Deep Learning First Assignment: CNN Architectures on the MAMe Dataset

Antonio Lobo–Santos
Alejandro Guzmán–Requena

April 25, 2025

Abstract

Deep learning, especially Convolutional Neural Networks (CNNs), has transformed image classification tasks. Optimal performance depends significantly on selecting suitable architectures and hyperparameters. This report compares three CNN approaches—standard sequential CNN, an InceptionNet-inspired architecture, and VGG-19 transfer learning—for image classification on the MAMe dataset. We outline implementation strategies, hyperparameter optimization, training processes, and evaluate models based on accuracy and expected saliency map features. The InceptionNet-inspired model achieved the highest test accuracy (**79.5%**), followed by the standard CNN (**79.1%**), and finally, the VGG-19 transfer learning model (**71.0%**). Results demonstrate the advantage of sophisticated architectures and pretrained feature extraction.

1 Introduction

Image classification assigns labels to images based on their content and is fundamental in computer vision applications, from autonomous driving to medical diagnostics. CNNs have significantly advanced this domain, surpassing traditional methods. Model performance varies with dataset characteristics and available computational resources. Tested CNN strategies include:

- Custom–designed CNNs tailored for specific tasks (Standard CNN).
- Complex architectures employing multi–scale processing and residual connections (InceptionNet–inspired CNN).
- Transfer learning by reusing pretrained feature extractors from large datasets (VGG19).

This report evaluates these three strategies using the MAMe dataset [6], highlighting architectural choices, hyperparameter optimization, and training outcomes.

2 Methodology

2.1 Dataset

Experiments utilized the MAMe dataset [6], characterized by cultural heritage images with variations in viewpoints, lighting, and backgrounds.

2.2 Data Augmentation, Optimizers, and Learning Rate Schedulers

For all experiments, unless specifically disabled, input images were subjected to stochastic augmentations during training to improve generalization. These included:

- **Random Resized Crop:** cropping to 256×256 with scale sampled uniformly from $[0.8, 1.0]$.
- **Random Horizontal Flip:** with probability $p = 0.5$.
- **Random Rotation:** up to $\pm 15°$.
- **Color Jitter:** random adjustments to brightness, contrast, saturation, and hue.

After augmentation, all images were converted to tensor format and normalized using per–channel means $[0.485, 0.456, 0.406]$ and standard deviations $[0.229, 0.224, 0.225]$.

Optimizers. We evaluated four gradient-based optimizers.

- **SGD** with momentum and weight decay.
- **Adam** with and optional weight decay.
- **AdamW** with parameters matching Adam.
- **RMSprop** with default PyTorch settings.

Each optimizer's base learning rate was swept over {0.1, 0.01, 0.001}, and the best setting per model was chosen based on validation accuracy. Finally, for most of our models we decided to keep a learning rate of 0.003 with AdamW.

Learning Rate Schedulers. To further improve convergence, we experimented with three scheduler types (also in `training_helpers.py`):

- **StepLR:** decay factor $\gamma = 0.1$ every 30 epochs.
- **CosineAnnealingLR:** cosine decay over the full training length ($T_{\max}$ = total epochs).
- **ReduceLROnPlateau:** monitor validation accuracy, reduce LR by factor 0.1 after 10 epochs without improvement.

Empirically, CosineAnnealingLR yielded the most stable training curves and was adopted as the default scheduler unless otherwise specified.

2.3 Weight Initialization

All models were initialized following best practices to promote stable gradients and faster convergence:

- **Convolutional layers:** Kaiming normal (He) initialization with `mode="fan_out"` and `nonlinearity="relu"`; any bias terms set to zero.
- **BatchNorm layers:** weight parameters set to 1, bias parameters set to 0.
- **Linear layers:** weights drawn from a normal distribution $\mathcal{N}(0, 0.01^2)$; biases set to zero.

2.4 Code Availability

The full implementation of all models, training scripts, and configuration files is publicly available at: https://github.com/alobo01/cnn-inception-pytorch

3 Architectural Overview

This section presents the architectural underpinnings of three CNN models implemented and evaluated in this work: a standard convolutional network, a modern Inception-inspired multi-branch CNN, and a transfer learning model based on VGG-19.

