
inflating:    train/7/00007_00006_00009.png
inflating:    train/7/00007_00006_00010.png
inflating:    train/7/00007_00006_00011.png
inflating:    train/7/00007_00006_00012.png
inflating:    train/7/00007_00006_00013.png
inflating:    train/7/00007_00006_00014.png
inflating:    train/7/00007_00006_00015.png
inflating:    train/7/00007_00006_00016.png
inflating:    train/7/00007_00006_00017.png
inflating: train/7/00007_00006_00018.png
```

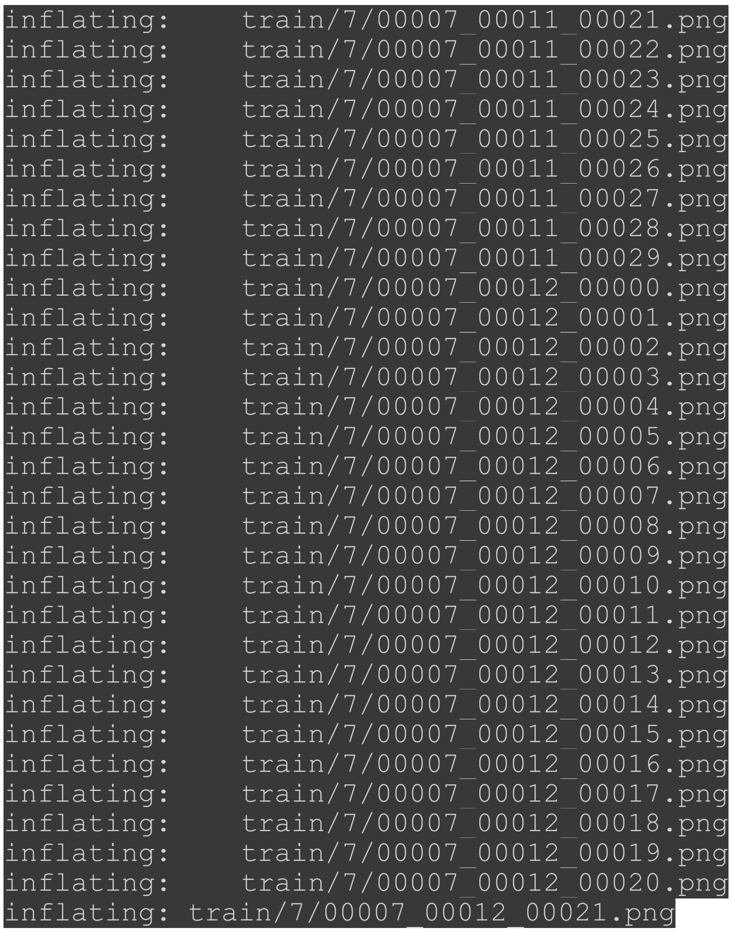
```
inflating:    train/7/00007_00006_00019.png
inflating:    train/7/00007_00006_00020.png
inflating:    train/7/00007_00006_00021.png
inflating:    train/7/00007_00006_00022.png
inflating:    train/7/00007_00006_00023.png
inflating:    train/7/00007_00006_00024.png
inflating:    train/7/00007_00006_00025.png
inflating:    train/7/00007_00006_00026.png
inflating:    train/7/00007_00006_00027.png
inflating:    train/7/00007_00006_00028.png
inflating:    train/7/00007_00006_00029.png
inflating:    train/7/00007_00007_00000.png
inflating:    train/7/00007_00007_00001.png
inflating:    train/7/00007_00007_00002.png
inflating:    train/7/00007_00007_00003.png
inflating:    train/7/00007_00007_00004.png
inflating:    train/7/00007_00007_00005.png
inflating:    train/7/00007_00007_00006.png
inflating:    train/7/00007_00007_00007.png
inflating:  train/7/00007_00007_00008.png
```

