Technical Challenge Font Classification and NSFW Content detection

Chebouti Boutheina

1 FONT CLASSIFICATION

Font classification is the first stage of our system, aiming to automatically recognize the font style used in each image. This task is challenging due to the dataset's diversity in scale (character-, word-, and image-level samples), variations in font weight and style, and the presence of noisy, low-contrast, or textured backgrounds. Accurate font classification is essential, as it directly impacts the performance of downstream OCR and text analysis stages.

To address this challenge, we progressively explored models of increasing complexity. We began with a baseline Convolutional Neural Network (CNN) trained from scratch, providing a solid reference point. We then leveraged transfer learning with ResNet50 to exploit richer pretrained visual features and then did a profound analysis on the results and model explainability where we explored on which fonts its making mistakes and why we also inspected the confusion matrix to see if the model is mistaken between what fonts, tsnee visualisation and also performed an analysis on character level using the char dataset and we inspected if the model is making mistakes due to the number of training data lastly we tried to combine the ocr on these images to extract the words and do a deep analysis to see on which words its mistaking , followed then by EfficientNet, chosen for its compound scaling strategy and superior accuracy efficiency balance. In later stages, we also considered vision transformers but due to the lack of time we wouldn't be able to get the results from it for fine-grained pattern recognition.

The following sections present each model's methodology, preprocessing strategies, training setup, performance results, and detailed error analyses, highlighting both strengths and limitations in handling the dataset's visual diversity

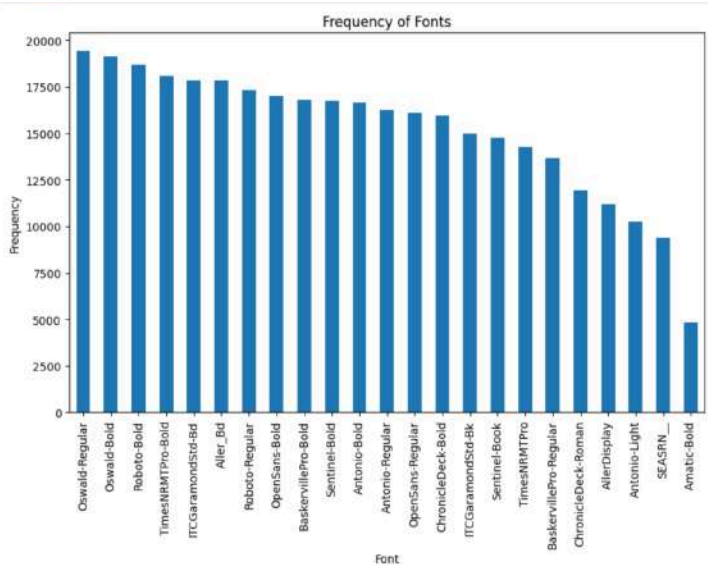
Convolutional Neural Networks (CNNs) are widely used in image recognition due to their ability to effectively detect patterns and features in images. This section details the implementation, hyperparameters, and performance of a CNN model trained to classify font images.

Data Inspection

Before model training, we conducted an initial inspection of the dataset to better understand its composition and potential biases.

Font Frequency Distribution:
The bar chart shows that the dataset is imbalanced, with certain fonts such as **Oswald Regular**, **Oswald-Bold**, and **Roboto-Bold** appearing significantly more frequently than others like **SEASRN__**

and **Amatic Bold**. This imbalance can influence model performance, as classes with fewer samples may be harder to learn and more prone to misclassification.



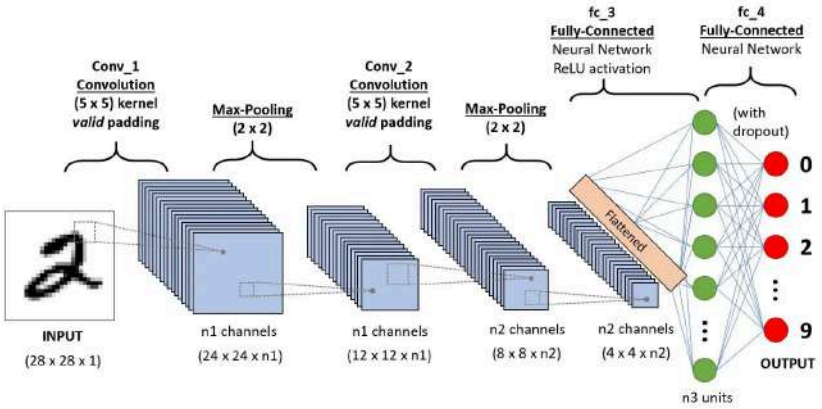
Sample Visualization:
The image grid presents representative samples from each font class. It highlights the high visual diversity in background textures, colors, and text content, which can make font classification more challenging. Some fonts have clean, high contrast examples, while others appear on noisy or complex backgrounds, potentially affecting recognition accuracy.

1.1 Baseline CNN Method

CNNs are particularly well-suited for image classification because they can automatically capture spatial patterns, ranging from simple edges to complex shapes.
A typical CNN includes:

- Convolutional layers for feature extraction

- Activation functions (e.g., ReLU) for non-linearity
- Pooling layers to reduce spatial dimensions
- Fully connected layers for classification
- Dropout to prevent overfitting



CNNs were chosen as the **baseline** in this project because they provide a proven, efficient starting point for image-based classification tasks, requiring minimal manual feature engineering while effectively leveraging the spatial structure of input data.

A.Data Preprocessing

Images were normalized by scaling pixel values by a factor of **1/255**. This ensures all inputs fall within the [0, 1] range, which helps the **network train more efficiently and improves numerical stability.**

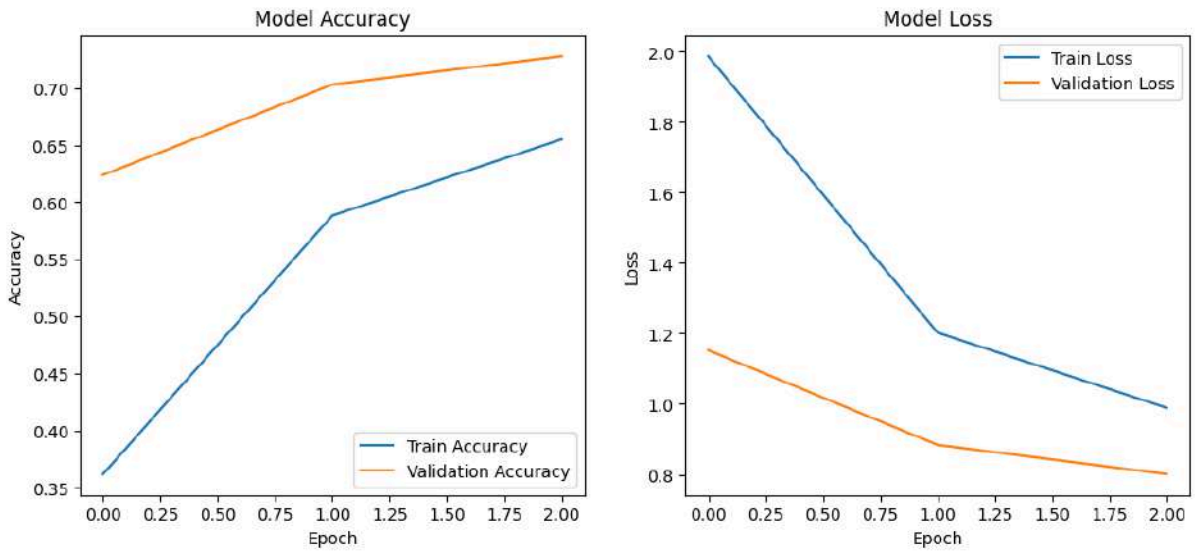
B. Model Architecture and Training

The model consists of three convolutional layers with 32, 64, and 128 filters (3×3 kernels, same padding). Each convolutional layer is followed by a 2×2 max pooling layer.
ReLU activations are applied throughout, and a **dropout rate of 0.5** is used before the final classification layer to prevent overfitting. The flattened features are passed into a dense layer of 256 units, followed by a softmax output layer for the 23 font classes.

Training was performed with the **Adam optimizer** (learning rate = 0.001), categorical cross-entropy loss, a batch size of 32, and 3 epochs.