3.0.1 Standard Convolutional Neural Network

The *StandardCNN* follows the canonical deep learning design inspired by AlexNet [5] and VGG [8], consisting of sequential 3×3 convolutions interleaved with batch normalization, ReLU activations, and max-pooling. Each block increases the number of filters (typically by a factor of 2), while halving the spatial resolution:

$$\text{Conv2d}(C_{i-1}, C_i, 3 \times 3) \rightarrow \text{BatchNorm2d} \rightarrow \text{ReLU} \rightarrow \text{MaxPool2d}(2 \times 2).$$

A global pooling layer (AdaptiveAvgPool2d) reduces the feature map to a fixed size, followed by a fully connected classifier with dropout. This design is simple but lacks the expressiveness and regularization mechanisms of more modern approaches.

3.0.2 Progressive Development of Inception-inspired Multi-Branch Network

Our architecture development began by drawing inspiration from the original Inception model [9], characterized by multi-branch modules that efficiently capture multi-scale features within each block. The initial implementation of the *InceptionBlock* featured four primary branches:

1. A single 1×1 convolutional layer,
2. A 1×1 convolution followed by a factorized 3×3 convolution, implemented as separate 1×3 and 3×1 convolutions,
3. A deeper branch comprising two consecutive factorized 3×3 convolutions,
4. A 3×3 max-pooling operation succeeded by a 1×1 projection convolution.

Subsequently, we incrementally introduced advanced modifications to enhance the flexibility, representational power, and training regularization of the network. These improvements included:

- Incorporation of **Squeeze-and-Excitation (SE)** blocks [3] to introduce channel-wise attention,
- Integration of **DropBlock** [1] for structured spatial dropout, enhancing robustness,
- Application of **Stochastic Depth** [4] to probabilistically bypass certain residual connections during training, thereby improving generalization,
- Inclusion of optional residual connections inspired by ResNet [2] to facilitate better gradient flow.

This gradual evolution ultimately culminated in a more sophisticated architecture closely aligned with the design principles of InceptionV3 [10]. The resulting network effectively balances complexity and computational efficiency through aggressive bottlenecking, separable convolutions, and carefully managed regularization, significantly enhancing performance without a proportional increase in parameter count.

3.0.3 Transfer Learning from VGG-19

Transfer learning is a strategy in which a model pretrained on a large source dataset is adapted to a new, typically smaller target task. By reusing the convolutional filters learned on ImageNet, the network can leverage low-level edges and textures in early layers and more abstract patterns in deeper layers, without training from scratch. This yields faster convergence, lower data requirements, and often better generalization, especially when the target dataset is limited in size.

In our third model, we initialize the convolutional backbone with VGG-19 weights pretrained on ImageNet [7,8]. VGG-19 was chosen because its uniform, deep architecture strikes a balance between expressive capacity and computational tractability, and its pretrained weights are well-validated across a wide range of vision tasks. To preserve generic feature extraction, we freeze the first several convolutional blocks and optionally fine-tune the last k blocks, allowing the model to specialize higher-level filters to our dataset:

$$\underbrace{[\mathrm{Conv}_1 \ \ldots \ \mathrm{Conv}_{b-k}]}_{\text{frozen}} \quad \underbrace{[\mathrm{Conv}_{b-k+1} \ \ldots \ \mathrm{Conv}_b]}_{\text{fine-tuned}}$$

We replace the original classifier head with a task-specific multilayer perceptron:

$$\text{Linear} \to \text{ReLU} \to \text{Dropout} \to \cdots \to \text{Linear}(\cdot, \texttt{num\_classes}).$$

This design significantly reduces training time and mitigates overfitting on our small dataset by reusing robust, pretrained representations while still allowing adaptation to domain-specific features. Empirical studies confirm that VGG-based transfer learning yields strong baselines and competitive performance across diverse image classification challenges.

3.0.4 Comparative Summary

Aspect	Standard CNN	Inception-style	VGG Transfer
Depth	3–6 conv blocks	4-8 Inception blocks	19 layers (fixed) + head
Parameter Cost	Moderate	Efficient via bottlenecks	High (most frozen)
Regularization	Dropout	SE, DropBlock, Stochastic Depth	Dropout on head
Training Flexibility	Full end-to-end	Modular tuning	Head tuning only
Best Use Case	Simple tasks	From-scratch training	Low-data regimes

Table 1: Comparison of implemented CNN architectures.

4 Training Methodology

This section details the training pipeline, including model construction, optimization strategy, learning-rate scheduling, early stopping, and hyperparameter search.

4.1 Model, Optimizer, and Loss

Our training pipeline instantiates the chosen architecture via a unified YAML-driven factory interface using PyTorch. We employ optimizers such as stochastic gradient descent (SGD), Adam, AdamW, and RMSProp. All training runs use automatic mixed precision (AMP) to halve

memory usage and accelerate throughput. We supervise with the categorical cross-entropy loss, which coincides with the multinomial logistic loss when using softmax.