```
inflating:    train/7/00007_00007_00009.png
inflating:    train/7/00007_00007_00010.png
inflating:    train/7/00007_00007_00011.png
inflating:    train/7/00007_00007_00012.png
inflating:    train/7/00007_00007_00013.png
inflating:    train/7/00007_00007_00014.png
inflating:    train/7/00007_00007_00015.png
inflating:    train/7/00007_00007_00016.png
inflating:    train/7/00007_00007_00017.png
inflating:    train/7/00007_00007_00018.png
inflating:    train/7/00007_00007_00019.png
inflating:    train/7/00007_00007_00020.png
inflating:    train/7/00007_00007_00021.png
inflating:    train/7/00007_00007_00022.png
inflating:    train/7/00007_00007_00023.png
inflating:    train/7/00007_00007_00024.png
inflating:    train/7/00007_00007_00025.png
inflating:    train/7/00007_00007_00026.png
inflating:    train/7/00007_00007_00027.png
inflating:    train/7/00007_00007_00028.png
inflating:    train/7/00007_00007_00029.png
inflating:    train/7/00007_00008_00000.png
inflating:    train/7/00007_00008_00001.png
inflating:    train/7/00007_00008_00002.png
inflating:    train/7/00007_00008_00003.png
inflating:    train/7/00007_00008_00004.png
inflating:    train/7/00007_00008_00005.png
inflating:    train/7/00007_00008_00006.png
inflating:    train/7/00007_00008_00007.png
inflating:    train/7/00007_00008_00008.png
inflating:    train/7/00007_00008_00009.png
inflating:    train/7/00007_00008_00010.png
inflating:    train/7/00007_00008_00011.png
inflating:    train/7/00007_00008_00012.png
inflating:    train/7/00007_00008_00013.png
inflating:    train/7/00007_00008_00014.png
inflating:    train/7/00007_00008_00015.png
inflating:    train/7/00007_00008_00016.png
inflating:    train/7/00007_00008_00017.png
inflating:    train/7/00007_00008_00018.png
inflating: train/7/00007_00008_00019.png
```

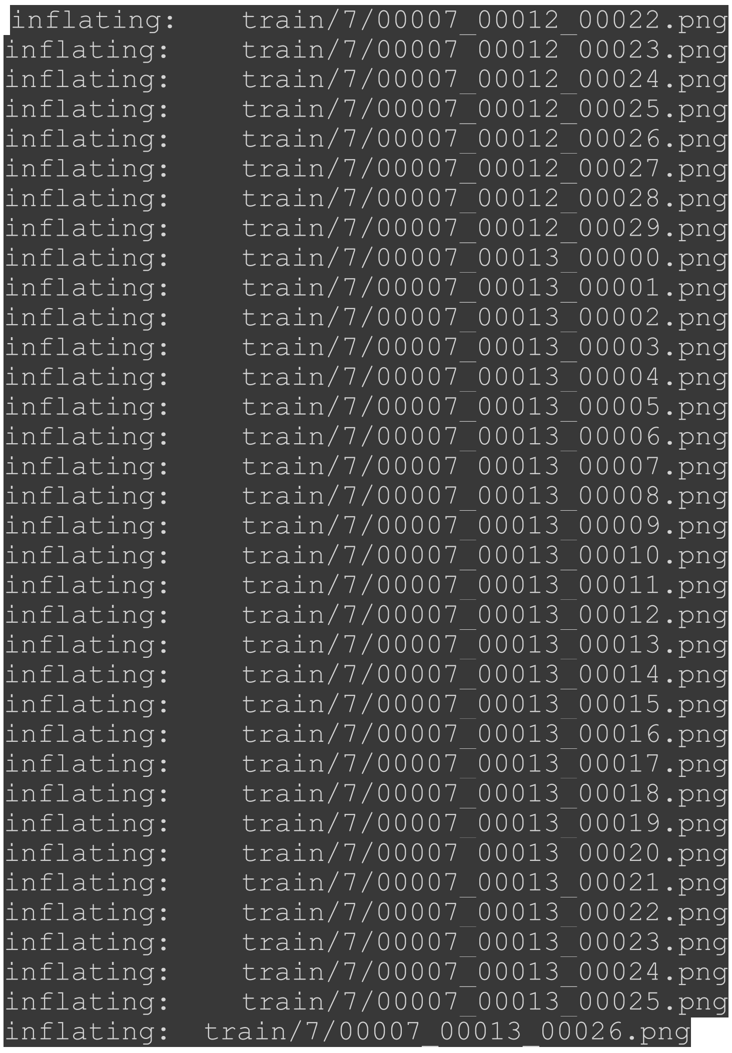
```
inflating:    train/7/00007_00008_00020.png
inflating:    train/7/00007_00008_00021.png
inflating:    train/7/00007_00008_00022.png
inflating:    train/7/00007_00008_00023.png
inflating:    train/7/00007_00008_00024.png
inflating:    train/7/00007_00008_00025.png
inflating:    train/7/00007_00008_00026.png
inflating:    train/7/00007_00008_00027.png
inflating:    train/7/00007_00008_00028.png
inflating:    train/7/00007_00008_00029.png
inflating:    train/7/00007_00009_00000.png
inflating:    train/7/00007_00009_00001.png
inflating:    train/7/00007_00009_00002.png
inflating:    train/7/00007_00009_00003.png
inflating:    train/7/00007_00009_00004.png
inflating:    train/7/00007_00009_00005.png
inflating:    train/7/00007_00009_00006.png
inflating:    train/7/00007_00009_00007.png
inflating:    train/7/00007_00009_00008.png
inflating:    train/7/00007_00009_00009.png
inflating:    train/7/00007_00009_00010.png
inflating:    train/7/00007_00009_00011.png
inflating:    train/7/00007_00009_00012.png
inflating:    train/7/00007_00009_00013.png
inflating:  train/7/00007_00009_00014.png
```


