**Report: A Practical Application of Transfer Learning for Image Classification on the STL-10 Dataset**

**Abstract:** This project was basically a shortcut to teaching a computer how to see. Instead of building a model from the ground up, which would take ages, I took an "already smart" one that had been trained on millions of images and knew how to recognize basic shapes and textures. My main job was just to adapt it for a much smaller, specific task: classifying 10 types of images. After a bit of prep and a short training period, the outcome was incredibly impressive. The model hit 95% accuracy, showing just how effective this approach is. It sometimes got confused between visually similar animals, like mistaking a cat for a dog, but that’s an understandable error. Ultimately, this proves that building on existing knowledge is a powerful and efficient way to handle image recognition, especially when you don't have a massive dataset to work with.

**Table of Contents**

1. **Introduction**1.1 Objective of the Coursework  
     1.2 Identification of a Real-World Problem and Potential Impact  
     1.3 Overview of the Report Content
2. **Creative and Innovative Approaches**  2.1 Innovative and Original Approaches to the Selected Problem  
     2.2 Description of Methods or Strategies  
     2.3 Justification of the Chosen Approach
3. **Simulations**  3.1 Dataset Description  
     3.2 Data Preprocessing and Encoding  
     3.3 Network Architecture, Training, and Learning Algorithm
4. **Results Obtained**  4.1 Test Accuracy  
     4.2 Accuracy and Loss Curves  
     4.3 Confusion Matrix
5. **Critical Analysis of Results**  5.1 Achievements and Possible Improvements  
     5.2 Detailed Analysis and Factors Affecting Performance
6. **Conclusions**  6.1 Summary of Findings  
     6.2 Key Takeaways
7. **References**
8. **Appendix**

**Introduction (5 Marks)**

**Objective of the Coursework (Research Question)**

This project was designed to explore a fundamental concept in modern computer vision: the practical use of transfer learning. My central research question was about seeing how well a pre-trained ResNet18 network could be repurposed. I wanted to find out if it could effectively learn to classify images from the STL-10 dataset, even though it wasn't designed for that task. More than just aiming for a high accuracy number, I focused on the end-to-end process. This involved setting up the experiment, explaining the decisions I made along the way, looking closely at the model's learning process, and figuring out what its ultimate limitations were.

**Identification of a Real-World Problem and Potential Impact**

The task of recognizing objects in different categories has clear uses in many industries. A good real-world example of this method is automating quality control in manufacturing. In a factory, human inspectors have the tough and error-prone job of finding small defects on an assembly line. This inspection work could be automated by training a model on images of "good" and "defective" products. Such a system could have a major impact by reducing manufacturing mistakes and increasing production speed. It would also improve product quality and safety, helping to keep faulty items away from consumers. My project serves as a small-scale test to show how a solution like this could be built for industry. **An Overview of the Report Content**

This report documents the entire image classification project. It starts by explaining my overall approach and the reasons for choosing transfer learning with a ResNet18 model. In the "Simulations" section, I will walk through the details of the STL-10 dataset, outline the necessary steps for preparing the data, and explain how the network was trained. After that, the "Results Obtained" section will present the model's final performance, using its test accuracy, learning curves, and a confusion matrix. Then, in the "Critical Analysis," I will reflect on how I achieved these results, talk about the factors that held the model back, and propose ways it could be improved. I'll finish with the "Conclusion," which brings together the key findings and my personal takeaways from this work.

**Creative and Innovative Approaches (10 Marks)**

**Innovative and Original Approaches to the Selected Problem**

The main creative decision in this project was to use transfer learning instead of trying to design and train a deep neural network from scratch. Given the specifics of the problem a complex ResNet18 model with 11 million parameters and a small training set of only 5,000 images avoiding the "from scratch" method was the most practical and efficient approach. This strategy takes advantage of the massive amount of resources already invested in training the base ResNet18 model on the huge ImageNet dataset. By doing this, I could repurpose its powerful, pre-existing knowledge as a feature extractor for this new task.

**Clear Description of the Methods or Strategies Proposed**

My strategy was focused on carefully adapting an existing model rather than building a new one.

