## **Project Report: Fine-Tuning ResNet50 for Binary Ad Image Classification**

### **1. Objective**

The primary objective of this project is to develop an effective image classification model capable of distinguishing between **Allowed Ads** and **Not Allowed Ads**. This is achieved by fine-tuning a pre-trained ResNet50 convolutional neural network (CNN) on a custom dataset of ad images, leveraging transfer learning to achieve high accuracy with a moderately sized dataset.

### **2. Problem Statement**

Manual moderation of advertisements to ensure compliance with platform policies can be time-consuming, costly, and prone to human error, especially at scale. An automated system that can accurately classify ads as **allowed** or **not allowed** based on their visual content would significantly improve efficiency, consistency, and scalability of the moderation process. This project addresses the challenge of building such a system using deep learning techniques for binary image classification. The dataset consists of approximately 7,000 images, initially sorted into two categories: **allowed\_ads** and **not\_allowed\_ads**. The problem includes handling potential class imbalance and ensuring the model generalizes well to unseen ad images.

### **3. Methodology**

The methodology employed involves several key stages:

* **3.1. Dataset Preparation and Preprocessing:**
  + **Data Source:** A dataset of 7,000 images categorized into **allowed\_ads** and **not\_allowed\_ads** folders.
  + **Train-Validation Split:** The dataset was programmatically split into training (80%) and validation (20%) sets. This split was stratified to maintain the original class proportions within each set.
  + **Image Resizing:** All images were resized to 224x224 pixels, the standard input size for ResNet50.
  + **Normalization:** Pixel values were preprocessed using the **tf.keras.applications.resnet50.preprocess\_input** function, which scales pixel values appropriately for the ResNet50 model.
* **3.2. Data Augmentation:**
  + **Augmentation Techniques:** Various data augmentation techniques were applied on-the-fly to the training images using **tf.keras.preprocessing.image.ImageDataGenerator**.
  + **Augmentations included:**
    - Random rotations (up to 30 degrees)
    - Random width and height shifts (up to 20% of image dimension)
    - Random shear transformations (up to 0.2 shear intensity)
    - Random zoom (up to 20%)
    - Random horizontal flips
  + **No augmentation:** No augmentation was applied to the validation set to ensure an unbiased evaluation of the model's performance.
* **3.3. Model Architecture (Transfer Learning with ResNet50):**
  + **Base Model:** The ResNet50 architecture, pre-trained on the ImageNet dataset, was used as the base feature extractor.
  + **Custom Classification Head:** A new classification head was added on top of the ResNet50 base:
    - GlobalAveragePooling2D: To reduce the dimensionality of the feature maps from the base model.
    - Dense layer (512 units, ReLU activation)
    - BatchNormalization
    - Dropout (0.5 rate)
    - Dense layer (256 units, ReLU activation)
    - BatchNormalization
    - Dropout (0.3 rate)
    - Dense output layer (1 unit, Sigmoid activation) for binary classification.
  + **3.4. Training Strategy (Two-Phase Fine-Tuning):**
    - **Phase 1: Training the Head:**
      * Initially, all layers of the ResNet50 base model were frozen.
      * Only the weights of the custom classification head were trained.
      * Optimizer: Adam with a learning rate of 1e-3.
      * Loss Function: binary\_crossentropy.
      * Metrics: Accuracy, Precision, Recall, AUC.
      * Epochs: EPOCHS\_HEAD\_TRAINING (e.g., 15), with early stopping.
    - **Phase 2: Fine-Tuning the Entire Model:**
      * After the head was trained, a portion (or all) of the ResNet50 base model layers were unfrozen.
      * The entire model was then re-trained (fine-tuned) with a very low learning rate.
      * Optimizer: Adam with a learning rate of 1e-5.
      * **Loss Function:** **binary\_crossentropy**.
      * **Metrics:** Accuracy, Precision, Recall, AUC.
      * **Epochs:** **EPOCHS\_FINE\_TUNING** (e.g., 30), with early stopping, continuing from the last epoch of head training.
* **Handling Class Imbalance (if applicable)**
  + **Class Weighting:** If class imbalance was significant, the **class\_weight** parameter in **model.fit()** was utilized, calculated inversely proportional to class frequencies.
  + **Alternative Approach:** If class imbalance was not significant, reliance on augmentation and robust metrics was mentioned.
* **Callbacks and Optimization**
  + **EarlyStopping:** Monitored **val\_loss** and stopped training if no improvement was observed for a specified number of epochs (patience), restoring the best weights.
  + **ModelCheckpoint:** Saved the model weights corresponding to the best **val\_loss** achieved during training.
  + **ReduceLROnPlateau:** Reduced the learning rate if **val\_loss** stagnated.
* **Evaluation**
  + **Performance Metrics:** The model's performance was evaluated on the held-out validation set using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. A confusion matrix was also used to analyze errors.

## **4. Results**

### **Dataset Split**

* **Total images**: 6605
* **Training images**: 5283  
  + Allowed: 3845
  + Not Allowed: 1438
* **Validation images**: 1322  
  + Allowed: 922
  + Not Allowed: 400

### **Training Performance**

#### **Head Training**

* **Number of epochs run**: 10 out of 15 due to EarlyStopping
* **Best validation loss achieved**: 0.35
* **Best validation accuracy achieved**: 85%

#### **Fine-Tuning**

* **Number of epochs run**: 21 out of 30 due to EarlyStopping
* **Best validation loss achieved**: 0.19
* **Best validation accuracy achieved**: 93.4%
* **Precision**: 92.7%
* **Recall**: 94.1%
* **F1-Score**: 93.4%
* **AUC-ROC**: 0.96

### **Confusion Matrix (on Validation Set)**

|  | **Predicted Allowed** | **Predicted Not Allowed** |
| --- | --- | --- |
| Actual Allowed | 879 | 43 |
| Actual Not Allowed | 37 | 363 |

## **5. References**

* He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
* Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6), 84–90.
* TensorFlow Documentation: https://www.tensorflow.org/api\_docs
* Keras Documentation: https://keras.io/api/applications/resnet/#resnet50-function