Università degli Studi di Milano

Machine Learning Experimental Project

Chihuahuas vs Muffins Image Classifier
with Convolutional Neural Networks

Statistical Methods for Machine Learning

Professor nicolò cesa-bianchi

Danial Forouzanfar, 12988A
Danial.forouzanfar@studenti.unimi.it

2023-2024

Contents

Abstract

This work presents a convolutional neural network[1] model designed for binary classification to distinguish between images of Chihuahuas and muffin. Leveraging the computational power of Google Colab's Graphics Processing Unit[2], the model is implemented using the PyTorch framework, with k-fold cross-validation facilitated by the sklearn library. The report provides an overview of the dataset and preprocessing techniques, followed by an explanation of neural network theory with a focus on CNNs. It details three different CNN architectures tested, along with experiments using three distinct learning rates for each model. The results demonstrate the model's performance and effectiveness in this classification task.

1. Introduction

1.1. Background

Data science, is the science that help the businesses to solve their problems with studying real-world data. This field blends different techniques from mathematics, statistics, artificial intelligence, and computer science to analyze vast datasets and discover hidden insights. Through the analysis, data scientists can explore and answer questions about past events, reasons behind them, future predictions, and potential actions based on the finding.

Machine learning[3], is a branch of artificial intelligence focused on creating and studying statistical algorithms that can learn from data and make predictions on new, unseen data, enabling tasks to be performed without explicit instructions. The term "machine learning" was introduced in 1959 by Arthur Samuel, an IBM employee and a pioneer in computer gaming and AI. Today, machine learning has two main goals:

1. classifying data based on developed models and predicting future outcomes using these models.
2. make predictions for future outcomes based on these models. (2)

Machine learning models fall into three primary categories:

1. Supervised machine learning

Supervised learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately. Supervised learning helps organizations solve a variety of real-world problems, one simple example is spam detection in emails.

2. Unsupervised machine learning

Unsupervised learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for

[1] Convolutional Neural Network - CNN
[2] Graphics Processing Unit - GPU
[3] Machine Learning - ML

human intervention. This method's ability to discover similarities and differences in information make it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction.

3. Semi-supervised learning

Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of not having enough labeled data for a supervised learning algorithm. It also helps if it's too costly to label enough data. (3)

Neural network, is a sophisticated machine learning model inspired by the workings of the human brain. It employs a network of interconnected nodes that function like neurons, collaborating to recognize patterns and phenomena. These artificial neurons process information by considering various factors and options before making a decision. This approach enables neural networks to tackle complex tasks, such as image recognition, by learning from data over time. (4)

Neural networks rely on training data to learn and improve their accuracy over time. Once they are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. (5)

Deep learning, builds upon the foundation of neural networks. With advancements in technology, particularly in computing power and algorithmic sophistication, it has become possible to delve deeper into these neural networks. This deeper exploration involves the utilization of multiple layers of interconnected nodes, allowing for more complex processing of data. Actually, deep learning extends the capabilities of traditional neural networks by enabling them to learn intricate features and hierarchies from vast amounts of data, resulting in more accurate and sophisticated predictions and classifications. (6)

Machine learning, neural networks, and deep learning are all sub-fields of artificial intelligence. However, neural networks is actually a sub-field of machine learning, and deep learning is a sub-field of neural networks. (7)

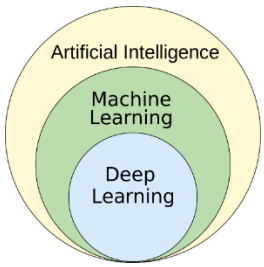

Figure 1. Artificial Intelligence, Machin Learning, Neural Network, and Deep Learning

Convolutional Neural Network[4], also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others. (8)

1.2. Problem statement and objective

In this project, the task is to develop a binary classification model using CNN to accurately distinguish between images of Chihuahuas and muffins. The challenge lies in training a robust model capable of accurately classifying diverse images of these two distinct categories as in many images they are really look like and sometimes it's hard to distinguish them even for children. Also, the publisher of this dataset wrote "How to confuse ML" in Kaggle.

