Deep Learning with Pytorch

Final Project Report

Image Classification of the Pet Breeds Using the Oxford-IIIT Pet Dataset

Prepared By:

Muhammad Osama

1. Abstract

In this project I have implemented image classification using deep learning models to classify different classes of pets. The data set that I have used is Oxford-IIIT Pet Dataset. I have used four different models for the task a custom CNN, ResNet18, ResNet34, and ResNet50. I have implemented a full pipeline for our task from data preprocessing to model training and finally the comparison between all the models. It shows how pre-trained models have better performance using transfer learning. ResNet50 outperformed among all the models with the highest accuracy.

2. Introduction

Image classification is one of the foundational tasks in computer vision it has applications in all domains such as smart automation systems, medical diagnostics, image detection and animal monitoring. In this project I aim to classify different breeds of cats and dogs based upon their features. The data set that I have used is Oxford-IIIT Pet Dataset and will be using different deep learning models for the task.

The dataset gives a real-world classification challenge like various posing, background and lighting which makes it very close to real world scenarios. Overall, the dataset contains 37 classes of cats and dogs. The dataset is suitable for evaluating the performance of our models that we have used for this image classification task.

Problem Statement

The goal of the project is to build an image classification model which can accurately predict and recognize and classify the breed of cats and dogs. For this task it requires to build suitable deep learning models with suitable architecture or using transfer learning using pretrained models, training them on labeled data and evaluating their ability to generalize on unseen data.

3. Dataset and Preprocessing

The dataset that I have used is **Oxford-IIIT Pet Dataset,** the dataset consists of approximately 7,400 images which include different breeds of cats and dogs. For each class there are approximately 200 images, which means the dataset is relatively balanced.

To prepare the dataset for training I have applied a various number of preprocessing and augmentation steps. I have created a transform pipeline for my data set, first is train data transform pipeline I have resized the shorter edge to 256 so it maintains the aspect ratio and standardizing the images makes it easier to process with CNNs. Then random crop of 224 it randomly takes 224 * 224 crops of image which encourages the model to focus on different parts of the pet in different epochs it adds robustness and generalization. For example, in each epoch the model will have different parts to focus on like one crop might center the face, another might crop near the tail this makes the model learn more general features. Adding RandomHorizontalFlip for data diversity. RandomRotation rotates the image randomly by up to ± 15 degrees so it helps model learn rotation invariant features. Finally convert the PIL image to pytorch tensor. And then adding channel wise normalization so we adjust brightness (mean) and contrast (std) to match ResNets training environment.

Transformsubset is used for the subset of classes that we are using. Transforms are applied at the time of access not during split and it allows the same image to be transformed differently in each epoch.

I have divided the dataset into three splits:

- 70% for Training
- 15% for Validation
- 15% for Testing

Data splitting helps us to monitor model performance during training and test set allows us to make unbiased evaluations on unseen data.

4. Model Architectures

In this project I have implemented four different types of models starting from Custom CNN built from scratch and then there are three different variants of ResNet (ResNet18, ResNet34, and ResNet50) which are pre trained models.

4.1 Custom CNN

The Custom CNN is a simple architecture composed of four convolutional layers. I have used 4 convolutional layers which provide enough depth to learn the features of our data set. Used BatchNorm, which improves the stability of the model by normalizing activations across the batch, it helps in faster convergence and reduces the chances of overfitting. Relu activation function is used it applies nonlinearity and helps model to learn complex nonlinear features. Relu keeps only positive values. Using MaxPooling to reduce spatial size and computation. Towards the end I am using AdaptiveAvgPool2d which makes the network independent of spatial size and drastically reduces parameters before they enter the classifier.

Finally, the last layer uses SoftMax activation to produce predictions for the 37 classes. In the classifier head I am using drop out 0.5 which drops 50% of the unit to reduce overfitting.

While this model is lightweight and computationally efficient, it lacks the depth and representational power needed for fine-grained image classification, especially for distinguishing between the pet's breeds which are visually similar.

