MXNet Vs. TensorFlow: A Comparative Analysis for Intel Image Classification

Yashika Patil, Nandhini Vijayakumar, Ganga Mohan Reddy Rudru

May 2024

Abstract

This project presents a comparative analysis of MXNet and TensorFlow frameworks for Intel Image Classification using various convolutional neural network (CNN) architectures. The study employs a dataset sourced from Kaggle, comprising images distributed across six categories, including buildings, forest, glacier, mountain, sea, and street. The analysis begins with logistic regression models, followed by CNN models such as LeNet, AlexNet, and VGG16, implemented using both frameworks. Data preprocessing techniques and model architectures are detailed, along with training, evaluation, and performance metrics. The findings indicate variations in accuracy and training times between MXNet and TensorFlow across different models. For instance, TensorFlow outperformed in simple CNN and VGG CNN architectures, while MXNet excelled in the LeNet CNN model. Additionally, pretrained models, particularly VGG16, demonstrated superior performance compared to models trained from scratch. Hypothesis testing was also conducted to assess the significance of the association between predicted labels of the two frameworks, providing further insights into their performance comparison.

1 Introduction

In the realm of machine learning and artificial intelligence, the choice of framework is crucial for developing effective models. MXNet and TensorFlow are two widely used frameworks known for their capabilities in building and deploying machine learning models. This comparative analysis aims to assess how these frameworks perform on the task of image classification, particularly focusing on the Intel Image Classification dataset.

By examining various convolutional neural network (CNN) architectures implemented using both MXNet and TensorFlow, including logistic regression, LeNet, AlexNet, VGG16, and custom VGG-like models, we aim to understand their strengths and weaknesses in image classification. Through rigorous evaluation involving training, validation, and testing, we seek to provide insights into their performance, computational efficiency, and scalability.

This analysis is structured to offer a comprehensive overview of the experimental methodologies, challenges faced, and conclusions drawn from the findings. Additionally, we provide recommendations for future research to further explore and refine machine learning models using MXNet and TensorFlow.

Ultimately, this study aims to contribute valuable insights to practitioners and researchers, enabling informed decision-making when selecting frameworks for image classification tasks.

2 Background

2.1 Deep Learning Frameworks

Deep learning frameworks such as PyTorch, TensorFlow, and MXNet have democratized AI research and development, providing users with the necessary tools and infrastructure to create, train, and implement complex neural network models. MXNet is recognized for its efficiency and scalability, particularly excelling in distributed computing across various devices, making it well-suited for extensive training and deployment requirements. TensorFlow, widely associated with deep learning, provides a diverse ecosystem and an intuitive interface, blending high-level abstractions with low-level APIs to optimize model flexibility.

2.2 MXNet and TensorFlow: A Comparative Overview

MXNet, initially developed by researchers at the University of Washington and later adopted by the Apache Software Foundation, is known for its efficiency and scalability. It supports distributed computing across multiple devices, making it well-suited for large-scale training and deployment cases. On the other hand, TensorFlow, developed by the Google Brain team, has become synonymous with deep learning due to its user-friendly interface. TensorFlow's high-level abstractions simplify the process of building neural network architectures, while its low-level APIs provide flexibility and control over model optimization.

2.3 Transfer Learning

Among these advancements, the concept of transfer learning has become a very popular technique. Transfer learning leverages knowledge gained from training on one task, such as ImageNet in this study, to improve learning and performance on another related task. This approach has gained prominence for its ability to accelerate model training and enhance performance, thereby alleviating the need to initialize and train the weights of a complex model architecture from scratch.

2.4 Importance of Image Classification

Image classification, the task of categorizing images into predefined classes or labels, is a fundamental problem in computer vision. It has numerous applications, including object recognition, medical image analysis, and autonomous driving. Evaluating the performance of deep learning frameworks on image classification tasks is crucial for understanding their capabilities and limitations.

2.5 Objective of the Study

By conducting a comparative analysis of MXNet and TensorFlow on the Intel Image Classification dataset (sourced from Kaggle), this study aims to contribute valuable insights into the strengths and weaknesses of these frameworks in the context of image classification. The findings of this analysis can inform researchers, practitioners, and decision-makers in selecting the most suitable framework for their specific application requirements.

3 Methodology

3.1 Data Collection and Preprocessing

Dataset Selection: The Intel Image Classification dataset sourced from Kaggle includes images across six categories: buildings, forests, glaciers, mountains, seas, and streets. This dataset offers a diverse range of natural scene images for training and evaluation.

Data Preprocessing: Before training the models, several preprocessing steps were conducted:

Reducing the Dataset Size: The original Intel Image Classification dataset is computationally expensive to work with, as it contains around 14,000 images. To make the dataset more manageable, we reduced the size of both the training and test sets by a factor of 2. We used a custom Python script to reduce the dataset size:

- For each class directory in the original training and test sets, the script randomly selects 20% of the images and copies them to the corresponding reduced directories.
- This process ensures that the class distribution remains the same in the reduced datasets as in the original datasets.