```
inflating:    train/7/00007_00009_00015.png
inflating:    train/7/00007_00009_00016.png
inflating:    train/7/00007_00009_00017.png
inflating:    train/7/00007_00009_00018.png
inflating:    train/7/00007_00009_00019.png
inflating:    train/7/00007_00009_00020.png
inflating:    train/7/00007_00009_00021.png
inflating:    train/7/00007_00009_00022.png
inflating:    train/7/00007_00009_00023.png
inflating:    train/7/00007_00009_00024.png
inflating:    train/7/00007_00009_00025.png
inflating:    train/7/00007_00009_00026.png
inflating:    train/7/00007_00009_00027.png
inflating:    train/7/00007_00009_00028.png
inflating:    train/7/00007_00009_00029.png
inflating:    train/7/00007_00010_00000.png
inflating:    train/7/00007_00010_00001.png
inflating:    train/7/00007_00010_00002.png
inflating:    train/7/00007_00010_00003.png
inflating:    train/7/00007_00010_00004.png
inflating:    train/7/00007_00010_00005.png
inflating:    train/7/00007_00010_00006.png
inflating:    train/7/00007_00010_00007.png
inflating:    train/7/00007_00010_00008.png
inflating:    train/7/00007_00010_00009.png
inflating:    train/7/00007_00010_00010.png
inflating:    train/7/00007_00010_00011.png
inflating:    train/7/00007_00010_00012.png
inflating:    train/7/00007_00010_00013.png
inflating:    train/7/00007_00010_00014.png
inflating:    train/7/00007_00010_00015.png
inflating:    train/7/00007_00010_00016.png
inflating:    train/7/00007_00010_00017.png
inflating:    train/7/00007_00010_00018.png
inflating:    train/7/00007_00010_00019.png
inflating: train/7/00007_00010_00020.png
```


```
inflating:    train/7/00007_00010_00021.png
inflating:    train/7/00007_00010_00022.png
inflating:    train/7/00007_00010_00023.png
inflating:    train/7/00007_00010_00024.png
inflating:    train/7/00007_00010_00025.png
inflating:    train/7/00007_00010_00026.png
inflating:    train/7/00007_00010_00027.png
inflating:    train/7/00007_00010_00028.png
inflating:    train/7/00007_00010_00029.png
inflating:    train/7/00007_00011_00000.png
inflating:    train/7/00007_00011_00001.png
inflating:    train/7/00007_00011_00002.png
inflating:    train/7/00007_00011_00003.png
inflating:    train/7/00007_00011_00004.png
inflating:    train/7/00007_00011_00005.png
inflating:    train/7/00007_00011_00006.png
inflating:    train/7/00007_00011_00007.png
inflating:    train/7/00007_00011_00008.png
inflating:    train/7/00007_00011_00009.png
inflating:    train/7/00007_00011_00010.png
inflating:    train/7/00007_00011_00011.png
inflating:    train/7/00007_00011_00012.png
inflating:    train/7/00007_00011_00013.png
inflating:    train/7/00007_00011_00014.png
inflating:    train/7/00007_00011_00015.png
inflating:    train/7/00007_00011_00016.png
inflating:    train/7/00007_00011_00017.png
inflating:    train/7/00007_00011_00018.png
inflating:    train/7/00007_00011_00019.png
inflating:  train/7/00007_00011_00020.png
```



```
inflating:    train/7/00007_00011_00021.png
inflating:    train/7/00007_00011_00022.png
inflating:    train/7/00007_00011_00023.png
inflating:    train/7/00007_00011_00024.png
inflating:    train/7/00007_00011_00025.png
inflating:    train/7/00007_00011_00026.png
inflating:    train/7/00007_00011_00027.png
inflating:    train/7/00007_00011_00028.png
inflating:    train/7/00007_00011_00029.png
inflating:    train/7/00007_00012_00000.png
inflating:    train/7/00007_00012_00001.png
inflating:    train/7/00007_00012_00002.png
inflating:    train/7/00007_00012_00003.png
inflating:    train/7/00007_00012_00004.png
inflating:    train/7/00007_00012_00005.png
inflating:    train/7/00007_00012_00006.png
inflating:    train/7/00007_00012_00007.png
inflating:    train/7/00007_00012_00008.png
inflating:    train/7/00007_00012_00009.png
inflating:    train/7/00007_00012_00010.png
inflating:    train/7/00007_00012_00011.png
inflating:    train/7/00007_00012_00012.png
inflating:    train/7/00007_00012_00013.png
inflating:    train/7/00007_00012_00014.png
inflating:    train/7/00007_00012_00015.png
inflating:    train/7/00007_00012_00016.png
inflating:    train/7/00007_00012_00017.png
inflating:    train/7/00007_00012_00018.png
inflating:    train/7/00007_00012_00019.png
inflating:    train/7/00007_00012_00020.png
inflating: train/7/00007_00012_00021.png
```



```
inflating:    train/7/00007_00012_00022.png
inflating:    train/7/00007_00012_00023.png
inflating:    train/7/00007_00012_00024.png
inflating:    train/7/00007_00012_00025.png
inflating:    train/7/00007_00012_00026.png
inflating:    train/7/00007_00012_00027.png
inflating:    train/7/00007_00012_00028.png
inflating:    train/7/00007_00012_00029.png
inflating:    train/7/00007_00013_00000.png
inflating:    train/7/00007_00013_00001.png
inflating:    train/7/00007_00013_00002.png
inflating:    train/7/00007_00013_00003.png
inflating:    train/7/00007_00013_00004.png
inflating:    train/7/00007_00013_00005.png
inflating:    train/7/00007_00013_00006.png
inflating:    train/7/00007_00013_00007.png
inflating:    train/7/00007_00013_00008.png
inflating:    train/7/00007_00013_00009.png
inflating:    train/7/00007_00013_00010.png
inflating:    train/7/00007_00013_00011.png
inflating:    train/7/00007_00013_00012.png
inflating:    train/7/00007_00013_00013.png
inflating:    train/7/00007_00013_00014.png
inflating:    train/7/00007_00013_00015.png
inflating:    train/7/00007_00013_00016.png
inflating:    train/7/00007_00013_00017.png
inflating:    train/7/00007_00013_00018.png
inflating:    train/7/00007_00013_00019.png
inflating:    train/7/00007_00013_00020.png
inflating:    train/7/00007_00013_00021.png
inflating:    train/7/00007_00013_00022.png
inflating:    train/7/00007_00013_00023.png
inflating:    train/7/00007_00013_00024.png
inflating:    train/7/00007_00013_00025.png
inflating:  train/7/00007_00013_00026.png
```