Architecture Summary:

- 4 Convolutional layers with increasing filters (32 → 256)
- Batch Normalization and ReLU after each conv layer

- Max Pooling after each block
- Adaptive Average Pooling to flatten output
- 1 Fully Connected layer (128 units)
- Dropout (p=0.5)
- Output layer (37-class softmax)

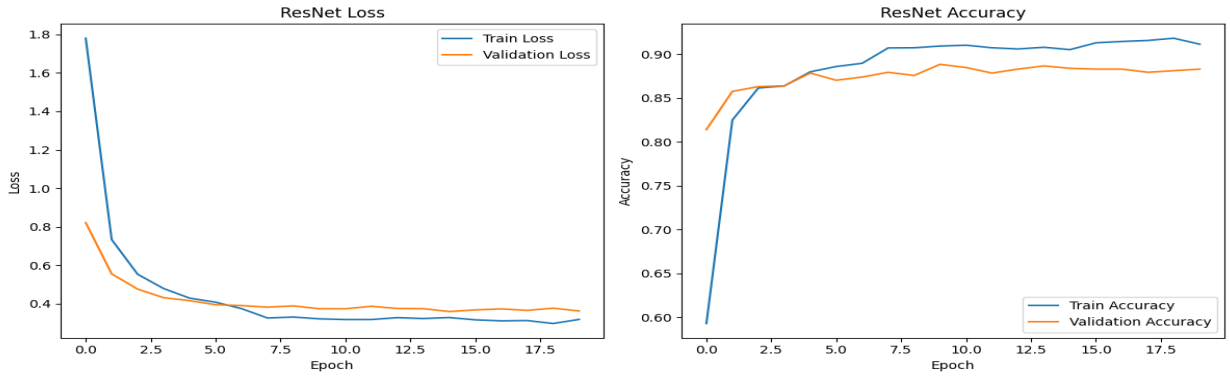
4.2 ResNet18

ResNet18 is a widely used architecture which is based on the concept of residual learning. It introduces shortcut connections that allow the model to learn identity mappings, making training deep networks more stable and faster. It has a total depth of 18 layers

In my implementation, I have used ResNet18 which is a pretrained model on the ImageNet dataset which contains more than 1 million images. I have used the fine-tuning technique in transfer learning which means I have frozen all layers of the model except the last layer which was replaced with output predictions for our 37 classes. The model leverages the pretrained weights, the model begins training with a strong understanding of visual features, which drastically improves performance.

Architecture Summary:

- 1 initial Conv layer with 64 filters + BatchNorm + MaxPool
- 4 Residual blocks (2 layers each) with shortcut connections
- Total depth: 18 layers
- Global Average Pooling layer
- Final FC layer (output: 37)
- Total Parameters: ~11.7 million



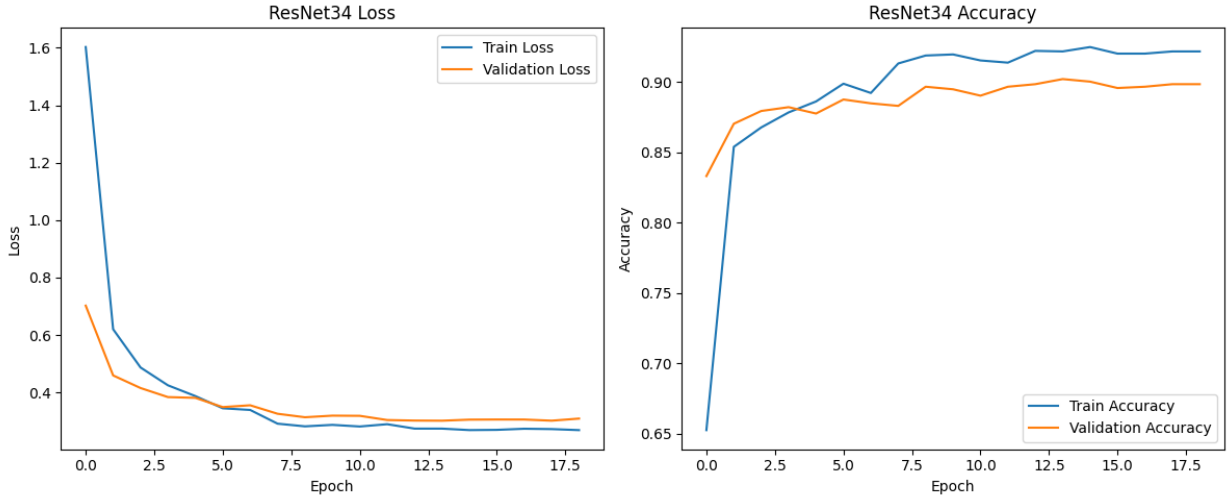
4.3 ResNet34

ResNet34 offers a middle ground between ResNet18 and ResNet50 in terms of depth and performance. It has 34 layers in total which means it is deeper than ResNet18, providing improved feature representation while being lighter and faster than ResNet50.

