Group 10

# COMPARISON OF NEURAL NETWORK ARCHITECTURES FOR ART GENRE CLASSIFICATION USING IMAGE METADATA

HOON HAN
JOONHYOUNG LEE
STEPHEN YEAP WELLER

# TABLE OF CONTENTS

# Introduction

## Project Objective

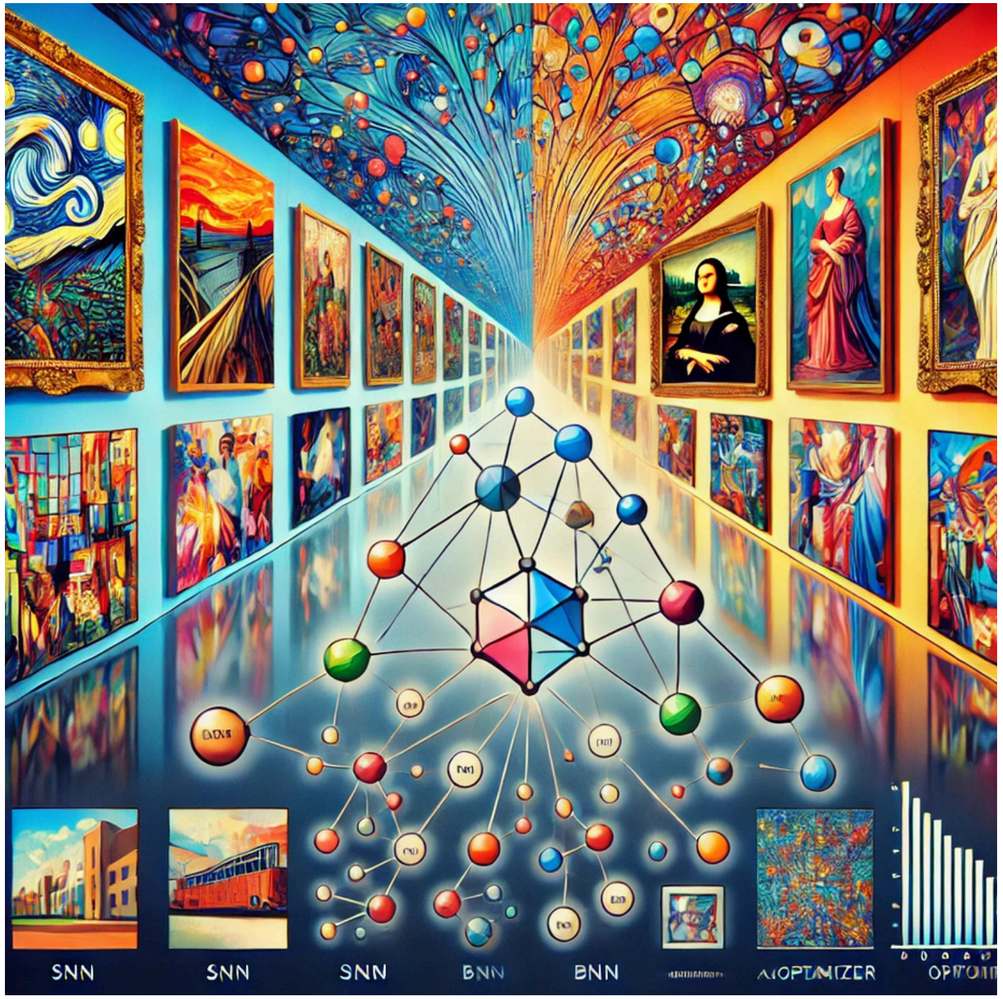
- Classify paintings into artistic genres using machine learning.
- Focus on stylistic patterns over artist identification.

## Our Approach

- **Simple Neural Network (SNN)**
  – Baseline model with straightforward architecture.
- **Boosted Neural Network (BNN)**
  – Ensemble method to correct misclassification.
- **SNN with Adam Optimizer**
  – Adaptive learning rate for improved convergence.