```
inflating:    train/7/00007_00013_00027.png
inflating:    train/7/00007_00013_00028.png
inflating:    train/7/00007_00013_00029.png
inflating:    train/7/00007_00014_00000.png
inflating:    train/7/00007_00014_00001.png
inflating:    train/7/00007_00014_00002.png
inflating:    train/7/00007_00014_00003.png
inflating:    train/7/00007_00014_00004.png
inflating:    train/7/00007_00014_00005.png
inflating:    train/7/00007_00014_00006.png
inflating:    train/7/00007_00014_00007.png
inflating:    train/7/00007_00014_00008.png
inflating:    train/7/00007_00014_00009.png
inflating:    train/7/00007_00014_00010.png
inflating:    train/7/00007_00014_00011.png
inflating:    train/7/00007_00014_00012.png
inflating:    train/7/00007_00014_00013.png
inflating:    train/7/00007_00014_00014.png
inflating:    train/7/00007_00014_00015.png
inflating:    train/7/00007_00014_00016.png
inflating:    train/7/00007_00014_00017.png
inflating:    train/7/00007_00014_00018.png
inflating:    train/7/00007_00014_00019.png
inflating:    train/7/00007_00014_00020.png
inflating:    train/7/00007_00014_00021.png
inflating: train/7/00007_00014_00022.png
```

```
inflating:    train/7/00007_00014_00023.png
inflating:    train/7/00007_00014_00024.png
inflating:    train/7/00007_00014_00025.png
inflating:    train/7/00007_00014_00026.png
inflating:    train/7/00007_00014_00027.png
inflating:    train/7/00007_00014_00028.png
inflating:    train/7/00007_00014_00029.png
inflating:    train/7/00007_00015_00000.png
inflating:    train/7/00007_00015_00001.png
inflating:    train/7/00007_00015_00002.png
inflating:    train/7/00007_00015_00003.png
inflating:    train/7/00007_00015_00004.png
inflating:    train/7/00007_00015_00005.png
inflating:    train/7/00007_00015_00006.png
inflating:    train/7/00007_00015_00007.png
inflating:    train/7/00007_00015_00008.png
inflating:    train/7/00007_00015_00009.png
inflating:    train/7/00007_00015_00010.png
inflating:    train/7/00007_00015_00011.png
inflating:    train/7/00007_00015_00012.png
inflating:    train/7/00007_00015_00013.png
inflating:    train/7/00007_00015_00014.png
inflating:    train/7/00007_00015_00015.png
inflating:    train/7/00007_00015_00016.png
inflating:    train/7/00007_00015_00017.png
inflating:    train/7/00007_00015_00018.png
inflating:    train/7/00007_00015_00019.png
inflating:    train/7/00007_00015_00020.png
inflating:    train/7/00007_00015_00021.png
inflating:    train/7/00007_00015_00022.png
inflating:    train/7/00007_00015_00023.png
inflating:    train/7/00007_00015_00024.png
inflating:    train/7/00007_00015_00025.png
inflating:    train/7/00007_00015_00026.png
inflating:    train/7/00007_00015_00027.png
inflating:    train/7/00007_00015_00028.png
inflating:    train/7/00007_00015_00029.png
inflating:    train/7/00007_00016_00000.png
inflating:    train/7/00007_00016_00001.png
inflating:  train/7/00007_00016_00002.png
```

```
inflating:    train/7/00007_00016_00003.png
inflating:    train/7/00007_00016_00004.png
inflating:    train/7/00007_00016_00005.png
inflating:    train/7/00007_00016_00006.png
inflating:    train/7/00007_00016_00007.png
inflating:    train/7/00007_00016_00008.png
inflating:    train/7/00007_00016_00009.png
inflating:    train/7/00007_00016_00010.png
inflating:    train/7/00007_00016_00011.png
inflating:    train/7/00007_00016_00012.png
inflating:    train/7/00007_00016_00013.png
inflating:    train/7/00007_00016_00014.png
inflating:    train/7/00007_00016_00015.png
inflating:    train/7/00007_00016_00016.png
inflating:    train/7/00007_00016_00017.png
inflating:    train/7/00007_00016_00018.png
inflating:    train/7/00007_00016_00019.png
inflating:    train/7/00007_00016_00020.png
inflating:    train/7/00007_00016_00021.png
inflating:    train/7/00007_00016_00022.png
inflating: train/7/00007_00016_00023.png
```