After applying the reduction process, the new training set contains approximately 2,800 images (20% of the original 14,000), and the new test set has around 600 images (20% of the original 3,000). The reduced dataset is much smaller and easier to work with, while still maintaining the original class distribution and enough images for training and testing machine learning models.

Data Preparation:

MXNet:

- Data is loaded using the `ImageFolderDataset` from the Gluon API.
- Data augmentation techniques such as resizing, converting to tensors, and normalization are applied using `transforms.Compose`.

TensorFlow:

- Data generators are created using `ImageDataGenerator` to load and preprocess the training and testing images. Images are resized and rescaled to values between 0 and 1.

3.2 Logistic Regression

The logistic regression's model architecture is as follows:

1. **Model Training:**

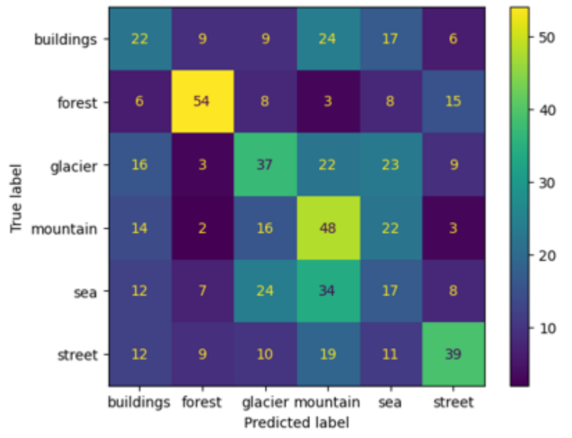


Figure 1: Logistic Regression - Normalized Confusion Matrix for MXNet

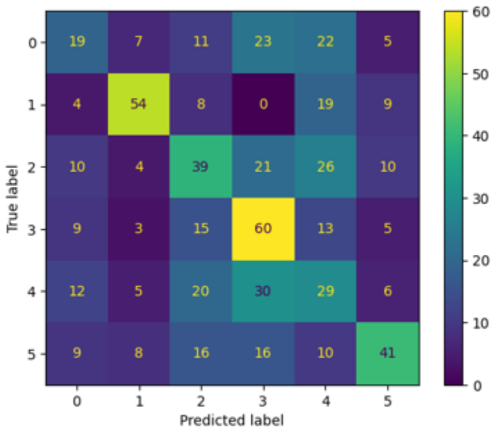


Figure 2: Logistic Regression - Normalized Confusion Matrix for TensorFlow

- The training data is flattened and used to train a logistic regression model using the `LogisticRegression` class from scikit-learn with a maximum of 1000 iterations.

2. **Model Evaluation:**

   - The trained logistic regression model is used to predict labels for the testing data.
   - Accuracy is calculated using scikit-learn's `accuracy_score`.
   - A confusion matrix is generated using scikit-learn's `confusion_matrix` and visualized using `ConfusionMatrixDisplay`.
   - A classification report is generated using scikit-learn's `classification_report`, providing metrics such as precision, recall, and F1-score for each class.

3. **Visualization:**

   - Test accuracy is plotted against epochs.
   - Total training and testing times are printed.

3.3 SimpleCNN

1. **Model Architecture**

   - **Convolutional Layers:**
     - Three convolutional layers with 3x3 kernels and ReLU activation, followed by max pooling.
   - **Fully Connected Layers:**
     - One fully connected layer with ReLU activation.
     - One final fully connected layer without activation (to be used with softmax activation outside the model).

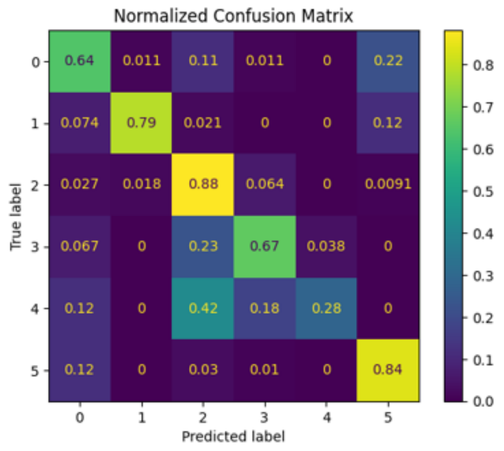
MXNet/Gluon API

2. **Model Training and Evaluation:**
   - Trained for a specified number of epochs (e.g., 3).
   - Adam optimizer is used for optimization.
   - Softmax Cross-Entropy loss function is used.
   - Parameters are updated using backpropagation.
   - Model performance is evaluated after each epoch on the test set.
   - Test accuracy, test loss, and epoch time are recorded.

