

Тематичний напрям № 3
Штучний інтелект, машинне навчання та інтелектуальний аналіз даних

UDC 004.93

SOFTWARE ARCHITECTURE OF A SYSTEM FOR AI-GENERATED IMAGE DETECTION BASED ON THE XCEPTION NETWORK

Vladyslav Rogachko
National University «Odesa Polytechnic» Odesa, Ukraine

Abstract: The paper specifies a software architecture for a system that detects AI-generated images using a configured Xception network. Because the discriminative evidence in this problem consists of low-amplitude generation artifacts, the software architecture is subject to correctness requirements that are not typically encountered in conventional image-classification systems. Three design principles are formulated; based on them, the system is decomposed into four components with stable interfaces. The architectural decisions dictated by the problem are summarized in a table, and the anti-patterns prevented by the proposed architecture are analyzed.

Introduction. Generative models can synthesise images that are virtually indistinguishable from real photographs. This capability introduces information-security risks and creates a need for automated detection of AI-generated visual content. The problem is formulated as a binary classification task: given an input image, the system estimates the probability that the image belongs to the class of AI-generated images [1]. A distinguishing characteristic of this problem is that the discriminative evidence is represented by low-amplitude, generation-specific artifacts in the texture and frequency characteristics of the image rather than by its semantic content [2]. This characteristic has direct implications for the software architecture: any inconsistency in image representation or processing along the execution path may alter or eliminate the evidence on which the classification decision depends.

The Xception network [3] is a suitable baseline for the problem. Its depthwise separable convolutions factorise a standard convolution into depthwise and pointwise operations, model spatial and cross-channel correlations independently and reduce the number of trainable parameters while preserving representational capacity. Weights pretrained on a large-scale corpus transfer to the problem because the lower layers encode edges, textures and colour gradients common to most computer vision problems [4]. Configuring the network for detection means structural tuning of its classification stage together with transfer learning.

Network configuration, however, represents only one development activity within the complete system. Data preparation, input validation, dataset integrity verification, model serialisation, experiment and run management, configuration management, and delivery of the trained model to the end user constitute a substantial part of the supporting software. These components directly affect maintainability, reproducibility, testability, and the cost of subsequent modifications. Technical debt arising in this supporting code when it is treated as secondary to model development has been documented in the software-engineering literature on machine-learning systems [5]. In contrast, publications on AI-generated image detection typically describe the neural-network architecture and report experimental performance, while providing little or no specification of the software architecture responsible for data processing, model execution, experiment management, and model delivery.

Aim. The aim of this paper is to specify a software architecture for a system that solves the AI-generated image detection problem with a configured Xception network: to state the design principles, to derive the decomposition into components and the interfaces between them, to identify the architectural

decisions that the problem itself dictates, and to name the anti-patterns that the architecture is intended to prevent.

Materials and methods. Three design principles drive the decomposition [6]. The first is separation of concerns by execution cost: normalising a large collection and training the network are measured in hours, recomputing a performance measure in seconds, so a component boundary is placed wherever re-execution would be expensive. The second is loose coupling through persistent interfaces: components exchange files and no component imports another, which yields a batch-sequential style over a shared repository and allows any component to be re-executed, reimplemented or relocated without invalidating the outputs of its predecessors. The third is a single source of truth: every rule obeyed by more than one component — the image preprocessing above all — has exactly one definition. For AI-generated image detection the third principle is a correctness requirement and not merely a maintainability one.

Component decomposition. The principles yield four components and the artefacts they exchange (Fig. 1). The implementation platform is Python: the network is configured and trained with a deep learning framework [7] through its high-level API [8], the performance measures are computed with a general machine learning library [9], and the inference service is built on a browser-oriented framework [10].