## Significance

- **Art Education:** Tools for learning and exploring art styles.
- **Digital Curation:** Efficient organization of art collections for professionals and enthusaiasts.

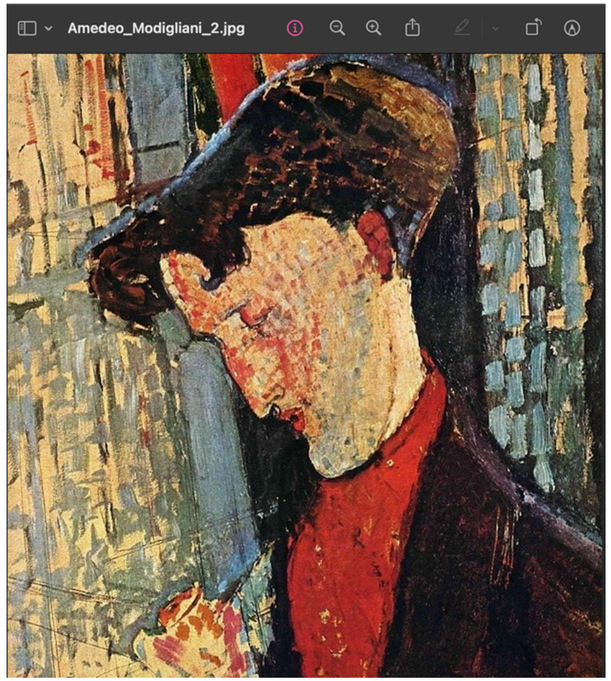


# Dataset Overview

artists

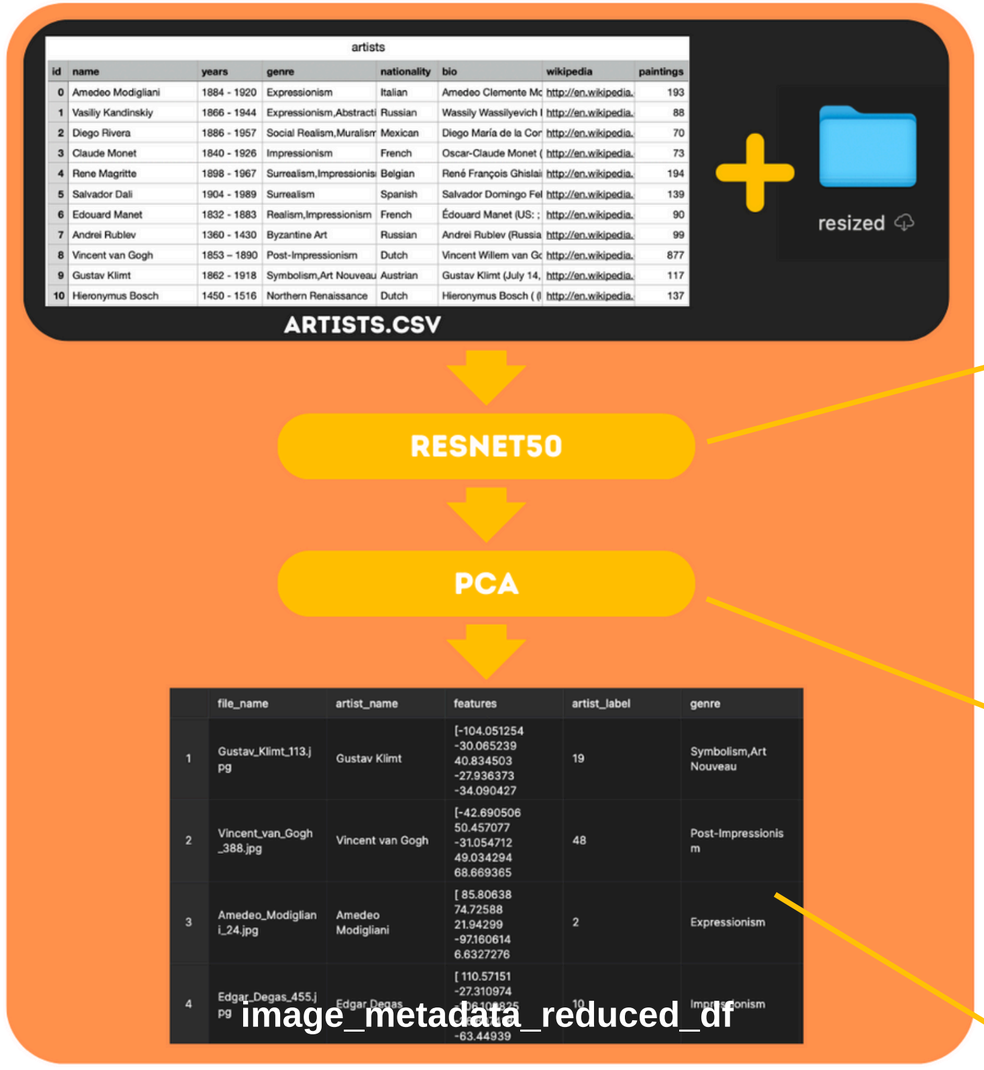
| id | name | years | genre | nationality | bio | wikipedia | paintings |
|---|---|---|---|---|---|---|---|
| 0 | Amedeo Modigliani | 1884 - 1920 | Expressionism | Italian | Amedeo Clemente Modigliani (Italian pronun | http://en.wikipedia.org/wiki/Amedeo_Modiglian | 193 |
| 1 | Vasiliy Kandinskiy | 1866 - 1944 | Expressionism,Abstractionism | Russian | Wassily Wassilyevich Kandinsky (Russian: Ba | http://en.wikipedia.org/wiki/Wassily_Kandinsky | 88 |
| 2 | Diego Rivera | 1886 - 1957 | Social Realism,Muralism | Mexican | Diego María de la Concepción Juan Nepomu | http://en.wikipedia.org/wiki/Diego_Rivera | 70 |
| 3 | Claude Monet | 1840 - 1926 | Impressionism | French | Oscar-Claude Monet (; French: [klod mɔnɛ]; | http://en.wikipedia.org/wiki/Claude_Monet | 73 |