4.2 Learning-Rate Scheduling

When enabled, we adjust the learning rate according to one of three policies:

- Cosine annealing with warm restarts as in SGDR,
- Step decay (drop by a factor γ every k epochs),
- Reduction on plateau, which lowers the rate when validation accuracy stalls.

4.3 Early Stopping and Checkpointing

To prevent overfitting, we monitor validation accuracy and halt training after a patience of p epochs with no improvement ("early stopping"). The best model state (by validation accuracy) is checkpointed via `torch.save()` and restored for final evaluation.

4.4 Hyperparameter Search

For automated tuning, we implement a recursive, coordinate-wise search:

1. Map wildcard paths in the YAML config to concrete parameter locations so that any configuration parameter can be tuned.
2. For each numeric parameter, given a range, the extremes and the middle point (rounded for integers parameters) are evaluated. Then the range is narrowed around the best value according to validation accuracy in a relatively small number of epochs $(30 - 50)$.
3. For boolean or categorical parameters, we perform one-shot evaluation.

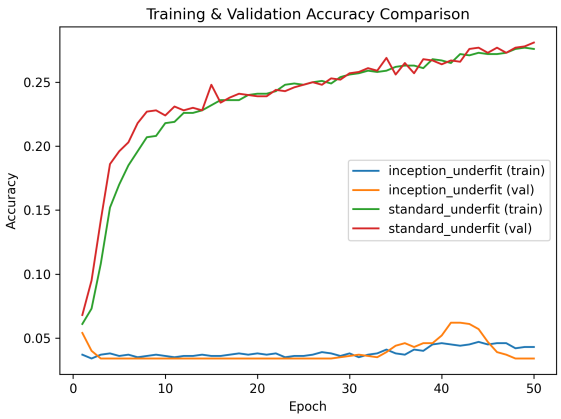
This strategy reduces the computational cost of classical grid Search, which has been shown to outperform exhaustive grids in high-dimensional spaces.

5 Extreme Configurations for Underfitting and Overfitting

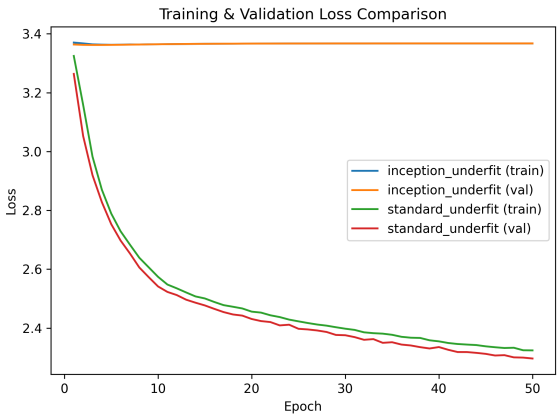
To systematically explore the behavior of extreme generalization failures, we designed deliberately exaggerated configurations using two architectures: `StandardCNN` and `InceptionNet`. These settings are crafted to clearly illustrate the characteristics of both underfitting and overfitting, as shown in Figure 1.

In the upper row of Figure 1, we observe the underfitting scenarios. Both architectures struggle to learn meaningful patterns, with validation accuracy stagnating around 0.05 for `StandardCNN` and 0.27 for `InceptionNet`. Despite the latter's higher representational capacity, neither model achieves sufficient training accuracy, and their loss curves plateau early, suggesting ineffective learning. `StandardCNN`, with its minimal depth and excessive regularization, fails to extract any useful features.

In contrast, the lower row illustrates overfitting behavior. Training accuracy approaches 100% and training loss approaches zero, while validation accuracy plateaus at suboptimal levels (approximately 0.55 for `StandardCNN` and 0.70 for `InceptionNet`). Notably, validation loss initially improves but quickly deteriorates, indicating that the models are memorizing the training data rather than generalizing.



(a) Underfitting Accuracy



(b) Underfitting Loss

(c) Overfitting Accuracy

(d) Overfitting Loss

Figure 1: Training and validation accuracy and loss curves for intentionally extreme underfitting and overfitting configurations.

5.1 Model Configuration

Table 2 summarizes the architectural block/stage patterns used in each model.