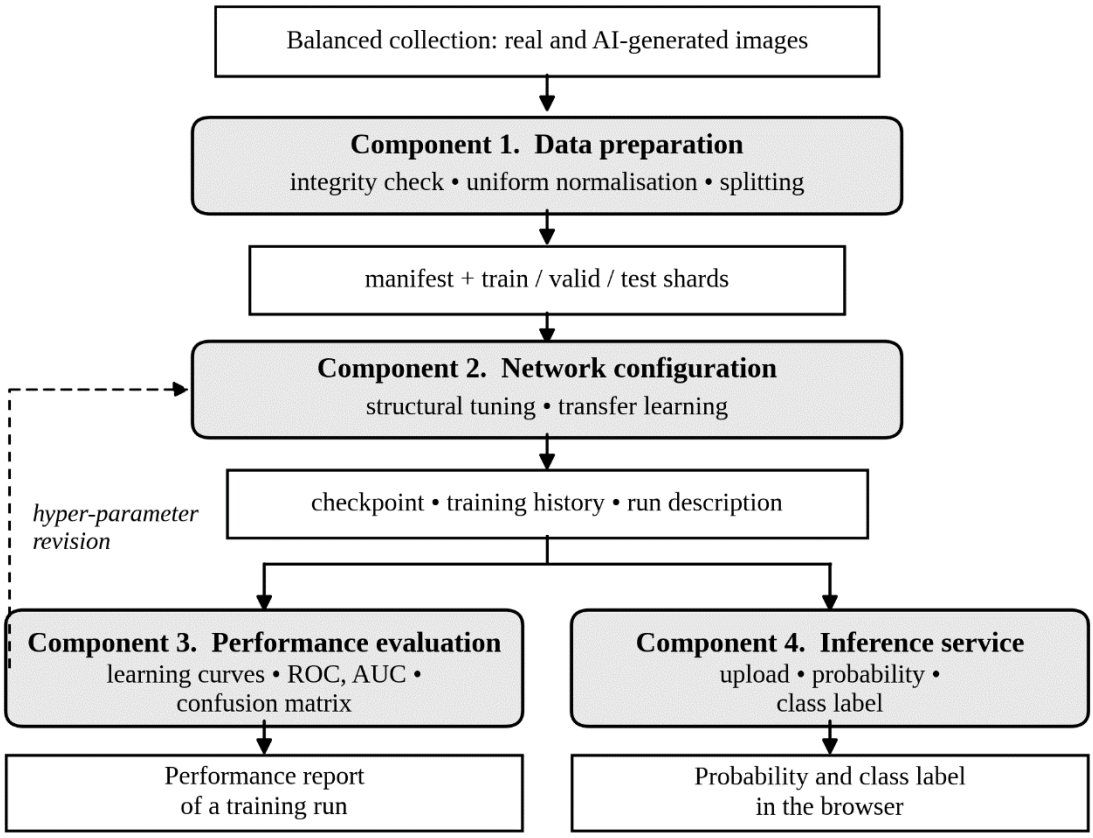


Fig. 1. Component decomposition of the software and the artefacts exchanged between the components.

Data preparation component. The component enumerates the balanced collection of real and AI-generated images, assigns to every image its class label, and verifies file integrity: each file is opened, its internal structure is parsed, and whatever cannot be decoded is logged and excluded. A collection assembled from several corpora invariably contains truncated files, and performing this verification once is cheaper than a decoder exception that aborts a training run after several hours.

Normalisation, ordinarily a routine activity, is the critical one in this problem. Unifying the geometry to the input resolution of the configured network and scaling the pixel values to the interval [0, 1] alter the very statistics on which the decision rests, so the operations must be identical for both classes: one resampling filter, one target resolution, one container format and one quality setting for the whole collection. If the two classes arrive in different formats or at different resolutions and are normalised differently, the classifier

learns the handling instead of the origin. This is a data leakage defect in the architectural sense: the class label reaches the network through a channel other than the image content, and the reported performance figures become meaningless. The decisions this consideration fixes are collected in Table 1.

The normalised collection is shuffled once and split into training, validation and test subsets in the proportion 70/20/10, and the prepared images are serialised into a small number of large sequential containers rather than kept as separate files: at a scale of hundreds of thousands of images, sequential access is what determines whether an epoch lasts minutes or hours. The split itself is recorded in a separate manifest that lists the records and the subset each belongs to, rather than being frozen into the containers; the manifest is the artefact that documents the absence of leakage between subsets and permits the proportions to be revised without regenerating the collection.

Network configuration component. The input pipeline is composed as a chain of parallel reading, caching, shuffling, parallel decoding, batching and prefetching. The decoding routine restores the tensor, sets its shape explicitly, since a deserialised tensor carries none, and applies the normalisation prescribed by the backbone.

