

# Oxford Pet Dataset Classification Report

Amr Ragab Abdelaziz

## 1. Oxford Pet Dataset Description

The Oxford-IIIT Pet Dataset is a well-known dataset for image classification and object detection tasks. It consists of 37 different pet breeds, with approximately 200 images per class. The dataset includes images of cats and dogs with varying poses, lighting conditions, and occlusions. Additionally, it provides annotations such as class labels, segmentation masks, and bounding boxes, making it useful for multiple computer vision tasks.

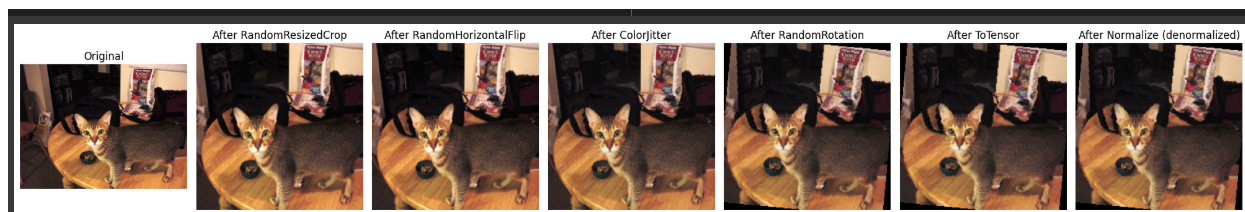
## 2. Task

The primary objective is to develop a machine learning model that performs multiclass classification of pet breeds using the Oxford-IIIT Pet Dataset. All the code can be found in this [Github Repository](#)

## 3. Dataset preparation

To prepare the dataset for training, we applied the following preprocessing steps:

- **Resizing:** All images were resized to a standard size (224,224) to ensure uniformity during training and to fit the model input layer.
- **Normalization:** Images were normalized using the mean and standard deviation of the ImageNet dataset to improve model convergence.
- **Data Augmentation:** Various augmentation techniques, such as random horizontal flips and rotations, were applied to improve the model's generalization ability.
- **Splitting:** The dataset was split into training and validation sets (80%,20%) respectively to evaluate model performance effectively.



## 3. Model Selection & Training

- ResNet 34

- **MobileNet**

We used a **ResNet-34** model pretrained on ImageNet for classification. This model was chosen because:

- It is pretrained on a large-scale dataset (ImageNet), which allows for faster convergence and better generalization.
- The architecture includes residual connections, which help mitigate the vanishing gradient problem in deep networks.
- The final fully connected layer was modified to classify 37 pet breeds instead of the original 1000 classes from ImageNet.

## Training

Framework used : Pytorch

## Parameters

- **Learning rate:** multiple values of learning rate were used (0.0001,0.001,0.012,0.0015) until we reached the best model. Learning rate decay was also used to reduce the **lr** as we move closer to the optimal point.
- **Optimizer:** Initially, I used Adam optimizer, but the performance was not progressing as expected. So, I switched to SGD optimizer.

## 4. Challenges and Problems Faced

### Overfitting

One of the primary challenges encountered was **overfitting**. While the model achieved high accuracy on the training set, validation performance was significantly lower, indicating that the model was memorizing training data rather than generalizing.

To address overfitting, we applied:

- **Data Augmentation:** Introducing variations in training images to improve generalization.
- **Dropout and Regularization:** Adding dropout layers with different values (0.3,0.4,0.5) and L2 regularization to prevent over-reliance on specific features.
- **Changing the optimizer :** Started with adam optimizer, then i changed it to SGD.
- **Learning Rate Scheduling:** Adjusting the learning rate dynamically to improve convergence and reduce overfitting.
- **Early Stopping :** the model's performance plateaus at some point. So, I added Early stopping to spare the wasted time and compute

## Slow training process

The model performance was not progressing quickly throughout the epochs, the difference

Further experiments are needed to fine-tune hyperparameters and improve validation accuracy.