| 4 | Rene Magritte | 1898 - 1967 | Surrealism,Impressionism | Belgian | René François Ghislain Magritte (French: [ʁə | http://en.wikipedia.org/wiki/René_Magritte | 194 |
| 5 | Salvador Dali | 1904 - 1989 | Surrealism | Spanish | Salvador Domingo Felipe Jacinto Dalí i Domè | http://en.wikipedia.org/wiki/Salvador_Dalí | 139 |
| 6 | Edouard Manet | 1832 - 1883 | Realism,Impressionism | French | Édouard Manet (US: ; UK: ; French: [edwaʁ m | http://en.wikipedia.org/wiki/Édouard_Manet | 90 |
| 7 | Andrei Rublev | 1360 - 1430 | Byzantine Art | Russian | Andrei Rublev (Russian: Андре́й Рублёв, IPA | http://en.wikipedia.org/wiki/Andrei_Rublev | 99 |
| 8 | Vincent van Gogh | 1853 – 1890 | Post-Impressionism | Dutch | Vincent Willem van Gogh (Dutch: [ˈvɪnsɛnt ˈʊɪ | http://en.wikipedia.org/wiki/Vincent_van_Gogh | 877 |
| 9 | Gustav Klimt | 1862 - 1918 | Symbolism,Art Nouveau | Austrian | Gustav Klimt (July 14, 1862 – February 6, 19 | http://en.wikipedia.org/wiki/Gustav_Klimt | 117 |
| 10 | Hieronymus Bosch | 1450 - 1516 | Northern Renaissance | Dutch | Hieronymus Bosch ( (listen); Dutch: [hijeːˈroːnɪ | http://en.wikipedia.org/wiki/Hieronymus_Bosch | 137 |