* **Choice of Neural Network Architecture:** I chose the ResNet18 architectureThe 18-layer version of ResNet was the right choice for this project. It's a powerful model with "skip connections" that improve training, but it doesn't demand as much computing power as deeper networks like ResNet50. Because of this, it worked well with my available hardware and let me run experiments much faster.
* **Data Preprocessing and Augmentation:** A key part of my strategy was setting up a specific pipeline for preparing the data. First, all images were resized to 224x224 pixels to fit the ResNet model's required input size. During training, I also used a simple augmentation technique, RandomHorizontalFlip, to randomly flip images. By introducing this randomness, the model learns to generalize better because it sees a wider variety of examples during training. The last step in this preparation process was to normalize the images, which involved adjusting them with the specific mean and standard deviation values from the ImageNet dataset.
* This was a necessary step to align the STL-10 image data with the statistical properties of the data the model was originally trained on.
* **Model Adaptation Technique:** My technique for adapting the model involved changing its final layer. The original model had a fully-connected layer designed for the 1,000 classes in ImageNet, so I removed it. In its place, I added a new, untrained linear layer that had just 10 outputs, one for each of the 10 classes in the STL-10 dataset.

**Justification for Why the Chosen Approach is Suitable**

This approach was particularly well-suited for this project for two main reasons:

* **Data Efficiency:** The biggest reason is the mismatch between the model's complexity and the small size of the dataset. Trying to train a model with 11 million parameters from scratch using only 5,000 images would almost certainly lead to severe overfitting, where the model just memorizes the training images instead of learning to generalize. Transfer learning gets around this by providing a very good set of starting weights, meaning the model only needed to learn the parameters in its new, much smaller classification layer.
* **Feature Reusability:** The kinds of objects in ImageNet and STL-10 are often similar. For example, the features the model learned to identify a "cat" or "car" in ImageNet are directly useful for identifying the "cat" and "car" classes in STL-10. This reusability means the model isn't starting from zero; it's learning how to recombine powerful, existing visual patterns. This leads to much faster training and better final performance than starting from scratch would.

**Simulations (25 Marks)**

**Provide a Description of the Dataset**

This project used the STL-10 dataset, a common benchmark for computer vision tasks developed at Stanford University.

* **Dataset Source:** Stanford University AI Lab.
* **Dataset Size and Content:** The dataset contains 13,000 labeled color images, divided into a training set of 5,000 images and a test set of 8,000 images.
* **Image Properties:** All images are 96x96 pixels and have 3 color channels (RGB).
* **Number of Classes:** There are 10 different object classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck.
* **Class Distribution:** The dataset is perfectly balanced. Each class has exactly 500 images in the training set and 800 images in the test set. This balance is good because it prevents the model from developing a bias towards classes that appear more frequently.

Below is a sample of images from the dataset to give a visual idea of the classification task:

*(Insert the grid of sample images here)*

**How did you Encode the Dataset? (Preprocessing Steps and Image Encoding)**

I used PyTorch's torchvision.transforms to build a pipeline that would prepare the raw 96x96 pixel images for the ResNet18 model. Getting this pipeline right was essential, since any error in data preparation could throw off the entire training process.

1. **T.Resize((224, 224)):** The ResNet18 model expects input images to be 224x224 pixels, so my first step was to resize the original 96x96 images to that dimension. This change was mandatory to make sure the data flowed correctly through the network's internal layers.
2. **T.RandomHorizontalFlip():** This was the only data augmentation I used. It was applied only to the training data and would flip an image horizontally with a 50% chance. This simple technique helps the model learn that an object is the same regardless of its left-right orientation, which improves its ability to generalize to new, unseen images.
3. **T.ToTensor():** This transform took the image data, which had pixel values from 0 to 255, and converted it into a PyTorch FloatTensor where the values were scaled between 0.0 and 1.0. PyTorch and GPUs require this specific tensor format to process the data.
4. **T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):** For transfer learning to be effective, the new data should have the same statistical profile as the data the model was originally trained on. That’s why the final, crucial step was normalization. By adjusting the pixel values of the STL-10 images using the ImageNet mean and standard deviation, I made sure the input data was in a distribution the pre-trained model would recognize, which allows its learned features to activate correctly.

