

Data Analytics – Final Project Report

Name: Shovan Shakya

Title: Analysis of Image Classification Methods on Custom Data

1. INTRODUCTION

This report is for the final project for Data Analytics (EEL4774). The objective of the project is to experiment and then fine tune various machine learning models to achieve the best accuracy on a provided dataset. There were 2 options for the dataset: (*Time Series, Images*). I choose to go with the image classification set.

The dataset contains roughly 2000 images of formats such as JPEG and PNG. There are 17 Classes (*Adult, Airplane, Alpaca, Bird, Bus, Car, Cat, Child, Elephant, Flower, Giraffe, Horse, Monkey, Panda, Reptile, Vessel*). Each image is inside a parent folder with its respective classes. The images have different sizes and resolutions.

Due to the non-linear nature of image data, the approach chosen went from using models of increasing complexity and sophistication that supported nonlinearity.

2. PERFORMED ANALYSES

As such, analyses using Support Vector Machines, Multilayer Perceptron, Convolutional Neural Networks and Vision Transformers were performed. Testing using different parameters, epochs, as well as transfer learning on pretrained models were performed wherever applicable.

Support Vector Machines (SVM)

SVM is a relatively simpler algorithm that aims to find a hyperplane in an N-dimensional space that effectively separates data points into two distinct classes. The goal is to identify a hyperplane that has the maximum margin, which is the greatest distance between data points of both classes. [1] GridSearchCV is a hyperparameter tuning technique that was used to optimize the performance of the SVM. It works by systematically searching through a range of hyperparameters to find the combination that results in the best performance. [2]

- For uniformity, the images from the dataset were initially resized to 150 x 150 x 3 and then flattened. [8]
- Initial GridSearchCV hyperparameters were $\{ 'C': [0.1, 1, 10, 100], 'gamma': [0.0001, 0.001, 0.01, 1], 'tol': [1e-3, 1e-4, 1e-5, 1e-6], 'kernel': ['rbf', 'sigmoid'] \}$. This setting had a wider range allowing for a more thorough search. However, the estimated time to complete the training process was

calculated to be around 5000 minutes. So, a reduction in the hyperparameters was made.

- The hyperparameters finally used were `{'C':[1,100], 'gamma':[1,0.1], 'tol':[1e-1, 1e-1, 1e-1, 1e-1], 'kernel':['rbf', 'sigmoid']}`. Total training time was around 600 minutes.

Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP) is a neural network designed to overcome the limitations of the Perceptron. Unlike the Perceptron, which has a linear mapping between inputs and outputs, the MLP has a non-linear mapping due to the use of one or more hidden layers containing multiple neurons with arbitrary activation functions. This flexibility allows the MLP to model complex relationships between inputs and outputs that cannot be modeled by the Perceptron. The MLP architecture consists of an input layer, one or more hidden layers, and an output layer. [3]

- For uniformity, the images from the dataset were resized to 300 x 300 x 3. [7]
- They were then divided by 255 as pixel intensity lies between 0 – 255. *LabelEncoder()* was used to numerically assign classes numbers.
- The first model architecture is as follows:
Flatten(input shape=(300,300,3))
Dense(256, activation=tanh)
Dense(16, activation= softmax)
- The second model architecture is as follows:
Flatten(input shape=(300,300,3))
Dense(256, activation= tanh)
Dense(64, activation=relu)
Dense(16, activation=softmax)
- Both models were tested on epoch sizes of 32 and 50.

Convolutional Neural Networks (CNN)

CNN is a type of neural network that is commonly used for processing grid-like data such as images and videos. The CNN architecture consists of several layers including the input layer, Convolutional layer, Pooling layer, and fully connected layers. The Convolutional layer applies filters to the input data to extract features, the Pooling layer reduces the size of the data, and the fully connected layer makes the final prediction. CNN learns the optimal filters through backpropagation and gradient descent. [4]

Without Pretrained Weights [5]

