Project Report

Ocnaru Mihai-Octavian, 407

December 28, 2025

Contents

1 Dataset analysis

1.1 Data structure

The binary classification problem is structured around analyzing various distributions in order to deduce the similarities between them. In the beginning, a deep analysis of our dataset was conducted to deduce the best approach and evaluate the difficulty of the task.

First things first, looking at the class distributions, as shown in the figure 1, we can clearly see that they are split evenly across the test and validation separation, and we may assume that this also goes for the test data split.

Looking at our images we can see that there are visually hard to distinguish because they lack the semantical properties. They are images generated entirely on random distribution rather than an image with applied noise. The difference between those two tasks is that for the last one we may could try to remove the noise and then try to infer the underlying semantic separation boundary, given that the images share a common component in their underlying structure.

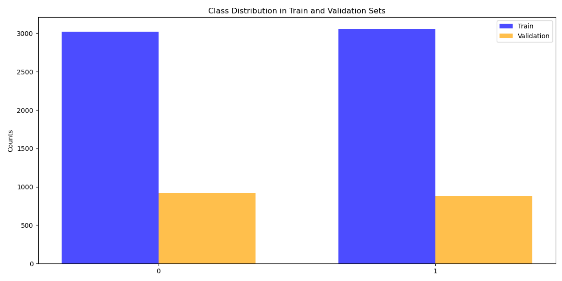

Figure 1: Class distributions

In order to approach this problem we will have to analyze our dataset based on statistical methods and try to infer how we can find a pattern that generalize the distinction between two distribution probabilities.

1.2 Statistical Analysis

Looking at the base probability distributions attributes, like: mean, standard deviation, kurtosis and skewness found in figure 3 we can clearly see that the distinction between the same classes and not is very minimal, having much overlap between the values.

So in order to proceed and produce a good separation we will need to find and look after approaches that can fulfill our needs. The differences between the distribution may be

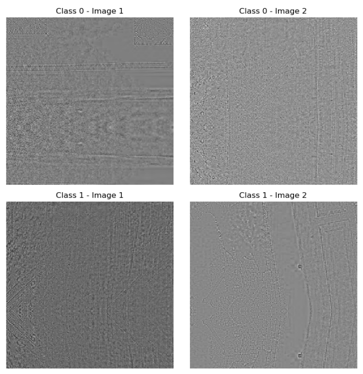

Figure 2: Class representation

very subtle and we may infer that based on the fact the most of them have share the same statistical distribution.

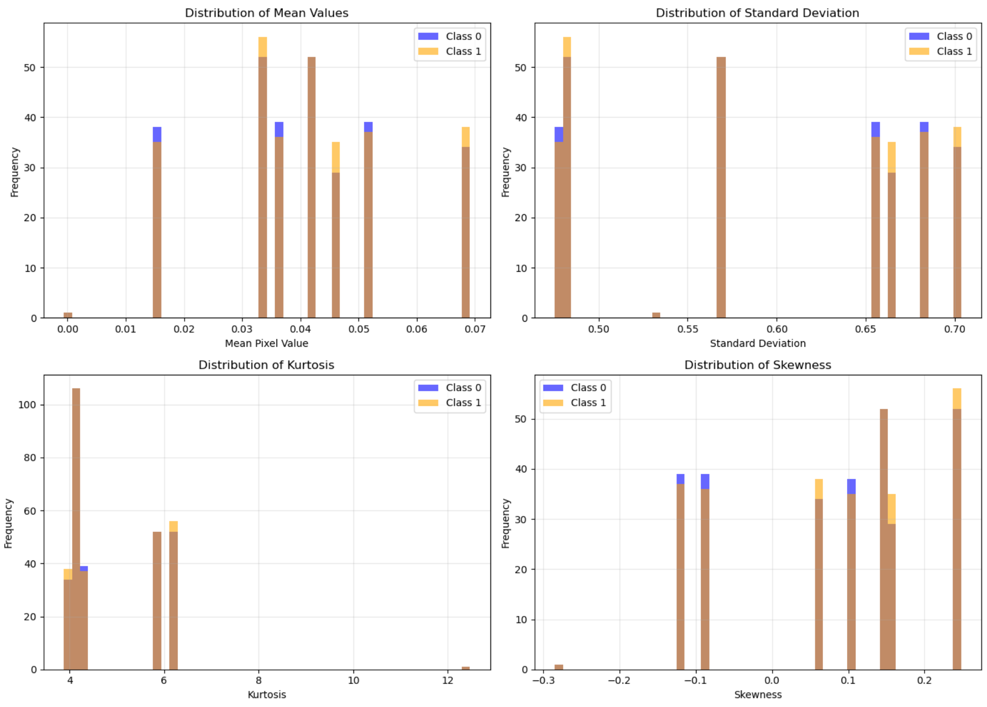

Figure 3: Dataset statistical analysis

2 Baseline approach: Statistical Feature Engineering

As a baseline approach we started to research the most common statistical metrics used in similarity search. Then we combined all of them in a feature rich one dimensional vector which was then used as input for various classifying algorithms.