Like ResNet18 I have retrained the final fully connected layer to predict output based on 37 classes in my dataset while freezing all the other layers.

Architecture Summary:

- 1 initial Conv layer with 64 filters + BatchNorm + MaxPool
- 4 Residual blocks (3, 4, 6, 3 layers respectively)
- Total depth: 34 layers
- Global Average Pooling layer
- Final FC layer (output: 37)
- Total Parameters: ~21.8 million



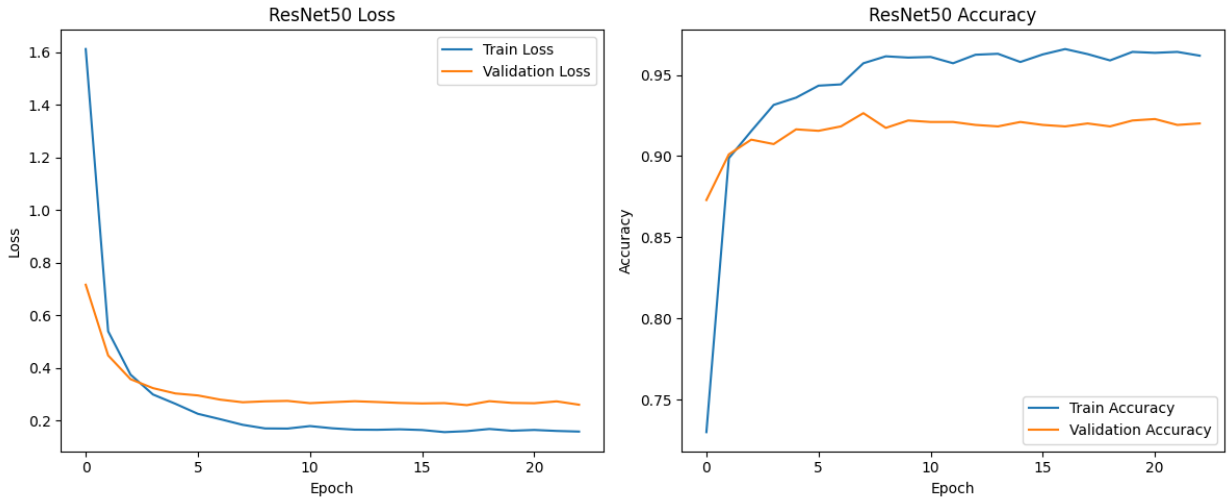
4.4 ResNet50

ResNet50 builds upon ResNet34 by comprising a deeper network with 50 layers. It uses bottleneck residual blocks to reduce computation while maintaining depth. Similar to ResNet18 and ResNet34, we used the pretrained version of ResNet50 and replaced the final classification layer.

Resnet50 has greater depth and complexity, due to which it can extract more abstract features, leading to higher accuracy but at the cost of longer training time and increased usage of memory.

Architecture Summary:

- 1 initial Conv layer with 64 filters + BatchNorm + MaxPool
- 16 Bottleneck Residual blocks (3 layers each)
- Total depth: 50 layers
- Global Average Pooling layer
- Final FC layer (output: 37)
- Total Parameters: ~23.5 million



5. Methodology

The method that I have used for training the models consists of best practices in deep learning. For all the models I have used Adam as an optimizer with a learning rate of 0.001. The loss function that I have used is cross-entropy loss because here we are solving a multi-class classification problem. For the learning rate decay, I have used the StepLR scheduler with gamma = 0.1 and step size of 10 for Custom CNN with 7 for all the other models. Controlling the learning rate decay helps our model to learn faster in the early stages and then makes smaller, stable updates. It helps with Faster Initial Learning, Finer Convergence Later, Helps Escape Local Minima in early stages and Prevents Overshooting.

The training is done on GPU using PyTorch for faster processing. I have used 30 epochs; the model is then evaluated on training and validation sets. For each epoch I have recorded the validation and training loss. I have also integrated early stopping with patience of 5 which reduces the chances of overfitting and reduces computational power if there is no improvement in validation loss for 5 epochs.