- TinyVGG model architecture was used as the CNN model.
- Train and Test dataset were resized to 64x64 and then was converted to Tensor. They were then converted to DataLoaders.

```

=====
Layer (type:depth-idx)                Output Shape                Param #
=====
TinyVGG                                [1, 16]                     --
├─Sequential: 1-1                      [1, 10, 32, 32]            --
│   └─Conv2d: 2-1                      [1, 10, 64, 64]            280
│       └─ReLU: 2-2                    [1, 10, 64, 64]            --
│           └─Conv2d: 2-3              [1, 10, 64, 64]            910
│               └─ReLU: 2-4            [1, 10, 64, 64]            --
│                   └─MaxPool2d: 2-5   [1, 10, 32, 32]            --
├─Sequential: 1-2                      [1, 10, 16, 16]            --
│   └─Conv2d: 2-6                      [1, 10, 32, 32]            910
│       └─ReLU: 2-7                    [1, 10, 32, 32]            --
│           └─Conv2d: 2-8              [1, 10, 32, 32]            910
│               └─ReLU: 2-9            [1, 10, 32, 32]            --
│                   └─MaxPool2d: 2-10  [1, 10, 16, 16]            --
├─Sequential: 1-3                      [1, 16]                     --
│   └─Flatten: 2-11                    [1, 2560]                  --
│       └─Linear: 2-12                 [1, 16]                     40,976
=====
Total params: 43,986
Trainable params: 43,986
Non-trainable params: 0
Total mult-adds (M): 6.78
=====
Input size (MB): 0.05
Forward/backward pass size (MB): 0.82
Params size (MB): 0.18
Estimated Total Size (MB): 1.04
=====

```

- The kernel size was 3 and the stride was 1 for a more thorough search.
- Analysis was done on both 32 and 50 epoch sizes.
- Data Augmentation was done and added to the increased dataset.
- The model was then again trained on the new data.

With Pretrained Weights [6]

- EfficientNet_B0 was used as the model architecture with the default(ImageNet) pretrained weights.
- The dataset was then transformed to fit the input shape (224 x 224)
- All parameters apart from the final classifier layer (*Dropout*, *Linear*) were frozen. The final layer's output shape was matched with the number of classes.
- Adam optimizer was used due to its efficiency.
- Analysis was done on both 32 and 50 epoch sizes.

Vision Transformers

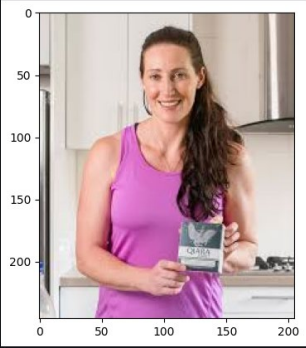
- A pretrained vision transformer (ViT_B_16) was used with transfer learning.
- Images were resized to 224 x 224 x 3.
- All parameters apart from the final classifier layer (*Linear*) were frozen. The final layer's output shape was matched with the number of classes.
- Initial test on 5 epochs were performed.

3. RESULTS AND DISCUSSION

SVM

The accuracy score on test data was 12.7 %.

	precision	recall	f1-score	support	...
adult	0.16	0.17	0.17	23	
airplane	0.17	0.17	0.17	23	
alpaca	0.00	0.00	0.00	21	
bird	0.04	0.05	0.04	20	
bus	0.28	0.27	0.27	26	
car	0.21	0.15	0.18	20	
cat	0.12	0.10	0.11	29	
child	0.06	0.10	0.07	21	
elephant	0.15	0.12	0.13	25	
flower	0.04	0.09	0.06	23	
giraffe	0.08	0.10	0.09	21	
horse	0.12	0.40	0.19	25	
monkey	0.22	0.08	0.12	25	
panda	0.20	0.04	0.06	26	
reptile	0.33	0.05	0.09	20	
vessel	0.00	0.00	0.00	20	
accuracy			0.12	368	
macro avg	0.14	0.12	0.11	368	
weighted avg	0.14	0.12	0.11	368	



adult = 5.543610055637133%
airplane = 8.409098889202247%
alpaca = 5.5202147462288655%
bird = 5.672114352436284%
bus = 6.950231789874425%
car = 6.497653460714579%
cat = 7.178988730816399%
child = 4.517507307931789%
elephant = 7.775993967306628%
flower = 7.7230144950815856%
giraffe = 5.801053943795739%
horse = 9.00909500162887%
monkey = 4.038647862014724%
panda = 5.546535009659828%
reptile = 5.7341734509648395%
vessel = 3.4812608706160395%
The predicted image is : flower