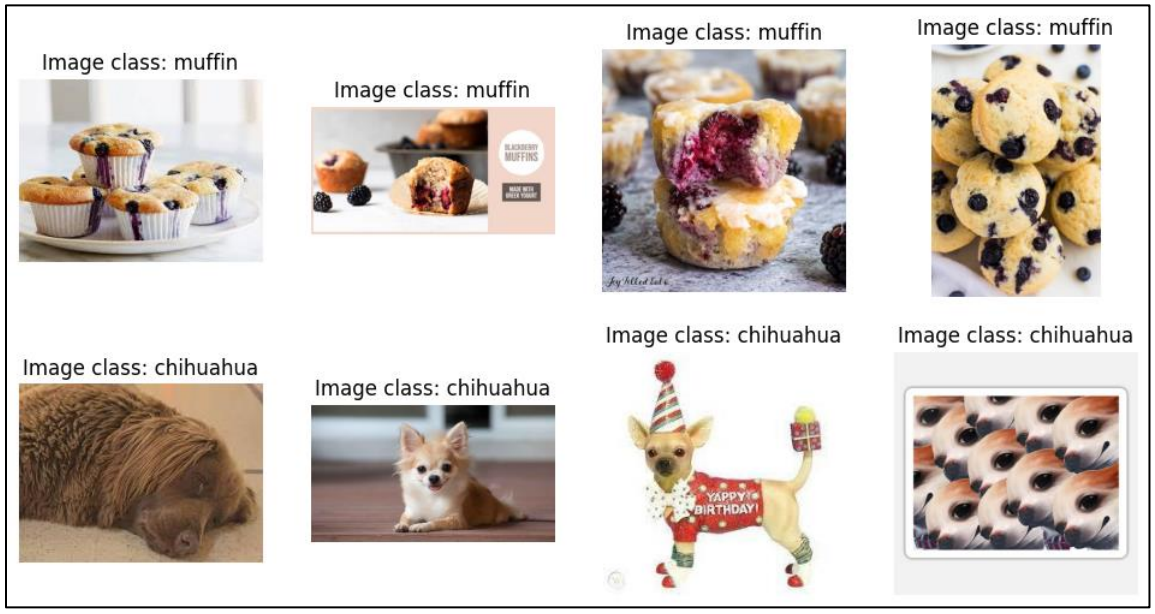

Figure 2. Some images of the dataset

The objective is to create different CNN models with different architectures that can effectively generalize from the training data to correctly classify unseen images into one of the two classes: Chihuahua or muffin. This task is significant as it represents a practical application of deep learning in image classification, with potential use cases in various fields such as e-commerce, animal recognition, and self-driving cars. For making those models we used PyTorch which is a framework for deep learning problems.

2. Methodology

2.1. Data Description

The dataset consists of two types of images, chihuahua and muffin, the dataset can be found in Kaggle website as "Muffin vs chihuahua". In this dataset, There are 2559 images of

[4] Convolutional Neural Network - CNN

chihuahua and 2174 images of muffins for training. Also, we have 640 images of chihuahua and 544 images of muffins for testing our models.

2.2. Data preprocessing

Each image has different size and first we had to resize each image and we decide to resize all the images to 120*120 pixels. After resizing the images, we had to turn them to tensors to use them in training section. Our tool to do these was `torchvision.transforms`. Torchvision supports common computer vision transformations and can be used to transform or augment data for training or inference of different tasks. (9)

The dataset is organized into well-defined directories, which facilitates straightforward loading. Specifically, the training and test data are stored in separate directories, each containing subdirectories for the two classes. This hierarchical structure enables us to use a simpler loading class, `ImageFolder`, that inherits from `torchvision.datasets` module, which is a generic data loader. (10)

Figure 3. Directory Structure

After organizing and loading the dataset, the next step was setting the batch size for training images. Processing an entire dataset all at once may not be feasible due to memory constraints, especially when dealing with large datasets or high-resolution images. Modern GPUs, although powerful, have limited memory, and trying to load too much data at once can result in memory overflow, causing the training process to fail. To manage this, the dataset is divided into smaller subsets called batches. By processing these smaller batches one at a time, we can efficiently utilize the available memory and computational resources (11). For this step we used ` torch.utils.data.DataLoader` which represents a Python iterable over a dataset. we set `batch_size=32`. (12)

2.3. Architectures

2.3.1. TinyVGG[5]

In this project, we utilized a CNN architecture called TinyVGG which is inspired by the demo CNN in Stanford CS231n class (13). This model is inspired by the more complex and deeper VGG architectures, such as VGG13 and VGG16, but is designed to be simpler and more lightweight, making it suitable for limited computational resources. The TinyVGG model architecture is based on the implementation from CNN Explainer. (14)

The TinyVGG model consists of two main convolutional blocks followed by a fully connected classifier. TinyVGG contains Two convolutional layers and one simple layer. Each of those convolutional layers has two convolutional layers, two ReLU layers, to introduce non-linearity, and one MaxPool layer, Downsamples the feature map by a factor of 2. In the classifier layer, we flatten the feature map into a 1D vector and connect it to the last layer which has only one node. and connect all of the nodes to the only node in the last layer.

2.3.2. AlexNet

For the second architecture, we implemented a CNN architecture called AlexNet, which is one of the pioneering models in the field of deep learning for image classification. The AlexNet model is known for its depth and ability to learn hierarchical features, making it highly effective for complex image recognition tasks. This model is inspired by the more advanced CNN architectures used in large-scale image classification tasks but is adapted for binary classification in our project.

The AlexNet model architecture consists of five convolutional blocks followed by three fully connected layers. Below is a detailed breakdown of its components:

Convolutional Block 1: This block contains a Conv2d layer, BatchNorm2d layer to normalize the output, a ReLU layer to introduce non-linearity, and a MaxPool2d layer to downsample the feature map by a factor of 2.

Convolutional Block 2: This block contains a Conv2d layer, followed by a BatchNorm2d layer, a ReLU layer, and a MaxPool2d layer for further downsampling.

Convolutional Block 3, 4: These two blocks contain a Conv2d layer, a BatchNorm2d layer, and a ReLU layer.

Convolutional Block 5: This block contains a Conv2d layer, a BatchNorm2d layer, a ReLU layer, and a MaxPool2d layer to downsample the feature map.

The first fully connected layer includes a Dropout layer with a probability of 0.5 to reduce overfitting, followed by a Linear layer and a ReLU activation.

The second fully connected layer also includes a Dropout layer, a Linear layer, and a ReLU activation.

[5] Visual Geometry Group

The final fully connected layer consists of a Linear layer that outputs the final predictions, with the number of output units equal to 1.