Finally, the models are saved at their best performing validation accuracy. Once the training process is completed, the best models are evaluated on the test set.

6. Evaluation

The evaluation of all the models is conducted using the test set, which the model never saw during training and is unseen data for the model. For each model, I have computed:

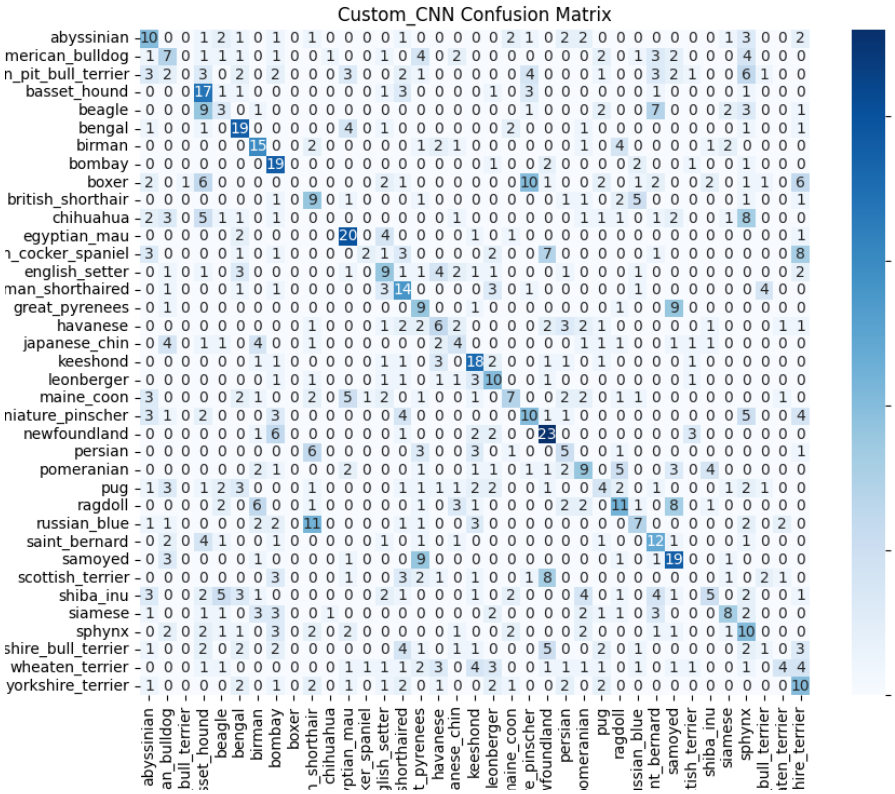
- Test Accuracy
- Confusion Matrix
- Classification Report (Precision, Recall, F1-score)

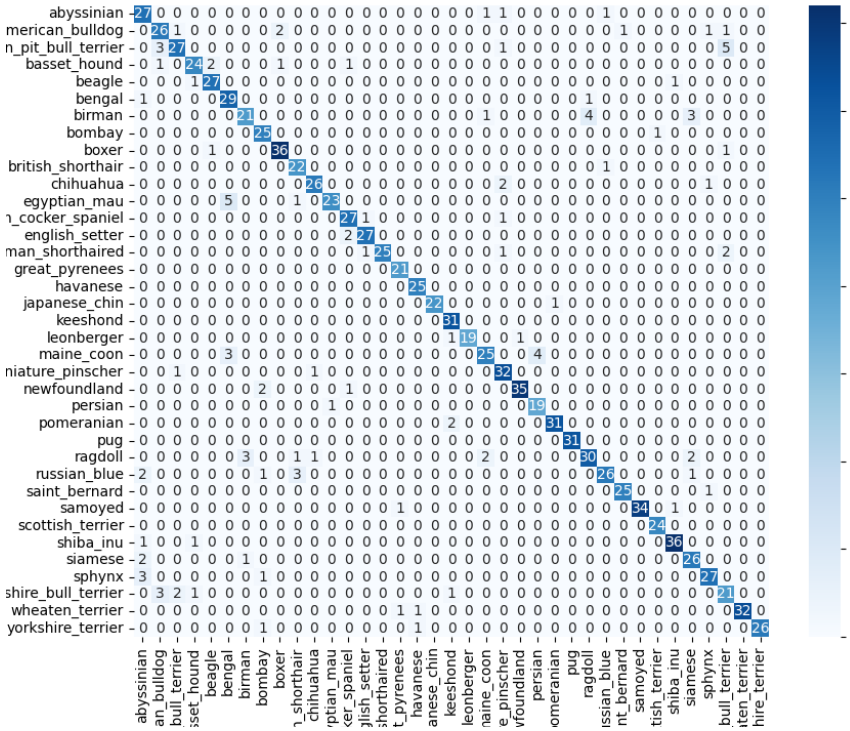
Test Accuracies:

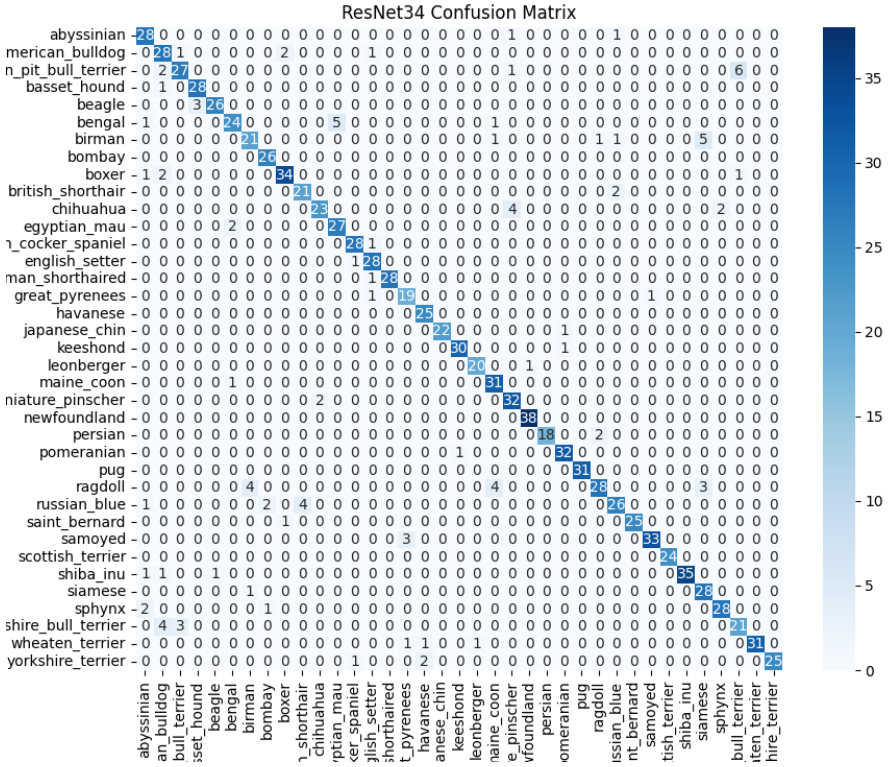
- **Custom CNN: ~30.46%**
- **ResNet18: ~89.76%**
- **ResNet34: ~90.57%**
- **ResNet50: ~92.38%**

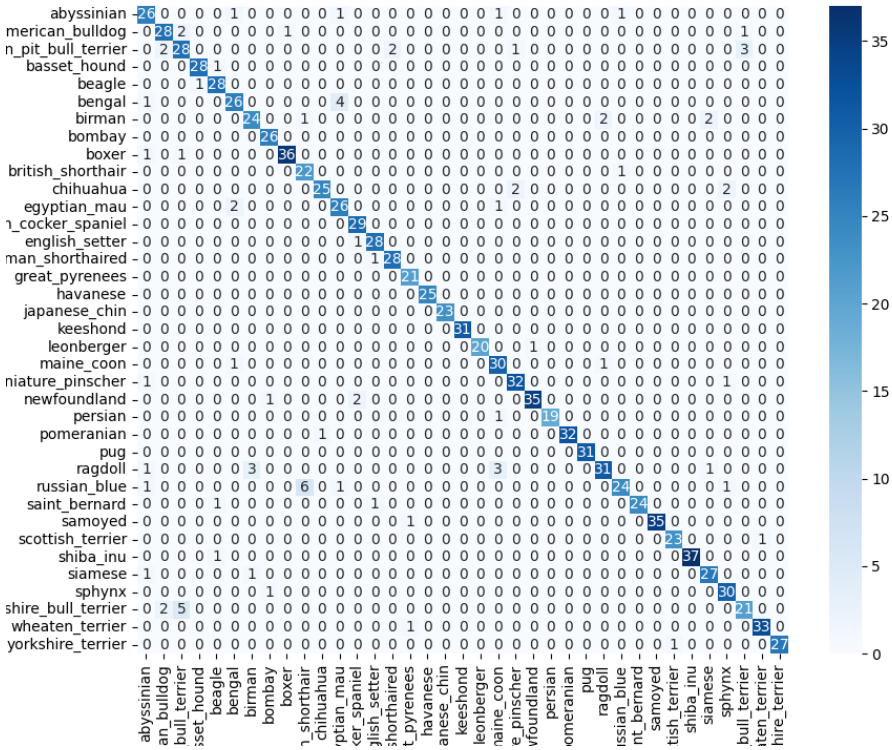
Observations:

- Custom CNN achieved the lowest accuracy among all, majorly due to limited capacity and lack of pretraining.
- ResNet18 demonstrated excellent performance with minimal tuning due to transfer learning.
- ResNet34 showed a solid improvement over ResNet18
- ResNet50 performed slightly better than ResNet34, benefiting from deeper layers.


Custom_CNN Confusion Matrix
abyssinian
american_bulldog
american_pit_bull_terrier
basset_hound
beagle
bengal
birman
bombay
boxer
british_shorthair
chihuahua
egyptian_mau
english_cocker_spaniel
english_setter
german_shorthaired
great_pyrenees
havanese
japanese_chin
keeshond
leonberger
maine_coon
miniature_pinscher
newfoundland
persian
pomeranian
pug
ragdoll
russian_blue
saint_bernard
samoyed
scottish_terrier
shiba_inu
siamese
sphynx
staffordshire_bull_terrier
wheaten_terrier
yorkshire_terrier
0
5
10
15
20

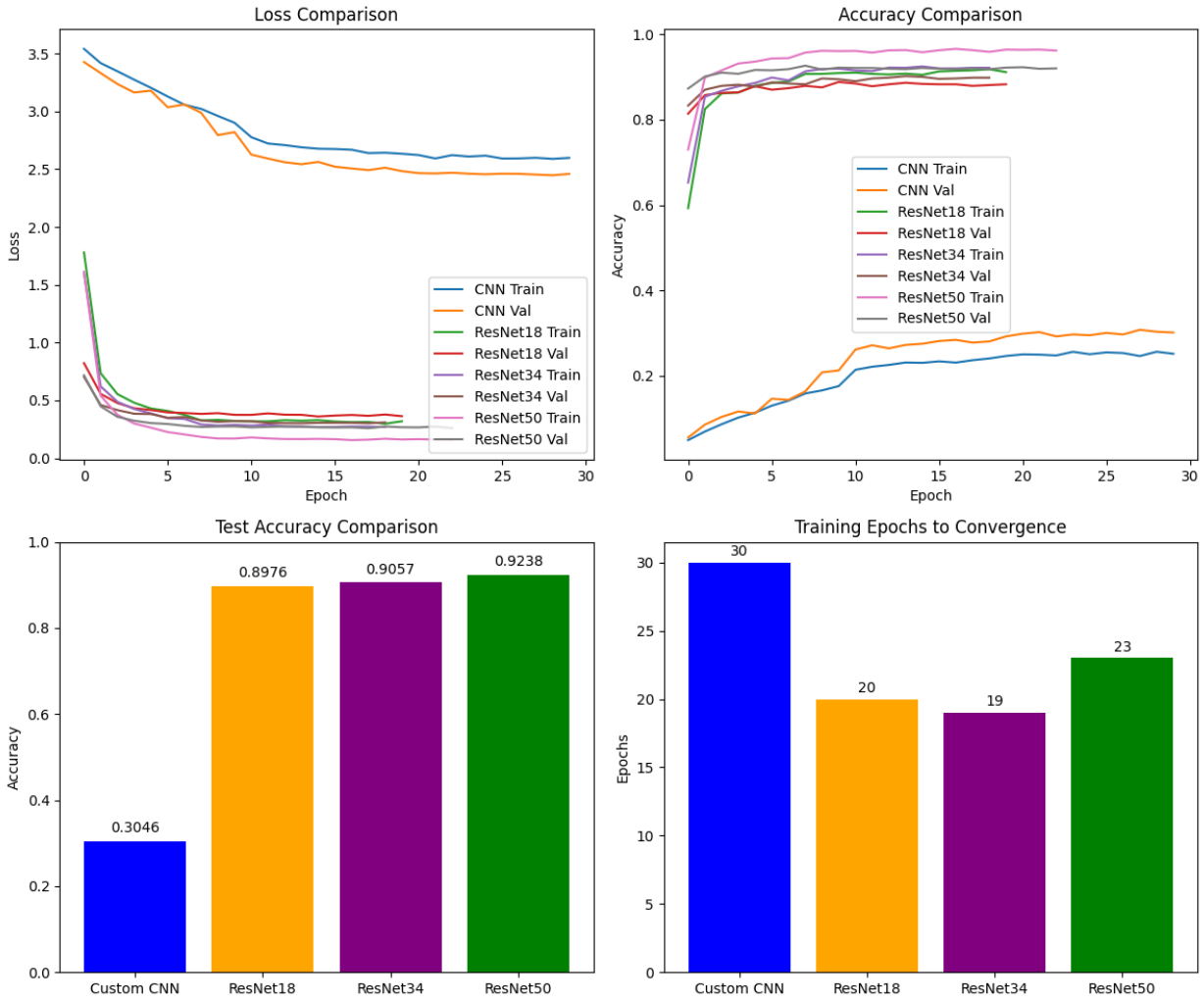

ResNet Confusion Matrix
abyssinian
american_bulldog
american_pit_bull_terrier
basset_hound
beagle
bengal
birman
bombay
boxer
british_shorthair
chihuahua
egyptian_mau
english_cocker_spaniel
english_setter
german_shorthaired
great_pyrenees
havanese
japanese_chin
keeshond
leonberger
maine_coon
miniature_pinscher
newfoundland
persian
pomeranian
pug
ragdoll
russian_blue
saint_bernard
samoyed
scottish_terrier
shiba_inu
siamese
sphynx
staffordshire_bull_terrier
wheaten_terrier
yorkshire_terrier
0
5
10
15
20
25
30
35


ResNet34 Confusion Matrix


ResNet50 Confusion Matrix

7. Comparison of Models

To compare models, we analyzed accuracy, number of parameters, depth, and convergence behavior:



Model	Accuracy	Depth	Training Epochs
Custom CNN	~30.46%	4 Conv	30
ResNet18	~88.76%	18	20
ResNet34	~90.57%	34	19

ResNet50	~92.38%	50	23

From the comparison we can clearly see the benefit of a pre-trained model, especially when we have limited training data or the task that we are doing is complex. Overall, from the results we can conclude that the custom cnn which I built from scratch needs more optimization in terms of adding layers and depth. RestNet18 showed a great performance. However, ResNet34 offers a sweet spot it terms of efficiency and performance although its accuracy was low compared to ResNet50 but the computational power of ResNet50 is high.