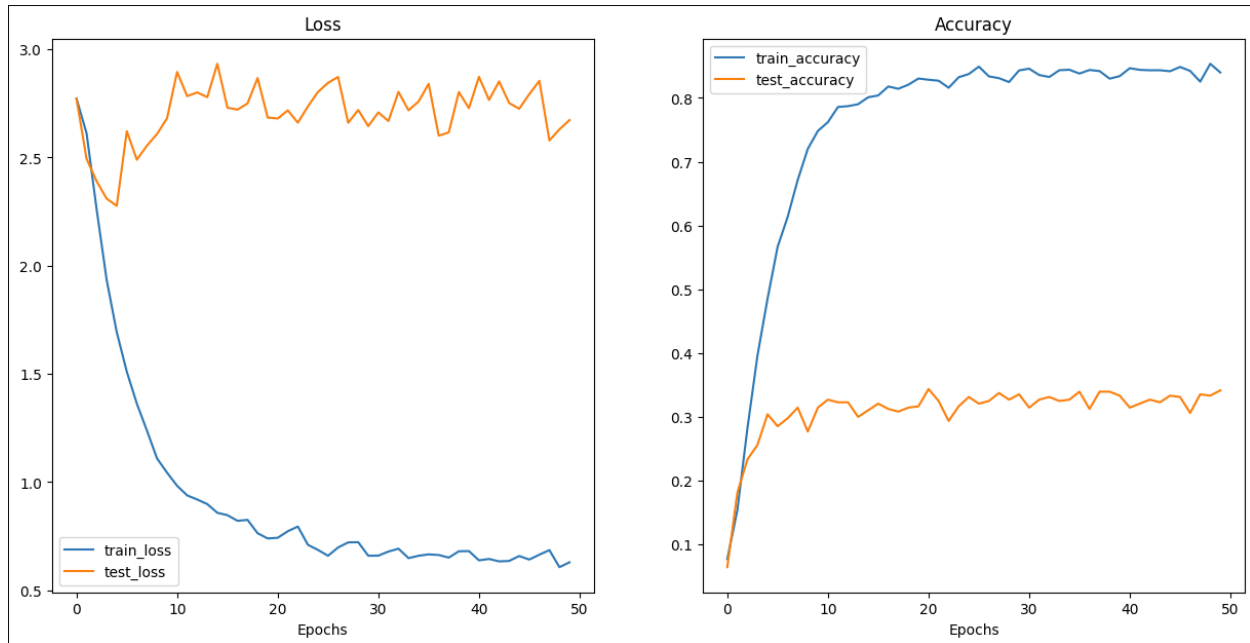
MLP

The accuracy averaged around 6-7 % for the first model.

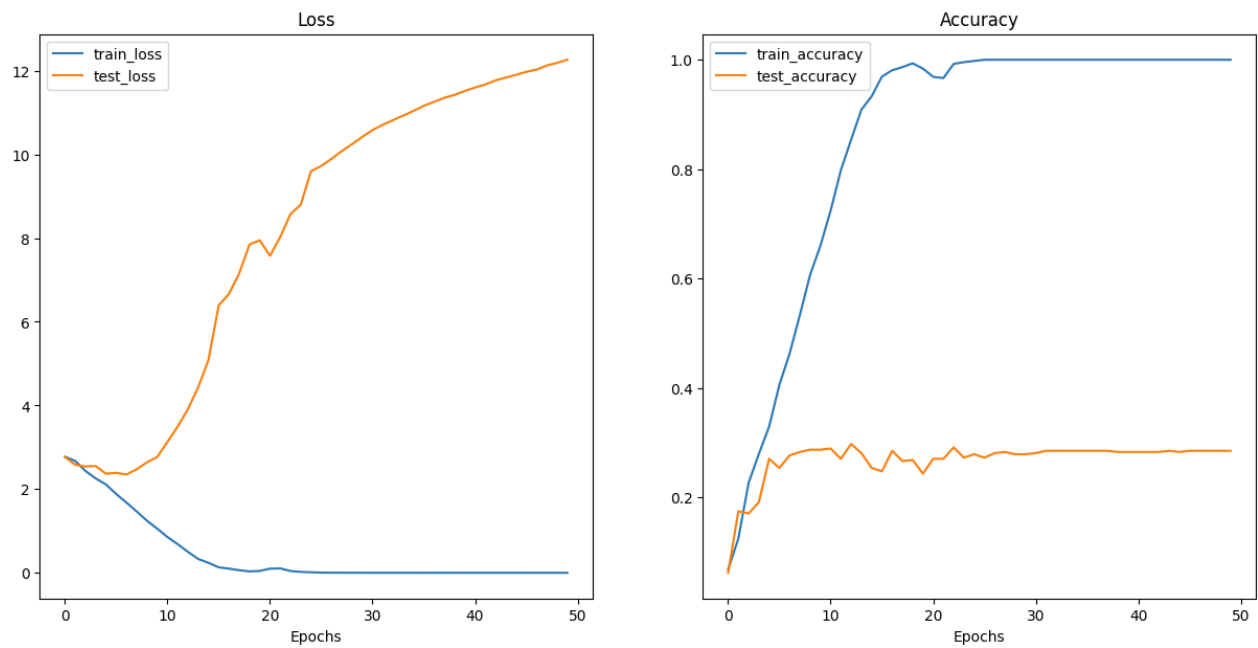
The second model with another layer had an accuracy of 8.64%.