Configuration proper consists of structural tuning and transfer learning. Structural tuning replaces the classification stage of the base architecture: the pooled feature maps are converted into a one-dimensional vector by a Flatten layer, a Dropout layer with a rate of 0.5 regularises the representation [11], an intermediate Dense layer with ReLU activation learns nonlinear combinations of the discriminative features, and a single-neuron Dense output layer with Sigmoid activation estimates the probability of the AI-generated class. Binary cross-entropy serves as the loss function, and an adaptive optimiser [12] is used for weight optimisation.

Training is organised in two stages within one executable. During the frozen-weights stage the base layers are fixed and only the new classification head is trained, so that a randomly initialised head does not disturb the pretrained filters with large initial gradients — a material risk in this problem, since those filters are what responds to the discriminating texture. During the fine-tuning stage all layers are unfrozen and jointly optimised until early stopping halts the run. Every run is written into a directory of its own holding the checkpoint, the per-stage training history and a description of the configuration; this journal is the traceability mechanism of the system, and without it a series of experiments cannot be reconstructed afterwards. The hyper-parameters are held in a configuration file read by this component and by the next one, rather than as literals in the source.

Performance evaluation component. Separated from training because it is cheap and re-executed often, the component reloads the checkpoint and the stored histories, joins the two stages into continuous learning curves and marks the epoch at which the base layers were unfrozen, the effect of that transition being easy to misread when the curves are presented apart. Inference over the test subset is performed once and the predicted probabilities are persisted before any threshold is applied. The evaluation is thereby made threshold-independent: the ROC curve, the AUC, the operating point and the confusion matrix are recomputed in seconds, whereas repeating inference over tens of thousands of images costs hours. Selection of the operating point requires separate attention, because the costs of the two error types are not symmetric - classifying an authentic photograph as AI-generated is the more damaging outcome - and a single accuracy figure conceals that asymmetry.

Inference service component. The component loads the artefact produced by training and returns the verdict to the user; delivery through a browser removes any installation requirement on the client side. The model is loaded once per session rather than once per request, and the predicted probability is reported alongside the class label, a value near the operating point being a different statement from a confident one. One rule governs this component above all others: the preprocessing it applies must be the same function that was used during training, imported from the shared module, and not an independent reimplementation.

Results and discussion. Table 1 collects the architectural decisions that the AI-generated image detection problem itself dictates, together with the rationale for each; everything outside that list is ordinary engineering and may be reused from any other image processing system. The architecture repays its cost where the components are genuinely expensive to execute; for a collection of a few thousand images that

trains in minutes, four executables and an intermediate format are an unnecessary ceremony. Its value grows with the execution time of the components and with the number of configurations to be compared.

Table 1 Architectural decisions dictated by the detection problem.

Element	Decision	Rationale
Geometry and resampling	one target resolution and one filter for both classes	rescaling alters the statistics that carry the decision
Re-encoding	one container format and one quality setting	otherwise compression traces become the discriminative feature
Classification stage	Flatten, Dropout, Dense with ReLU, single-neuron Dense with Sigmoid	a compact head preserves the pretrained features
Loss function	binary cross-entropy	the output is interpreted as a class probability
Persisted evaluation output	the predicted probabilities, before any threshold	the operating point is selected after inference
Preprocessing	one function shared by training and inference	a divergence is undetectable by offline measurement

Of the anti-patterns the architecture prevents, one is specific to this problem and deserves emphasis. When the preprocessing is implemented once in the training pipeline and a second time in the inference service, the two implementations diverge sooner or later - a different scaling of the pixel values or a different resampling filter is sufficient. For an ordinary classifier the consequence is a modest loss of accuracy; in AI-generated image detection it is worse, because the decision rests on regularities of the same order as the divergence itself, so the deployed system may be reading the artefacts of its own serving path instead of those of the generator. Every offline measurement remains intact, since the offline path never executes the code of the inference service. This training-serving skew is a recognised form of technical debt in machine learning systems [5], and the architectural remedy is the one stated above: a single preprocessing function in a shared module, imported by both paths.