```
inflating:    train/7/00007_00016_00024.png
inflating:    train/7/00007_00016_00025.png
inflating:    train/7/00007_00016_00026.png
inflating:    train/7/00007_00016_00027.png
inflating:    train/7/00007_00016_00028.png
inflating:    train/7/00007_00016_00029.png
inflating:    train/7/00007_00017_00000.png
inflating:    train/7/00007_00017_00001.png
inflating:    train/7/00007_00017_00002.png
inflating:    train/7/00007_00017_00003.png
inflating:    train/7/00007_00017_00004.png
inflating:    train/7/00007_00017_00005.png
inflating:    train/7/00007_00017_00006.png
inflating:    train/7/00007_00017_00007.png
inflating:    train/7/00007_00017_00008.png
inflating:    train/7/00007_00017_00009.png
inflating:    train/7/00007_00017_00010.png
inflating:    train/7/00007_00017_00011.png
inflating:    train/7/00007_00017_00012.png
inflating:    train/7/00007_00017_00013.png
inflating:    train/7/00007_00017_00014.png
inflating:    train/7/00007_00017_00015.png
inflating:    train/7/00007_00017_00016.png
inflating:    train/7/00007_00017_00017.png
inflating:    train/7/00007_00017_00018.png
inflating:    train/7/00007_00017_00019.png
inflating:    train/7/00007_00017_00020.png
inflating:    train/7/00007_00017_00021.png
inflating:    train/7/00007_00017_00022.png
inflating:    train/7/00007_00017_00023.png
inflating:    train/7/00007_00017_00024.png
inflating:    train/7/00007_00017_00025.png
inflating:    train/7/00007_00017_00026.png
inflating:    train/7/00007_00017_00027.png
inflating:    train/7/00007_00017_00028.png
inflating:    train/7/00007_00017_00029.png
inflating:    train/7/00007_00018_00000.png
inflating:    train/7/00007_00018_00001.png
inflating:    train/7/00007_00018_00002.png
inflating:    train/7/00007_00018_00003.png
inflating:    train/7/00007_00018_00004.png
inflating:    train/7/00007_00018_00005.png
inflating:    train/7/00007_00018_00006.png
inflating:    train/7/00007_00018_00007.png
inflating:  train/7/00007_00018_00008.png
```

```
inflating:    train/7/00007_00018_00009.png
inflating:    train/7/00007_00018_00010.png
inflating:    train/7/00007_00018_00011.png
inflating:    train/7/00007_00018_00012.png
inflating:    train/7/00007_00018_00013.png
inflating:    train/7/00007_00018_00014.png
inflating:    train/7/00007_00018_00015.png
inflating:    train/7/00007_00018_00016.png
inflating:    train/7/00007_00018_00017.png
inflating:    train/7/00007_00018_00018.png
inflating:    train/7/00007_00018_00019.png
inflating:    train/7/00007_00018_00020.png
inflating:    train/7/00007_00018_00021.png
inflating:    train/7/00007_00018_00022.png
inflating:    train/7/00007_00018_00023.png
inflating: train/7/00007_00018_00024.png
```

```
inflating:    train/7/00007_00018_00025.png
inflating:    train/7/00007_00018_00026.png
inflating:    train/7/00007_00018_00027.png
inflating:    train/7/00007_00018_00028.png
inflating:    train/7/00007_00018_00029.png
inflating:    train/7/00007_00019_00000.png
inflating:    train/7/00007_00019_00001.png
inflating:    train/7/00007_00019_00002.png
inflating:    train/7/00007_00019_00003.png
inflating:    train/7/00007_00019_00004.png
inflating:    train/7/00007_00019_00005.png
inflating:    train/7/00007_00019_00006.png
inflating:    train/7/00007_00019_00007.png
inflating:    train/7/00007_00019_00008.png
inflating:    train/7/00007_00019_00009.png
inflating:    train/7/00007_00019_00010.png
inflating:    train/7/00007_00019_00011.png
inflating:    train/7/00007_00019_00012.png
inflating:    train/7/00007_00019_00013.png
inflating:    train/7/00007_00019_00014.png
inflating:    train/7/00007_00019_00015.png
inflating:    train/7/00007_00019_00016.png
inflating:    train/7/00007_00019_00017.png
inflating:    train/7/00007_00019_00018.png
inflating:    train/7/00007_00019_00019.png
inflating:    train/7/00007_00019_00020.png
inflating:    train/7/00007_00019_00021.png
inflating:    train/7/00007_00019_00022.png
inflating:    train/7/00007_00019_00023.png
inflating:    train/7/00007_00019_00024.png
inflating:    train/7/00007_00019_00025.png
inflating:    train/7/00007_00019_00026.png
inflating:    train/7/00007_00019_00027.png
inflating:    train/7/00007_00019_00028.png
inflating:    train/7/00007_00019_00029.png
inflating:    train/7/00007_00020_00000.png
inflating:    train/7/00007_00020_00001.png
inflating:    train/7/00007_00020_00002.png
inflating:    train/7/00007_00020_00003.png
inflating:    train/7/00007_00020_00004.png
inflating:    train/7/00007_00020_00005.png
inflating:    train/7/00007_00020_00006.png
inflating:    train/7/00007_00020_00007.png
inflating:    train/7/00007_00020_00008.png
inflating:    train/7/00007_00020_00009.png
inflating:    train/7/00007_00020_00010.png
inflating:    train/7/00007_00020_00011.png
inflating:    train/7/00007_00020_00012.png
inflating:    train/7/00007_00020_00013.png
inflating:  train/7/00007_00020_00014.png
```