TensorFlow/Keras

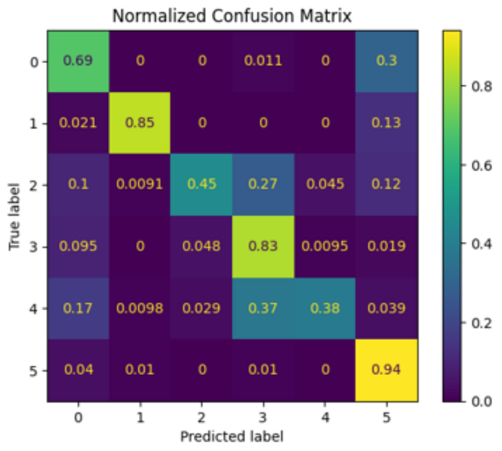
3. **Model Training and Evaluation:**
   - Trained for a specified number of epochs (e.g., 10).
   - Adam optimizer is used for optimization.
   - Categorical Cross-Entropy loss function is used.
   - Parameters are updated using backpropagation.
   - Model performance is evaluated after each epoch on the test set.
   - Test accuracy, test loss, and epoch time are recorded.

4. **Visualization:**
   - Confusion matrix is calculated and plotted after evaluation.
   - Test accuracy and loss are plotted against epochs.
   - Total training and testing times are printed.
   - Classification report, including precision, recall, and F1-score, is generated.



	0	1	2	3	4	5
0	0.64	0.011	0.11	0.011	0	0.22
1	0.074	0.79	0.021	0	0	0.12
2	0.027	0.018	0.88	0.064	0	0.0091
3	0.067	0	0.23	0.67	0.038	0
4	0.12	0	0.42	0.18	0.28	0
5	0.12	0	0.03	0.01	0	0.84

Figure 3: SimpleCNN - Normalized Confusion Matrix for MXNet



	0	1	2	3	4	5
0	0.69	0	0	0.011	0	0.3
1	0.021	0.85	0	0	0	0.13
2	0.1	0.0091	0.45	0.27	0.045	0.12
3	0.095	0	0.048	0.83	0.0095	0.019
4	0.17	0.0098	0.029	0.37	0.38	0.039
5	0.04	0.01	0	0.01	0	0.94