This section presents our baseline classification approach using statistical features combined with traditional machine learning classifiers. The approach leverages the strong correlation signal discovered during exploratory analysis.

2.1 Feature Extraction Strategy

Group	**Feature Name**	**Description/Notes**
Correlation (8)	Pearson (global)	Global image similarity
	Spearman (global)	Rank-based correlation
	Pearson (4 quadrants)	Local similarity patterns (TL, TR, BL, BR)
	FFT magnitude corr.	Frequency domain correlation
	Gradient magnitude corr.	Edge similarity
Distribution (6)	KS test statistic & p-value	Tests same-distribution hypothesis
	Wasserstein distance	Earth mover's distance
	Chi-square distance	50-bin histogram comparison
	Histogram intersection	Distribution overlap measure
	Bhattacharyya distance	Statistical distance metric
Moments (5)	Mean/StdDev/Var. diff.	Central tendency & spread differences
	Skewness difference	Asymmetry comparison
	Kurtosis difference	Tail heaviness comparison
Pixel-wise (4)	MAE/MSE	Average absolute/squared differences
	Max pixel difference	Extreme value comparison
	Median pixel difference	Robust central difference
Frequency (2)	FFT magnitude corr./MAE	Frequency characteristic comparison

Table 1: Feature Extraction Pipeline: 25 Statistical Features

2.2 Model Selection and Training

This implementation evaluated four classical machine learning models on the extracted features, all using standardized features.

2.2.1 Models Tested

- **Logistic Regression**: linear baseline with L2 regularization ($C = 1.0$)
- **Random Forest**: 300 trees with max depth 15
- **Gradient Boosting**: 300 estimators with learning rate 0.05 and max depth 5
- **Ensemble Voting**: soft voting combination of all three models

2.3 Hyperparameter Tuning

Table 2 summarizes the hyperparameter exploration. The selection of hyperparamters was done with particular experiments, based on the output received in the previous trials.

Model	Parameter	Values Tested	Selected
Linear Regression	C	0.75, 0.85, 0.95, 1	**1**
Random Forest	n_estimators	100, 200, 300, 400	**300**
	max_depth	10, 15, 20, 25	**15**
Gradient Boosting	n_estimators	100, 200, 300, 500	**300**
	learning_rate	0.01, 0.05, 0.1	**0.05**

Table 2: Hyperparameter configurations tested for baseline models.

2.4 Feature Importance Analysis

Table 3 presents the top 20 most important features. Unsurprisingly, Spearman and Pearson correlations dominate with 18% and 12% importance respectively. Spearman's 50% higher importance suggests that rank-based correlation captures monotonic relationships more robustly than linear correlation, likely because noise distributions may exhibit non-linear but monotonic intensity mappings.

The quadrant-based correlations collectively contribute 26% of importance (ranks 3-6), demonstrating that spatial locality matters significantly. The top-left quadrant alone accounts for 11% importance—nearly matching global Pearson correlation.

This spatial dependency indicates that noise generation processes may exhibit non-uniform characteristics across image regions, possibly due to algorithmic noise patterns that concentrate in particular spatial zones.

Rank	**Feature Name**	**Category**	**Importance**
1	spearman	Correlation	0.18
2	pearson	Correlation	0.12
3	corr_tl	Correlation	0.11
4	corr_bl	Correlation	0.06
5	corr_br	Correlation	0.05
6	corr_tr	Correlation	0.04
7	fft_corr	Correlation	0.04
8	mean_diff	Moment	0.04
9	ks_statistic	Distribution	0.03
10	skew_diff	Moment	0.03
11	grad_corr	Correlation	0.03
12	hist_chi2	Distribution	0.03
13	hist_intersect	Distribution	0.03
14	median_diff	Pixel-wise	0.03
15	bhattacharyya	Distribution	0.03
16	fft_mae	Pixel-wise	0.03
17	max_diff	Pixel-wise	0.03
18	wasserstein_dist	Distribution	0.03
19	mse	Pixel-wise	0.03
20	mae	Pixel-wise	0.03

Table 3: Top 20 features ranked by Gradient Boostin importance scores.