```
inflating:    train/7/00007_00020_00015.png
inflating:    train/7/00007_00020_00016.png
inflating:    train/7/00007_00020_00017.png
inflating:    train/7/00007_00020_00018.png
inflating:    train/7/00007_00020_00019.png
inflating:    train/7/00007_00020_00020.png
inflating:    train/7/00007_00020_00021.png
inflating:    train/7/00007_00020_00022.png
inflating:    train/7/00007_00020_00023.png
inflating:    train/7/00007_00020_00024.png
inflating: train/7/00007_00020_00025.png
```

```
inflating:    train/7/00007_00020_00026.png
inflating:    train/7/00007_00020_00027.png
inflating:    train/7/00007_00020_00028.png
inflating:    train/7/00007_00020_00029.png
inflating:    train/7/00007_00021_00000.png
inflating:    train/7/00007_00021_00001.png
inflating:    train/7/00007_00021_00002.png
inflating:    train/7/00007_00021_00003.png
inflating:    train/7/00007_00021_00004.png
inflating:    train/7/00007_00021_00005.png
inflating:    train/7/00007_00021_00006.png
inflating:    train/7/00007_00021_00007.png
inflating:    train/7/00007_00021_00008.png
inflating:    train/7/00007_00021_00009.png
inflating:    train/7/00007_00021_00010.png
inflating:    train/7/00007_00021_00011.png
inflating:    train/7/00007_00021_00012.png
inflating:    train/7/00007_00021_00013.png
inflating:    train/7/00007_00021_00014.png
inflating:    train/7/00007_00021_00015.png
inflating:    train/7/00007_00021_00016.png
inflating:    train/7/00007_00021_00017.png
inflating:    train/7/00007_00021_00018.png
inflating:    train/7/00007_00021_00019.png
inflating:    train/7/00007_00021_00020.png
inflating:    train/7/00007_00021_00021.png
inflating:    train/7/00007_00021_00022.png
inflating:    train/7/00007_00021_00023.png
inflating:    train/7/00007_00021_00024.png
inflating:    train/7/00007_00021_00025.png
inflating:    train/7/00007_00021_00026.png
inflating:    train/7/00007_00021_00027.png
inflating:    train/7/00007_00021_00028.png
inflating:    train/7/00007_00021_00029.png
inflating:    train/7/00007_00022_00000.png
inflating:    train/7/00007_00022_00001.png
inflating:    train/7/00007_00022_00002.png
inflating:    train/7/00007_00022_00003.png
inflating:    train/7/00007_00022_00004.png
inflating:    train/7/00007_00022_00005.png
inflating:    train/7/00007_00022_00006.png
inflating:    train/7/00007_00022_00007.png
inflating:    train/7/00007_00022_00008.png
inflating:    train/7/00007_00022_00009.png
inflating:    train/7/00007_00022_00010.png
inflating:    train/7/00007_00022_00011.png
inflating:    train/7/00007_00022_00012.png
inflating:    train/7/00007_00022_00013.png
inflating:    train/7/00007_00022_00014.png
inflating:    train/7/00007_00022_00015.png
inflating:    train/7/00007_00022_00016.png
inflating:    train/7/00007_00022_00017.png
inflating:    train/7/00007_00022_00018.png
inflating:    train/7/00007_00022_00019.png
inflating:  train/7/00007_00022_00020.png
```

```
inflating:    train/7/00007_00022_00021.png
inflating:    train/7/00007_00022_00022.png
inflating:    train/7/00007_00022_00023.png
inflating:    train/7/00007_00022_00024.png
inflating:    train/7/00007_00022_00025.png
inflating: train/7/00007_00022_00026.png
```

```
inflating:    train/7/00007_00022_00027.png
inflating:    train/7/00007_00022_00028.png
inflating:    train/7/00007_00022_00029.png
inflating:    train/7/00007_00023_00000.png
inflating:    train/7/00007_00023_00001.png
inflating:    train/7/00007_00023_00002.png
inflating:    train/7/00007_00023_00003.png
inflating:    train/7/00007_00023_00004.png
inflating:    train/7/00007_00023_00005.png
inflating:    train/7/00007_00023_00006.png
inflating:    train/7/00007_00023_00007.png
inflating:    train/7/00007_00023_00008.png
inflating:    train/7/00007_00023_00009.png
inflating:    train/7/00007_00023_00010.png
inflating:    train/7/00007_00023_00011.png
inflating:    train/7/00007_00023_00012.png
inflating:    train/7/00007_00023_00013.png
inflating:    train/7/00007_00023_00014.png
inflating:    train/7/00007_00023_00015.png
inflating:    train/7/00007_00023_00016.png
inflating:    train/7/00007_00023_00017.png
inflating:    train/7/00007_00023_00018.png
inflating:    train/7/00007_00023_00019.png
inflating:    train/7/00007_00023_00020.png
inflating:    train/7/00007_00023_00021.png
inflating:    train/7/00007_00023_00022.png
inflating:    train/7/00007_00023_00023.png
inflating:    train/7/00007_00023_00024.png
inflating:    train/7/00007_00023_00025.png
inflating:    train/7/00007_00023_00026.png
inflating:    train/7/00007_00023_00027.png
inflating:    train/7/00007_00023_00028.png
inflating:    train/7/00007_00023_00029.png
inflating:    train/7/00007_00024_00000.png
inflating:    train/7/00007_00024_00001.png
inflating:    train/7/00007_00024_00002.png
inflating:    train/7/00007_00024_00003.png
inflating:    train/7/00007_00024_00004.png
inflating:    train/7/00007_00024_00005.png
inflating:    train/7/00007_00024_00006.png
inflating:    train/7/00007_00024_00007.png
inflating:    train/7/00007_00024_00008.png
inflating:    train/7/00007_00024_00009.png
inflating:    train/7/00007_00024_00010.png
inflating:    train/7/00007_00024_00011.png
inflating:    train/7/00007_00024_00012.png
inflating:    train/7/00007_00024_00013.png
inflating:    train/7/00007_00024_00014.png
inflating:    train/7/00007_00024_00015.png
inflating:    train/7/00007_00024_00016.png
inflating:    train/7/00007_00024_00017.png
inflating:    train/7/00007_00024_00018.png
inflating:    train/7/00007_00024_00019.png
inflating:    train/7/00007_00024_00020.png
inflating:    train/7/00007_00024_00021.png
inflating:    train/7/00007_00024_00022.png
inflating:    train/7/00007_00024_00023.png
inflating:    train/7/00007_00024_00024.png
inflating:    train/7/00007_00024_00025.png
inflating:  train/7/00007_00024_00026.png
```