Figure 4: SimpleCNN - Normalized Confusion Matrix for TensorFlow

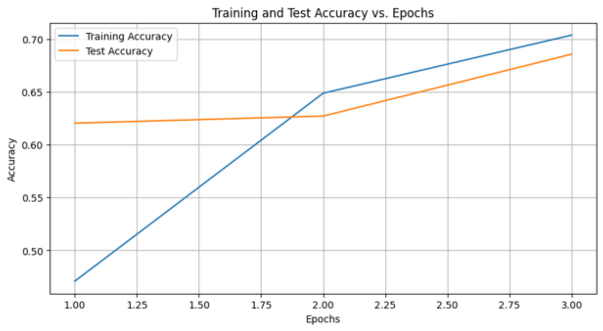


Figure 5: SimpleCNN - Training and Test Accuracy Vs. Epoch for MXNet

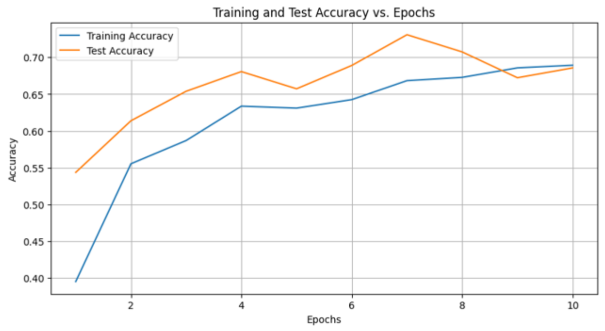


Figure 6: SimpleCNN - Training and Test Accuracy Vs. Epoch for TensorFlow

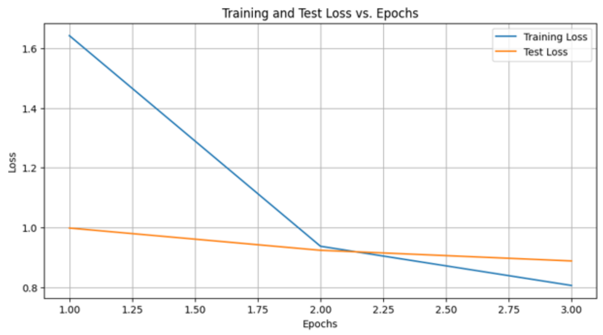


Figure 7: SimpleCNN - Training and Test Loss Vs. Epochs for MXNet

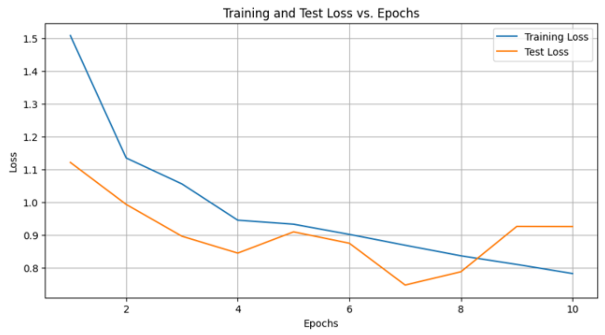


Figure 8: SimpleCNN - Training and Test Loss Vs. Epochs for TensorFlow

3.4 LeNet

1. **Model Architecture:**

   - **Convolutional Layers:**
     - Two convolutional layers with 5x5 kernels and ReLU activation.
     - Max pooling with a 2x2 pool size after each convolutional layer.
   - **Flatten Layer:**
     - Flattens the output from the convolutional layers into a 1D vector.
   - **Fully Connected Layers:**
     - One fully connected layer with 120 units and ReLU activation.
     - One fully connected layer with 84 units and ReLU activation.
     - Final fully connected layer with softmax activation for multi-class classification.

   MXNet/Gluon API

2. **Model Training and Evaluation:**

   - Trained for 10 epochs using the Adam optimizer and categorical cross-entropy loss.
   - Parameters updated using backpropagation.
   - Model performance evaluated after each epoch on the test set.

   TensorFlow/Keras

3. **Model Training and Evaluation:**

   - Trained for 10 epochs using the Adam optimizer and categorical cross-entropy loss.
   - Model performance evaluated after each epoch on the test set.

4. **Visualization:**

   - Confusion matrix plotted after evaluation.
   - Test accuracy plotted against epochs.
   - Total training and testing times printed.