CNN

TinyVGG Architecture (No Pretrained Weights)

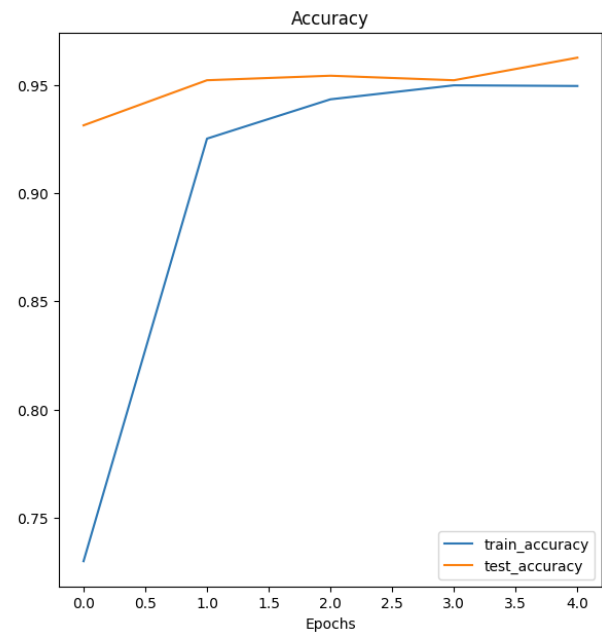
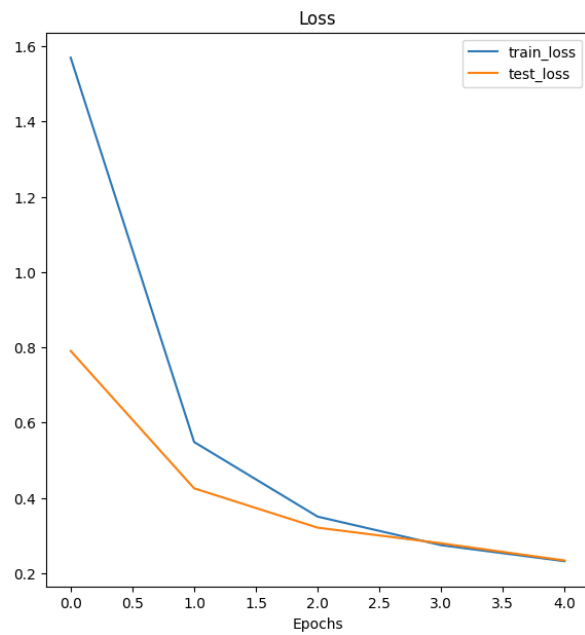