2.3.3. Resnet

For the third architecture, we implemented a CNN architecture called residual neural network. A ResNet is a seminal deep learning model in which the weight layers learn residual functions with reference to the layer inputs. ResNet specifically designed to address the vanishing gradient problem by using residual connections. ResNet allows training of very deep networks by introducing skip connections that bypass one or more layers. This architecture is known for its robustness and efficiency in various image classification tasks.

The ResNet model architecture consists of several key components, including the ResNet head, multiple residual blocks (both identity and skip), and the ResNet tail. Below is a detailed breakdown of these components:

ResNet Head: A Conv2d Layer, BatchNorm2d Layer, ReLU Activation, MaxPool2d Layer

ResIdentity Block: This block includes two convolutional layers with batch normalization and ReLU activation. The input is added directly to the output (skip connection), ensuring the identity mapping.

ResSkip Block: This block includes two convolutional layers with batch normalization and ReLU activation, similar to the ResIdentity block. However, it also includes a convolutional layer in the skip path to match the dimensions and a stride of 2 for downsampling.

NIdentityBlocks: This module consists of multiple ResIdentity blocks repeated sequentially.

SkipAndNIdentityBlocks: This module starts with a ResSkip block followed by multiple ResIdentity blocks.

ResNet Tail: AdaptiveAvgPool2d Layer which applies adaptive average pooling to reduce the feature map. Also, it has one fully connected linear layer to produce the final output.

2.4. Training process

2.4.1. Hyperparameter Selection

For training our models, the first crucial step was setting our hyperparameters, particularly the optimizer and learning rate as those can significantly influence on the output. Hyperparameter Selection:

- Optimizer

We chose Stochastic Gradient Descent[6] as our optimizer for all the models. SGD is widely used in training neural networks due to its efficiency and simplicity. It updates the model parameters iteratively based on the gradient of the loss function with respect to the parameters.

[6] Stochastic Gradient Descent - SGD

- Learning Rate

A learning rate that is too high might cause the training process to converge too quickly to a suboptimal solution, while a learning rate that is too low can result in a prolonged training process. To optimize the performance of our models, we experimented with three different learning rates: 0.01, 0.05, and 0.001. By comparing the performance of the models trained with these learning rates, we aimed to identify the learning rate that yields the best results.

- Number of epochs

We set this hyperparameter to 20. So in each fold the selected model should trained for 20 times.

2.4.2. Training Procedure

We started by loading our dataset using the torchvision.datasets.ImageFolder which conveniently organizes images into folders for each class. This utility helps in directly converting our data into PyTorch tensors. The dataset was divided into training and validation sets using stratified 5-fold cross-validation to ensure each fold is representative of the overall distribution.

For each fold, the model was trained using the training data loader, and its performance was evaluated on the validation data loader. The loss function used was Binary Cross-Entropy with Logits[7], appropriate for binary classification tasks.

By systematically experimenting with different learning rates and using SGD as our optimizer, we were able to fine-tune our models to achieve optimal performance. The use of stratified k-fold cross-validation ensured robust and generalizable results. This approach ensured that we selected the most effective model configuration for our binary classification task, facilitating a thorough and rigorous evaluation of our models' performance.

2.4.3. Evaluation metrics

In order to assess the performance of our models, we employed several key evaluation metrics. These metrics provided a comprehensive understanding of how well the models performed in both training and validation phases.

- Loss function

In our binary classification task, we used the Binary Cross-Entropy with Logits Loss. This loss function is particularly suitable for binary classification as it combines a Sigmoid layer and the Binary Cross-Entropy Loss in a single class, making it numerically stable.

- Accuracy

Accuracy is a widely used metric in classification tasks, representing the proportion of correct predictions out of the total number of predictions. This metric gives a

[7] Binary Cross-Entropy with Logits - BCEWithLogitsLoss

straightforward indication of how many predictions are correct but does not account for the distribution of false positives and false negatives.

- Zero-One Loss

Zero-One Loss is another binary classification metric, which counts the number of misclassifications. This metric essentially measures the fraction of incorrect predictions.

3. Result

In this project, we trained and evaluated a total of nine models using three different architectures and three distinct learning rates. The architectures used were AlexNet, TinyVGG, and ResNet. Each architecture was trained with learning rates of 0.05, 0.01, and 0.001. Thorough experimentation, we found that the best learning rate for all architectures was 0.001. Consequently, we provide detailed results for the models trained with this optimal learning rate and present the test set results for the remaining six models.

3.1. TinyVGG

3.1.1. LR = 0.05

Our first model was TinyVGG with SGD optimizer and the learning rate of 0.05. As we told before, This architecture is a small version of the other VGG architectures so we can't expect much from this models. With these learning rate we got the worst result between these 9 models that we trained.

Table 1. Test result (TinyVGG, LR=0.05)

Model	Learning rate	Test Loss	Test ZOL	Test acc
TinyVGG	0.05	0.6899	0.459459	54.05405

As we see in the table, the model couldn't learn anything and just randomly choose the value.

3.1.2. LR = 0.01

The next model is also trained with TinyVGG but with a learning rate of 0.01. We obtain a much better result with this hyperparameter but still we need better results.

