ChatGPT

Fake News Image Detection System

Project Overview

This project presents an ensemble-based deep learning framework for detecting fake news images. It combines four complementary models—ResNet50, Vision Transformer (ViT-B/16), EfficientNet-B4, and a custom multi-scale CNN—via three fusion strategies (early, late, and attention-based) to leverage their joint strengths [1] [2]. We train and evaluate on a curated dataset of ~2,050 real/fake images [3] [4]. Experiments demonstrate state-of-the-art performance: for example, the attention-based fusion achieves an F1-score ≈0.9902 and AUC-ROC ≈0.9981 on the test set [5] [6]. The system delivers robust, real-time inference (≈42 ms/image) and an explainable ensemble pipeline suitable for deployment [7] [4].

Problem Statement

Fake news images—manipulated, miscaptioned, or AI-generated visuals—undermine information integrity and influence public opinion [8]. Prior single-model detectors often fail to generalize across diverse manipulation techniques and new generation methods (e.g. GANs, diffusion models) [9] [10]. Simple fusion approaches or unimodal systems overlook complementary features and lack interpretability. This work addresses these gaps by building a unified, multi-model system that improves generalization and explainability in fake-image detection [9] [11].

System Architecture

The pipeline converts each input image into multiple feature embeddings and then fuses them for classification. Specifically:
- **ResNet50:** A 50-layer pretrained CNN. We remove its final classification head and use the global average pooling output (2048-D embedding) as features [12] [13].
- **Vision Transformer (ViT-B/16):** A 12-layer transformer that splits the 224×224 image into 16×16 patches. We extract the class-token embedding (768-D) after the final encoder [14] [15].
- **EfficientNet-B4:** A compound-scaled CNN architecture pretrained on ImageNet, yielding a 1792-D feature vector from its penultimate layer [16] [17].
- **Custom CNN:** A from-scratch convolutional network with parallel multi-scale branches (kernel sizes 3×3, 5×5, 7×7) and channel-attention modules, producing a 1024-D embedding [18] [19].
- **Fusion Module:** One of three fusion strategies (see below) that combines the four embeddings and outputs a binary fake/real prediction.

Each model brings unique strengths (texture vs. global context vs. efficiency vs. specialized artifact detection) into the ensemble, making the system robust to a wide range of manipulations [12] [17].

Fusion Strategies

We explore three fusion methods to combine model outputs:

- **Early Fusion (Concatenation):** All four embeddings (2048 + 768 + 1792 + 1024 = 5632 dimensions) are concatenated into one vector. A small fully-connected network (with layers 5632→2048→512→128→1, ReLU activations, BatchNorm, and dropout) learns to fuse these features for final classification [2].

- **Late Fusion (Weighted Voting):** Each model independently produces a probability P_i of "fake". We compute the final score as a weighted sum:
$$P_{\text{final}} = \sum_{i=1}^4 w_i\,P_i,\quad \sum_i w_i=1,$$
where the weights w_i are learned during training to emphasize more reliable models [20].

- **Attention-Based Fusion:** An adaptive strategy that computes input-dependent weights. A small attention network processes each embedding e_i and outputs scores $f_{\text{att}}(e_i)$. We apply a softmax:
$$\alpha_i = \frac{\exp(f_{\text{att}}(e_i))}{\sum_{j=1}^4 \exp(f_{\text{att}}(e_j))},$$
then form the fused embedding $$e_{\text{fused}} = \sum_{i=1}^4 \alpha_i \, e_i.$$
This lets the system emphasize the most relevant model(s) for each image [21]. For example, ViT might get higher α on images with global inconsistencies, while ResNet50 could dominate for fine-grained texture artifacts [22] [23].

Dataset & Preprocessing

- **Source:** We use Roboflow's "Fake News Image Classifier" dataset (2,050 images) [3] [4].
- **Classes:** Binary labels (Real vs. Fake), roughly balanced (1023 real, 1027 fake) [4] [24].
- **Diversity:** Images come from various social media contexts, covering GAN-generated and diffusion-based fakes as well as Photoshop manipulations [25].
- **Preprocessing:** Images are resized to 224×224 pixels and normalized to ImageNet's mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225] [26].
- **Augmentation:** During training we apply random rotations (±15°), horizontal/vertical flips, scaling (0.8–1.2×), color jitter (brightness/contrast/saturation), occasional Gaussian blur, and noise injection to improve robustness [27].
- **Splits:** We use a stratified split: 87.5% for training (1795 images), 8.5% for validation (174 images), and 4% for testing (81 images) [28]. This maintains the real/fake ratio in each subset.