```
inflating: train/7/00007_00024_00027.png
```

```
inflating:    train/7/00007_00024_00028.png
inflating:    train/7/00007_00024_00029.png
inflating:    train/7/00007_00025_00000.png
inflating:    train/7/00007_00025_00001.png
inflating:    train/7/00007_00025_00002.png
inflating:    train/7/00007_00025_00003.png
inflating:    train/7/00007_00025_00004.png
inflating:    train/7/00007_00025_00005.png
inflating:    train/7/00007_00025_00006.png
inflating:    train/7/00007_00025_00007.png
inflating:    train/7/00007_00025_00008.png
inflating:    train/7/00007_00025_00009.png
inflating:    train/7/00007_00025_00010.png
inflating:    train/7/00007_00025_00011.png
inflating:    train/7/00007_00025_00012.png
inflating:    train/7/00007_00025_00013.png
inflating:    train/7/00007_00025_00014.png
inflating:    train/7/00007_00025_00015.png
inflating:    train/7/00007_00025_00016.png
inflating:    train/7/00007_00025_00017.png
inflating:    train/7/00007_00025_00018.png
inflating:    train/7/00007_00025_00019.png
inflating:    train/7/00007_00025_00020.png
inflating:    train/7/00007_00025_00021.png
inflating:    train/7/00007_00025_00022.png
inflating:    train/7/00007_00025_00023.png
inflating:    train/7/00007_00025_00024.png
inflating:    train/7/00007_00025_00025.png
inflating:    train/7/00007_00025_00026.png
inflating:    train/7/00007_00025_00027.png
inflating:    train/7/00007_00025_00028.png
inflating:    train/7/00007_00025_00029.png
inflating:    train/7/00007_00026_00000.png
inflating:    train/7/00007_00026_00001.png
inflating:    train/7/00007_00026_00002.png
inflating:    train/7/00007_00026_00003.png
inflating:    train/7/00007_00026_00004.png
inflating:    train/7/00007_00026_00005.png
inflating:    train/7/00007_00026_00006.png
inflating:    train/7/00007_00026_00007.png
inflating:    train/7/00007_00026_00008.png
inflating:    train/7/00007_00026_00009.png
inflating:    train/7/00007_00026_00010.png
inflating:    train/7/00007_00026_00011.png
inflating:    train/7/00007_00026_00012.png
inflating:    train/7/00007_00026_00013.png
inflating:    train/7/00007_00026_00014.png
inflating:    train/7/00007_00026_00015.png
inflating:    train/7/00007_00026_00016.png
inflating:    train/7/00007_00026_00017.png
inflating:    train/7/00007_00026_00018.png
inflating:    train/7/00007_00026_00019.png
inflating:    train/7/00007_00026_00020.png
inflating:    train/7/00007_00026_00021.png
inflating:    train/7/00007_00026_00022.png
inflating:    train/7/00007_00026_00023.png
inflating:    train/7/00007_00026_00024.png
inflating:    train/7/00007_00026_00025.png
inflating:    train/7/00007_00026_00026.png
inflating:    train/7/00007_00026_00027.png
inflating: train/7/00007_00026_00028.png
```

```
inflating:    train/7/00007_00026_00029.png
inflating:  train/7/00007_00027_00000.png
```