C. Results

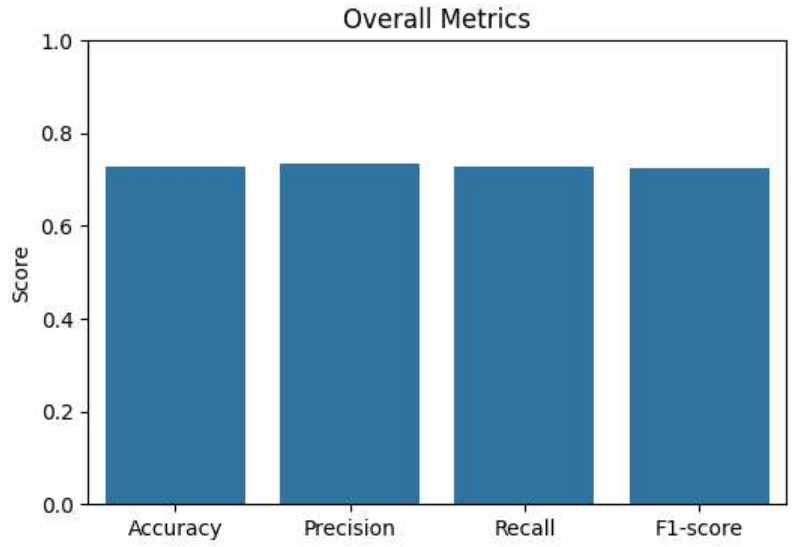
The CNN model was trained for three epochs with a batch size of 32. Training and validation loss steadily decreased, while validation accuracy improved from 62% to over 72% (training accuracy reached approximately 66%). This indicates stable learning and decent generalization for a baseline model without **overfitting.**



.

- **Training Accuracy:** 66.02%
- **Validation Accuracy:** 72.31%
- **Training Loss:** 0.99
- **Validation Loss:** 0.80

The CNN baseline achieved balanced performance across all metrics, with accuracy of ~73%, precision of 74%, recall of 73%, and F1-score of 73%. This shows that the model was able to correctly identify most font classes while maintaining consistent precision and recall.



D. Result Analyses and model interpretation

Best Detected Fonts

These fonts achieve the highest F1-scores, meaning the model almost always identifies them correctly:

1. SEASRN__ — Precision: 91%, Recall: 88%, F1-score: 0.90

When looking at the pictures of **SEASRN__**, we can see clear, angular letter shapes with wide spacing and bright colors standing out from the background. This strong contrast and distinctive style make the font very easy for the model to recognize.

2. AllerDisplay — Precision: 90%, Recall: 79%, F1-score: 0.84

The font is bold and clear so it can be distinguished easily from other fonts

Least Detected Fonts

These fonts have the lowest F1-scores, indicating frequent confusion with other fonts:

1. Aller_Bd — Precision: 55%, Recall: 56%, F1-score: 0.55

The font is very to other sans-serif fonts in the dataset (like OpenSans or Roboto) as we will see in the confusion matrix later. The variations between them are subtle so the CNN may confuse them which led to a drop in the performance .

2. OpenSans-Bold — Precision: 72%, Recall: 47%, F1-score: 0.56

Same issue: it shares many features with other common sans-serif fonts and some backgrounds are textured which explain the poor performance of the model .

The baseline CNN helped identify key challenges in the dataset, such as the difficulty in distinguishing visually similar fonts.
To address these limitations, we moved to **ResNet**, a deeper architecture pretrained on ImageNet, with the goal of leveraging its richer feature representations.

1.2 ResNet:

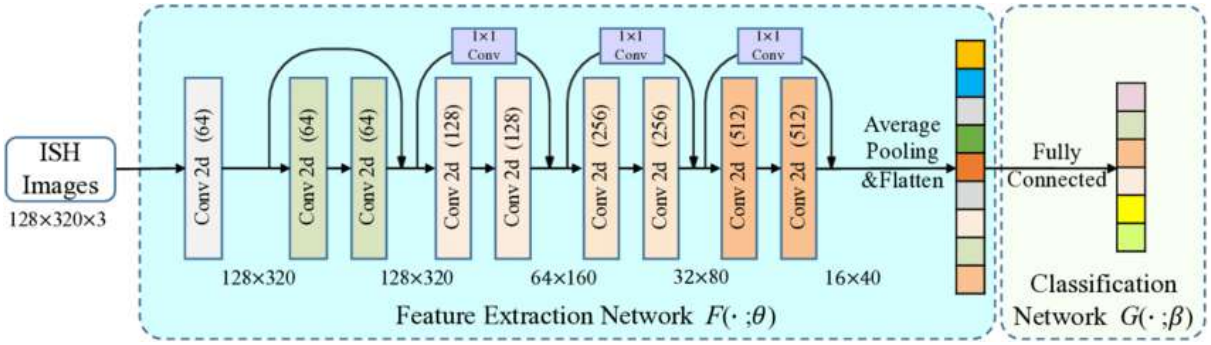
We chose **ResNet50** as it offers better accuracy than ResNet34 while being less computationally heavy than ResNet101 or ResNet152 [1] .

A.Data preprocessing

In addition to the previous processing we applied **horizontal flipping** and **brightness variation (±10%)** to the training images to improve generalization, while keeping validation and test images unchanged except for ResNet50 preprocessing.

B.Model architecture and training

Initial Attempt (Frozen Backbone)
Our first experiment used ResNet purely as a fixed feature extractor, keeping all backbone parameters frozen and training only the new classification head.
While this setup converged quickly, the performance was poor, reaching only **34% accuracy** on the validation set.



This indicates that the pretrained features alone were insufficient to capture the fine-grained differences between fonts in our dataset, and the model could not adapt enough to our specific task.

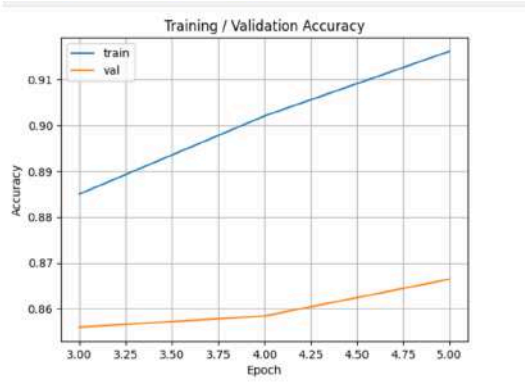
Refined Approach (Selective Fine-Tuning)
We then adopted **selective fine-tuning**:

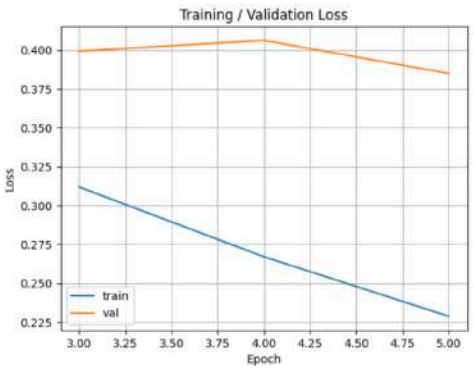
- Keeping **Batch Normalization layers frozen** for stability
- Unfreezing only the **last residual stage (conv5)** so the model could adapt high-level features to font-specific patterns
- Adding a **Dropout layer (0.3)** before the final Dense layer to reduce overfitting

For training we trained the model with batch **size 64 and 5 epochs .**

This approach allowed the model to retain the general visual knowledge from ImageNet while learning to separate similar fonts more effectively, resulting in a significant performance boost compared to both the frozen-ResNet and baseline CNN.

C.Results :





From epoch 3 to epoch 5, both training and validation accuracy improved steadily, with training accuracy rising from about 88.5% to 91.6% and validation accuracy from 85.6% to 86.6%. Training loss decreased consistently, and validation loss showed a slight drop by the final epoch. The small, stable gap between training and validation curves indicates that the model continued to learn effectively during fine-tuning without overfitting and generalises well (the training and validation accuracy/loss plots shown here are a zoomed view from epoch 3 to epoch 5, corresponding to the resumed fine-tuning phase of the model)

	Accuracy	**Loss**
Train	**91.61**	**0.23**
validation	**86.64**	**0.38**
test	**86.63**	**0.38**

D.Deep Analysis and Model Explainability

Best and Worst Detected Fonts by the ResNet50 Model

Best Detected Fonts

These fonts achieve the highest F1-scores, meaning the model almost always identifies them correctly:

SEASRN__ Precision: 98%, Recall: 97%, F1-score: 0.98

In the CNN baseline, **SEASRN__** was already one of the best-detected fonts, but ResNet50 pushed its performance from an F1-score of 0.90 to 0.98. The sharp, angular shapes, wide spacing, and high contrast against the background remain key features, but the deeper ResNet features capture these details even more which proves that the residuals help in capturing more details .