Conclusions. A software architecture has been specified for a system that solves the AI-generated image detection problem by means of a configured Xception network. Three design principles - separation of concerns by execution cost, loose coupling through persistent interfaces, and a single source of truth for every shared rule - yield a decomposition into four components whose boundaries fall where re-execution would be expensive. The distinguishing property of the problem is that the evidence carrying the decision is faint enough to be destroyed by the software itself, which converts the uniform normalisation of the collection and the sharing of a single preprocessing function between training and inference from questions of style into correctness requirements. Further work is seen in extending the architecture to an ensemble of configured backbones and to periodic retraining as new generative models appear.

References

1. J. J. Bird and A. Lotfi. CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. IEEE Access, vol. 12, pages 15642–15650, 2024.

2. F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi. Do GANs Leave Artificial Fingerprints? In IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pages 506–511, 2019.

3. F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1251–1258, 2017.

4. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How Transferable Are Features in Deep Neural Networks? In Advances in Neural Information Processing Systems, volume 27, pages 3320–3328, 2014.

5. D. Sculley, G. Holt, D. Golovin, et al. Hidden Technical Debt in Machine Learning Systems. In Advances in Neural Information Processing Systems, volume 28, pages 2503–2511, 2015.

6. L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. 4th edition. Addison-Wesley, Boston, 2021, 464 pages.

7. M. Abadi, P. Barham, J. Chen, et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 265–283, 2016.

8. Keras API documentation [Electronic resource]. URL: https://keras.io/.

9. F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, vol. 12, pages 2825–2830, 2011.

10. Streamlit documentation [Electronic resource]. URL: https://docs.streamlit.io/.

11. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, vol. 15, pages 1929–1958, 2014.

12. T. Dozat. Incorporating Nesterov Momentum into Adam. In International Conference on Learning Representations, Workshop Track, 2016.

УДК 004.932

QUATERNION CONVOLUTIONS IN LARGE-MASK IMAGE INPAINTING

Дмитро Колодочка, Марина Полякова

Національний університет «Одеська політехніка», м. Одеса, Україна

Introduction. Image inpainting is the process of restoring missing or corrupted image regions so that the result is visually plausible and semantically consistent with its surroundings. The problem splits into two broad regimes. In large-mask inpainting the unknown regions are wide and reconstruction depends on long-range propagation of structure and color. In the fine-grained missing-pixel regime many pixels are absent but the gaps between known pixels stay small. Large-mask inpainting is the harder and more practically relevant case: it is used in tasks like object removal, film rig removal, and region editing in generative systems, where large gaps require semantic reconstruction of background, geometry, and lighting rather than simple texture copying.

Recent quaternion inpainting work covers different missing-pattern regimes: Miao et al. [1] study a matrix-completion setting with evenly distributed missing pixels and small local gaps, while Wang et al. [2] explore hole-like masks closer to large-mask inpainting. Building on that direction, we study quaternion convolution (QConv) inside a LaMa-style architecture rather than a standard QConv GAN, and push the evaluation toward larger mask gaps. We chose LaMa-Fourier because its Fast Fourier Convolution (FFC) design [3] targets wide missing regions and global context, making it a good baseline for studying QConv beyond local completion.

QConv treats RGB information as a coupled entity rather than three independently mixed scalar channels. This may improve color consistency in broad smooth regions, but stronger channel coupling may reduce the flexibility needed for fine local texture. We therefore expect to measure a trade-off between color consistency and texture detail.

Aim. The aim of this work is to introduce selective QConv components into a LaMa-Fourier generator and to measure their effect on reconstruction quality, inference speed, and model size. Training stays fixed. Only the locations at which QConv replaces standard convolutions in the up-scaling, down-scaling, and local FFC paths are varied. We report FID, LPIPS, PSNR, SSIM, parameter count, throughput, and qualitative behavior on thin, medium, wide, and ultrawide masks, against a LaMa-Fourier baseline trained under the same data and optimization settings.

Materials and methods. The baseline generator follows LaMa's FFC design, which is strong for large-mask completion because global context is injected early through Fourier operators [3]. The LaMa-Fourier generator (Fig. 1) takes a three-channel masked image and a binary mask, uses a 7x7 convolution that lifts the input to 64 channels, downsamples through three 2D convolutions, processes the result with a stack of residual FFC blocks, upsamples through three transpose convolutions, and shrinks the image back to 3 channels with a 7x7 transpose convolution. Each FFC block splits features into local and global branches