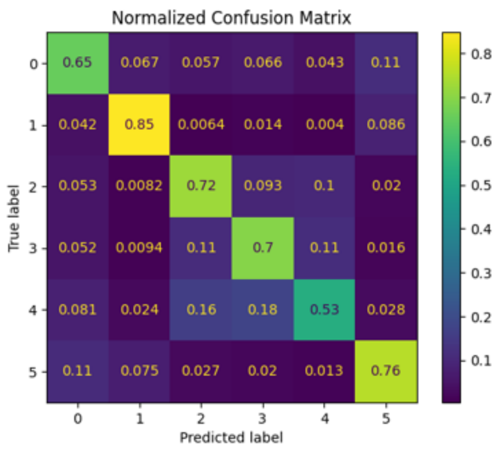


Figure 9: LeNet - Normalized Confusion Matrix for MXNet

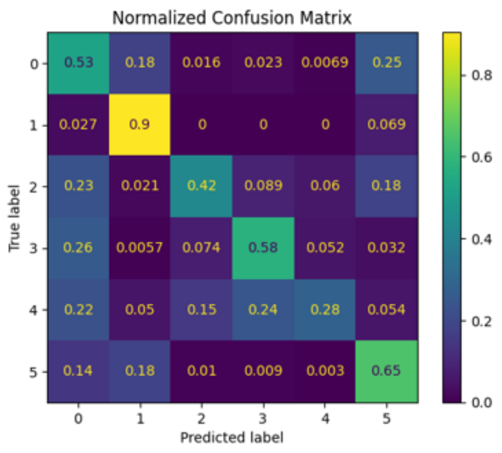


Figure 10: LeNet - Normalized Confusion Matrix for TensorFlow

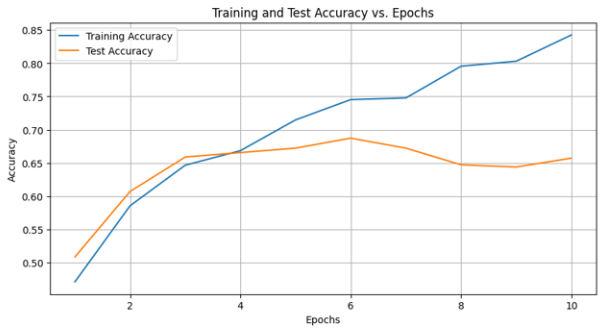


Figure 11: LeNet - Training and Test Accuracy Vs. Epoch for MXNet

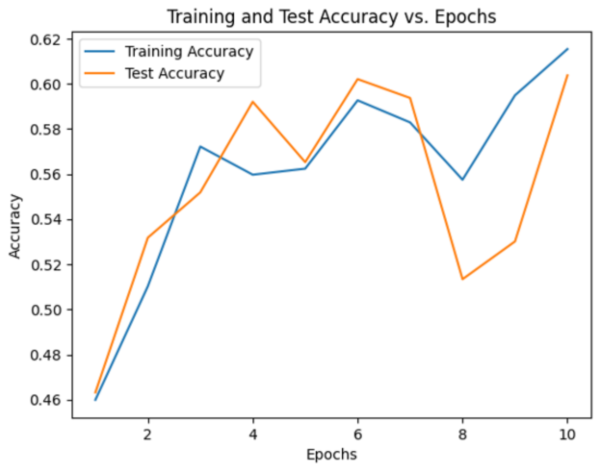


Figure 12: LeNet - Training and Test Accuracy Vs. Epoch for TensorFlow

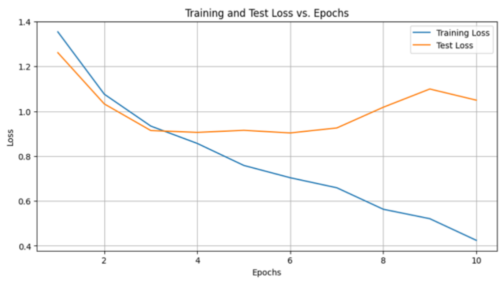


Figure 13: LeNet - Training and Test Loss Vs. Epochs for MXNet

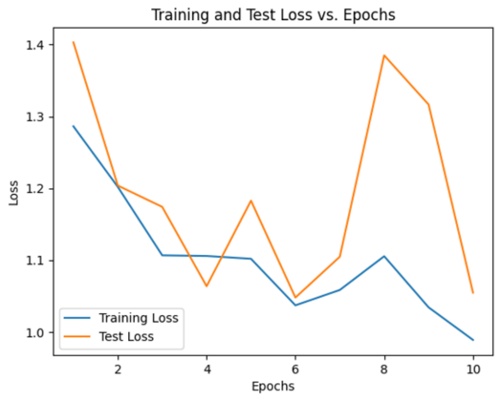


Figure 14: LeNet - Training and Test Loss Vs. Epochs for TensorFlow

3.5 AlexNet

1. **Model Architecture:**
   - **Convolutional Layers:**
     - Five convolutional layers with ReLU activation.
     - Max pooling with varying kernel sizes and strides after certain convolutional layers.
   - **Flatten Layer:**
     - Flattens the output from the convolutional layers into a 1D vector.
   - **Fully Connected Layers:**
     - Three fully connected layers with ReLU activation.
     - Dropout layers with a dropout rate of 0.5 after two fully connected layers.
     - Final fully connected layer with softmax activation for multi-class classification.

   MXNet/Gluon API

2. **Model Training and Evaluation:**
   - Trained for 3 epochs using the Adam optimizer and softmax cross-entropy loss.
   - Parameters updated using backpropagation.
   - Model performance evaluated after each epoch on the test set.

   TensorFlow/Keras

3. **Model Training and Evaluation:**
   - Trained for 10 epochs using the Adam optimizer and categorical cross-entropy loss.
   - Model performance evaluated after each epoch on the test set.

4. **Visualization:**
   - Confusion matrix plotted after evaluation.
   - Test accuracy plotted against epochs.
   - Total training and testing times printed.