**Artists.csv: 50 Artists, genre lables**


**resized folder: 8355 image files of paintings**

# Data Preprocessing




| id | name | years | genre | nationality | bio | wikipedia | paintings |
|---|---|---|---|---|---|---|---|
| 0 | Amedeo Modigliani | 1884 - 1920 | Expressionism | Italian | Amedeo Clemente Mc | http://en.wikipedia. | 193 |
| 1 | Vasiliy Kandinskiy | 1866 - 1944 | Expressionism,Abstracti | Russian | Wassily Wassilyevich I | http://en.wikipedia. | 88 |
| 2 | Diego Rivera | 1886 - 1957 | Social Realism,Muralism | Mexican | Diego María de la Cor | http://en.wikipedia. | 70 |
| 3 | Claude Monet | 1840 - 1926 | Impressionism | French | Oscar-Claude Monet ( | http://en.wikipedia. | 73 |
| 4 | Rene Magritte | 1898 - 1967 | Surrealism,Impressionis | Belgian | René François Ghislai | http://en.wikipedia. | 194 |
| 5 | Salvador Dali | 1904 - 1989 | Surrealism | Spanish | Salvador Domingo Fel | http://en.wikipedia. | 139 |
| 6 | Edouard Manet | 1832 - 1883 | Realism,Impressionism | French | Édouard Manet (US: ; | http://en.wikipedia. | 90 |
| 7 | Andrei Rublev | 1360 - 1430 | Byzantine Art | Russian | Andrei Rublev (Russia | http://en.wikipedia. | 99 |
| 8 | Vincent van Gogh | 1853 – 1890 | Post-Impressionism | Dutch | Vincent Willem van Gc | http://en.wikipedia. | 877 |
| 9 | Gustav Klimt | 1862 - 1918 | Symbolism,Art Nouveau | Austrian | Gustav Klimt (July 14, | http://en.wikipedia. | 117 |
| 10 | Hieronymus Bosch | 1450 - 1516 | Northern Renaissance | Dutch | Hieronymus Bosch ( | http://en.wikipedia. | 137 |

| | file_name | artist_name | features | artist_label | genre |
|---|---|---|---|---|---|
| 1 | Gustav_Klimt_113.jpg | Gustav Klimt | [-104.051254 -30.065239 40.834503 -27.936373 -34.090427 | 19 | Symbolism,Art Nouveau |
| 2 | Vincent_van_Gogh_388.jpg | Vincent van Gogh | [-42.690506 50.457077 -31.054712 49.034294 68.669365 | 48 | Post-Impressionism |
| 3 | Amedeo_Modigliani_24.jpg | Amedeo Modigliani | [ 85.80638 74.72588 21.94299 -97.160614 6.6327276 | 2 | Expressionism |
| 4 | Edgar_Degas_455.jpg | Edgar_Degas | [ 110.57151 -27.310974 | 10 | Impressionism |

```python
# Load the pretrained ResNet50 model (without the top classification layer)
model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Function to extract features from an image
def extract_features(image_path):
    img = load_img(image_path, target_size=(224, 224))  # Resize image to 224x224
    img_array = img_to_array(img)
    img_array = preprocess_input(img_array)  # Preprocess for ResNet50
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    features = model.predict(img_array)
    return features.flatten()  # Flatten the feature map to a 1D vector
```