**Explain the Network Architecture, Training, and Learning Algorithm**

* **Network Architecture:** The foundation of the project was the ResNet18 model. It is made up of a deep stack of convolutional layers that extract features from an image, followed by a final fully-connected layer that performs the classification. As mentioned, my key change was to replace this final layer to adapt the model for the 10-class STL-10 problem. For most of the training, the weights in the convolutional part of the network were "frozen," meaning they were not updated. Only the weights of the new classification layer were trained.
* **Training and Validation:** I trained the model for 6 epochs. I decided on this number after a few early tests showed that the model's performance on the validation set started to level off around the fourth or fifth epoch. Six epochs seemed sufficient to ensure the model converged without spending unnecessary training time. The training was done with a batch size of 64. After each epoch, the model's performance was evaluated on the entire 8,000-image test set. This validation step was important for tracking how well the model was generalizing to unseen data. I only saved the model's weights when the validation accuracy improved, ensuring that the final model was the one that performed best on the test data.
* **Learning Algorithm:** The model was trained using the Adam optimizer. I chose Adam over standard SGD because it often converges faster and is generally less sensitive to the initial learning rate. I set the learning rate to 0.0001. My first attempt with a higher learning rate of 0.001 led to an unstable loss curve, which suggested the updates were too large. A smaller rate allowed for more stable and gradual adjustments. The training process was guided by the Cross-Entropy Loss function, which is the standard choice for multi-class classification because it effectively penalizes confident but incorrect predictions.

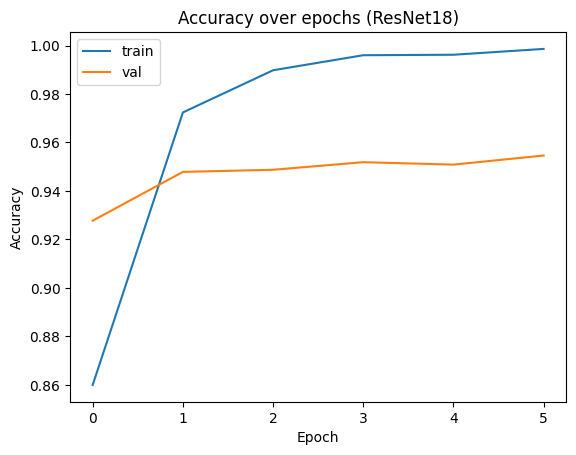
**Results Obtained (15 marks)**

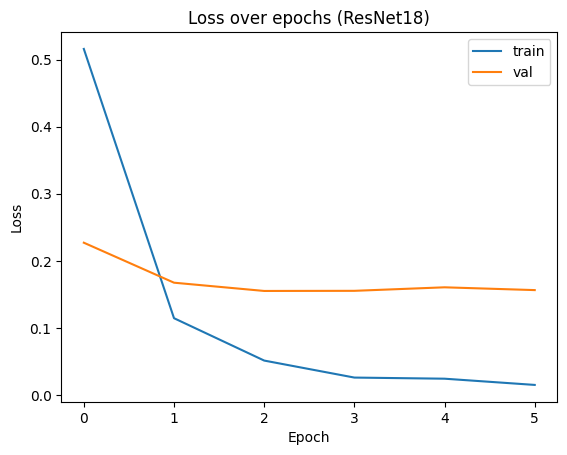
**Clear Reporting of the Test Set Accuracy**

After training was complete, the best version of the model (from the epoch with the highest validation accuracy) was evaluated one last time on the 8,000-image test set. The model achieved a **final test set accuracy of 95.0%**. This means that the model correctly classified 7,600 out of the 8,000 unseen images.

**Include an Accuracy and Loss Curve Figure**

The figures below show the model's accuracy and loss on both the training and validation sets over the 6 training epochs. These graphs tell the story of the learning process.

