This project implements convolutional neural networks (CNNs) and transfer learning using VGG16 to classify cats and dogs with high accuracy. Data augmentation, normalization, and fine-tuning techniques improve model generalization and performance, achieving up to 98% validation accuracy.

**Part 1 – Data Loading and Preprocessing**

* **Dataset**: 25K Cats and Dogs Images were loaded into a PyTorch Dataset and batched using a DataLoader. Training set of 80%, Validation Set 20%, External Test Set.
* **Normalization**: Pixel values were standardized (mean, std) to stabilize and speed up training.
* **Data Augmentation**: Applied transformations (random flips, rotations, crops) to reduce overfitting and improve generalization.
* **Batching**: Used mini-batches instead of feeding the full dataset at once. Batch size of 64 was chosen to balance memory usage and convergence stability.

**Part 2.1 – Model Design**

* **Architecture**: Implemented a CNN with 2 convolutional layers, activation functions (ReLU), and fully connected layers.
* **Motivation**: Convolutional layers capture spatial features, and fully connected layers perform classification.
* **Initialization**: Weights initialized properly to avoid vanishing/exploding gradients.????

**Part 2.2 – Training and Evaluation**

* **Loss Function**: BCE loss was used for classification.
* **Optimizer**: Adam for weight updates.
* **Learning Rate**: Tuned carefully; too high caused divergence, too low slowed convergence. 0.001
* **Epochs**: Multiple passes through the dataset ensured the network learned effectively. 20 epochs
* **Metrics**: Training accuracy reached ~63%, validation accuracy peaked at ~61%. Loss curves indicated good convergence without instability. Probs not enough epochs

**Part 3 – Improvements**

* **Regularization**: 2 Additional Residual Blocks to improve convergence
* **Batch Normalization**: Used before ReLu to stabilize training and allow higher learning rates.
* **Pooling:** Max Pooling of kernel size 2 and stride 2 after each residual block or activation.
* **Result**: Improvements raised validation accuracy to ~80%

**Part 4 – Transfer Learning with VGG16**

* **Architecture**: Leveraged **VGG16 pretrained on ImageNet** as the base model, with the original classifier replaced by a custom head with ReLU, BatchNorm, Dropout, and final Sigmoid.
* **Motivation**: Training a deep model from scratch with limited data leads to overfitting. Transfer learning starts from pretrained weights, allowing much faster convergence and higher accuracy.
* **Feature Freezing**: Initially froze the convolutional feature extractor and trained only the custom classifier head.

**Results (Frozen Features):**

* Achieved **~98% training accuracy** and **~98% validation accuracy** after 20 epochs.
* Both training and validation curves showed smooth convergence, with minimal overfitting.

**Part 4.2 – Fine-Tuning**

* **Unfreezing**: After training the head, all convolutional layers were unfrozen and trained with a **very small learning rate (0.0001)** to fine-tune weights.
* **Reasoning**: Small LR ensures pretrained filters are adjusted gradually, preventing catastrophic forgetting of the ImageNet features.

**Results (Fine-tuned):**

* Training accuracy increased up to **99%**.
* Validation accuracy peaked at **~98%**
* Compared to frozen-feature training, fine-tuning yielded slightly higher peak validation accuracy but less stability.

**Overall Comparison:**

* Transfer learning (both frozen and fine-tuned) **outperformed scratch models** by a large margin (scratch: ~80% validation vs. transfer: ~98%).
* **Frozen features training** was already very effective, fast, and stable.
* **Fine-tuning** achieved the highest training accuracy
* **The final model could be submitted to Kaggle to evaluate accuracy**

**Interview Preparation: Key Concepts**

**Data Handling**

* **What is a DataLoader?**  
  A PyTorch utility that feeds mini-batches of data to the model. It supports shuffling (improves learning by breaking data order correlations) and parallel loading (faster training).
* **Why Normalization?**  
  Keeps input values in a consistent range, helping gradients flow better and training converge faster.
* **Why Data Augmentation?**  
  Simulates new data without collecting more samples. Reduces overfitting by making the model invariant to transformations like flips or rotations.
* **Batch Size Considerations**:
  + Small batch → noisier updates, better generalization, but slower.
  + Large batch → faster, smoother updates, but may overfit.

**Model & Training**

**1. Why are CNNs suitable for image classification?**  
CNNs exploit local connectivity and weight sharing, making them efficient at capturing spatial patterns in images.

**2. Why use ReLU instead of sigmoid or tanh?**  
ReLU avoids vanishing gradients, allowing faster training and better convergence compared to sigmoid/tanh.

**3. What is Dropout and why is it used?**  
Dropout randomly deactivates neurons during training to prevent co-adaptation and reduce overfitting.

**4. What is Batch Normalization and how does it help?**  
BatchNorm normalizes activations in each mini-batch, stabilizing training, allowing higher learning rates, and often improving accuracy.

**5. What is the difference between an epoch and an iteration?**

* Epoch: one full pass through the dataset.
* Iteration: one update step (i.e., processing one batch).

**6. Why is learning rate important?**  
It controls the step size in gradient descent. Too high → divergence; too low → slow convergence.

**Data & Preprocessing**

**7. Why use a DataLoader?**  
Supplies mini-batches and parallelizes I/O (with num\_workers) so GPU stays fed efficiently.

**8. Why apply data augmentation?**  
Effectively increases dataset size without new images; improves generalization and validation accuracy.

**9. Why is normalization important, especially with pretrained networks?**  
Pretrained networks expect input normalized with ImageNet statistics (mean [0.485,0.456,0.406], std [0.229,0.224,0.225]); otherwise, performance drops.

**10. How do you choose batch size?**  
Depends on memory and model size. Common defaults: 32 for large models on GPU, up to 512 for small images or CPU prototyping.

**Transfer Learning & Fine-Tuning**

**11. When is transfer learning beneficial?**  
When target data is limited and tasks are similar to the source (e.g., ImageNet natural images). Can boost accuracy significantly (~30% in your project).

**12. When might transfer learning be unsuitable?**  
If the source and target domains are very different (e.g., medical signals, NLP), pretrained features may not transfer well.

**13. Why freeze layers in a pretrained network before training the classifier head?**  
To prevent large updates to pretrained weights and focus training on the new top layers (classifier head).

**14. Why use a very small learning rate when fine-tuning pretrained layers?**  
The pretrained weights are already close to a good solution; small learning rates make subtle adjustments without destroying learned features.

**15. How do you decide which layers to fine-tune versus keep frozen?**

* Freeze early layers (capture general features like edges/textures).
* Fine-tune later layers (task-specific features).

**Training Strategies & Evaluation**

**16. How do epochs, batch size, and early stopping interact to prevent overfitting?**

* Multiple epochs allow learning, but too many may overfit.
* Appropriate batch size balances memory usage and convergence.
* Early stopping monitors validation performance to stop training before overfitting.

**17. How do training and validation accuracy/loss curves indicate underfitting or overfitting?**

* Training loss high + low accuracy → underfitting.
* Training accuracy high + validation accuracy low → overfitting.
* Smooth convergence without divergence indicates well-tuned training.