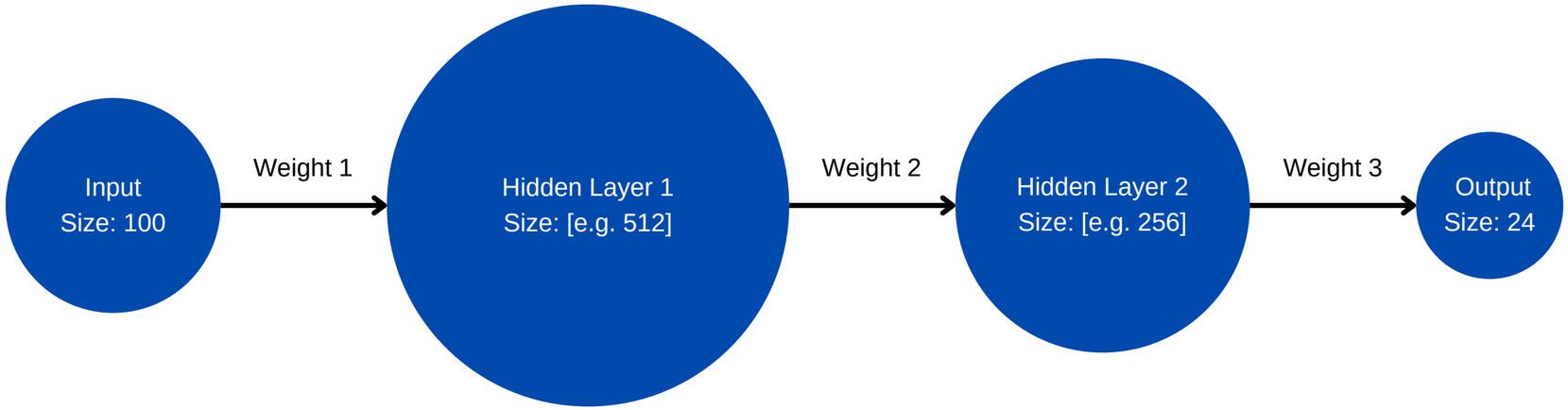
```python
# Step 1: Extract feature matrix from image_metadata_df
X = np.stack(image_metadata_df['features'].values)  # Shape: [n_samples, n_features]

# Step 2: Apply PCA to reduce dimensionality to 100 components
pca = PCA(n_components=100, random_state=42)
X_reduced = pca.fit_transform(X)  # Shape: [n_samples, 100]

# Step 3: Replace the 'features' column with the reduced features
image_metadata_df['features'] = list(X_reduced)  # Store reduced features as lists
```

```python
# One-hot encode the genres
mlb = MultiLabelBinarizer()
one_hot_genres = mlb.fit_transform(image_metadata_reduced_df['genre_list'])
```

# Models’ Architecture

# Model 1: Simple Neural Network (SNN)

<Structure of out SNN>




## Why Only Two Layers?

1. Pre-trained ResNet50 & PCA
2. Gradient Vanishing
3. Insufficient data (8,355)
4. Computational coast
5. Imbalanced genre labels

# Model 1: Simple Neural Network (SNN)

## 1) Initalization of Parameters

```
self.weights = {
    "W1": np.random.randn(input_size, hidden_sizes[0]) * 0.01,
    "W2": np.random.randn(hidden_sizes[0], hidden_sizes[1]) * 0.01,
    "W3": np.random.randn(hidden_sizes[1], output_size) * 0.01,
}
self.biases = {
    "b1": np.zeros((1, hidden_sizes[0])),
    "b2": np.zeros((1, hidden_sizes[1])),
    "b3": np.zeros((1, output_size)),
}
```

## 2) Forward Propagation

```
def forward(self, X):
    # Forward propagation
    self.Z1 = np.dot(X, self.weights["W1"]) + self.biases["b1"]
    self.A1 = self.relu(self.Z1)

    self.Z2 = np.dot(self.A1, self.weights["W2"]) + self.biases["b2"]
    self.A2 = self.relu(self.Z2)

    self.Z3 = np.dot(self.A2, self.weights["W3"]) + self.biases["b3"]
    self.A3 = self.sigmoid(self.Z3)  # Sigmoid activation for multi-label classification
```

**Theory**

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}, \quad A^{[l]} = \mathrm{ReLU}(Z^{[l]}), \quad A^{[L]} = \sigma(Z^{[L]})$$

# Model 1: Simple Neural Network (SNN)

## 3) Loss Computation

```python
def binary_cross_entropy_loss(self, y_pred, y_true):
    n_samples = y_true.shape[0]
    return -np.sum(y_true * np.log(y_pred + 1e-10) + (1 - y_true) * np.log(1 - y_pred + 1e-10)) / n_samples
```

**Theory**

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(y_{\text{pred},i}) + (1 - y_i)\log(1 - y_{\text{pred},i})]$$