Oswald-Bold Precision: 96%, Recall: 97%, F1-score: 0.97

Oswald-Bold was not among the very top fonts in the CNN model but still performed well. With ResNet50, it became one of the best overall, improving to an F1-score of 0.97. Its tall, condensed style and thick strokes are now captured more accurately, even across varied backgrounds.

Least Detected Fonts

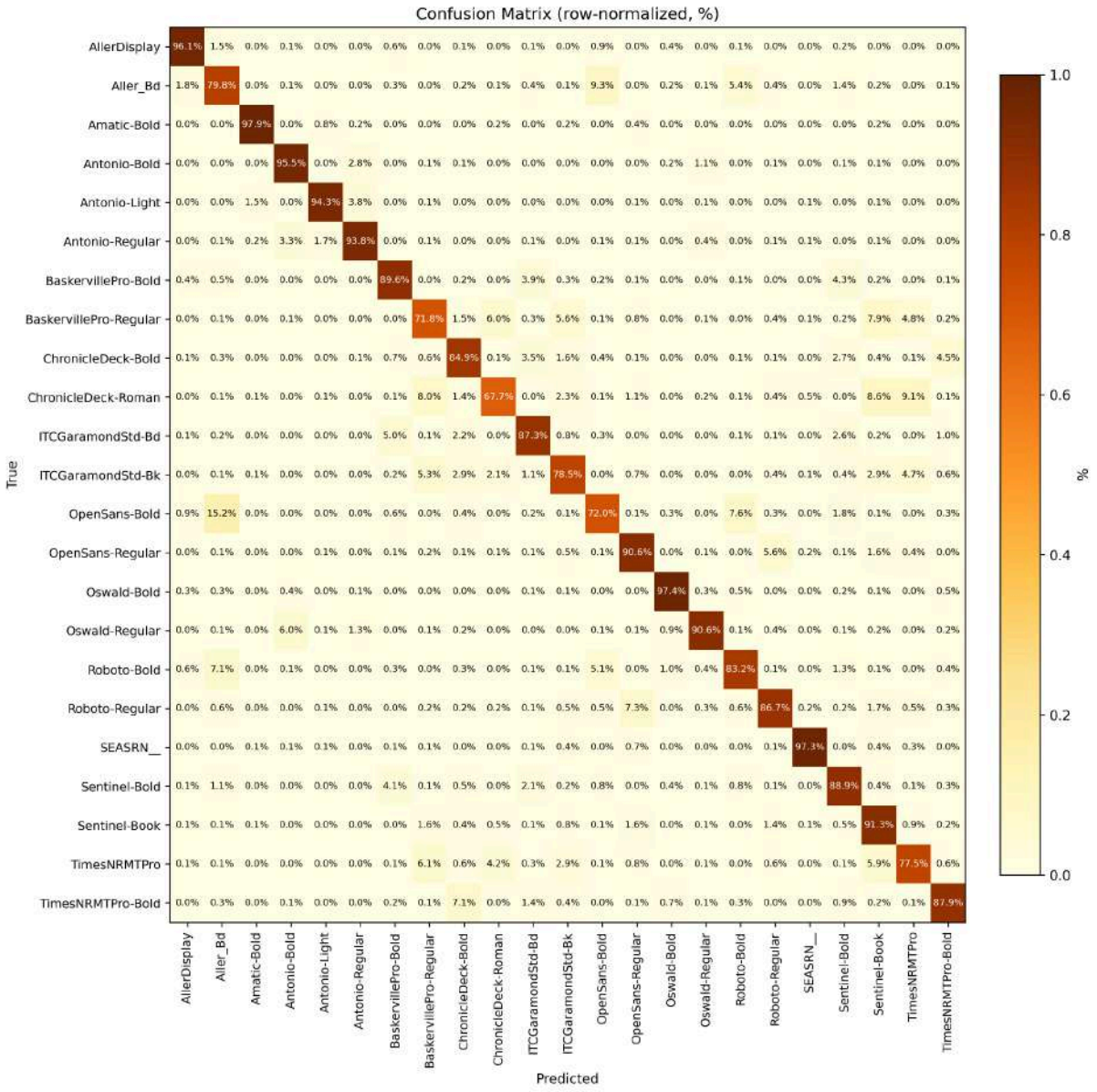
Even tho the F1 score is more important than the CNN baseline one but here are the fonts that are the least detected by our ResNet this indicates they are more often confused with other classes as we are going to show in the confusing matrix later on :

BaskervillePro-Regular Precision: 76%, Recall: 72%, F1-score: 0.74
This font did not appear among the worst in the CNN baseline, but ResNet50's errors shifted toward serif fonts with subtle design differences. **BaskervillePro Regular** shares proportions and fine details with other serif styles, making them harder to separate even for a deeper model.

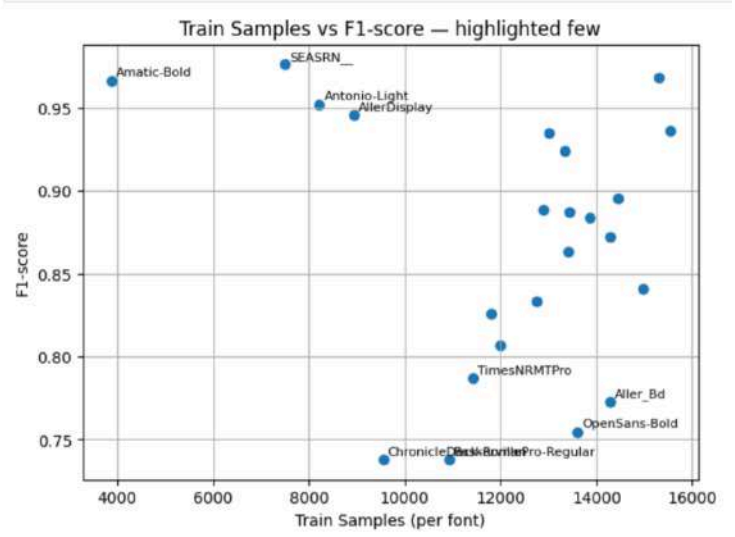
ChronicleDeck-Roman Precision: 81%, Recall: 68%, F1-score: 0.74
*(*Similar to *BaskervillePro Regular*, this serif font has small variations in curves and stroke endings that resemble other serif fonts in the dataset. While its performance is much better than the CNN's lowest scoring fonts, it remains one of the harder classes for ResNet50.

Analysis of Font Confusions in ResNet50 Predictions

The confusion matrix shows that some fonts are still difficult to tell apart. For example, **OpenSans-Bold** is often confused with **Aller_Bd** (15.2%), just like in the CNN results, because both are clean sans serif fonts with very similar shapes. **BaskervillePro-Regular** is frequently mistaken for **ChronicleDeck-Bold** (15.5%) and **ChronicleDeck-Roman** (15.6%), as these serif fonts share thin strokes and similar letter forms. The model also mixes up **Roboto-Bold** and **Roboto-Regular** (14.4% in both directions), which is expected since the only difference is the thickness , often hard to see in smaller or textured images. The same applies to **ChronicleDeck-Roman** and **ChronicleDeck-Bold** (14.6%), where the bold/regular distinction remains subtle.

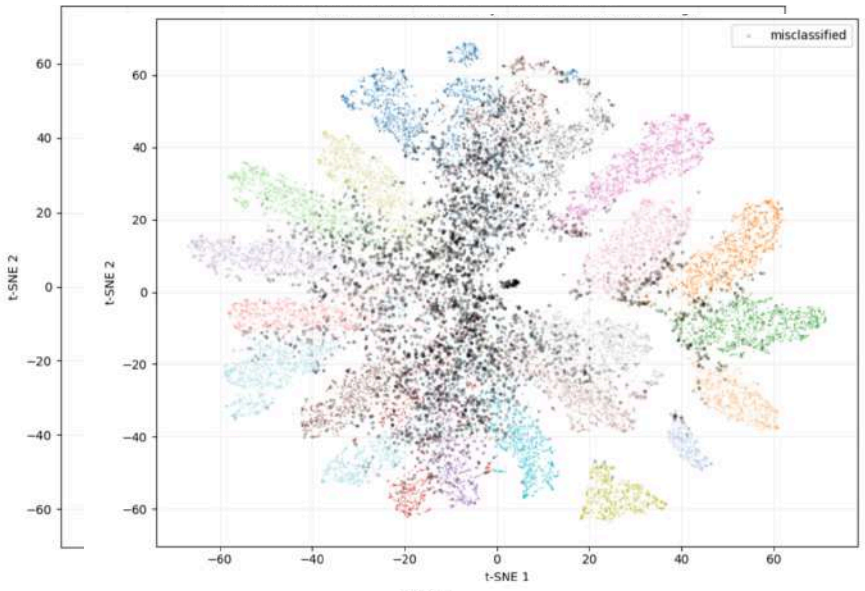

Confusion Matrix (row-normalized, %)
True
Predicted
%
AllerDisplay
Aller_Bd
Amatic-Bold
Antonio-Bold
Antonio-Light
Antonio-Regular
BaskervillePro-Bold
BaskervillePro-Regular
ChronicleDeck-Bold
ChronicleDeck-Roman
ITCGaramondStd-Bd
ITCGaramondStd-Bk
OpenSans-Bold
OpenSans-Regular
Oswald-Bold
Oswald-Regular
Roboto-Bold
Roboto-Regular
SEASRN__
Sentinel-Bold
Sentinel-Book
TimesNRMTPro
TimesNRMTPro-Bold
1.0
0.8
0.6
0.4
0.2
0.0