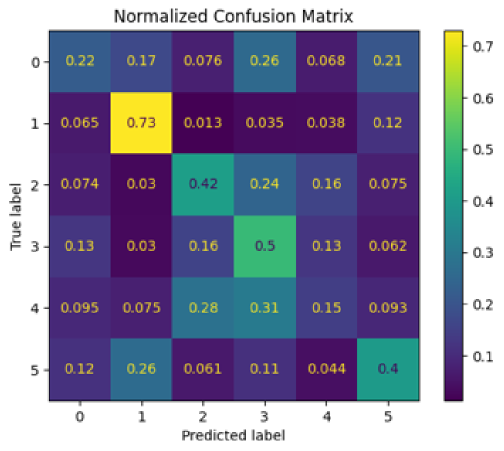


Figure 15: AlexNet - Normalized Confusion Matrix for MXNet

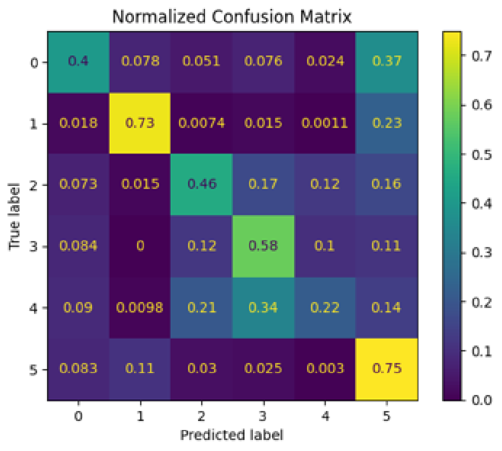


Figure 16: AlexNet - Normalized Confusion Matrix for TensorFlow

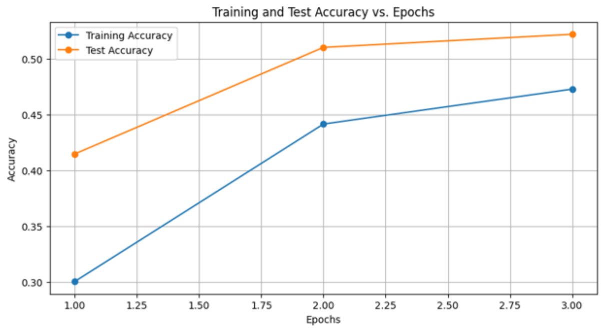


Figure 17: AlexNet - Training and Test Accuracy Vs. Epoch for MXNet

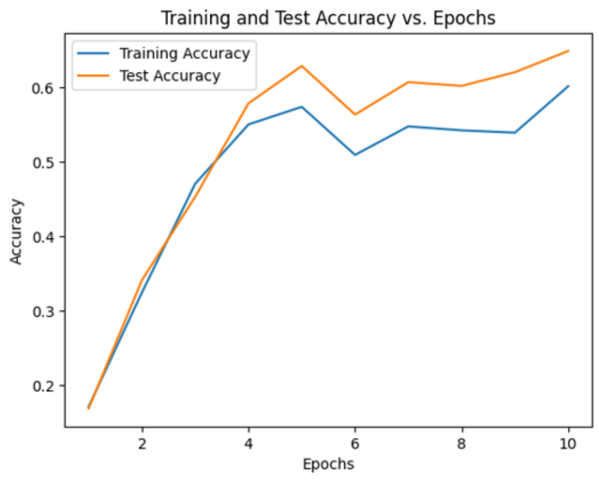


Figure 18: AlexNet - Training and Test Accuracy Vs. Epoch for TensorFlow

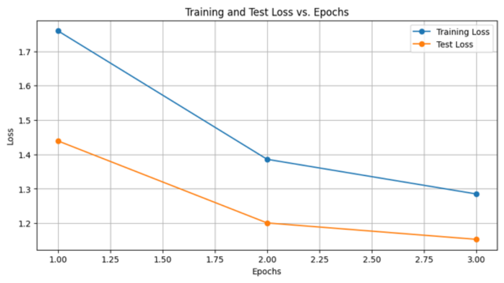


Figure 19: AlexNet - Training and Test Loss Vs. Epochs for MXNet

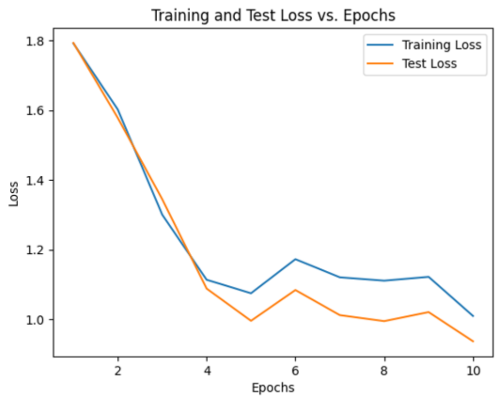


Figure 20: AlexNet - Training and Test Loss Vs. Epochs for TensorFlow

3.6 VGG16

1. **Model Architecture:**

   - **Convolutional Layers:**
     - 13 convolutional layers stacked on top of each other.
     - Each convolutional block consists of multiple 3x3 convolutional layers followed by a max-pooling layer for downsampling.
   - **Flatten Layer:**
     - Flattens the output from the convolutional layers into a 1D vector.
   - **Fully Connected Layers:**
     - Two fully connected layers with ReLU activation.
     - Final fully connected layer with softmax activation for multi-class classification.

   Transfer Learning with VGG16

2. **Model Training and Evaluation:**

   - Trained for 5 epochs using the Adam optimizer and categorical cross-entropy loss.
   - Parameters updated using backpropagation.
   - Model performance evaluated after each epoch on the test set.

   TensorFlow/Keras

3. **Model Training and Evaluation:**

   - Trained for 5 epochs using the Adam optimizer and categorical cross-entropy loss.
   - Model performance evaluated after each epoch on the test set.