## 4) Backward Propagation and Weights & Bias Update

```python
def backward(self, X, y_true, y_pred):
    # Backward propagation
    n_samples = y_true.shape[0]

    # Gradients for the output layer
    dZ3 = y_pred - y_true  # Binary cross-entropy gradient
    dW3 = np.dot(self.A2.T, dZ3) / n_samples
    db3 = np.sum(dZ3, axis=0, keepdims=True) / n_samples

    # Gradients for the second hidden layer
    dA2 = np.dot(dZ3, self.weights["W3"].T)
    dZ2 = dA2 * self.relu_derivative(self.Z2)
    dW2 = np.dot(self.A1.T, dZ2) / n_samples
    db2 = np.sum(dZ2, axis=0, keepdims=True) / n_samples

    # Update weights and biases
    self.weights["W1"] -= self.learning_rate * dW1
    self.biases["b1"] -= self.learning_rate * db1

    self.weights["W2"] -= self.learning_rate * dW2
    self.biases["b2"] -= self.learning_rate * db2

    self.weights["W3"] -= self.learning_rate * dW3
    self.biases["b3"] -= self.learning_rate * db3
```

**Theory**

$$\delta^{[L]} = A^{[L]} - Y$$

$$\frac{\partial \mathcal{L}}{\partial W^{[L]}} = A^{[L-1]}\delta^{[L]} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b^{[L]}} = \delta^{[L]}$$

$$\delta^{[l]} = (W^{[l+1]})^T \delta^{[l+1]} \odot \text{ReLU}'(Z^{[l]})$$

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = A^{[l-1]}\delta^{[l]} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \delta^{[l]}$$

$$W^{[l]} = W^{[l]} - \eta\frac{\partial \mathcal{L}}{\partial W^{[l]}} \quad \text{and} \quad b^{[l]} = b^{[l]} - \eta\frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

# Model 1: Simple Neural Network (SNN)

## 6) Training Loop

```python
def train(self, X, y, epochs=20, batch_size=32):
    n_samples = X.shape[0]

    for epoch in range(epochs):
        # Shuffle the data
        indices = np.arange(n_samples)
        np.random.shuffle(indices)
        X = X[indices]
        y = y[indices]

        # Mini-batch gradient descent
        for i in range(0, n_samples, batch_size):
            X_batch = X[i:i + batch_size]
            y_batch = y[i:i + batch_size]

            # Forward and backward propagation
            y_pred = self.forward(X_batch)
            self.backward(X_batch, y_batch, y_pred)


        # Compute loss
        y_pred_full = self.forward(X)
        loss = self.binary_cross_entropy_loss(y_pred_full, y)
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss:.4f}")
```

### Theory

$$W^{[l]} = W^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}} \quad \text{and} \quad b^{[l]} = b^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

### Theory

- Mini-batching
- Epochs repeat

# Model 2: Boosted Neural Network- Ensemble Method (BNN)

## Theory

*Train multiple neural networks sequentially, with each subsequent model addressing the errors of its predecessors.*

## 1,2,3) Initalization of Parameters, Forward Propagation, Loss Computation

Same with Model 1

## 4) Backward Propagation and Weight & Bias Updates:

Gradient clipping ensures that gradients do not exceed a predefined threshold, avoiding unstable updates. For this model, the clipping value is set to 0.5

```
dW3 = np.clip(dW3, -gradient_clip_value, gradient_clip_value)
dW2 = np.clip(dW2, -gradient_clip_value, gradient_clip_value)
dW1 = np.clip(dW1, -gradient_clip_value, gradient_clip_value)
```

## 5) (Ensemble) Training Residual Updates:

Gradient clipping ensures that gradients do not exceed a predefined threshold, avoiding unstable updates. For this model, the clipping value is set to 0.5

```
def train(self, X, y, epochs=10, batch_size=32, residual_threshold=1e-3):
    residuals = y.astype(np.float64)
    for model_idx, model in enumerate(self.models):
        print(f"Training Model {model_idx + 1}/{len(self.models)}")
        model.train(X, residuals, epochs, batch_size, self.gradient_clip_value)
        predictions = model.forward(X)
        residuals -= predictions
        residual_norm = np.linalg.norm(residuals, axis=0, keepdims=True)
        if np.all(residual_norm < residual_threshold):
            print("Residuals below threshold. Stopping further training.")
            break
        residuals /= residual_norm
```