Per class analysis :



This plot shows that having more training images for a font does not always mean better performance. For example, *Amatic Bold* scores very high with few samples, while *Aller_Bd* and *OpenSans-Bold* still perform poorly despite having many samples.
This suggests that the main difficulty comes from visual similarity between fonts and from challenging visual characteristics such as blurred letters, distracting backgrounds, or low contrast which make it harder for the model to correctly distinguish them.

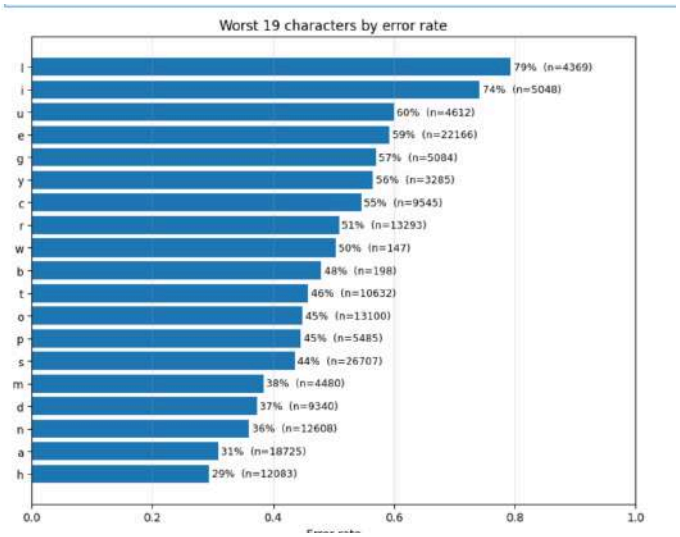
Feature space visualisation with tsne



The t-SNE visualizations show that some fonts, like **Amatic Bold** and **SEASRN__**, form very compact, isolated clusters, meaning the model recognizes them with ease. In contrast, similar-looking fonts such as **OpenSans Bold** and **Aller_Bd**, or **Roboto Bold** and **Roboto Regular**, have overlapping clusters, which explains their high confusion rates observed earlier. Most misclassifications appear along the borders between clusters, where differences between fonts are subtle and the decision boundaries are harder to define. However, some mistakes also occur inside otherwise well-separated clusters, often due to image quality issues such as blur, background noise , or poor contrast. These patterns align with the CNN results, confirming that the main challenge lies in the visual similarity between certain fonts rather than limitations of the model architecture itself.

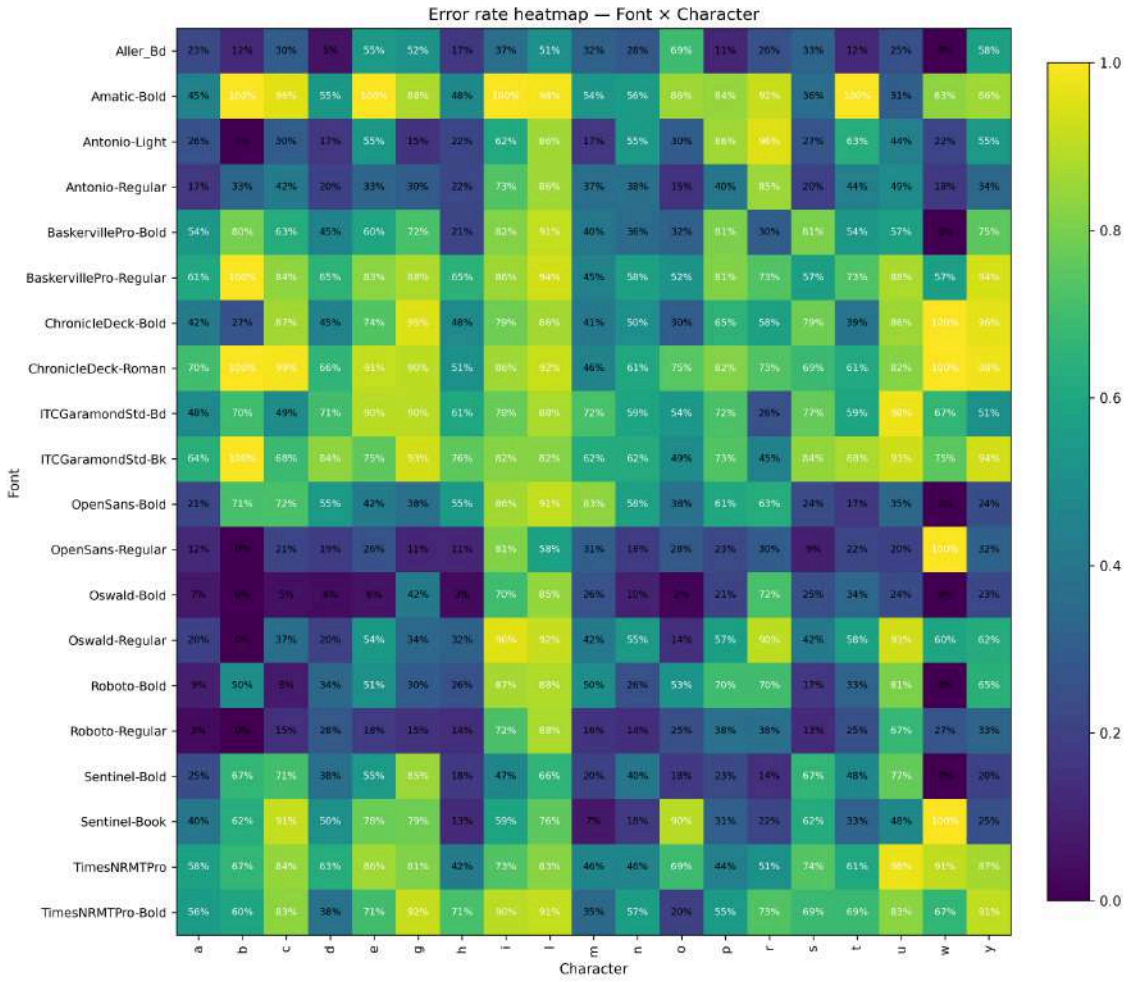
Per char analysis :

Here we are going to use the Char dataset to analyse the model deeply on char level



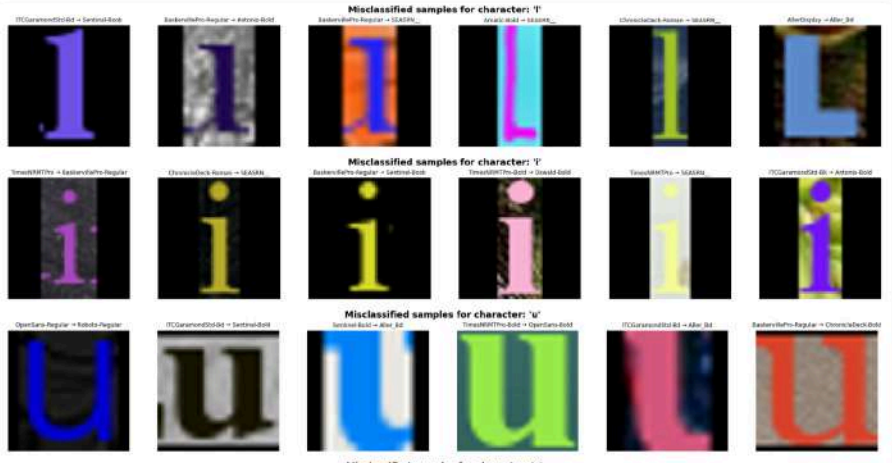
The char level analysis shows that some letters are much harder for the model to classify correctly, with the highest error rates for **l (79%)**, **I (74%)**, and **i (60%)**. These characters are visually similar, often differing only by subtle details like height, thickness, or the presence of small serifs, which become even harder to detect in low resolution or noisy images
Other problematic letters include rounded shapes like **u** and **e**, where slight curvature variations between fonts cause confusion, and characters like **g** and **y**

Error heat map per char :



The heatmap confirms that these errors are often concentrated between specific font pairs, especially when both fonts share similar stroke patterns.