🏋 Training Strategy

- **Two-Stage Protocol:**
- *Stage 1 (Warm-up, epochs 1–10):* Freeze all pretrained model weights (ResNet50, ViT, EfficientNet) and train only the Custom CNN and fusion layers [29]. This preserves ImageNet features while initializing the fusion.
- *Stage 2 (Fine-Tuning, epochs 11–50):* Unfreeze all networks and continue end-to-end training with a lower learning rate [30]. Early stopping (patience 7) is applied on validation F1 to prevent overfitting.
- **Optimization:** We use the Adam optimizer with carefully tuned learning rates [31]. In Stage 1, we use a higher LR (1e-3) for the new custom and fusion parameters, while pretrained models remain frozen. In Stage 2, we set LR≈1e-5 for pretrained backbones and 1e-4 for new layers, with a stepwise decay (reduce by factor 0.1 every 15 epochs) [32].

- **Regularization:** Dropout (rates 0.3, 0.2, 0.1) and BatchNorm are used in the fusion network to prevent overfitting [33]. We also fix random seeds and use deterministic PyTorch operations for reproducibility [34]. Intermediate embeddings are cached (HDF5) after Stage 1 to speed up experimentation [29] [34].

Evaluation & Results

- **Metrics:** We evaluate using accuracy, precision, recall, F1-score, and AUC-ROC [35]. Five-fold cross-validation on the training set ensures robust hyperparameter tuning [36].
- **Individual Models:** Table II (below) shows test performance of each standalone model. The Vision Transformer achieved the highest single-model performance (F1≈0.9685, AUC≈0.9912) [37]. ResNet50 and EfficientNet-B4 also perform well (F1≈0.9388, 0.9530) via transfer learning, while our custom CNN (trained from scratch) achieves F1≈0.9265 [37].
- **Fusion Strategies:** With identical training budgets, all ensemble methods surpass individual models. In our experiments, early fusion reached F1≈0.9878 (AUC≈0.9967), late fusion F1≈0.9756 (AUC≈0.9923), and attention-based fusion F1≈0.9902 (AUC≈0.9981) [6]. The dynamic attention fusion notably outperforms the best single model (ΔF1 ≈ +2.17%) [6].
- **Comparative Gain:** Overall, the attention-fusion ensemble improves F1 by ~4.5 percentage points over baseline methods reported in literature [5], demonstrating the value of multi-model fusion. The final model runs at ~42 ms/image on a GPU [7], suitable for real-time deployment.

📊 Model Comparison Table

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
ResNet50	93.83%	92.68%	95.12%	93.88%	98.21%
ViT-B/16	97.04%	96.15%	97.56%	96.85%	99.12%
EfficientNet-B4	95.06%	94.29%	96.34%	95.30%	98.67%
Custom CNN	92.59%	91.43%	93.90%	92.65%	97.56%

Table: Test-set performance of each model (excerpt from Table II) [38].

🔍 Technical Details

- **Loss Function:** We use binary cross-entropy (BCE) loss for training the fake/real classifier [39].
- **Optimizer:** The Adam optimizer is used (β1=0.9, β2=0.999, weight decay=1e-5) [40].
- **Learning Rate Schedule:** A multi-step LR schedule reduces the rate by 0.1 every 15 epochs [32]. Pretrained backbones are trained with a smaller LR (1e-5), while new layers use a larger LR (1e-3 warm-up, 1e-4 fine-tune) [41] [32].
- **Regularization:** The fusion network uses dropout (0.3, 0.2, 0.1 in successive layers) and BatchNorm [33]. We also apply early stopping on validation F1 [42] to avoid overfitting.
- **Implementation:** Models are implemented in PyTorch. We fix all random seeds and enable deterministic operations for reproducibility. Embeddings from Stage 1 are cached to speed up fusion experiments [34].

Key Contributions

- **Unified Multi-Model Framework:** Combines four diverse image models (ResNet50, ViT, EfficientNet-B4, custom CNN) for robust fake-image classification [12] [17].
- **Novel Fusion Techniques:** Introduces three fusion methods (especially an attention-based fusion) to integrate model outputs; attention fusion yields the highest accuracy (F1≈0.9902, AUC≈0.9981) [6].
- **Advanced Training Protocol:** A two-stage transfer-learning scheme that preserves pretrained knowledge while adapting to the fake-image task [29].
- **State-of-the-Art Results:** Outperforms existing methods by ~4.5% in F1 on the target dataset [5]. Achieves high inference speed (~42 ms/image) with clear explainability (e.g., model attention weights) [7] [23].
- **Extensible & Reproducible:** Embedding-based design allows easy addition of new models [43]. All code and trained models are structured for reproducibility and reuse.