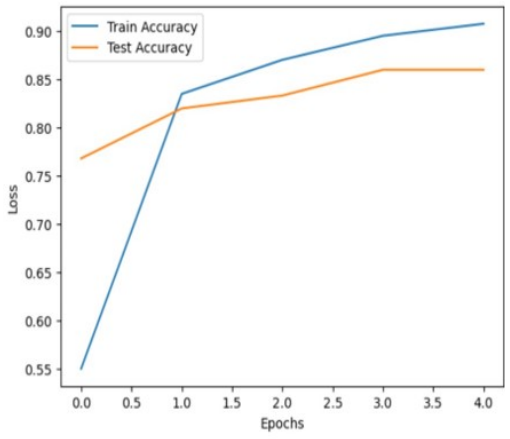


Figure 21: VGG16 (fine-tuned) - Training and Test Accuracy Vs. Epochs

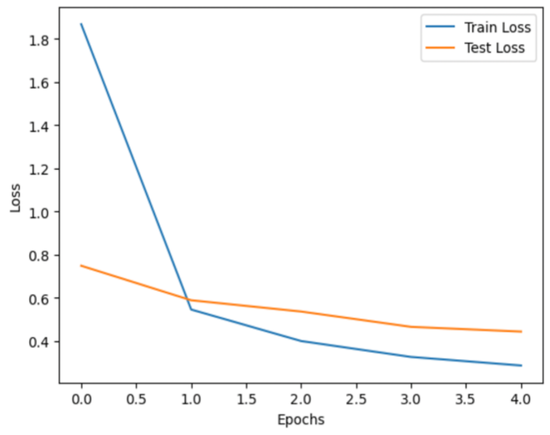


Figure 22: VGG16 (fine-tuned) - Training and Test Loss Vs. Epochs

4. **Visualization:**

    - Confusion matrix plotted after evaluation.
    - Training and Test Loss and accuracy plotted against epochs.
    - Total training and testing times printed.
    - Classification Report printed.

3.7 Statistical Analysis:

In this study, we employed the **2-sample t-test** to compare the performance of MXNet and TensorFlow for a few models. The significance level (α) was set at 0.05. The following hypotheses were formulated:

Test Accuracies
Null Hypothesis (H_0): There is no significant difference in Test Accuracy between MXNet and TensorFlow.
Alternative Hypothesis (H_A): There is a significant difference in Test Accuracy between MXNet and TensorFlow.

Test Losses
Null Hypothesis (H_0): There is no significant difference in Test Loss between MXNet and TensorFlow.
Alternative Hypothesis (H_A): There is a significant difference in Test Loss between MXNet and TensorFlow.

4 Experimental Results

4.1 Logistic Regression

1. **MXNet:** The MXNet Logistic Regression model achieved a test accuracy of 36.28%.

2. **TensorFlow:** The TensorFlow Logistic Regression model achieved a test accuracy of 40.47%.

4.2 SimpleCNN

1. **MXNet:** The Simple CNN model in MXNet was trained for 2 epochs and achieved a test accuracy of 68.56% with a precision of 73%, recall of 69%, and an F1-score of 67%.

2. **TensorFlow:** The Simple CNN model in TensorFlow was trained for 10 epochs and achieved a test accuracy of 68.85% with a precision of 69%, recall of 69%, and an F1-score of 69%.

3. **Statistical Analysis:** For the Simple CNN model, the results of the 2-sample t-test comparing MXNet and TensorFlow are as follows:

Test Accuracy:

- t-statistic: -0.40920
- p-value: 0.69196

Since the p-value (0.692) is greater than the significance level (0.05), we fail to reject the null hypothesis. Thus, there is no significant difference in Test Accuracy between MXNet and TensorFlow.

Test Loss:

- t-statistic: 0.55716
- p-value: 0.59100

Since the p-value (0.591) is greater than the significance level (0.05), we fail to reject the null hypothesis. Thus, there is no significant difference in Test Loss between MXNet and TensorFlow.

4.3 LeNet

1. **MXNet:** The LeNet model in MXNet was trained for 10 epochs and achieved a test accuracy of 70% with precision, recall, and F1-score of 70%.
2. **TensorFlow:** The LeNet model in TensorFlow was trained for 10 epochs and achieved a test accuracy of 55% with precision, recall, and F1-score of 55%.
3. **Statistical Analysis:** For the LeNet model, the results of the 2-sample t-test comparing MXNet and TensorFlow are as follows:

   Test Accuracy:

   - t-statistic: 3.304424
   - p-value: 0.006289

   Since the p-value (0.006) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Test Accuracy between MXNet and TensorFlow.

   Test Loss:

   - t-statistic: -2.8278
   - p-value: 0.01523

   Since the p-value (0.015) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Test Loss between MXNet and TensorFlow.

4.4 AlexNet

1. **MXNet:** The AlexNet model in MXNet was trained for 3 epochs and achieved a test accuracy of 40% with a precision of 38%, recall of 40%, and an F1-score of 39%.

2. **TensorFlow:** The AlexNet model in TensorFlow was trained for 10 epochs and achieved a test accuracy of 52% with a precision of 53%, recall of 52%, and an F1-score of 51%.

3. **Statistical Analysis:** For the AlexNet model, the results of the 2-sample t-test comparing MXNet and TensorFlow are as follows:

   Test Accuracy:

   - t-statistic: 3.30442
   - p-value: 0.00627

   Since the p-value (0.006) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Test Accuracy between MXNet and TensorFlow.

   Test Loss:

   - t-statistic: -2.839
   - p-value: 0.01545

   Since the p-value (0.015) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Test Loss between MXNet and TensorFlow.