Visual inspection

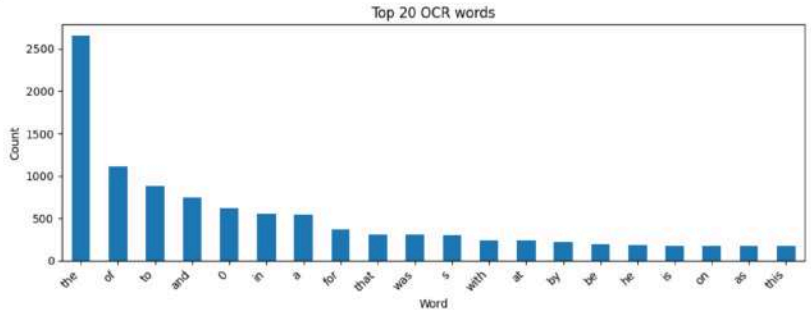


Visual inspection of misclassified samples reveals that image quality issues such as blur, compression artifacts, or distracting backgrounds also contribute to mistakes, as they mask the fine details that distinguish one font from another

Combining OCR with Model Predictions

As an extension of this analysis, I initiated an experiment to integrate Optical Character Recognition (OCR) (in the Ocr analysis notebook) with the ResNet results. The objective was to extract the text present in each image and analyze which specific words were associated with lower detection accuracy. This could provide additional insights into whether certain textual elements, font combinations, or letter shapes are systematically misclassified. Although this analysis was not completed in the current study, it remains a promising avenue for deeper error diagnosis and model improvement.

As a first attempt to analyse the result we could find the most repeated words



While the ResNet-based model provided a strong starting point for the font classification task, the detailed analyses (t-SNE clustering, per-character error rates, and per-font confusion heatmaps) highlighted several limitations:

- Significant overlap in the feature space for visually similar fonts.
- High misclassification rates for thin characters such as *l*, *i*, and *u*, often affected by blur or low resolution.
- Confusion concentrated among fonts sharing similar weight or style.

To address these issues, the model architecture was switched to **EfficientNet**, known for its improved parameter efficiency, better handling of image scale variations, and stronger performance on fine-grained classification tasks.

Given the time constraints of the 72-hour challenge, the same in-depth interpretation pipeline was not fully applied to the EfficientNet model. Instead, the evaluation focused on overall accuracy and Loss .

1.3.Efficient Net :

A.Data preprocessing

Used EfficientNet's preprocessing with light augmentations: rotation ±5°, small width/height shifts (±5%), zoom ±5%, and shear ±2%. Validation and test sets were left unaltered except for preprocessing.

B.Model Architecture and training

We selected EfficientNet-B0 for its compound scaling strategy, which optimizes depth, width, and resolution together, offering a strong accuracy–efficiency balance compared to heavier variants like B3 or B7. This made it well-suited for our 73h challenge constraints while still providing a performance boost over ResNet50.

Fine-Tuning Setup
For this experiment, we directly adopted a selective fine-tuning approach:

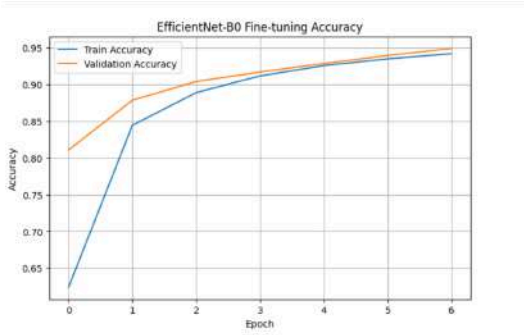
- Kept all Batch Normalization layers frozen to maintain training stability.
- Unfroze layers starting from block 5 onward, allowing the model to adapt higher-level features to the nuances of font shapes.
- Added a Dropout layer (0.3) before the Dense output layer to reduce overfitting.

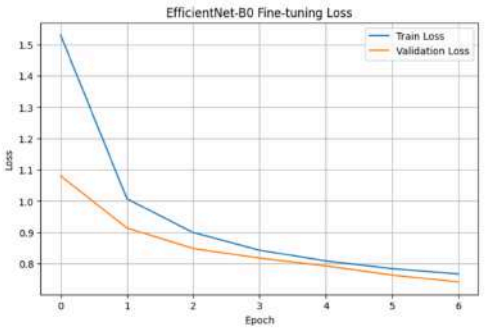
We trained the model with a batch size of 64 for 7 epochs, using Adam optimizer with a learning rate of 5×10^{-5} and label smoothing of 0.1.

C.Results

The validation accuracy started at **81.1%** after the first epoch and increased steadily, reaching **94.9%** by the final epoch. Training accuracy improved from **43.2%** (due to frozen early layers) to **94.1%**, with both curves showing parallel growth and no signs of divergence. Training and validation loss consistently decreased, with only a small gap between them — a strong indicator of good generalization and stability.

This smooth learning curve, compared to the more gradual gains of ResNet, suggests that EfficientNet's architecture is better at capturing fine-grained font differences when fine-tuned selectively.





1..4 Conclusion and Model benchmarking

The font classification experiments showed a clear performance progression as we moved from a baseline CNN to more advanced architectures. The CNN provided a strong starting point, achieving stable training and decent generalization, but struggled with visually similar fonts and subtle style variations. ResNet50 significantly improved accuracy by leveraging pretrained features and selective fine-tuning, capturing more nuanced font characteristics and reducing confusion for most classes. EfficientNet-B0 further enhanced performance, offering faster convergence, better generalization, and

superior handling of fine-grained differences thanks to its balanced scaling strategy.

Across all models, the main challenges remained consistent: fonts with minimal stylistic differences (e.g., regular vs. bold variants) and those with low contrast or noisy backgrounds were prone to misclassification. Detailed analyses, including confusion matrices, per-character error rates, and t-SNE visualizations, confirmed that many errors stemmed from inherent visual similarities rather than model shortcomings.

Overall, while EfficientNet delivered the highest accuracy within our constraints, future work could explore Vision Transformers, larger EfficientNet variants, and combined visual–textual representations (e.g., OCR-informed embeddings) to further reduce errors in challenging font pairs.

Model	Epochs	Batch Size	Training Accuracy	Validation Accuracy	Test Accuracy	Precision	Recall	F1-score
Baseline CNN	3	32	66.02%	72.31%	72.85%	74%	73%	73%
ResNet50 (fine-tuned)	5	64	91.61%	86.64%	86.63%	87%	86%	86%
EfficientNet-B0	7	64	94.10%	94.90%				

The Ressources used : kaggle, colab, and a local gpu .

2- NSFW Content detector :

We present a lightweight, fully open source pipeline to flag images that **contain NSFW text**, addressing cases where harmful language is embedded in the image itself (memes, screenshots, banners) and may bypass text only moderation systems. The task is formulated as **binary classification** NSFW vs. SFW using a **small dataset (89 images)** and since the dataset is small no training was needed to classify it because the models would have overfit too early.

Our approach begins with OCR to extract the text, followed by text normalization and classification without training. We tested multiple OCR engines **PyTesseract**, **EasyOCR**, and **Qwen-OCR** and found Qwen-OCR to be subjectively the most accurate and consistent in visual inspection, particularly for stylized or noisy text. We did not perform quantitative OCR accuracy evaluation due to the lack of ground truth transcriptions.

For classification, we started with **FlashText** keyword matching, progressed to **RoBERTa** for zero-shot natural language inference, and finally evaluated **LLMs** such as Qwen for richer contextual understanding.The best results in our current setup were obtained with the robust zero-shot classifier **MoritzLaurer/deberta-v3-large-zeroshot-v2.0**, which outperformed the other tested models without requiring any training. We expect that using more capable LLMs, such as LLaMA3 or similar models on suitable hardware, could further improve results.

The final configuration combines OCR output with zero-shot scores and applies a calibrated decision threshold, achieving **F1 = 0.97** and accuracy 0.966 on the held out set.

2.1 Optical character recognition (OCR)

The pipeline begins with **Optical Character Recognition (OCR)** to extract text from images. We focused on open-source, locally runnable tools and tested **PyTesseract**, **EasyOCR**, and **Qwen-OCR**, each offering different strengths for stylized or noisy text .