Getting Started

1. **Clone the Repository:**

   ```
   git clone <your-repo-url>
   cd Fake-News-Image-Detection
   ```

2. **Install Dependencies:**

   ```
   pip install -r requirements.txt
   ```

   Ensure you have Python 3.8+ and PyTorch (with GPU support for training). The `requirements.txt` lists core libraries (e.g., `torch`, `torchvision`, `timm`, `numpy`, etc.).
3. **Set Up Data:**
   Download the dataset (or place your images) as described (e.g., in `data/real/` and `data/fake/` folders). The code assumes a train/val/test split; update paths in `config.py` if needed.
4. **Environment:**
   (Optional) Create a virtual environment: `python -m venv venv && source venv/bin/activate` before installing requirements.

💻 Usage Instructions

- **Training Models:** Use the provided scripts or notebooks. For example, to train the attention-fusion ensemble:

  ```
  python train.py --fusion attention
  ```

  This runs the two-stage training protocol on the training set (flags `--fusion early/late/attention` select the fusion strategy). Check `config.py` or `--help` for hyperparameter options.
- **Running Inference:** After training, run inference on new images:

```
python predict.py --fusion attention --input_dir /path/to/images
```

Replace `attention` with `early` or `late` to use a different fusion method. The script outputs fake/real labels (and probabilities) for each image.

- **Switching Configurations:** You can enable/disable individual models or change learning rates in `config.py`. For advanced use, there are separate CLI flags for model components (e.g. `--train_resnet False`) and fusion parameters. Refer to docstrings or `--help` outputs for details.

Paper Access

The full paper is available here (Google Drive) and in the repository under `docs/FakeNewsImageDetection.pdf`. Please cite it when referencing this work.

Citation

If you use this work, please cite our paper:

```
@inproceedings{naeem2025fakenews,
  title={{Fake News Image Detection: A Comprehensive Study of Deep Learning Approaches}},
  author={Ahmad Naeem and Abdullah},
  booktitle={Proceedings of the IEEE International Conference on Fake News Detection (ICFN)},
  year={2025}
}
```

Future Work

- **Dataset Expansion:** Incorporate additional datasets, especially images from GAN/diffusion models, to further improve generalization [44].
- **Enhanced Fusion:** Design deeper fusion architectures (e.g. multi-head attention or residual-fusion hybrids) to capture complex manipulation patterns [44].
- **Multimodal Analysis:** Extend to multimodal fake news detection by fusing text (captions, posts) with image features.
- **Lightweight Models:** Explore model compression or distillation to deploy detection on edge/IoT devices.
- **Explainability:** Add visualization tools (e.g., Grad-CAM on CNNs, attention maps on ViT) to highlight which image regions or models contributed to decisions [23].

FAQ

Q: What types of fake images can be detected?
A: The model is trained on a diverse set including GAN-generated and diffusion-based images as well as conventional photoshopped fakes [4]. It learns subtle artifacts across these categories, so it can flag manipulations like face swaps, deepfakes, and unnatural edits.

Q: Which fusion strategy should I use?
A: In our experiments, **attention-based fusion** performed best overall [6]. However, early fusion is nearly as good and may be simpler to implement, while late fusion is more interpretable. You can set `--fusion` to `early`, `late`, or `attention` to compare results.

Q: Can I add other models (e.g. Xception, ResNet101) to the ensemble?
A: Yes. Because we fuse embeddings, you can integrate any new model by extracting its feature vector and concatenating it. You would update the fusion network's input dimension and retrain. The embedding-based design makes adding models straightforward [43].

Q: What hardware is required?
A: We trained on an NVIDIA T4 GPU (16GB VRAM) [45]. Training takes on the order of an hour per model. For inference, the ensemble runs quickly (~42 ms per image on GPU) [7]. You can run inference on CPU, but a GPU is recommended for reasonable speed.

Q: How do I reproduce the results?
A: Set the same random seed (see `config.py`) and follow the two-stage training procedure. Fixed seeds and deterministic flags are enabled in code [34]. Checkpoint models and training logs are saved by default, so you can load a checkpoint (`*_best.pth`) and run `predict.py` to verify outcomes.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [33] [34] [35] [36] [37] [38] [42] [43] [44] [45] DLP_Paper (22L-7500 - 21L-6239).pdf
file://file_0000000072b07246b5bd6cc7c2b783ed

[31] [32] [39] [40] [41] DLP_Project(22L_7500_&_21L_6239).ipynb
file://file-R3R6dLxXwZgTGsyKpoN8Ws