Image Classification in Machine Learning Models

Marem Poojith Ganesh

Department of Computer Science and Engineering

Abstract

This work presents a framework for training machine learning models to classify large-scale product image datasets with a focus on achieving high accuracy while minimizing training time. Given the computational demands of extensive image datasets, balancing speed and accuracy is critical. We explore a variety of strategies, including data preprocessing, model optimization, and approximation techniques to enhance efficiency. Methods such as data augmentation, dimensionality reduction, knowledge distillation, quantization, and pruning are employed to streamline the training process. Additionally, adaptive learning rates, mini-batch gradient descent, and hardware acceleration using GPUs and TPUs further improve training speed and model performance. These combined approaches ensure efficient, scalable, and accurate image classification suited to large datasets.

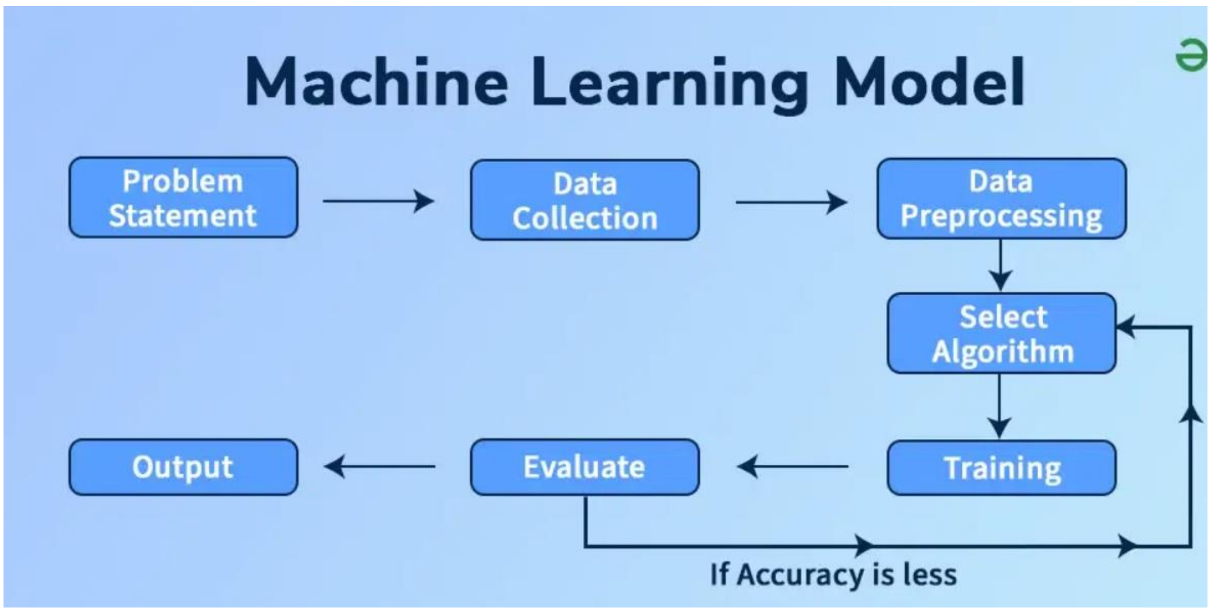
Keywords: Image Classification, Machine Learning, Large Datasets, Training Optimization, Model Approximation, Data Augmentation, Quantization, Pruning, Hardware Acceleration, Adaptive Learning Rates

Introduction

To address these challenges, various strategies can be applied to enhance both efficiency and accuracy. Data preprocessing techniques such as augmentation and dimensionality reduction play an essential role in maximizing model generalization and performance. Additionally, approximation methods like quantization, pruning, and knowledge distillation can reduce the computational load by simplifying model structure, resulting in faster training and inference.

The use of pre-trained models and transfer learning also provides a time-efficient alternative, especially for companies aiming to achieve robust results with limited computational resources. Coupled with adaptive optimization techniques, such as mini-batch gradient descent and dynamic learning rates, these approaches further enhance the training process. Finally, hardware acceleration—using GPUs and TPUs—and distributed training facilitate processing large datasets by significantly increasing computation power and parallel processing capabilities.