2.1.1 Tesseract OCR

Tesseract OCR is an open-source engine based on LSTM networks for text recognition. We used it through the Pytesseract Python . The engine outputs a raw text string from each image.

A . Processing Techniques Tested:

Initially, we applied preprocessing to improve recognition on noisy or stylized text. This included:

- Converting the image to **grayscale**.
- Applying a **median blur (3×3)** to reduce salt-and-pepper noise.
- Using **adaptive Gaussian thresholding** (block size 31, C=3) to binarize text against the background.

These steps often helped with low-contrast images but sometimes degraded fine details in stylized fonts. To balance speed and accuracy, we later tested a **direct OCR run without preprocessing**, passing the RGB image directly to Tesseract but the results were worse .

B. Results:

To illustrate Tesseract's output, we plotted some original images alongside the cleaned OCR text it produced. In addition, we generated **word clouds** from the recognized text to visualize the most frequent tokens across the dataset, and separately for NSFW vs. SFW subsets.

OCR (cleaned)

```
SS ee ere Greer EL LU To ee Nr ese et amet Hpi seth eo Se Wo EE ee Se Sa oo JSS
pee Bos te es on ST
```

the visual review confirmed earlier observations: Tesseract handled clean, high contrast text reasonably well but often or most of the times produced fragmented or incorrect tokens for stylized fonts and complex backgrounds or when the text is blended in the background and not clear enough .

The NSFW word cloud showed scattered terms with limited semantic clarity, highlighting the OCR’s difficulty in capturing accurate word forms in challenging images.

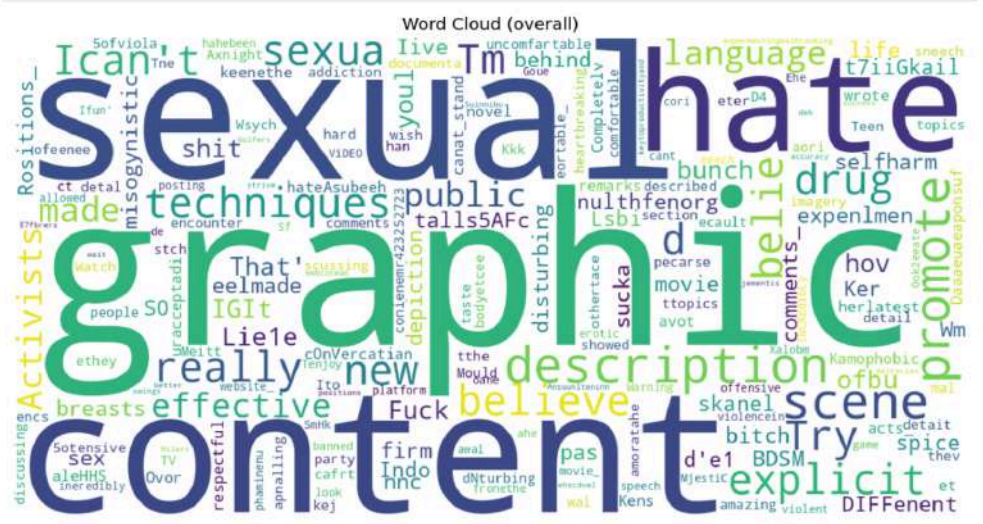
2.1.2 Easy Ocr

A. Method explanation

EasyOCR is a deep learning based text recognition library that uses convolutional and recurrent neural networks, along with attention mechanisms, to detect and read text in images. Unlike Tesseract, which is rule- and model based with fixed language packs, EasyOCR can better handle variable fonts, curved text, and complex backgrounds thanks to its learned features and end to end training on diverse scripts.

B. Results

We applied EasyOCR directly to the original images without any preprocessing. The output was more complete than Tesseract's for stylized, noisy, or low contrast text, and it improved coverage across the dataset.



The generated **word cloud** from all EasyOCR outputs clearly highlighted frequent NSFW-related terms such as sexual, graphic, and content, showing that the OCR successfully captured meaningful words for downstream classification. However, some noise and misread tokens remained, especially for very small text or heavy background patterns.

Compared to Tesseract's word cloud, EasyOCR's results showed far fewer random or fragmented tokens and a stronger concentration of relevant keywords. This suggests better text integrity and

higher semantic value in the extracted content, especially for NSFW related detection.

2.1.3 Qwen OCR

A.How it works

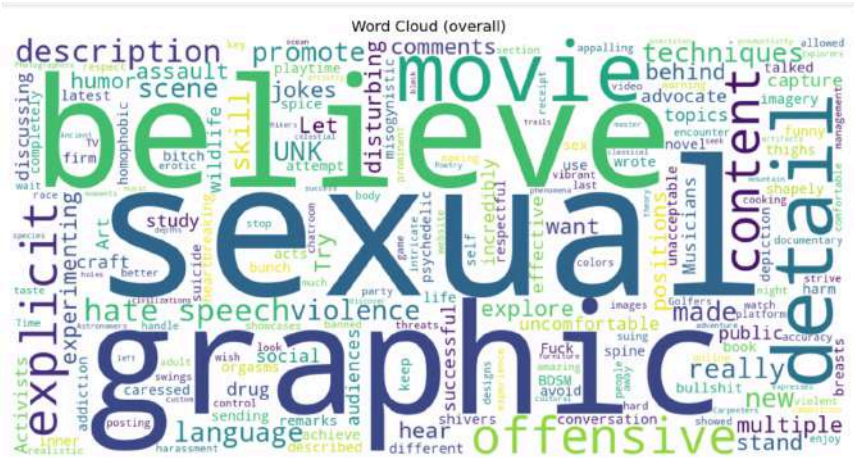
Qwen-OCR is a vision language model that combines image understanding and text generation. Unlike Tesseract or EasyOCR, which are specialized OCR pipelines, Qwen processes the image through a multimodal transformer and generates the recognized text token by token. This approach makes it more flexible with **stylized fonts, irregular layouts, curved text, and noisy backgrounds**, and it can better preserve structure such as line breaks when prompted.

We passed each RGB image directly to Qwen with the instruction:

“Return ONLY the exact text you see. Preserve line breaks. If unreadable, write [UNK].”

B.Results

Qualitative review showed Qwen outperforming Tesseract and often surpassing EasyOCR, especially on long, complex text blocks and decorative designs. It preserved wording and spacing better, resulting in text closer to the original image content.



The **word cloud** from Qwen outputs displayed a dense cluster of meaningful NSFW related terms*sexual*, *graphic*, *explicit*, *offensive*, *hate speech*, *violence* alongside contextual words like *movie*, *description*, *techniques*.

Compared to Tesseract's noisier cloud and EasyOCR's cleaner but slightly less rich vocabulary, Qwen's results combined **semantic relevance with textual completeness**, offering stronger input for the downstream classification step.

C.Limitations

Qwen requires more GPU memory and compute time than the other OCR tools. In our limited hardware setup, we could not run larger versions of the model, meaning even better performance could be expected on high end GPUs.

2.1.4 Conclusion

Overall, the OCR evaluation showed a clear progression in performance from Tesseract to EasyOCR to Qwen-OCR. Tesseract, while fast and lightweight, struggled with stylized fonts and complex backgrounds, leading to fragmented or incorrect tokens. EasyOCR's deep learning based approach significantly improved accuracy and semantic clarity, especially for irregular text layouts, though some errors persisted in extreme conditions. Qwen-OCR delivered the most contextually rich and complete outputs, effectively capturing nuanced and NSFW-related terms while preserving text structure, albeit at a higher computational cost. These results indicate that for NSFW content detection from images, richer OCR outputs like those from Qwen can provide a stronger foundation for downstream classification, especially when hardware resources permit.

2.2 NSFW content detector from text

2.2.1. Dataset Overview & Class Balance

Class inspection :

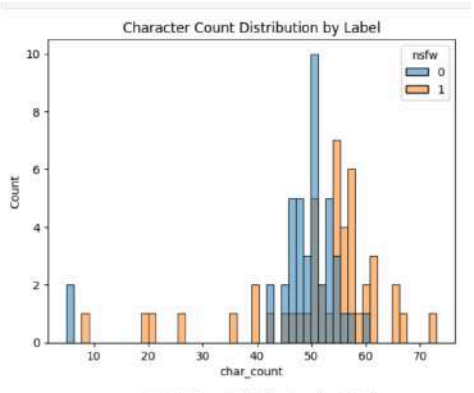
After extracting textual content from the images using **Qwen OCR**, we first inspected the dataset to understand its structure

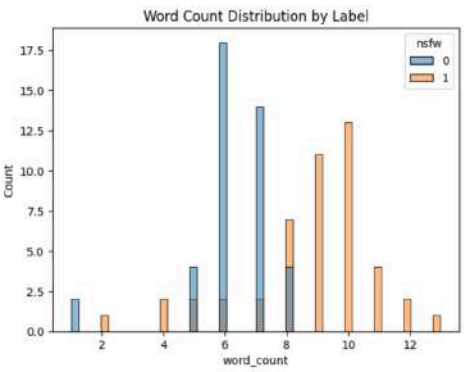
The **class distribution** analysis shows that NSFW and SFW samples are almost evenly represented.