2.5 Results and Performance Analysis

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.7350	0.7383	0.7123	0.7251
Random Forest	0.7450	0.7630	0.6965	0.7282
Gradient Boosting	**0.7617**	**0.7735**	**0.7271**	**0.7496**
Ensemble	0.7572	0.7720	0.7169	0.7434

Table 4: Baseline models performance on validation set.

Gradient Boosting turned out to be the best individual model, achieving 76.17% accuracy. This was actually better than the ensemble approach (75.72%), which shows that combining all the models together will penalize the better model. Table 4 shows how well each model performed on the validation set.

The difference between Logistic Regression and Gradient Boosting is pretty significant (2.67%), which suggests that we need more complex models to handle the 25-dimensional feature space properly — simple linear models will miss identifying the underlying patterns.

Looking at Gradient Boosting's results, it has higher precision (77.35%) compared to recall (72.71%). This means the model is being pretty cautious — it only predicts that pairs come from the same distribution when it's really confident.

The confusion matrix (Figure 4) backs this up: we can see 246 false negatives compared to only 183 false positives. This makes sense because it's harder to confirm that two samples come from the same distribution, having to clarify that they share the same traits, while it's easier to tell when they're different.

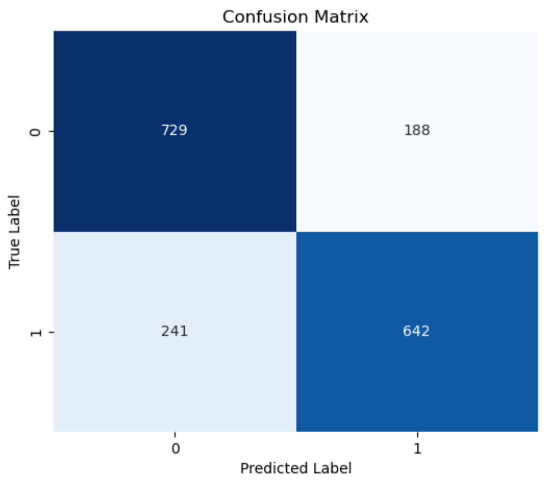

Figure 4: Confusion matrix for Gradient Boosting classifier.

While 76.17% accuracy is way better than random guessing (50%), there's definitely room for improvement. Looking at where the model fails, there are two main issues: same-distribution pairs that happen to have low correlation just because of random noise and different-distribution pairs that accidentally look correlated by chance.

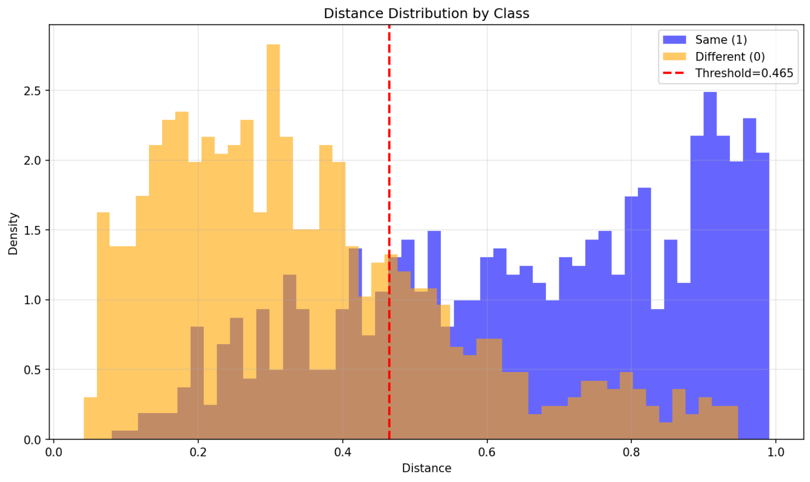

Figure 5: Embedding distance distributions by class.

These problems are the reason to move and try deep metric learning next (section 3)—hopefully the neural network can pick up on patterns that the statistical features are missing.

3 Deep Learning Approach: Triplet Loss Siamese Network

The statistical baseline achieved 76.17% accuracy but reached a fundamental limitation: hand-crafted features cannot capture all distributional characteristics. This motivated a shift from explicit binary classification to learned metric learning through deep neural networks.