Table 2. Test result (TinyVGG, LR=0.01)

Model	Learning rate	Test Loss	Test ZOL	Test acc
TinyVGG	0.01	0.668822	0.152872	84.71284

3.1.3. LR = 0.001

As we told before, the best learning rate was 0.001. So the best result between these three models obtained from this model. Let's see the results on the training and validation set:

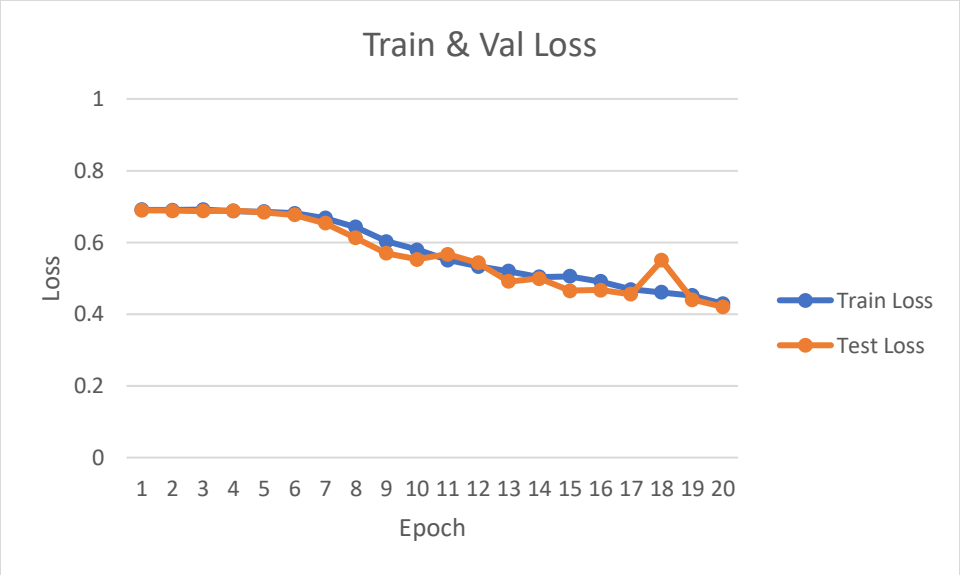

Figure 4. Training & Validation BCE Loss

Our BCE loss function decrease over time in both training and validation sets.

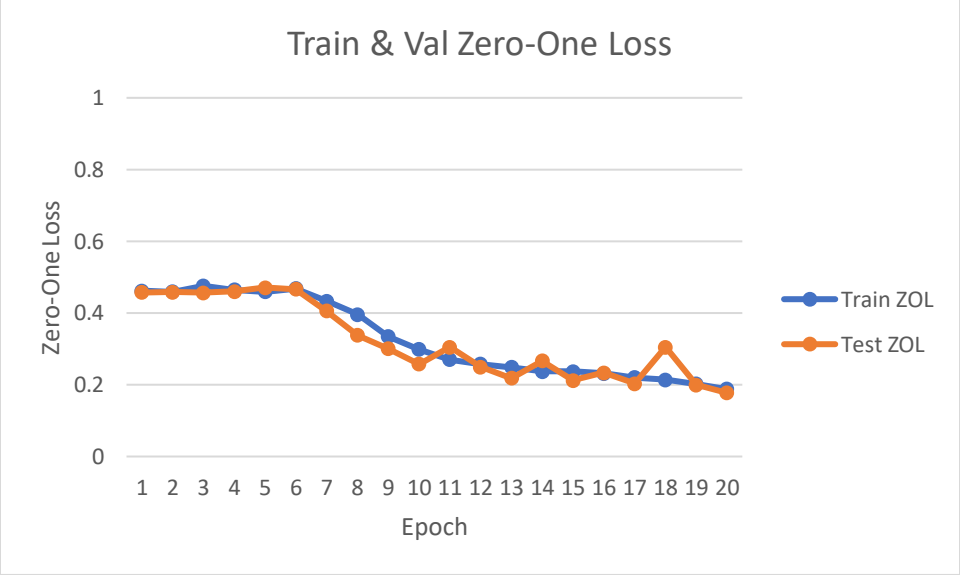

Figure 5. Training & Validation Zero-One Loss

Also our zero-one loss decrease over time.

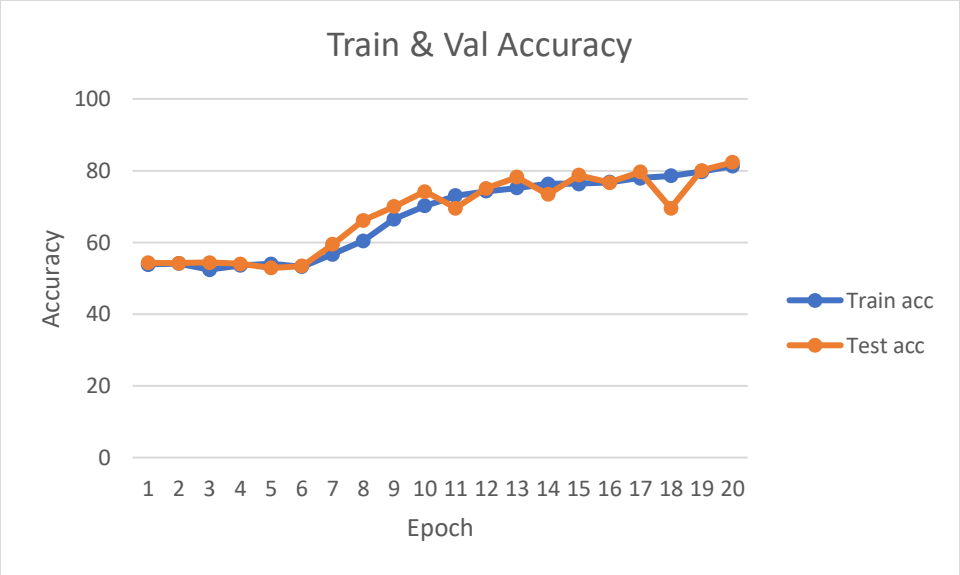

Figure 6. Training & Validation Accuracy

Our accuracy for this model is increasing and reach 82 percent in the last epoch.