Finally, we can see how each model has converged over epoch and loss and accuracy of all the models. ResNet50 turned out to be the best compared to accuracy compared to all other models

8. Future Improvements

There are several improvements that we can make in my models for future improvements:

- We can fine tune the pretrained models, for example instead of freezing all the layers we make changes to those layers according to dataset.
- We can use more efficient or advanced architecture EfficientNet, DenseNet, or Vision Transformers
- Hyperparameter is one of the important aspects that we can work on to improve the model's efficiency by using techniques like grid search or Bayesian optimization
- We can use model ensembles, which means combining multiple networks for more robust predictions.
- For custom CNN we can increase the depth and add more layers for better performance and results.
- We can also use Grad-Cam to visualize the focus area of model through which we can have better insight

9. Conclusion

In this project I have demonstrated an end-to-end deep learning pipeline using pytorch to classify different breeds of pet's images. I started with the data processing, applied different transforms to enhance generalization in images which matches with real world scenarios. Then I trained 4 different models starting from building custom CNN from scratch and moving on to pre trained models like ResNet18, Resnet34 and ResNet50. Overall, the result

showed that the worse performing was the custom CNN which needed more depth or adding more layer and ResNet50 performed best out of the lots depending on the accuracy.

Through careful preprocessing, modularized development and evaluation of different models, we have achieved high accuracy in multi-class classification. Further work we can build on the project using more advanced models and interpretability methods.

GitHub Link:

https://github.com/MuhammadOsama380/Image-Classification-of-the-Pet-Breeds-