Without Data Augmentation

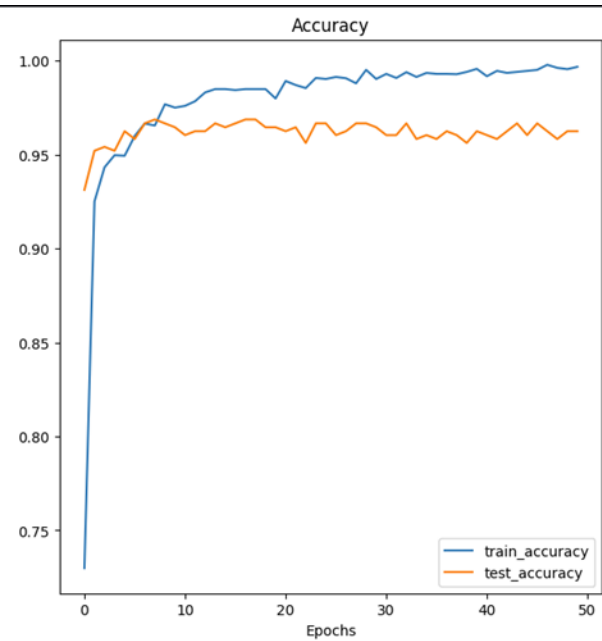
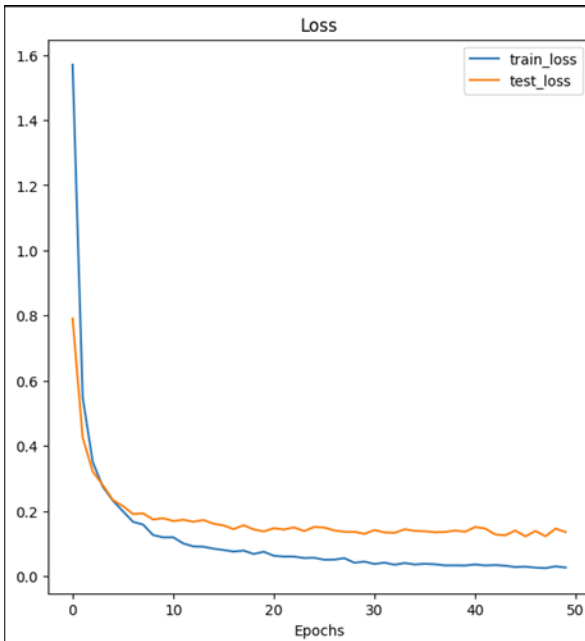


With Data Augmentation

EfficientNet_B0 Architecture (Transfer Learning)

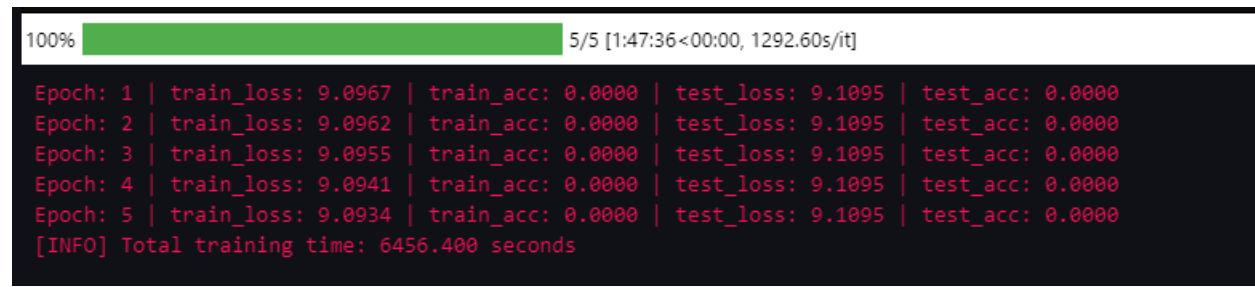


5 Epochs



50 Epochs

Vision Transformer



Test accuracy remained 0% throughout the 5 epochs. There seems to be a misconfiguration while setting up the model. Could not figure out the error by the deadline.

4. **CONCLUSION**

Transfer Learning on the EfficientNet model proved to be the best approach with an accuracy on the test data of 95%. I wanted to debug the Vision Transformer error but did not find resources online that had used Pytorch's ViT model. I could not configure Pytorch to use the computer's GPU even though following the related tutorials and had to use the CPU for training which was a hassle as epochs of size 100 or more had an estimated training time of >7 hours.

The project provided a good foundation to build and test the performance of various models. Experimentation with tweaking various parameters allowed for a deeper understanding of the inner workings of the models along with what parameters are better suited for the given task. I believe I can get improvements in analysis by increasing the diversity in parameters, using different architectures and models as well as using larger epoch sizes.

REFERENCES

- [1] <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- [3] <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>
- [4] <https://www.geeksforgeeks.org/introduction-convolution-neural-network/>
- [5] https://www.learnpytorch.io/04_pytorch_custom_datasets/
- [6] https://www.learnpytorch.io/06_pytorch_transfer_learning/
- [7] <https://youtu.be/H-p0drovpP8>
- [8] <https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/>