Let's check the model's performance on the test set:

Table 3. Test result (TinyVGG, LR=0.001)

Model	Learning rate	Test Loss	Test ZOL	Test acc
TinyVGG	0.001	0.297709	0.111486	88.85135

As we see, the accuracy of this model is 89 percent which is a good result considering that our architecture is a small version of VGG's family. The confusion matrix for the test set is as follow:

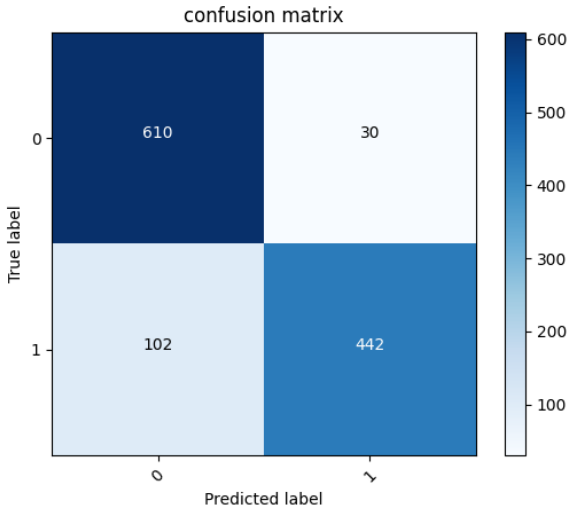

Figure 7. Confusion matrix (TinyVGG, LR=0.001)

3.2. AlexNet

3.2.1. LR = 0.05

This model is slightly better than TinyVGG with LR=0.05. But it could be much better. The test result for this architecture with learning rate of 0.05 is as follow:

Table 4. Test result (AlexNet, LR=0.05)

Model	Learning rate	Test Loss	Test ZOL	Test acc
AlexNet	0.05	0.59355	0.385135	61.48649

As we see, the model achieved the accuracy of 61 percent which is not good enough but it is slightly better than TinyVGG with LR=0.05.

3.2.2. LR = 0.01

This architecture with the learning rate of 0.01 performs better than the last ones. The test result for this architecture with learning rate of 0.01 is as follow:

Table 5. Test result (AlexNet, LR=0.01)

Model	Learning rate	Test Loss	Test ZOL	Test acc
AlexNet	0.01	0.309226	0.113176	88.68243

It's accuracy is almost equal as the accuracy of TinyVGG with LR=0.001. So, we can have a better result with this architecture.

3.2.3. LR = 0.001

It is the best model that we trained for this dataset. It has a great performance on the training and validation set and also done a good prediction on the test set.

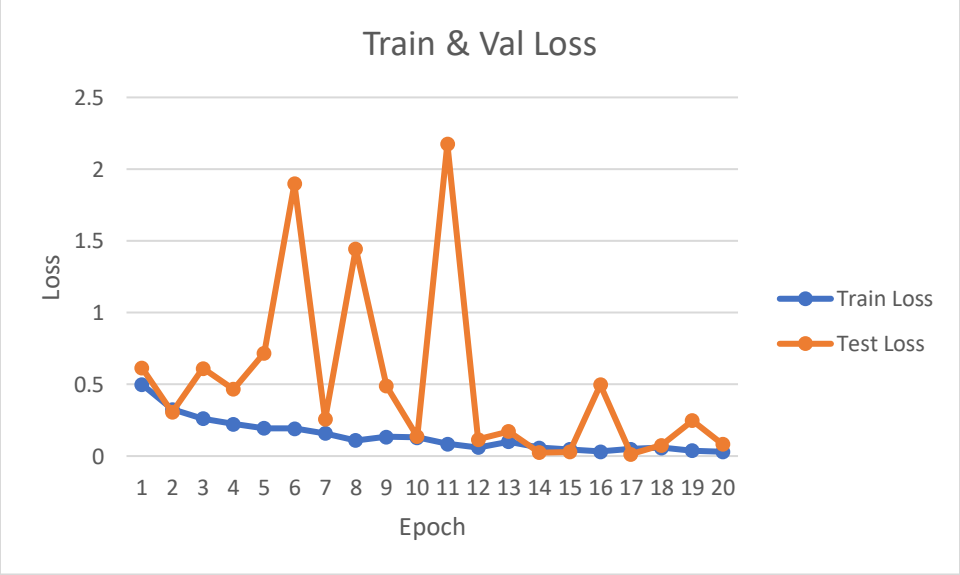

Figure 8. Training & Validation BCE Loss

This model continuously decrease over time on the training set and reached the loss of 0.02 in the last epoch. The validation loss has some fluctuation but it has a good performance on the last epochs.