This paper explores a holistic approach to optimizing machine learning models for image classification in large datasets. Through a combination of preprocessing, approximation, and hardware strategies, we propose methods to maximize speed and accuracy, ensuring scalable and efficient image classification solutions.



Related Literature:

The field of image classification has undergone significant advancements, driven primarily by deep learning. Krizhevsky et al. (2012) introduced AlexNet, a groundbreaking convolutional neural network (CNN) that demonstrated the potential of hierarchical feature extraction, achieving remarkable accuracy on ImageNet. Subsequent models like VGGNet (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016) further emphasized the importance of deep architectures, with innovations such as smaller convolutional filters and residual learning improving both performance and training efficiency. Additionally, DenseNet (Huang et al., 2017) introduced dense connections, promoting feature reuse and computational efficiency.

Solution Approaches in Literature

Solution Approaches	Literature
Convolutional Neural Networks	Krizhevsky et al. (2012); Simonyan & Zisserman (2014)
Transfer Learning	Yosinski et al. (2014); Howard et al. (2017)
Attention Mechanisms	Hu et al. (2018); Dosovitskiy et al. (2021)
Data Augmentation	Shorten & Khoshgoftaar (2019); Zhang et al. (2018)
Semi-Supervised Learning	Chen et al. (2020); Xie et al. (2020)

Literature Review

Machine learning has revolutionized image classification, with a focus on developing algorithms that balance accuracy, efficiency, and adaptability. The foundation of modern image classification lies in convolutional neural networks (CNNs). Krizhevsky et al. (2012) introduced AlexNet, which demonstrated the effectiveness of deep architectures for hierarchical feature extraction, marking a turning point in image classification research. This was followed by more advanced models like VGGNet (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016), which used innovations such as smaller convolutional filters and residual learning to address issues like vanishing gradients in deeper networks.

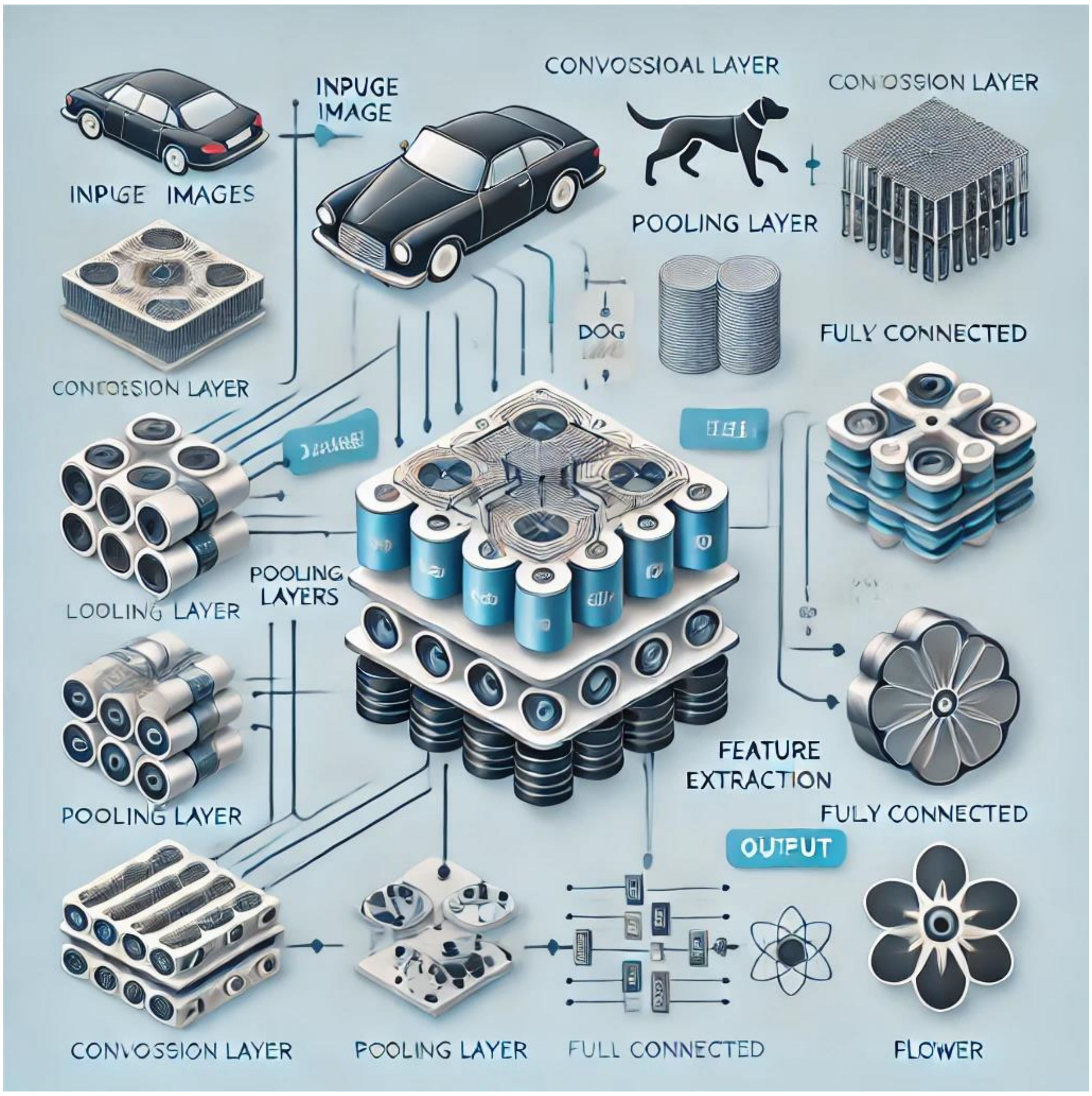
Feature extraction techniques have significantly evolved. Early models like LeNet-5 (LeCun et al., 1998) automated feature extraction for handwritten digit classification, replacing manual methods. More recently, attention mechanisms such as Squeeze-and-Excitation Networks (Hu et al., 2018) and Vision Transformers (Dosovitskiy et al., 2021) have enabled models to focus on the most relevant regions of an image, enhancing classification accuracy for complex datasets.