## 5. Results

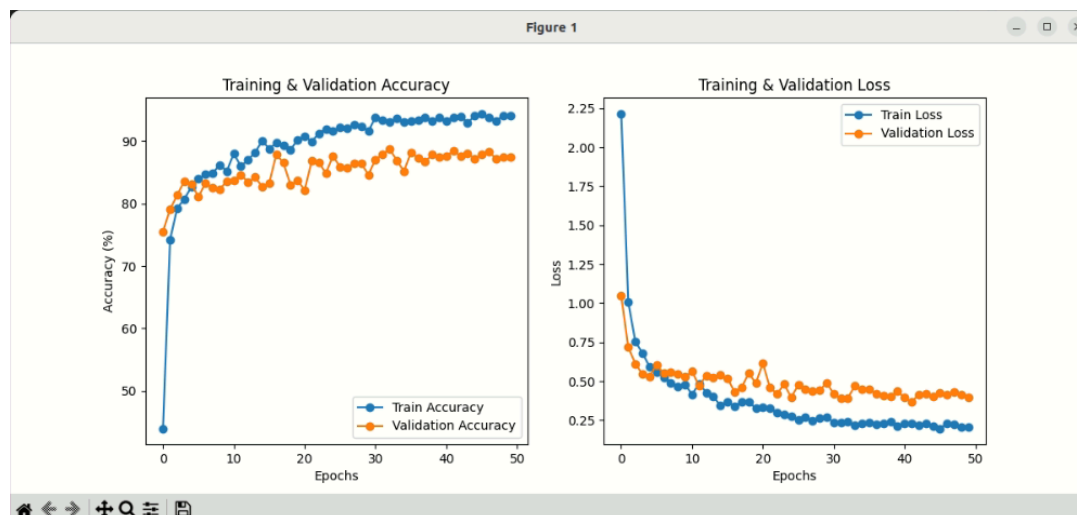
### ResNet34

After applying the mentioned procedures to reduce the overfitting, the model performed much better ,achieving:

- **training accuracy of 91%**
- **validation accuracy of 88%.**

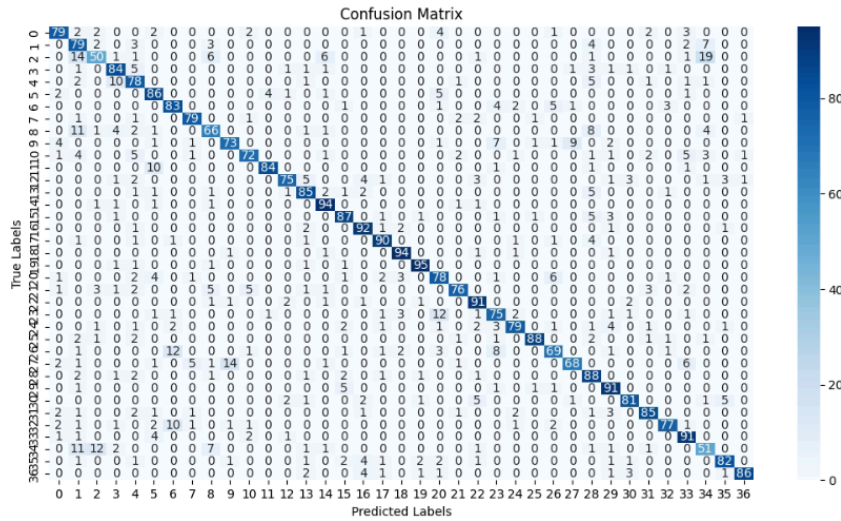
#### Training process

```
Epoch [26/30] | Train Loss: 0.3303 | Train Acc: 90.49% | Val Loss: 0.5799 | Val Acc: 83.97%  
Epoch [27/30] | Train Loss: 0.2805 | Train Acc: 91.81% | Val Loss: 0.5315 | Val Acc: 83.42%  
Epoch [28/30] | Train Loss: 0.2882 | Train Acc: 91.61% | Val Loss: 0.4267 | Val Acc: 87.09%  
Epoch [29/30] | Train Loss: 0.3022 | Train Acc: 90.96% | Val Loss: 0.4032 | Val Acc: 87.64%  
Epoch [30/30] | Train Loss: 0.2856 | Train Acc: 91.81% | Val Loss: 0.4448 | Val Acc: 88.18%  
Training complete. Model saved as resnet34_pets.pth
```



### Confusion matrix

A confusion matrix was generated to analyze misclassification patterns. It showed that most misclassified breeds had visual similarities, indicating that further feature extraction techniques could improve results.



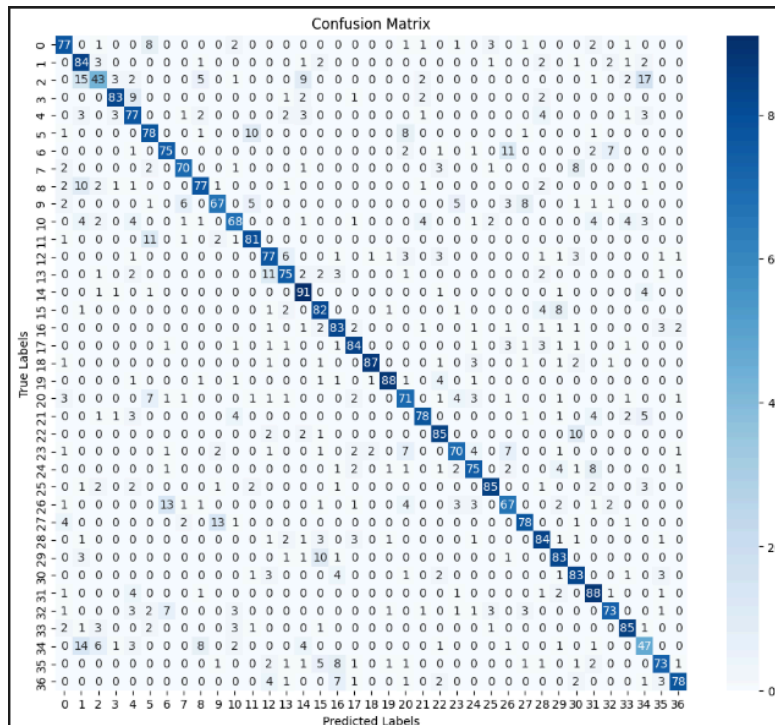
## MobileNet

Initially, MobileNet showed mediocre performance reaching ( **Training Acc : 84%, Val Acc: 74%**) which is less than ResNet34.