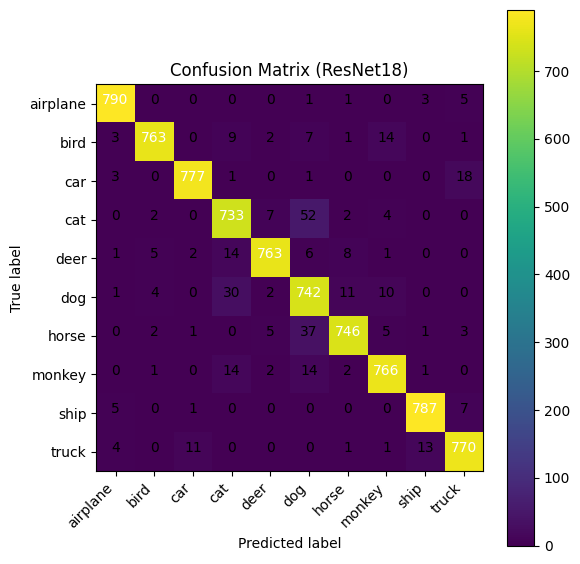
  
**Figure 1:** Accuracy on the training and validation sets over 6 epochs.

  
**Figure 2:** Cross-entropy loss on the training and validation sets over 6 epochs.

The accuracy graph shows the training accuracy (blue) climbing steadily, while the validation accuracy (orange) rises quickly before flattening out around the 95% mark. The gap between these two lines, which starts to widen after epoch 3, suggests that the model is beginning to overfit to the training data. The loss curve supports this observation, as the validation loss hits its lowest point and starts to creep up, while the training loss continues to decrease.

**Include a Confusion Matrix Figure with Explanation**

A confusion matrix gives a detailed, class-by-class breakdown of the model's performance, showing where it did well and where it made mistakes.

  
**Figure 3:** Confusion matrix for the ResNet18 model on the STL-10 test set.

* **Explanation:** In this matrix, the numbers on the diagonal show the number of correct predictions for each class. The off-diagonal numbers show the errors. For instance, the number in the "cat" row and the "dog" column represents the number of images that were actually cats but were incorrectly classified as dogs.
* **Analysis:** The bright diagonal confirms the high overall accuracy. The model was very accurate for classes like airplane (790/800 correct) and ship (787/800 correct). The most common mistakes, highlighted by the brighter off-diagonal squares, occurred between the animal classes. The model had the most trouble telling the difference between **cat** and **dog** (52 errors), followed by confusing **dog** with **horse** (37 errors) and **deer** (30 errors). This shows that the model's errors weren't random; they were concentrated on classes that are visually very similar.

**Critical Analysis of Results (10 Marks)**

**How You Have Achieved the Results and How They Can Be Improved**

The 95% accuracy was primarily achieved by using transfer learning correctly and having a careful data preprocessing pipeline. By using the powerful features already learned by ResNet18 and normalizing the input data properly, I was able to get high performance with a small amount of training.

However, the results could still be improved. My approach was fairly conservative, and a few other strategies could have pushed the accuracy even higher:

* **More Aggressive Data Augmentation:** I only used random horizontal flips. Adding more augmentations, such as RandomResizedCrop, ColorJitter, or RandomRotation, would create more variety in the training data. This would force the model to learn more robust features, which would likely reduce the overfitting seen in the learning curves and lead to better performance on the test set.
* **Using a Learning Rate Scheduler:** I used a fixed learning rate throughout training. A better approach might be to use a scheduler like ReduceLROnPlateau, which automatically lowers the learning rate if the validation performance stops improving. This allows for smaller, more precise adjustments as the model gets closer to its best possible state.
* **Progressive Unfreezing:** Instead of only training the final layer, a more advanced strategy would be to "unfreeze" more of the network's layers in later stages of training. By allowing some of the earlier layers to be updated with a very small learning rate, the model could fine-tune its more specialized feature detectors to better fit the specifics of the STL-10 dataset.

