***Oxford-IIIT Pet Dataset Classification and Segmentation Project***

**Project Overview**

This project implements both breed classification and semantic segmentation on the Oxford-IIIT Pet Dataset, which contains images of 37 different pet breeds. The implementation consists of two main components:

1. A CNN-based classification model for breed identification /

2. A U-Net architecture for pet segmentation

**Data Overview**

- Dataset: Oxford-IIIT Pet Dataset

- Classes: 37 pet breeds

- Images: ~7,400 images

- Resolution: Variable (standardized to 224x224 during preprocessing)

- Annotations: Trimap segmentation masks for semantic segmentation

**Classification Implementation Details**

This project implements two different approaches for pet breed classification:

1. Custom CNN architecture from scratch
2. Fine-tuned ResNet34 pre-trained model

The goal is to classify pet breeds from the Oxford-IIIT Pet Dataset, comparing the performance of a custom-built solution against a transfer learning approach.

**Firstly: Custom CNN architecture from scratch**

* A screenshot of a computer

  AI-generated content may be incorrect.Architecture:
* Design Decisions

- Three convolutional blocks with increasing filter sizes (32 → 64 → 128)

- MaxPooling layers to reduce spatial dimensions

- Dropout layer (0.5) to prevent overfitting

- Final dense layer with softmax activation for multi-class classification

**Secondly: Fine-tuned ResNet34 pre-trained model**

**Training Configuration**

* Input image size: 224x224x3
* Batch size: 32
* Learning rate: 0.001
* Optimizer: Adam
* Loss function: Categorical Crossentropy
* Epochs: 50 (with early stopping)

**Data Augmentation**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Training Strategy**

1. Initial Training Phase:
   * Base model frozen
   * Only top layers trained
   * Learning rate: 0.001
   * Epochs: 10
2. Fine-tuning Phase:
   * Unfreeze last few layers of ResNet34
   * Lower learning rate: 0.0001
   * Additional epochs: 20

**Performance Comparison between the two methods of implementation:**

**Metrics Tracked**

* A graph of a graph showing the results of training and validation accuracy

  AI-generated content may be incorrect.Accuracy

A graph of a graph

AI-generated content may be incorrect.

CNN Pretrained- ResNet

* A graph of a graph

  AI-generated content may be incorrect.A graph of a training loss

  AI-generated content may be incorrect.Loss

CNN Pretrained- ResNet

* A purple and yellow grid with numbers

  AI-generated content may be incorrect.A screen shot of a graph

  AI-generated content may be incorrect.Confusion Matrix

CNN Pretrained- ResNet

**Pet Image Segmentation Using U-Net**

**Project Overview**

This project implements a deep learning solution for pet image segmentation using a U-Net architecture. The model segments pet images into three classes, separating the pet from the background and identifying key features.

**Project Structure**

**Data Processing**

The project uses a dataset organized as follows:

pet\_seg/

├── images/

│ └── [pet\_images].jpg

├── annotations/

│ ├── trainval.txt

│ ├── test.txt

│ └── trimaps/

│ └── [pet\_masks].png

└── results/

└── [segmentation\_results].jpg

**Core Components**

**Data Loading and Preprocessing**

* process\_data(): Processes file paths for images and their corresponding masks
* load\_data(): Splits data into training, validation, and test sets
* read\_image(): Loads and preprocesses images to 256x256 resolution
* read\_mask(): Loads and preprocesses segmentation masks
* tf\_dataset(): Creates TensorFlow dataset pipeline for efficient training

**Model Architecture**

The U-Net architecture consists of:

* Encoder: 4 convolutional blocks with max pooling
* Bridge: Connection between encoder and decoder
* Decoder: 4 upsampling blocks with skip connections
* Output layer: Softmax activation for 3-class segmentation (mask – background – boarder)

**Training Configuration**

* Image dimensions: 256x256x3
* Number of classes: 3
* Learning rate: 1e-4
* Batch size: 8
* Epochs: 10
* Optimizer: Adam
* Loss function: Categorical crossentropy

**Performance Metrics**

The model's performance is evaluated using:

* IoU (Intersection over Union): Measures overlap between predicted and ground truth masks

A graph of a line

AI-generated content may be incorrect.

* Dice coefficient: Measures segmentation accuracy

A graph of a line

AI-generated content may be incorrect.