# Model 3: Adam Optimizer Method

## Theory

*Adam optimizer leverages an adaptive learning rate optimization algorithm.*

## 1) Initalization of Parameters:

Additional parameters for moment estimates.

```
# Initialize Adam parameters
self.m_weights = {key: np.zeros_like(value) for key, value in self.weights.items()}
self.v_weights = {key: np.zeros_like(value) for key, value in self.weights.items()}
self.m_biases = {key: np.zeros_like(value) for key, value in self.biases.items()}
self.v_biases = {key: np.zeros_like(value) for key, value in self.biases.items()}
```

## 2, 3) Forward Propagation, Loss Computation

Same with Model 1

# Model 3: Adam Optimizer Method

## 4) Backward Propagation

Adopt Adam Optimization algorithm

```python
def adam_update(self, gradients, params, m_params, v_params):
    self.t += 1
    updated_params = {}
    for key in params.keys():
        m_params[key] = self.beta1 * m_params[key] + (1 - self.beta1) * gradients[key]
        v_params[key] = self.beta2 * v_params[key] + (1 - self.beta2) * (gradients[key] ** 2)
        m_hat = m_params[key] / (1 - self.beta1 ** self.t)
        v_hat = v_params[key] / (1 - self.beta2 ** self.t)
        updated_params[key] = params[key] - self.learning_rate * m_hat / (np.sqrt(v_hat) + self.epsilon)
    return updated_params, m_params, v_params


def backward(self, X, y_true, y_pred):
    n_samples = y_true.shape[0]
    dZ3 = y_pred - y_true
    dW3 = np.dot(self.A2.T, dZ3) / n_samples
    db3 = np.sum(dZ3, axis=0, keepdims=True) / n_samples
    dA2 = np.dot(dZ3, self.weights["W3"].T)
    dZ2 = dA2 * self.relu_derivative(self.Z2)
    dW2 = np.dot(self.A1.T, dZ2) / n_samples
    db2 = np.sum(dZ2, axis=0, keepdims=True) / n_samples
    dA1 = np.dot(dZ2, self.weights["W2"].T)
    dZ1 = dA1 * self.relu_derivative(self.Z1)
    dW1 = np.dot(X.T, dZ1) / n_samples
    db1 = np.sum(dZ1, axis=0, keepdims=True) / n_samples

    gradients = {"W1": dW1, "W2": dW2, "W3": dW3, "b1": db1, "b2": db2, "b3": db3}
    self.weights, self.m_weights, self.v_weights = self.adam_update(
        {key: gradients[key] for key in self.weights.keys()}, self.weights, self.m_weights, self.v_weights
    )
    self.biases, self.m_biases, self.v_biases = self.adam_update(
        {key: gradients[key] for key in self.biases.keys()}, self.biases, self.m_biases, self.v_biases
    )
```

### Theory

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

# Model 3: Adam Optimizer Method

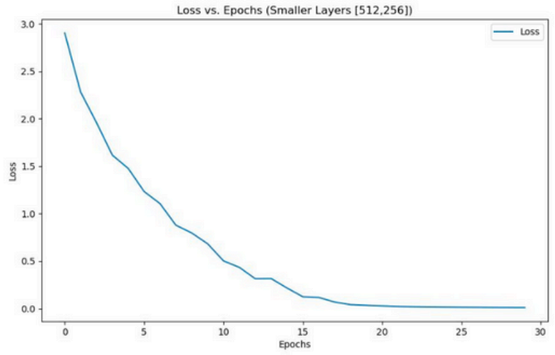
## 5) Training

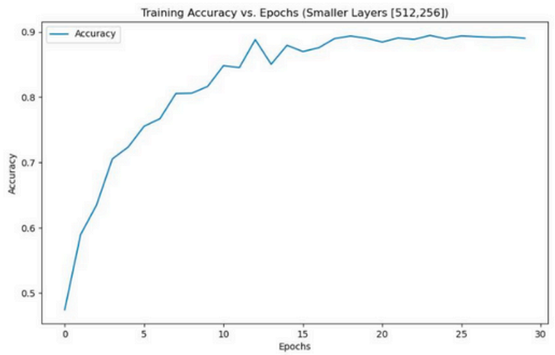
Incorporates the unique Adam update mechanism during the parameter update.