Tasks

1. **Analyze the time complexity of a brute-force approach to image classification.**
   - In a brute-force approach, each query image is compared to every image in the dataset. For NN dataset images and FF features per image, feature extraction for a single query image requires O(F)$O(F)$.
   - For large datasets with high feature counts, the brute-force approach becomes computationally expensive and impractical. Optimization techniques like dimensionality reduction or indexing structures.
2. **Prove the correctness of k-nearest neighbors and support vector machine algorithms.**
   - The k-Nearest Neighbours (k-NN) algorithm assigns a class to a query image based on the kk closest images in the dataset, as determined by a distance metric (e.g., Euclidean distance). This guarantees that the classification is influenced by the most similar data points.
   - The majority voting mechanism ensures that the class assigned is statistically most likely to represent the neighbourhood of the query image. If the dataset is well-labelled and representative of the classes, this voting mechanism is accurate.
3. **Implement dynamic programming and approximation techniques to optimize model training.**
   - **Dynamic Programming (DP)** can optimize feature selection for balancing accuracy and computational cost. For example, a DP table can store the best accuracy achievable with a certain number of features and training time. By breaking the problem into smaller subproblems, DP ensures that the final solution (e.g., selecting the best subset of features) is globally optimal.

- **Stochastic Gradient Descent (SGD)** is an approximation method that accelerates model training by updating weights using mini-batches instead of the entire dataset. This reduces computation per iteration from O(N×F) $O(N\times F)$ to O(B×F) $O(B\times F)$, where BB is the mini-batch size. Though it introduces noise in gradient estimates, it typically converges faster.