* Pixel accuracy: Percentage of correctly classified pixels

A graph of a graph

AI-generated content may be incorrect.

***A screen shot of a computer program

AI-generated content may be incorrect.Key Functions***

* **Model Building:**
* A computer screen shot of a program code

  AI-generated content may be incorrect.**Convolution Block**

**Training Process**

1. Data Preparation:
   * Images are resized to 256x256
   * Pixel values are normalized to [0,1]
   * Masks are one-hot encoded
2. Model Training:
   * Implements early stopping
   * Reduces learning rate on plateau
   * Saves best model weights
   * Monitors validation loss
3. Callbacks:
   * ModelCheckpoint: Saves best model
   * ReduceLROnPlateau: Adjusts learning rate
   * EarlyStopping: Prevents overfitting

**Inference and Visualization**

The project includes visualization tools for:

* Training metrics over time
* Side-by-side comparisons of:
  + Original images
  + Ground Truth Masks
  + Predicted segmentation masks

A collage of images of a cat and a dog

AI-generated content may be incorrect.

**Results Storage**

Segmentation results are saved in the following format:

* Combined image showing original, ground truth, and prediction
* Saved to the specified output folder
* Maintains original image filename

**Best Practices and Notes**

1. Data Management:
   * Maintain consistent directory structure
   * Use appropriate train/validation/test split
2. Training:
   * Monitor validation metrics
   * Use appropriate batch size for memory
   * Implement early stopping
3. Inference:
   * Save predictions systematically
   * Maintain original image names
   * Include visualization tools
4. Performance Optimization:
   * Use TensorFlow dataset API for efficient data loading
   * Implement data prefetching
   * Use appropriate image preprocessing

**Combined Model Testing Implementation**

**Overview**

This section documents the implementation of a unified testing script that combines both the classification and segmentation models for pet image analysis. The script provides concurrent model inference and visualization capabilities.

**Key Features**

1. Concurrent Processing
   * Uses ThreadPoolExecutor for parallel model inference
   * Optimizes execution time for multiple models
   * Efficient resource utilization
2. Comprehensive Visualization
   * Original image display
   * Classification result overlay
   * Segmentation mask visualization
   * Side-by-side comparison
3. Result Storage
   * Saves original image
   * Stores classification result in text format
   * Exports segmentation mask as image
   * Organized output directory structure
4. Flexible Input/Output
   * Supports various image formats
   * Configurable output directory
   * Adaptable to different model architectures
   * Easy to extend for additional models

***Challenges and Limitations***

**Task’s requirements not finished:**

* Applying test units for different steps of preprocessing, training and testing.

**Computational Constraints**

1. Training Resource Limitations
   * Limited GPU availability significantly impacted experimentation
   * Long training cycles restricted the number of experiments
   * Memory constraints affected batch size selection
   * Trial and error approach was time-consuming due to training duration
2. Model Development Challenges
   * Restricted ability to test multiple architectures thoroughly
   * Limited hyperparameter optimization due to time constraints
   * Difficulty in implementing extensive cross-validation
   * Constrained experimentation with different model configurations

**Technical Challenges**

1. Custom CNN Development
   * Finding optimal architecture through trial and error was time-intensive
   * Limited ability to test different layer configurations
   * Resource constraints affected depth and width choices
   * Balancing model capacity with training efficiency
2. ResNet34 Fine-tuning
   * Limited experimentation with different freezing strategies
   * Restricted ability to test various learning rate schedules
   * Memory constraints affected batch size optimization
   * Time-consuming process of finding optimal training parameters

**Areas for Improvement**

With additional computational resources, several enhancements could be explored:

1. Model Architecture
   * Test more modern architectures (EfficientNet, Vision Transformer)
   * Experiment with different custom CNN configurations
   * Implement ensemble methods
   * Try different backbone networks
2. Training Strategy
   * Conduct more extensive hyperparameter tuning
   * Implement cross-validation
   * Test different optimization strategies
   * Experiment with various learning rate schedules
3. Data Processing
   * Test more augmentation techniques
   * Implement advanced preprocessing methods
   * Experiment with different input resolutions
   * Try various normalization strategies
4. Performance Optimization
   * Implement mixed precision training
   * Test different model pruning strategies
   * Experiment with knowledge distillation
   * Optimize inference speed