This balance reduces the risk of class imbalance but since we are going to do training it shouldn't cause any problems .

Text Length Analysis

We evaluated both **character count** and **word count** distributions for each class Both classes center around 45–55 characters, but NSFW examples tend to be slightly longer
NSFW texts also show a tendency for higher word counts, suggesting more descriptive or detailed content.



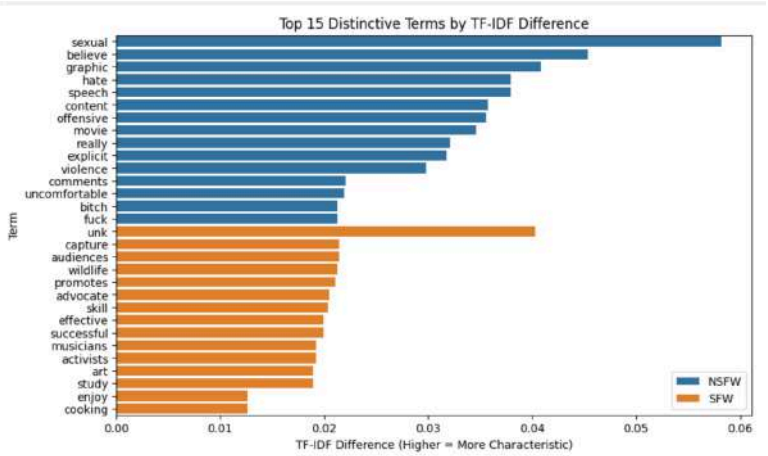


TF-IDF Distinctive Terms

We computed the **top distinctive terms** for each class using **TF-IDF difference**:

- **NSFW:** Words like *sexual*, *graphic*, *hate*, *explicit*, and *violence* have the highest TF-IDF differences, making them highly characteristic of this class.

- **SFW:** Words like *capture*, *wildlife*, *advocate*, *study*, and *musicians* dominate.



2.2.2 Flashtext (lexicon)baseline

A. Method

We built a profanity lexicon by merging three public lists (LDNOOBW, Google-profanity words by Robert J. Gabriel, and Coffee-and-Fun), lower-casing, deduplicating, and adding a small **whitelist** . We then used **FlashText** to scan text for exact term matches and flagged an image as NSFW if any matched term appeared. We first tried a lexicon derived **only from the first dataset** LDNOOBW, then expanded to the large public lists; results were essentially unchanged..

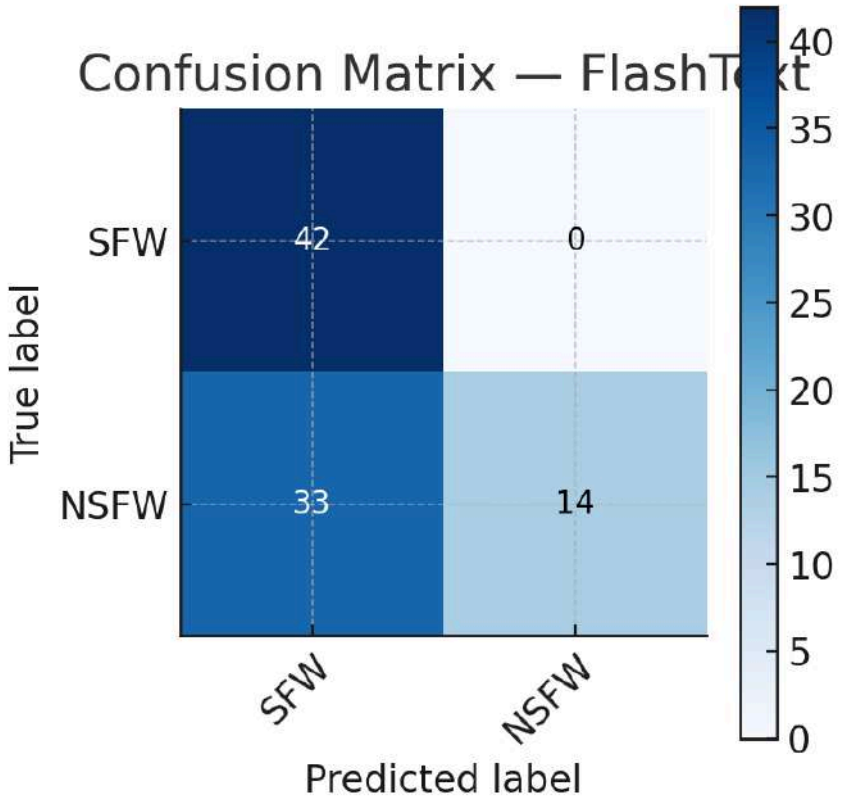
B.Results

We evaluated FlashText in two settings:

- **Baseline (small list):** Lexicon built only from NSFW terms from the first data .
- **BIG list:** Merged public lists (LDNOOBW, Google-profanity, Coffee-and-Fun) plus whitelist filtering.

	Accuracy	Precision (NSFW)	Recall (NSFW)	F1 (NSFW)
Small list	62.9%	1.00	0.30	0.46
BIG list merged	62.9%	1.00	0.30	0.46

Confusion matrix



FlashText underperformed because it relies solely on exact keyword matching. NSFW content in our dataset often uses euphemisms, misspellings, or context-dependent terms that are not captured by

static lexicons. As a result, many harmful sentences contain no exact “bad word” from the list, leading to low recall despite high precision .

2.2.3 Context-Aware Classification with RoBERTa (Detoxify) and DeBERTa

A.detoxify :

Detoxify (unbiased) is a **RoBERTa transformer** fine-tuned for multilingual toxicity detection, outputting probabilities for categories such as toxicity, obscene, threat, and insult.

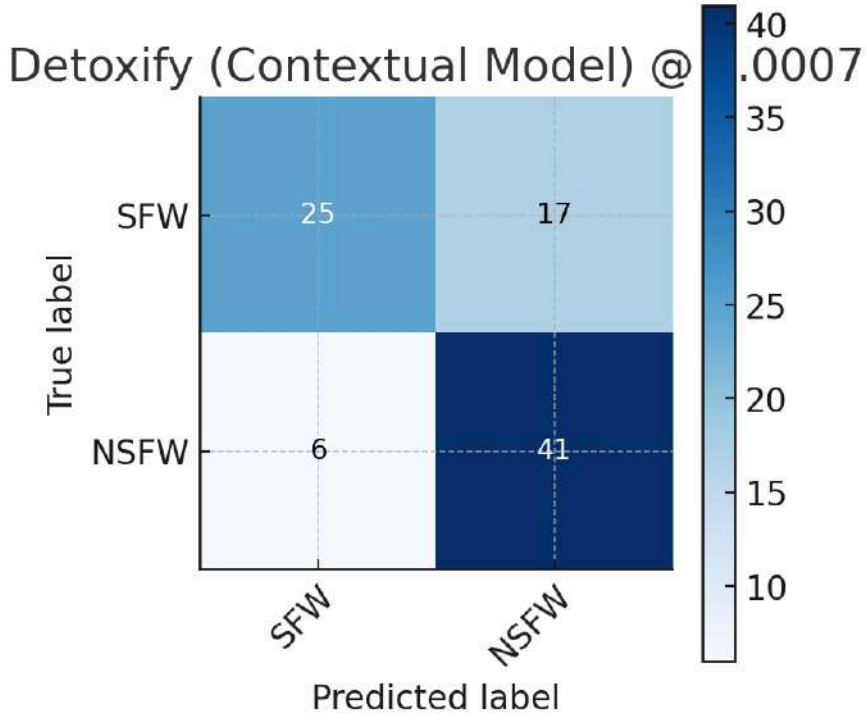
Our approach.

1. **Input:** Qwen-OCR text from each image.
2. **Scoring:** Ran Detoxify to obtain multiple context-related scores for each sample.
3. **NSFW signal:** Used ctx_nsfw_score as the main decision feature which was obtained as the max score of all context related scores .
4. **Threshold tuning:** Performed a **grid search** over threshold values to find the one maximizing the F1 score on our labeled dataset.