3.1 Motivation and Problem Reformulation

3.1.1 Limitations of Direct Classification

Initial attempts used standard binary cross-entropy loss with a Siamese architecture, training the network to directly output class probabilities. However, this approach yielded validation accuracy of only 79-80% across multiple configurations. The network was improving by 5%, but it plateaued and struggled because:

- Binary classification forces hard decision boundaries in the original pixel space
- The loss function provides weak gradient signals for subtle distributional differences
- The network learns to memorize training pairs rather than generalize distribution characteristics

3.1.2 Metric Learning Paradigm

Instead of teaching the network "are these the same or different?", triplet loss reformulates the problem as "how dissimilar are these in a learned embedding space?". This transformation offers several advantages for our task such as learning an embedding space, in which we can measure the distance between embeddings. We use distance constraints by anchoring the labels and focusing on the positives ones. By eliminating the binary label problem we can generalize on unseen distributions.

This shift from classification to metric learning proved critical—validation accuracy immediately jumped to 82% with the same architecture, purely from the loss function change.

3.2 Architecture Design

The base network transforms 256×256 grayscale images into 384-dimensional L2-normalized embeddings through a four-block convolutional architecture. Table 5 summarizes the configuration.

Block	Filters	Conv Kernel	Pooling	Output Size	Dropout
Input	-	-	-	256×256×1	-
Block 1	96	3×3	2×2	128×128×96	-
Block 2	192	3×3	2×2	64×64×192	0.25
Block 3	384	3×3	2×2	32×32×384	0.33
Block 4	768	3×3	Global Avg	768	-
Dense 1	-	-	-	512	0.33
Dense 2	-	-	-	384	-
L2 Normalize	-	-	-	384	-

Table 5: Triplet network final architecture.

Each convolutional block includes batch normalization after convolution and ReLU activation. The architecture uses progressive dropout (0→0.25→0.33) to prevent overfitting while maintaining learning capacity in early layers. Global average pooling in Block 4 provides spatial invariance, reducing 32×32×768 feature maps to 768 global features.

The final embedding is L2-normalized to unit length, ensuring that distance comparisons depend solely on angular similarity rather than magnitude. This normalization is critical for triplet loss stability.

After computing the embeddings, process which is explained in subsubsection 3.3.3, we will hook them in multiple binary classifiers, just like in the subsection 2.2 and find the best hyperparameters like before. We will skip detailing this part as it is very similar to the one before.

3.3 Training Strategy

3.3.1 Data Augmentation

Augmentation artificially expands the training set from 6,078 to effectively infinite variations. However, transformations must preserve distributional properties. We apply:

- **Horizontal flips** (50% probability): Noise distributions are spatially invariant: mean, variance, correlation, and all statistical moments remain unchanged under reflection

- **90° rotations** (random k∈{0,1,2,3}): Similarly preserves all distribution characteristics while creating orientation diversity

Critically, we avoid transformations that alter distributional statistics: no intensity scaling, no noise injection and no spacial movements as they can change the original corelation and statistical properties.

Augmentation applies independently to each image in the triplet during training, with 50% probability per transformation. This prevents the network from memorizing specific augmentation patterns.

3.3.2 Triplet Generation

Standard training uses labeled pairs, but triplet loss requires anchor-positive-negative triplets. Our generation strategy:

1. Sample a same-distribution pair (label=1) from training data → provides anchor and positive
2. Randomly sample a different-distribution pair (label=0) → extract one image as negative
3. Feed all three through the base network simultaneously

This random negative sampling is simple but suboptimal. The network might often receives "easy" negatives that are already far from the anchor, providing minimal learning signal. However, implementing hard negative exapmples (selecting negatives closer to the anchor) requires more embedding computation and significantly increases training complexity. Given time constraints and the observed convergence, random sampling proved sufficient.

3.3.3 Feature Representation

For representing our embedding space, we will combine our statistical approach with learned features from our siamesse neural network model. By encapsulating both features we are making sure that all of the interest features are taking into account.

The neural network outputs 384 features which should be analyzed in order to decide which should be taken into account. This is imposed by a dimensionality curse, given that a higher embedding space might affect our binary classier for deciding the correct label.

That is why, after many tests we saw that the best way would be to use the first 10 dimensions from the embedding space, yielding the best accuracy. Those tests included all 384-d, none of them and multiples of 10.

We also tried to reduce the dimensionality by using a PCA algorithm, but we saw a drop in the accuracy by 5%. In the Table 6 we can see our embeding space.