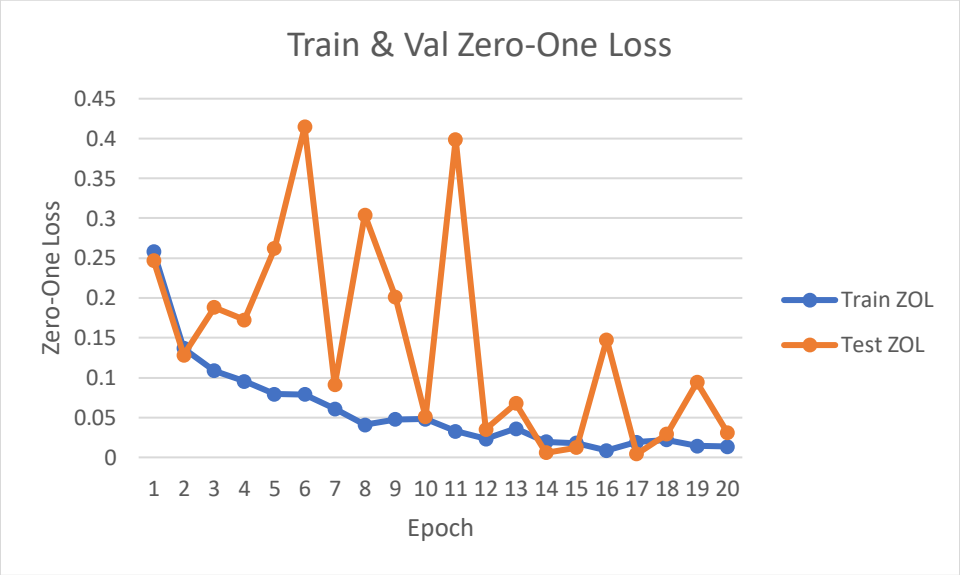

Figure 9. Training & Validation Zero-One Loss

The zero one loss is like the BCE loss and reached to a good point on both training and validation set.

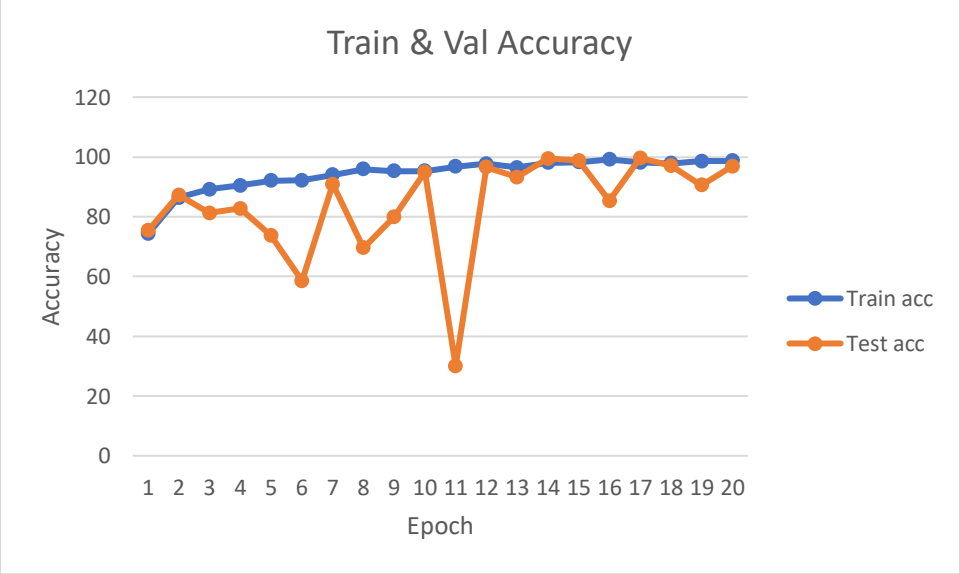

Figure 10. Training & Validation Accuracy

The accuracy of last epoch on training set is 98.6 percent and the accuracy of last epoch on validation set is 96.87 percent. The best result that we obtain during the training of these 9 models. Let's check the performance of the model on the test set:

Table 6. Test result (AlexNet, LR=0.001)

Model	Learning rate	Test Loss	Test ZOL	Test acc
AlexNet	0.001	0.222621	0.054899	94.51014

The model also perform a good prediction on the test set and t's accuracy is 94.5 percent which consider a good precision. Let's check the confusion matrix on the test set:

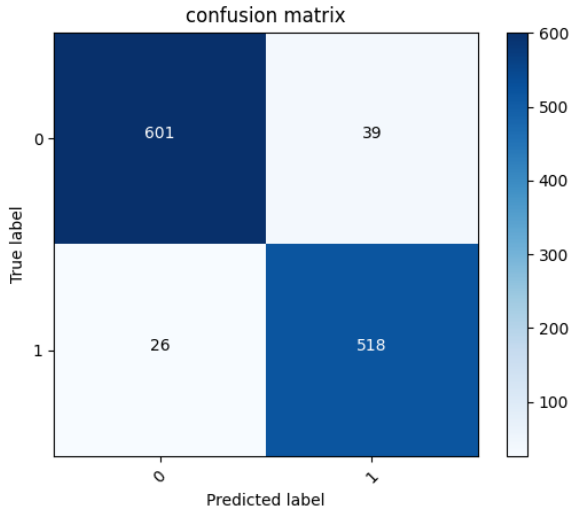

Figure 11. Confusion matrix (AlexNet, LR=0.001)

3.3. ResNet

3.3.1. LR = 0.05

This architecture with the learning rate of 0.05 has the best performance among the other architectures with the same learning rate. But we will see that this architecture performs better when we decrease the learning rate. The test result is as follow:

Table 7. Test result (ResNet, LR=0.05)

Model	Learning rate	Test Loss	Test ZOL	Test acc
ResIdentity	0.05	0.410134	0.174831	82.51689

The accuracy of this model is 82.5 which is a good accuracy considering that it's learning rate is not good as the other learning rates.