Best configuration (all signals):

- Threshold (τ) = **0.0007**
- F1 = **0.781**, Precision = **0.707**, Recall = **0.872**

Result :

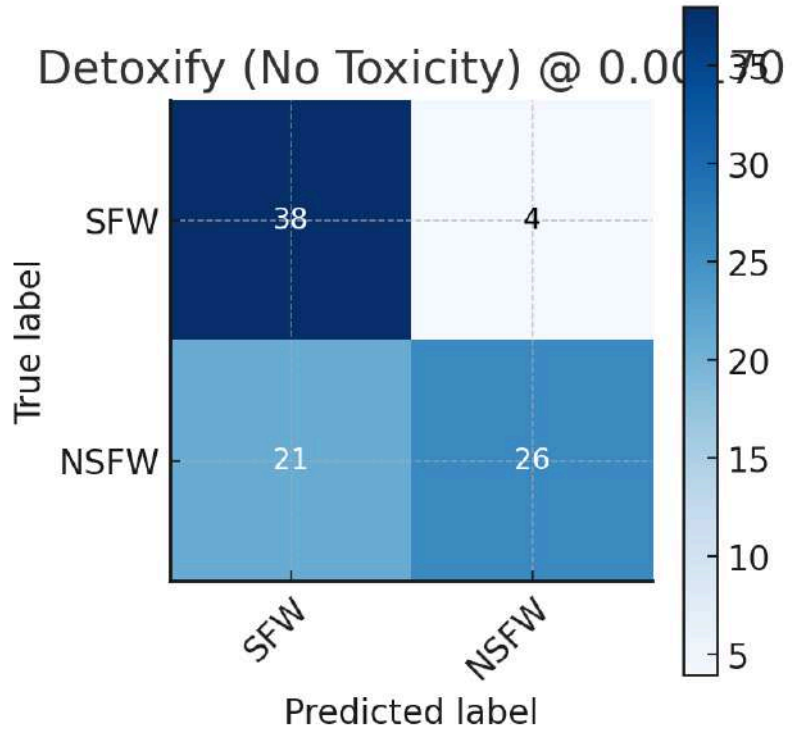


at τ = 0.0007, the model correctly classified **25/42 SFW** and **41/47 NSFW** samples, but misclassified 17 SFW as NSFW (false positives) and missed 6 NSFW (false negatives), indicating strong recall for NSFW content but lower precision due to **overflagging safe text.**

Further improvement :

After reviewing the errors from the τ = 0.0007 run, we found that many false positives came from SFW samples with very low but non zero toxicity scores. To address this, we removed the generic toxicity score from the decision features and repeated the same threshold tuning process.

- **F1 = 0.675, Precision = 0.867, Recall = 0.553, Accuracy = 0.719**



B. DeBERTa v3 Zero-Shot NLI Approach

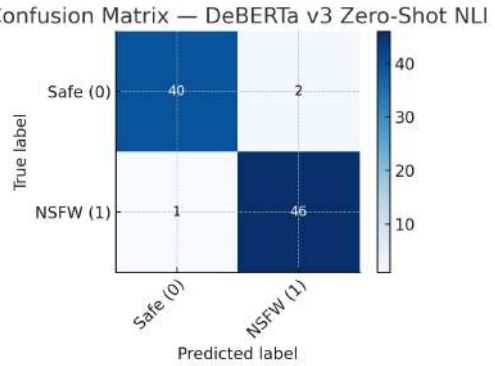
The **DeBERTa v3 zero-shot NLI model** is trained to understand the relationship between two sentences and decide if one supports the other, contradicts it, or is unrelated.
Here, we give it the text and the statement *"This text is nsfw"*, and it returns how likely this statement is true.

This means it can classify NSFW content **without being specifically trained for it**, just by changing the statement (prompt).

We computed the **NSFW probability score** for each text by comparing it against the “nsfw” and “safe” labels. Then, we performed a **grid search over possible thresholds** to maximize the F1-score.

At the best threshold (**0.2934**), the model achieved:

Metric	Value
Precision	0.958
Recall	0.979
F1-score	0.968
Accuracy	0.966



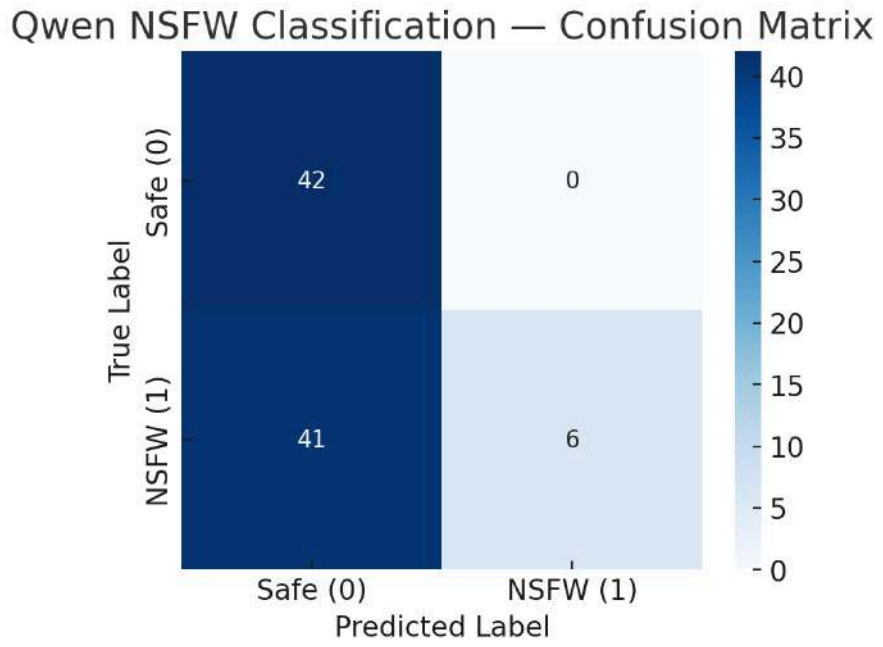
showing very few misclassifications and excellent separation between safe and NSFW texts

2.2.4Qwen-based NSFW Classification

We used **Qwen2.5-1.5B-Instruct**, a general-purpose instruction-tuned LLM, and prompted it to return a binary decision (1 = NSFW, 0 = Safe). Unlike Detoxify or DeBERTa NLI, Qwen was not specifically trained for content safety detection, so its classification ability relies entirely on understanding and following the prompt.

The results show that Qwen predicted all safe samples correctly (recall = 1.0 for class 0) but missed most NSFW cases (recall = 0.128 for class 1). This suggests that, in its default form, the model is **highly conservative** erring on the side of labeling content as safe unless very explicit as a remedy for

this we can try to change the prompt and make it more strict .



The results could likely be improved by using a more efficient LLM such as LLaMA 3 or a larger variant of Qwen, which would provide better language understanding and context handling.

2.2.5 Conclusion

Across all tested approaches, keyword-based FlashText proved inadequate for NSFW detection, as it failed to capture contextual or nuanced expressions. Detoxify, an encoder-only transformer trained on toxicity data, performed better by leveraging semantic understanding, but its results varied depending on whether toxicity signals were included either achieving higher recall with more false positives or higher precision with reduced recall. The DeBERTa v3 zero-shot NLI model delivered the best balance, with both precision and recall above 0.95, showing strong adaptability without task-specific training. In contrast, the Qwen-2.5 1.5B Instruct model, despite being capable of following classification prompts, underperformed due to its small size, missing most NSFW cases; however, its results could likely improve with larger or more advanced LLMs such as Qwen-14B or LLaMA 3.

2.3 Overall conclusion

In summary, the combined OCR and text classification pipeline demonstrates that both stages are critical for accurate NSFW content detection. On the OCR side, moving from Tesseract to EasyOCR to Qwen-OCR consistently improved the quality, completeness, and semantic relevance of extracted text, with Qwen providing the most reliable inputs despite higher computational demands. On the classification side, simple keyword matching via FlashText proved insufficient, as it lacked contextual understanding. Contextual models like Detoxify improved recall but remained sensitive to noisy signals and threshold tuning, while zero-shot NLI with DeBERTa delivered the highest accuracy and balanced performance without task-specific retraining. Together, these results highlight that the optimal approach pairs a high-quality OCR method (e.g., Qwen) with a robust, context-aware text classifier (e.g., zero-shot NLI), maximizing detection accuracy even on diverse and challenging image-text inputs.