```
inflating:    train/7/00007_00027_00001.png
inflating:    train/7/00007_00027_00002.png
inflating:    train/7/00007_00027_00003.png
inflating:    train/7/00007_00027_00004.png
inflating:    train/7/00007_00027_00005.png
inflating:    train/7/00007_00027_00006.png
inflating:    train/7/00007_00027_00007.png
inflating:    train/7/00007_00027_00008.png
inflating:    train/7/00007_00027_00009.png
inflating:    train/7/00007_00027_00010.png
inflating:    train/7/00007_00027_00011.png
inflating:    train/7/00007_00027_00012.png
inflating:    train/7/00007_00027_00013.png
inflating:    train/7/00007_00027_00014.png
inflating:    train/7/00007_00027_00015.png
inflating:    train/7/00007_00027_00016.png
inflating:    train/7/00007_00027_00017.png
inflating:    train/7/00007_00027_00018.png
inflating:    train/7/00007_00027_00019.png
inflating:    train/7/00007_00027_00020.png
inflating:    train/7/00007_00027_00021.png
inflating:    train/7/00007_00027_00022.png
inflating:    train/7/00007_00027_00023.png
inflating:    train/7/00007_00027_00024.png
inflating:    train/7/00007_00027_00025.png
inflating:    train/7/00007_00027_00026.png
inflating:    train/7/00007_00027_00027.png
inflating:    train/7/00007_00027_00028.png
inflating:    train/7/00007_00027_00029.png
inflating:    train/7/00007_00028_00000.png
inflating:    train/7/00007_00028_00001.png
inflating:    train/7/00007_00028_00002.png
inflating:    train/7/00007_00028_00003.png
inflating:    train/7/00007_00028_00004.png
inflating:    train/7/00007_00028_00005.png
inflating:    train/7/00007_00028_00006.png
inflating:    train/7/00007_00028_00007.png
inflating:    train/7/00007_00028_00008.png
inflating:    train/7/00007_00028_00009.png
inflating:    train/7/00007_00028_00010.png
inflating:    train/7/00007_00028_00011.png
inflating:    train/7/00007_00028_00012.png
inflating:    train/7/00007_00028_00013.png
inflating:    train/7/00007_00028_00014.png
inflating:    train/7/00007_00028_00015.png
inflating:    train/7/00007_00028_00016.png
inflating:    train/7/00007_00028_00017.png
inflating:    train/7/00007_00028_00018.png
inflating:    train/7/00007_00028_00019.png
inflating:    train/7/00007_00028_00020.png
inflating:    train/7/00007_00028_00021.png
inflating:    train/7/00007_00028_00022.png
inflating:    train/7/00007_00028_00023.png
inflating:    train/7/00007_00028_00024.png
inflating:    train/7/00007_00028_00025.png
inflating:    train/7/00007_00028_00026.png
inflating:    train/7/00007_00028_00027.png
inflating:    train/7/00007_00028_00028.png
inflating: train/7/00007_00028_00029.png
```

```
inflating:    train/7/00007_00029_00000.png
inflating:    train/7/00007_00029_00001.png
inflating:    train/7/00007_00029_00002.png
inflating:    train/7/00007_00029_00003.png
inflating:    train/7/00007_00029_00004.png
inflating:    train/7/00007_00029_00005.png
inflating:  train/7/00007_00029_00006.png
```

```
inflating:    train/7/00007_00029_00007.png
inflating:    train/7/00007_00029_00008.png
inflating:    train/7/00007_00029_00009.png
inflating:    train/7/00007_00029_00010.png
inflating:    train/7/00007_00029_00011.png
inflating:    train/7/00007_00029_00012.png
inflating:    train/7/00007_00029_00013.png
inflating:    train/7/00007_00029_00014.png
inflating:    train/7/00007_00029_00015.png
inflating:    train/7/00007_00029_00016.png
inflating:    train/7/00007_00029_00017.png
inflating:    train/7/00007_00029_00018.png
inflating:    train/7/00007_00029_00019.png
inflating:    train/7/00007_00029_00020.png
inflating:    train/7/00007_00029_00021.png
inflating:    train/7/00007_00029_00022.png
inflating:    train/7/00007_00029_00023.png
inflating:    train/7/00007_00029_00024.png
inflating:    train/7/00007_00029_00025.png
inflating:    train/7/00007_00029_00026.png
inflating:    train/7/00007_00029_00027.png
inflating:    train/7/00007_00029_00028.png
inflating:    train/7/00007_00029_00029.png
inflating:    train/7/00007_00030_00000.png
inflating:    train/7/00007_00030_00001.png
inflating:    train/7/00007_00030_00002.png
inflating:    train/7/00007_00030_00003.png
inflating:    train/7/00007_00030_00004.png
inflating:    train/7/00007_00030_00005.png
inflating:    train/7/00007_00030_00006.png
inflating:    train/7/00007_00030_00007.png
inflating:    train/7/00007_00030_00008.png
inflating:    train/7/00007_00030_00009.png
inflating:    train/7/00007_00030_00010.png
inflating:    train/7/00007_00030_00011.png
inflating:    train/7/00007_00030_00012.png
inflating:    train/7/00007_00030_00013.png
inflating:    train/7/00007_00030_00014.png
inflating:    train/7/00007_00030_00015.png
inflating:    train/7/00007_00030_00016.png
inflating:    train/7/00007_00030_00017.png
inflating:    train/7/00007_00030_00018.png
inflating:    train/7/00007_00030_00019.png
inflating:    train/7/00007_00030_00020.png
inflating:    train/7/00007_00030_00021.png
inflating:    train/7/00007_00030_00022.png
inflating:    train/7/00007_00030_00023.png
inflating:    train/7/00007_00030_00024.png
inflating:    train/7/00007_00030_00025.png
inflating:    train/7/00007_00030_00026.png
inflating:    train/7/00007_00030_00027.png
inflating:    train/7/00007_00030_00028.png
inflating:    train/7/00007_00030_00029.png
inflating: train/7/00007_00031_00000.png
```