3.3.2. LR = 0.01

ResNet architecture with the learning rate of 0.01 performs a good prediction on test set but it can be better. Test result:

Table 8. Test result (ResNet, LR=0.01)

Model	Learning rate	Test Loss	Test ZOL	Test acc
ResIdentity	0.01	0.296602	0.11402	88.59797

As we see, the model's accuracy is like the accuracy of AlexNet with this learning rate. It is but it can be better.

3.3.3. LR = 0.001

The second best model for this dataset is the model with ResNet architecture with learning rate of 0.001.

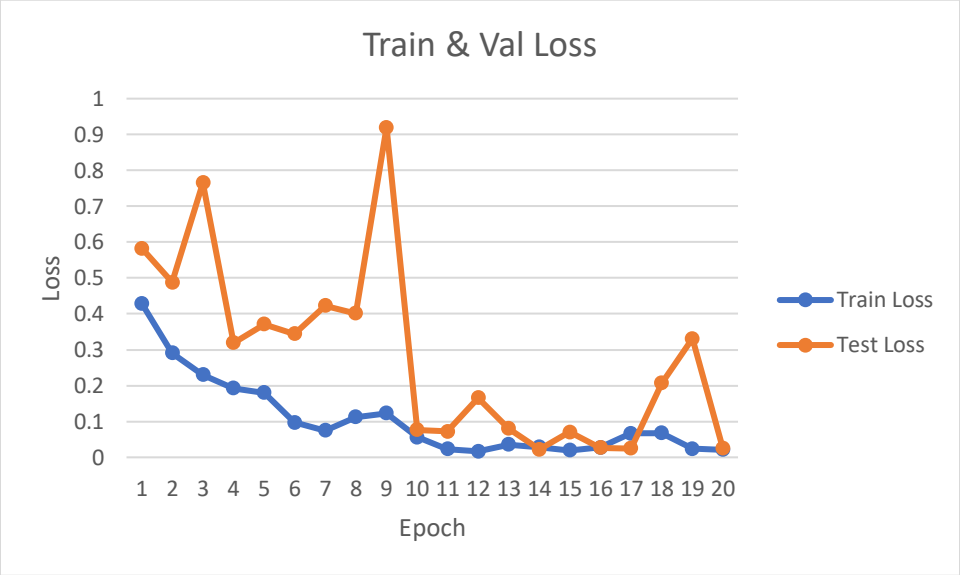

Figure 12. Training & Validation BCE Loss

As we see, The BCE loss for this model is like the BCE loss of AlexNet with the learning rate of 0.001. It predict the images of training set very well but has some fluctuation on validation set.

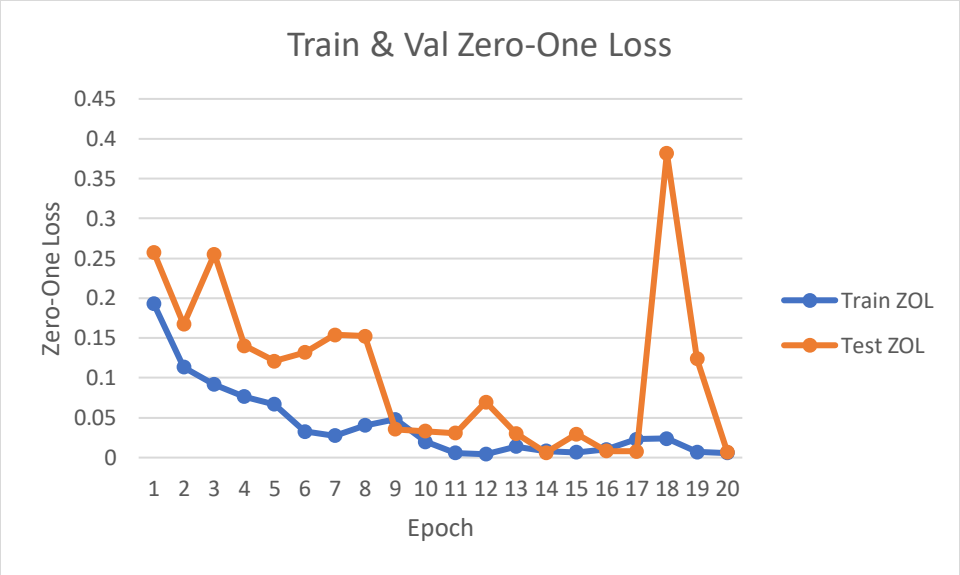

Figure 13. Training & Validation Zero-One Loss

The zero-one loss shows the same result as the BCE loss.