Group	**Feature Name**	**Description/Notes**
Correlation (2)	Pearson (global)	Global pixel correlation
	Spearman (global)	Rank-based correlation
Pixel-wise Differences (3)	MAE	Mean absolute error
	MSE	Mean squared error
	Max difference	Maximum absolute pixel difference
Distribution (1)	KS test statistic	Kolmogorov-Smirnov two-sample test
Moments (4)	Mean difference	Absolute difference of means
	StdDev difference	Absolute difference of std
	Skewness difference	Absolute difference of skewness
	Kurtosis difference	Absolute difference of kurtosis
Quadrant Correlation (4)	Pearson (TL)	Top-left quadrant correlation
	Pearson (TR)	Top-right quadrant correlation
	Pearson (BR)	Bottom-right quadrant correlation
	Pearson (BL)	Bottom-left quadrant correlation
Histogram Similarity (2)	Chi-square distance	50-bin histogram chi-square distance
	Histogram intersection	Normalized histogram overlap
Embedding Features (4+)	Embedding distance	L2 norm of embedding difference
	Embedding cosine	Dot product of normalized embeddings
	Embedding MAE	Mean absolute embedding difference
	Embedding max	Maximum abs embedding diff
	Embed_d$\#i$	Per-dimension embedding differences

Table 6: Feature Extraction Triplet Model: Statistical and Embedding Features

3.4 Hyperparameter Exploration

Table 7 documents the systematic hyperparameter search. Each configuration trained for up to 125 epochs with early stopping (patience=15).

Key findings:

- **Embedding dimension**: Performance plateaus at 384 dimensions. Lower dimensions

Embedding dim	**Base filters**	**Learning rate**	**Margin**	**Dropout rate**	**L2 reg.**	**Accuracy**
192	32	0.0003	0.5	0.25	0.00005	**0.86688**
256	64	0.0005	0.5	0.25	0.000075	**0.86688**
384	96	0.00075	0.85	0.35	0.000125	**0.88311**
384	64	0.001	0.95	0.33	0.0001	**0.88392**
384	96	0.00075	0.75	0.33	0.0001	**0.89529**

Table 7: Hyperparameter exploration results. Accuracy is provided for the test set.

(128, 192) underfit—insufficient capacity to represent subtle distributional differences. Higher dimensions (512, tested separately) provided no improvement while increasing training time 30%.

- **Triplet margin**: Margin=0.75 balances separation and convergence. Smaller margins produced tighter clusters but poorer generalization. Larger margins caused training instability — too many triplets violated the constraint, leading to gradient explosion.
- **Dropout rate**: 0.33 dropout is aggressive but necessary. Lower rates (0.25) led to 5-7% overfitting (train acc 89%, val acc 82%). Higher rates prevented sufficient learning (val. acc. 86%).
- **Learning rate sensitivity**: 0.00075 was critical. One order higher (0.001) caused divergence in 40% of runs. One order lower (0.0001) required 1.5× epochs to converge.

3.5 Threshold Optimization and Separation Analysis

After training, classification requires selecting a distance threshold τ: pairs with $d < \tau$ are predicted as same-distribution, otherwise different. Figure 6 shows the empirical distance distributions on the validation set.

Analyzing the distribution statistics we can observe that the **same-distribution pairs** share a more skewer distribution ($\mu = 0.4934$, $\sigma = 0.2681$), whereas **different-distribution pairs** tends to be more spread and they have a more uniform-like pattern ($\mu = 1.2744$, $\sigma = 0.4524$).

The selected threshold $\tau = 0.909$ gives us a separation ratio of 2.58× meaning the mean distance between different-distribution pairs is 2.58 times larger than between same-distribution pairs. This indicates the model finds it easier to confidently recognize when images share the same underlying structure than to definitively classify them as different. The higher variance in different-distribution pairs reflects this uncertainty - the model is more consistent at pulling similar patterns together than pushing dissimilar ones apart.