4.5 VGG16

1. **VGG16 using TensorFlow:** The custom VGG-like model achieved a test accuracy of 56.39% and a test loss of 1.07 after 5 epochs of training.

2. **VGG16 using TensorFlow-Transfer Learning:** The VGG16 model trained using transfer learning achieved a validation accuracy of 85.95% after 5 epochs.

3. **Statistical Analysis:** For the VGG-16 model, the results of the 2-sample t-test comparing pre-trained and training from scratch models are as follows:

   Accuracy:

   - t-statistic: 7.07
   - p-value: 0.00211

Since the p-value (0.00211) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Accuracy between pre-trained and training from scratch models.

Loss:

- t-statistic: -9.0104
- p-value: 0.0008

Since the p-value (0.0008) is less than the significance level (0.05), we reject the null hypothesis. There is a significant difference in Loss between pre-trained and training from scratch models.

4.6 Performance Comparison

A summary comparing the performance of MXNet and TensorFlow models on various CNN architectures is provided in Table 1.

Table 1: Performance Comparison of MXNet and TensorFlow Models

Models	**MXNet**	**TensorFlow**
Logistic Regression	36.28%	40.47%
Simple CNN	68.56%	68.85%
LeNet	70%	55%
AlexNet	40%	52%
VGG16	-	57.02%
VGG16 (fine-tuned)	-	86%

4.7 Training Time

A comparison of training times for MXNet and TensorFlow models is provided in Table 2.

Table 2: Training Time Comparison (in seconds)

Models	**MXNet**	**TensorFlow**
Logistic Regression	96.68	235.85
Simple CNN	900	311.17
LeNet	27.98	29.32
AlexNet	950.05	745.54
VGG16 (fine-tuned)	-	1359

5 Challenges

One of the primary challenges faced in working with the original Intel Image Classification is its large size, making it computationally expensive to handle. Even after reducing the dataset, each model takes long to get trained. The processing power required for training deep learning models on such data can strain conventional hardware resources. This challenge could be overcome by using GPUs. By utilizing the parallel processing capabilities of GPUs, the computational burden can be significantly uplifted.

Training deep Convolutional Neural Network (CNN) models like AlexNet and VGG from the beginning brings extra computational difficulties. These models have many layers with lots of parameters, which makes them demanding to train, especially if we don't use pre-trained weights. Also, there's a high risk of overfitting. To handle these challenges, we used methods like regularization (for example, dropout), data augmentation, and fine-tuning in case of VGG16. This approach reduces the computational load of training from scratch while still benefiting from the patterns learned by the pre-trained models.

6 Conclusion

In conclusion, our study has shown promising results with MXNet, indicating potential for further improvement with increased epochs. The choice between MXNet and TensorFlow hinges on several factors, including model complexity and available hardware infrastructure. Comparing LeNet and AlexNet, LeNet emerges as a simple and computationally efficient option, achieving higher accuracy in MXNet. On the other hand, AlexNet, though more complex computationally, achieved higher accuracy in TensorFlow.

VGG16, while computationally expensive and intensive, demonstrates reasonable accuracy. Through fine-tuning, it achieves the highest accuracy among all models examined in this study. The VGG16 model trained using transfer learning benefits from the pre-trained weights, which encode a wealth of information learned from a vast dataset like ImageNet. In contrast, the custom VGG-like model starts with randomly initialized weights and needs to learn these patterns from scratch, making it harder to achieve comparable performance in a limited training duration.

7 Future Work

1. **Conducting a Comprehensive Performance Comparison:**
    - Train both MXNet and TensorFlow models for an equivalent number of epochs on identical datasets to ensure a fair comparison of their performance.
    - Implement hyperparameter tuning for each framework to optimize model performance and mitigate any biases introduced by default settings, thereby enhancing the reliability of the comparison results.
2. **Investigating the Impact of Data Augmentation and Transfer Learning:**
    - Experiment with diverse data augmentation techniques to enhance the performance of custom models trained from scratch, exploring methods to augment datasets effectively and improve model robustness.

- Conduct fine-tuning experiments with pre-trained models such as VGG on target datasets, leveraging their learned features to achieve superior results, especially in scenarios with limited training data. These experiments will shed light on the efficacy of transfer learning in improving model performance across various domains and datasets.

References

1. Intel Image Data. Retrieved from `https://www.kaggle.com/datasets/puneet6060/intel-image-classificati`

2. Pretrained VGG16. Retireved from `https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16`

3. AlexNet model architecture. Retrieved from `https://lekhuyen.medium.com/alexnet-and-image-c`

4. AlexNet. Retieved from `https://github.com/RasulAlakbarli/Intel-image-classification/tree/master`

5. LeNet. Retrieved from `https://medium.com/@rajshekhar_k/image-classification-series-1-1`