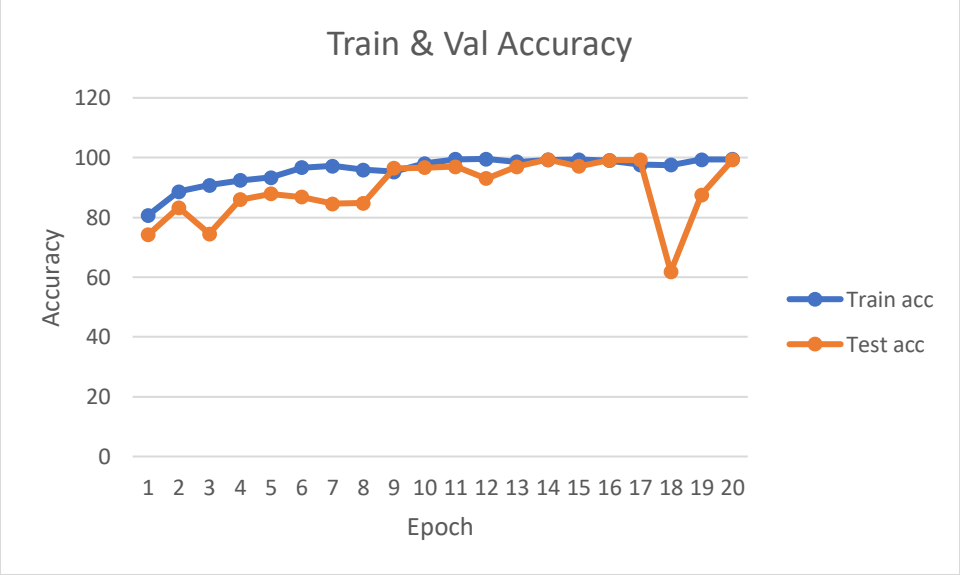

Figure 14. Training & Validation Accuracy

The accuracy of this model on the training set is 99.4 percent and on the validation set is 99.2 percent. It has a better accuracy on the both these dataset than the model with AlexNet architecture and learning rate of 0.001. Let's check the result on the test set:

Table 9. Test result (ResNet, LR=0.001)

Model	Learning rate	Test Loss	Test ZOL	Test acc
ResIdentity	0.001	0.286922	0.088682	91.13176

As we see, the accuracy of this model on the test set is lower than the accuracy of AlexNet with LR=0.001. But, It is also a good prediction. The confusion matrix on the test set is as follow:

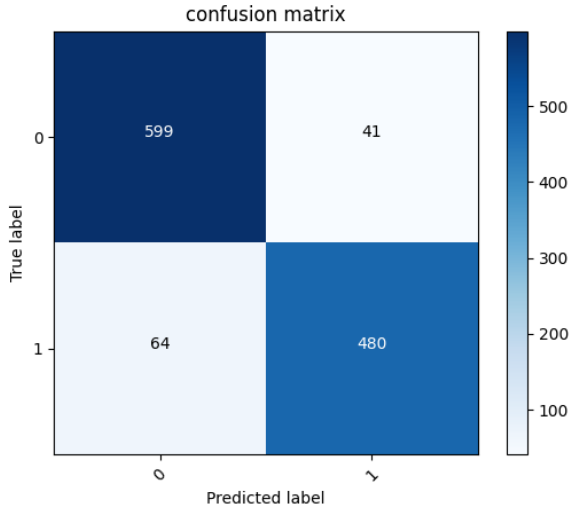

Figure 15. Confusion matrix (ResNet, LR=0.001)

4. Discussion

In this section, I want to compare the result of the models on the test set.

Table 10. Ranking of the model based on their performance on the test set

RANK	MODEL	LR	BCE LOSS	ZERO-ONE LOSS	ACCURACY
1	AlexNet	0.001	0.222621	0.054899	94.51014
2	ResNet	0.001	0.286922	0.088682	91.13176
3	TinyVGG	0.001	0.297709	0.111486	88.85135
4	AlexNet	0.01	0.309226	0.113176	88.68243
5	ResNet	0.01	0.296602	0.11402	88.59797
6	TinyVGG	0.01	0.668822	0.152872	84.71284
7	ResNet	0.05	0.410134	0.174831	82.51689
8	AlexNet	0.05	0.59355	0.385135	61.48649
9	TinyVGG	0.05	0.6899	0.459459	54.05405

As we see in the table, The best performance is for the model with AlexNet architecture and the LR=0.001 which has a accuracy of 94.5 and the worst performance between these models is for the model with the architecture of TinyVGG and LR=0.05.

5. Conclusion

In this project, we trained and evaluated a total of nine models using three different architectures and three distinct learning rates. The architectures used were AlexNet,

TinyVGG, and ResNet. Each architecture was trained with learning rates of 0.01, 0.001, and 0.05. For each model, we tracked the following performance metrics during training and validation:

- Training Loss: Measures how well the model is fitting the training data.
- Validation Loss: Indicates how well the model generalizes to unseen data.
- Training Accuracy: The proportion of correct predictions on the training set.
- Validation Accuracy: The proportion of correct predictions on the validation set.
- Zero-One Loss: The fraction of misclassified samples, both during training and validation.

After extensive training and evaluation, we found that the best-performing model was AlexNet with a learning rate of 0.001. This model achieved the highest validation accuracy and the lowest validation loss among all the models trained. The results of our experiments highlight the importance of selecting an appropriate learning rate for training deep learning models. While all three architectures demonstrated potential, the AlexNet architecture with a learning rate of 0.001 proved to be the most effective for our binary classification task. This model's superior performance underscores its ability to generalize well to new data, making it a robust choice for practical deployment.

6. References

1. https://aws.amazon.com/what-is/data-science/
2. https://en.wikipedia.org/wiki/Machine_learning
3. https://www.ibm.com/topics/machine-learning
4. https://www.ibm.com/topics/neural-networks
5. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
6. https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-deep-learning/
7. https://www.researchgate.net/figure/Relationship-between-artificial-intelligence-machine-learning-neural-network-and-deep_fig3_354124420
8. https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns
9. https://pytorch.org/vision/main/transforms.html
10. https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html
11. https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa
12. https://pytorch.org/docs/stable/data.html
13. https://cs231n.stanford.edu/
14. https://github.com/poloclub/cnn-explainer/tree/master