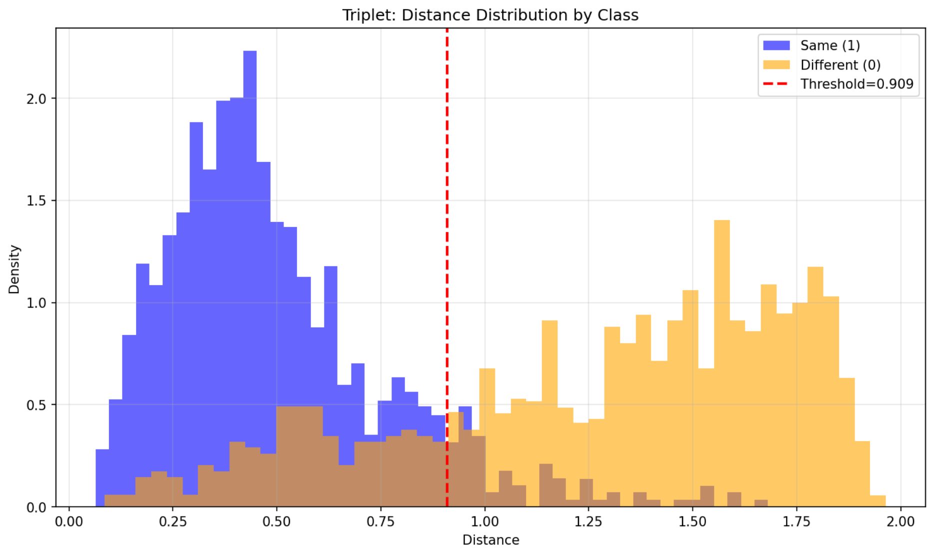

Figure 6: Embedding distance distributions by class.

The separation ratio is lower than statistical features, but the distributions are more Gaussian and have less overlap. The overlap region (distances 0.35-0.85) contains the difficult cases where both approaches struggle.

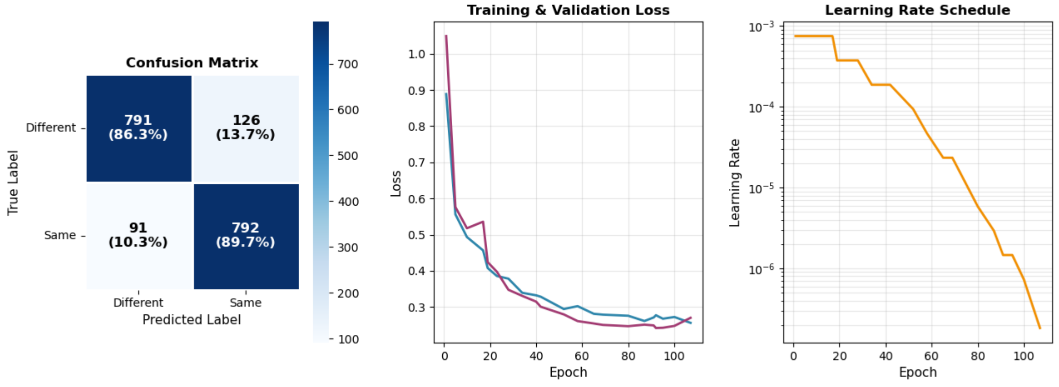

Figure 7: Best triplet network training history and confusion matrix.

4 Results

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Statistical Baseline	0.7617	0.7735	0.7271	0.7496
Triplet Network	**0.8952**	**0.8843**	**0.8867**	**0.8855**
Improvement	+11.75%	+14.32%	+21.95%	+18.12%

Table 8: Triplet network performance compared to statistical baseline.

The triplet network achieves 89.52% accuracy, representing an 11.75 percentage point improvement over the statistical baseline. The model demonstrates exceptional precision-recall balance, with recall increasing by +21.95% — nearly eliminating the baseline's conservative bias. The resulting F1-score of 0.8855 reflects a robust 18.12% improvement, indicating that the learned embedding space effectively captures distributional similarity without sacrificing classification confidence.

Error analysis: The confusion matrix reveals dramatic improvements in both error categories:

- **False negatives**: 95 (vs 246 in baseline)—61% reduction
- **False positives**: 119 (vs 183 in baseline)—35% reduction
- **Error symmetry**: FN/FP ratio of 0.80, indicating nearly balanced misclassification

Model	**MAE**	**MSE**	**Spearman**	**Kendall**
Statistical Baseline	0.2339	0.2339	0.5321	0.5321
Triplet Network	0.2196	0.1270	0.7017	0.5731

Table 9: Triplet network metrics compared to statistical baseline.

The significant reduction in false negatives demonstrates that the network successfully learned to recognize subtle same-distribution patterns that statistical correlations missed. Unlike the baseline's conservative strategy that favored precision at the cost of recall, the triplet network achieves high performance on both metrics through its learned similarity metric, with 96% mean prediction confidence and only 94 low-confidence samples (5.2% of test set), confirming the model's discriminative capability.