Model	**Block / Stage pattern***
StandardCNN Underfit	Single C_4 conv block with 4 filters and batch-norm.
InceptionNet Underfit	Minimal stem ($C_{\text{init}} = 8$), then 3 lightweight Inception modules (each with $b_0, b_1, b_2 \leq 8$ channels), down-sampling at stages 1 and 2; no SE or residual.
StandardCNN Overfit	Four repeated conv blocks: Conv (128 filters) $\rightarrow 2 \times 2$ max-pool (stride 2); no batch-norm or global pooling.
InceptionNet Overfit	Stem with 64 channels; two high-capacity Inception stages (no BN, SE, or residual): Stage 0: $\{b_0 = 2048 \mid b_1 = [256, 1024] \mid b_2 = [256, 1024]$, pool_proj=1024$\}$ Stage 1: $\{b_0 = 8192 \mid b_1 = [256, 2048] \mid b_2 = [256, 2048]$, pool_proj=2048$\}$

Table 2: Condensed block/stage configurations for each model. C_k denotes a $k \times k$ conv layer; down-sampling is indicated by $\downarrow$ (not shown explicitly here).

Table 3 presents the key training hyper-parameters, classifier designs, and (where applicable) weight decay for each model.

Model	**Trainable**	**Classifier**	lr	w_d	**Epochs**
StandardCNN Underfit	[1024]	1e-3	1e-4	50	
InceptionNet Underfit	[1024]	1e-3	1.0	50	
StandardCNN Overfit	[65 536]	1e-3	0	50	
InceptionNet Overfit	[32 768, 4 096]	1e-3	0	300	

Table 3: Essential hyper-parameters and classifier configurations.

6 Results

This section reports the experimental setup, compares model performances, and analyzes learning dynamics to assess the effectiveness of different backbone configurations on the MAMe dataset.

6.1 Model Configuration

Table 4 summarizes the backbone architectures used across models, detailing their block and stage configurations.

Model	Block / Stage pattern*
Standard	C64-P → C128-P → C256-P
Inception	$S_1\downarrow$: 2×{64 | 48-64 | 64-96} → $S_2\downarrow$: 2×{128 | 96-128} → S_3: 2×{192 | 96-160}
InceptionV3	As above, but with SE+DropPath and truncated S_3
Transfer Learn.	VGG19 backbone (frozen)

Table 4: Condensed block/stage configurations for each model's backbone. Ck = 3×3 conv with k filters, P = max-pool. S_i are Inception macro-stages; '↓' marks spatial down-sampling. *Within braces, columns denote $b0$ | ($b1\_1$-$b1\_2$) | ($b2\_1$-$b2\_2$) channels. †Only the 6.4 M parameters of the classifier are trainable.

Table 5 presents the key training hyper-parameters for each model, including the number of trainable parameters and classifier design.

Model	Params (M)	Trainable	Classifier	lr	w_d	Epochs
Standard	0.45	all	[256]	5e-3	1e-4	300
Inception	6.7	all	[2048,1024]	3e-3	2e-2	300
InceptionV3	4	all	[2048,1024]	3e-3	1e-2	300
Transfer Learn.	143†	0	[256,128]	5e-3	2e-2	300

Table 5: Essential hyper-parameters.

6.2 Evaluation Metrics

Model	Accuracy	Macro Precision	Macro Recall	Macro F1-score
Standard	0.7908	0.7662	0.8002	0.7753
Inception	0.7840	0.7665	0.7925	0.7696
InceptionV3	0.7947	0.7717	0.8030	0.7791
Transfer Learning	0.7102	0.6966	0.7229	0.6930

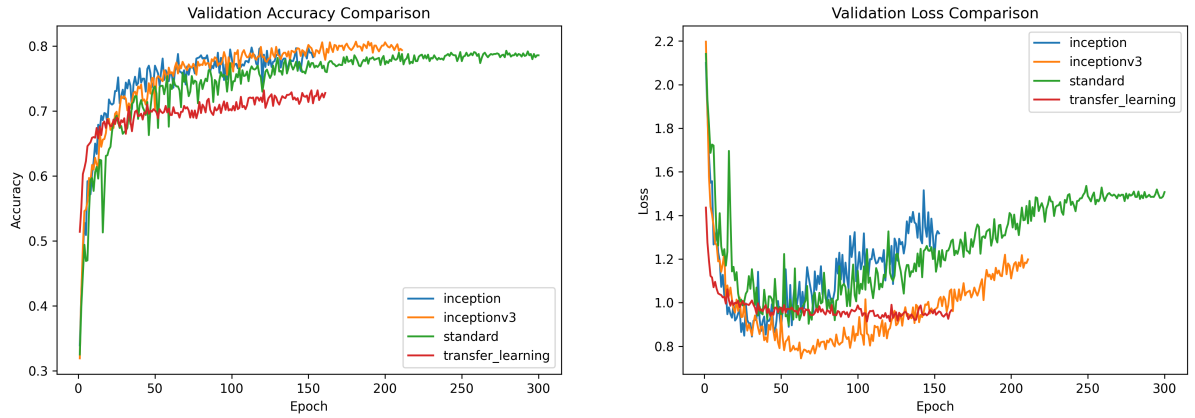
Table 6: Accuracy, precision, recall, and F1-score for each model.