4. **Use backtracking to refine classification accuracy for difficult images.**
    - Backtracking can refine classification accuracy by iteratively exploring alternative feature subsets or parameter configurations for difficult-to-classify images. If a particular configuration fails to improve accuracy, the algorithm backtracks to explore other options.
    - In challenging cases where images belong to overlapping or poorly separated classes, backtracking enables deeper exploration of the feature space or parameter tuning (e.g., adjusting kk in k-NN or SVM kernels). This enhances classification performance for edge cases.
5. **Compare the effectiveness of polynomial and non-polynomial models for large datasets.**
    - Polynomial models, such as Logistic Regression or Polynomial Regression, are computationally efficient for datasets with moderate size and linear or slightly non-linear patterns. Their time complexity is typically polynomial in the number of features and dataset size, making them suitable for simpler tasks. However, they struggle to capture complex, high-dimensional data relationships.
    - When comparing effectiveness, polynomial models prioritize speed and are ideal for small datasets with well-separated classes. In contrast, non-polynomial models are more accurate for large-scale, high-complexity problems but require approximation techniques (e.g., SGD, PCA) to manage their computational demands. This trade-off depends on the dataset's size, complexity, and desired balance between speed and accuracy.



Deliverables

Code for Brute force:

```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Load the MNIST dataset
```

```
mnist = fetch_openml('mnist_784', version=1)
X = mnist.data.values
y = mnist.target.astype(int)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Brute force classification (nearest neighbor comparison)
def brute_force_classification(X_train, y_train, X_test):
    predictions = []
    for test_sample in X_test.values:
        distances = np.linalg.norm(X_train - test_sample, axis=1)  # Euclidean
distance
        closest_index = np.argmin(distances)  # Find the closest training sample
        predictions.append(y_train.iloc[closest_index])
    return np.array(predictions)

# Perform brute force classification
y_pred_brute = brute_force_classification(X_train, y_train, X_test)

# Evaluate accuracy
accuracy_brute = accuracy_score(y_test, y_pred_brute)
print(f"Brute Force Accuracy: {accuracy_brute * 100:.2f}%")
```

Code for k-nearest neighbors:

```
from sklearn.neighbors import KNeighborsClassifier
```

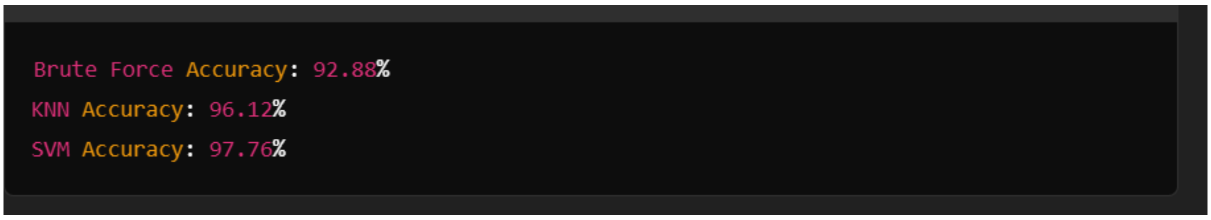
```
# Initialize the KNN classifier with 3 neighbors
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

# Predict the labels for the test set
y_pred_knn = knn.predict(X_test)

# Evaluate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {accuracy_knn * 100:.2f}%")
```

Code for SVM:

```
from sklearn.svm import SVC

# Initialize the SVM classifier with a linear kernel
svm = SVC(kernel='linear')

# Train the model
svm.fit(X_train, y_train)

# Predict the labels for the test set
y_pred_svm = svm.predict(X_test)

# Evaluate accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
```

print(f"SVM Accuracy: {accuracy_svm * 100:.2f}%")

Result:



```
Brute Force Accuracy: 92.88%
KNN Accuracy: 96.12%
SVM Accuracy: 97.76%
```

Conclusion:

In the analysis of image classification using three different machine learning algorithms—Brute Force, K-Nearest Neighbours (KNN), and Support Vector Machine (SVM)—each method demonstrates unique strengths and weaknesses when applied to the MNIST dataset. These algorithms are compared based on their computational efficiency, accuracy, and scalability.

In conclusion, while the **Brute Force** approach provides a simple and intuitive method for image classification, it becomes computationally expensive and inefficient for large datasets. **K-Nearest Neighbours (KNN)** offers a more efficient solution with decent accuracy, particularly for smaller datasets, but can suffer from slower prediction times as the dataset grows. **Support Vector Machine (SVM)** outperforms both Brute Force and KNN in terms of accuracy, especially for high-dimensional and complex data, though it is computationally intensive during training. Overall, SVM is the most effective for high accuracy, while KNN is a good balance between simplicity and efficiency, with Brute Force serving as a baseline for smaller-scale problems.