To further enhance the performance, I **unfroze all the layers** and tuned the **learning rate**. And the performance increased around 10% reaching **Training Acc : 91%, Val Acc: 82%**

```
Epoch [38/40] | Train Loss: 0.2802 | Train Acc: 91.92% | Val Loss: 0.5992 | Val Acc: 82.47%
Epoch [39/40] | Train Loss: 0.2788 | Train Acc: 91.85% | Val Loss: 0.5598 | Val Acc: 83.02%
Epoch [40/40] | Train Loss: 0.2790 | Train Acc: 91.85% | Val Loss: 0.5643 | Val Acc: 82.47%
```





## 6. Inference on test set

The trained model was tested on a separate test set. Inference was performed using images outside the training and validation set. The model demonstrated moderate classification performance.



ResNet 34 sample predictions against ground truth GT



Mobile Net sample predictions against ground truth GT

## 7. Testing the pipeline

A python script was added to test the pipeline of the code. It loads the data, initializes the model and ensures that a model was saved. Then it runs a quick test on dummy data.

```
FAILED (errors=2)
(pclass) hpc-01@hpc-lab-lux:/media/hpc-01/HDD/PClass/Pet-MultiClassification/tests$ python test_pipeline.py
../home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
../home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing 'weights=ResNet34_Weights.IMAGENET1K_V1'. You can also use 'weights=ResNet34_Weights.DEFAULT' to get the most up-to-date weights.
  warnings.warn(msg)
../home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing 'weights=None'.
  warnings.warn(msg)
..
Ran 6 tests in 2.488s

OK
```


## 8. CI pipeline

The CI pipeline in GitHub Actions automates testing for our project. It's set up in a `.github/workflows/ci.yml` file and runs on pushes or pull requests to the TestPipeline branch. Using Ubuntu, it grabs the code, sets up Python 3.9, installs stuff like PyTorch, and runs unit tests with **mock data** (via `CI=true`). This checks the data prep, training, and prediction parts quickly without real data, catching problems early.

```
- name: Set Up Python
  uses: actions/setup-python@v4
  with:
    python-version: "3.9" # Kept the same as your original

- name: Install Dependencies
  run: |
    python -m pip install --upgrade pip
    pip install torch torchvision # Explicitly install required packages
    # If you have a requirements.txt, uncomment the next line and adjust
    pip install -r requirements.txt

- name: Run Unit Tests with Unittest
  env:
    CI: "true" # Enable mock mode for tests
  run: |
    python -m unittest discover -s . -p "test_*.py" # Discover all test files
```

 **testing on mock data4**

Mock CI Pipeline #10: Commit `c53bf89` pushed by AmrRagab0

TestPipeline

6 minutes ago

2m 38s

...



## 9. Model Optimization

So far, the model was trained using **pytorch** which is a training framework. To use it for inference, we should use an inference framework such as **ONNX** to improve the inference speed of the model. So, We are going to:

- Convert the model to **ONNX**

```
Successfully installed onnx-1.17.0
(pclass) hpc-01@hpclablux:/media/hpc-01/HDD/PClass/Pet-MultiClassification/src$ python convert_to_onnx.py
/home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning:
  precated since 0.13 and may be removed in the future, please use 'weights' instead.
    warnings.warn(
/home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning:
  um or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior
  ResNet34_Weights.IMAGENET1K_V1'. You can also use 'weights=ResNet34_Weights.DEFAULT' to get the most up-to-
    warnings.warn(msg)
Model successfully converted to ONNX!
```

- Run the **ONNX** version and **compare** inference time with **pytorch**

```
(pclass) hpc-01@hpclablux:/media/hpc-01/HDD/PClass/Pet-MultiClassification/src$ python run_onnx_model.py
/home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning:
  precated since 0.13 and may be removed in the future, please use 'weights' instead.
    warnings.warn(
/home/hpc-01/anaconda3/envs/pclass/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning:
  um or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior
  None'.
    warnings.warn(msg)
Maximum output difference: 0.000001
PyTorch Inference Time: 0.037155 seconds
ONNX Inference Time: 0.021834 seconds
Speedup Factor (PyTorch/ONNX): 1.70x
```

Comparing the inference time (onnx/pytorch). The ONNX version was 1.7x faster than pytorch.

## 10. Future Work

To further improve model performance and deployment efficiency:

- **Model Optimization:** Apply **knowledge distillation** to train a smaller yet accurate model and use **quantization** to reduce inference time.
- **Training Visualization:** Integrate **TensorBoard** for better monitoring of training progress, loss trends, and gradient flow.
- **Enhanced Data Augmentation:** Utilize **color jittering**, **random erasing**, and **CutMix** to boost generalization.
- **Alternative Architectures:** Explore **EfficientNet**, or **Vision Transformers (ViTs)** for improved classification accuracy.