Table 6 compares the three CNN architectures. **InceptionV3** achieves the best overall accuracy (79.5%) and macro F1-score (0.779), narrowly surpassing the Standard CNN (79.1%) and the original Inception variant (78.4%). Transfer learning lags behind at 71.0%, indicating that ImageNet pretraining transfers sub-optimally to the cultural-heritage imagery in the MAMe dataset.

6.3 Class-level Insights

• **Consistently easy classes.** *Albumen photograph* and both hand-coloured print categories record F1 > 0.95 across all models, thanks to distinctive tone and texture cues.
• **Metal vs. stone confusion.** *Bronze*, *Silver*, and particularly *Iron* display reduced precision due to confusion with other metallic or polished stone objects. InceptionV3 improves *Bronze* precision to 0.77, yet recall remains modest.
• **Wood derivatives.** Wood-based printmaking techniques (*Woodcut*, *Woodblock*, *Wood engraving*) achieve 0.79–0.90 recall, whereas raw *Wood* artifacts underperform, indicating the models focus more on fine-grained texture than global shape.
• **Minority textile class.** *Silk and metal thread* maintains recall around 0.72 but precision below 0.32, highlighting severe class imbalance and visual overlap with metallic embroidery.
• **Carved stone artifacts.** *Marble* recall rises with network depth (0.74 in InceptionV3) while precision falls ($<$0.45) due to frequent confusion with *Limestone* surfaces.

Note: A confusion matrix analysis was considered but omitted due to space constraints.
vspace-0.5em



(a) Validation accuracy

(b) Validation loss

Figure 2: Validation curves across epochs for all models.

Figure 2 illustrates validation accuracy and loss trajectories during training. Early stopping was triggered based on validation accuracy, while validation loss trends provided insights into generalization.

InceptionV3 (orange) converges most rapidly, reaching 80% validation accuracy within 120 epochs and maintaining the lowest loss throughout training. The Standard CNN (green) ultimately approaches 78% accuracy but with higher variance, suggesting weaker regularization. The original Inception model (blue) exhibits a similar trend but stagnates earlier. InceptionV3's regularization strategies (e.g., DropPath) slow convergence but promote better generalization, as indicated by smoother loss curves.

The transfer-learning baseline (red) achieves over 50% accuracy within a single epoch, indicating rapid initial adaptation. However, it plateaus around 73% and maintains a relatively high validation loss, confirming limited domain adaptation despite robustness against overfitting.

Overall, deeper models incorporating inception modules balance rapid early learning with sustained generalization. The gradual improvement of the Standard CNN suggests that, given sufficient training time and careful learning-rate decay, simpler backbones could eventually close

the gap—at the cost of longer runtimes.

Meanwhile, the transfer-learning setup saturates early, reflecting that feature reuse from ImageNet is insufficient to bridge the domain gap for fine-grained material classification in MAMe. This limitation likely arises from freezing all backbone parameters during training.

Future work could involve a posterior fine-tuning phase after classifier stabilization, allowing backbone parameters to adapt progressively to the new domain, which may lead to significant performance improvements.

7 Generative AI Acknowledge

We used ChatGPT to assist with commenting, organizing, documenting and typing the code, as well as improving our writing and LaTeX presentation. No code was directly copied; instead, numerous questions regarding the code were asked to better understand and refine our work.

References

[1] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *NeurIPS*. Curran Associates Inc., 2018.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[3] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[4] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *ECCV*, 2016.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[6] F. Parés, A. Arias-Duart, D. Garcia-Gasulla, G. Campo-Francés, N. Viladrich, J. Labarta, and E. Ayguadáe. The mame dataset: On the relevance of high resolution and variable shape image properties. *arXiv preprint arXiv:2007.13693*, 2020.

[7] O. Russakovsky, J. Deng, H. Su, and et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CVPR*, 2016.