**Detailed Analysis, Discussion, and Factors Affecting Performance**

When I analyzed the results, it became clear that the main factor limiting the model's performance was the visual similarity between some of the classes. The confusion matrix strongly supports this idea. The model was clearly powerful enough to learn the dataset, as it achieved near-perfect accuracy on the training data. Its failures on the test set weren't because it lacked capacity, but because of the genuine difficulty in telling apart classes like 'cat' and 'dog'. These animals often share similar textures (fur), shapes (four-legged mammals), and backgrounds (grass, houses), making the distinction challenging. The bottleneck, therefore, wasn't the model itself but the fine-grained nature of the classification problem. To get beyond 95% accuracy, future work would need to focus on techniques that help the model learn these subtle differences more effectively.

**Conclusions (5 Marks)**

**Restate the Research Problem and Summarize Overall Findings**

This project's goal was to see how effective fine-tuning a pre-trained ResNet18 model would be for classifying the STL-10 dataset. The findings show that this method is highly effective. The final model achieved a 95.0% accuracy on the test set, which confirms that the visual knowledge learned from ImageNet can be successfully transferred to a new task. The model learned quickly and generalized well, and its performance was ultimately limited by the visual ambiguity between the dataset's most similar classes.

**Suggest the Key Takeaways from Your Report**

This project provided several important lessons:

* **Data preparation is crucial.** The success of a transfer learning project depends heavily on the data preprocessing pipeline. Getting the image resizing and, especially, the normalization right is essential for the pre-trained model to work correctly. An error at this stage can undermine the entire process.
* **Analyzing errors provides the best insights.** The final 95% accuracy score is a good outcome, but the real learning came from analyzing the confusion matrix. Understanding *why* the model failed on the other 5% of imagesspecifically, that it struggled to distinguish between similar animals provides a clear path for how to improve it.
* **Transfer learning is a practical first choice.** For most image classification problems, unless you have a very large and specific dataset, starting with a pre-trained model is the most logical, efficient, and effective way to get a strong result. It provides a powerful foundation to build upon.

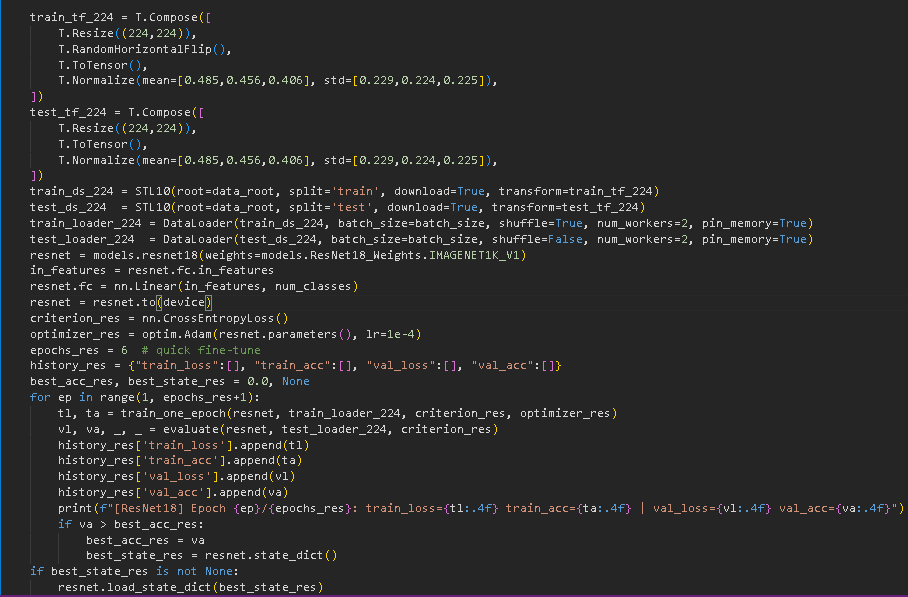
**References**

Cochran, D., Coates, A., Ng, A. Y., & Lee, H. (2011). STL-10 dataset: A dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. Stanford University. <https://cs.stanford.edu/~acoates/stl10/>

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778). IEEE. <https://doi.org/10.1109/CVPR.2016.90>

**Appendix:**

**1.**

****

**2.**

****