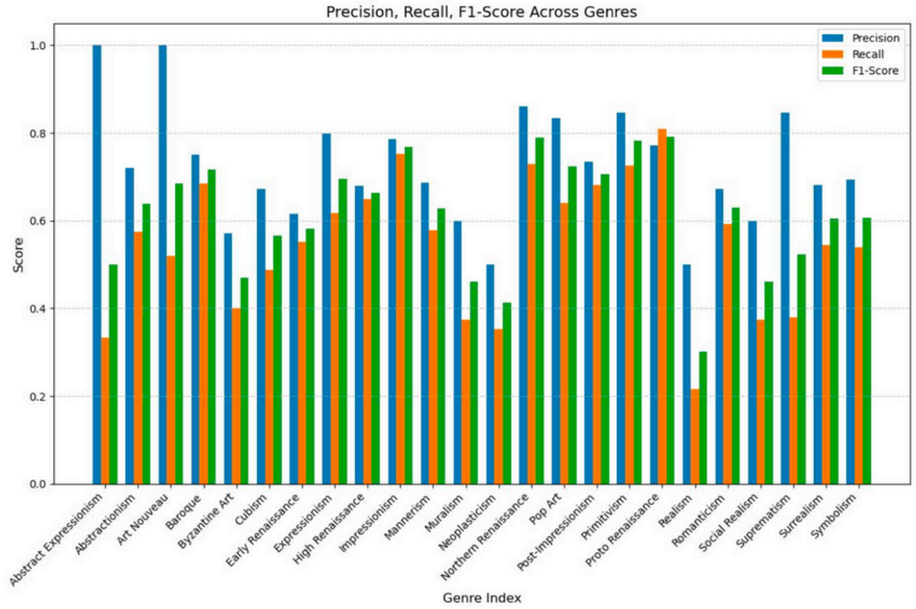
```
def train(self, X, y, epochs=20, batch_size=32):
    n_samples = X.shape[0]
    for epoch in range(epochs):
        indices = np.arange(n_samples)
        np.random.shuffle(indices)
        X = X[indices]
        y = y[indices]
        batch_losses = []
        batch_accuracies = []
        for i in range(0, n_samples, batch_size):
            X_batch = X[i:i + batch_size]
            y_batch = y[i:i + batch_size]
            y_pred = self.forward(X_batch)
            self.backward(X_batch, y_batch, y_pred)
```

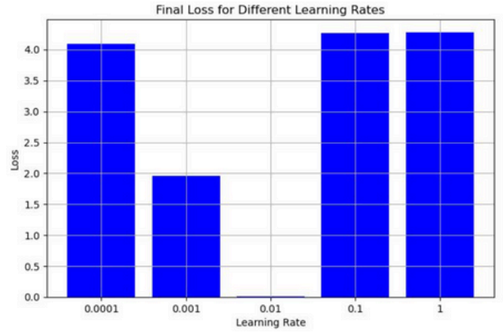
# Evaluation & Results

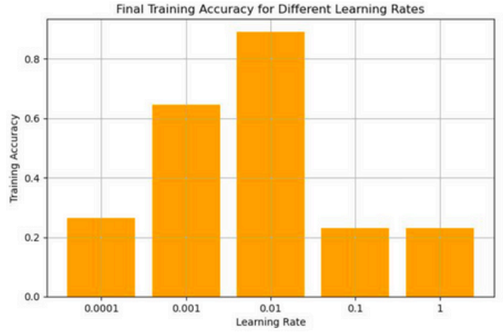
# Key results: Simple Neural Network

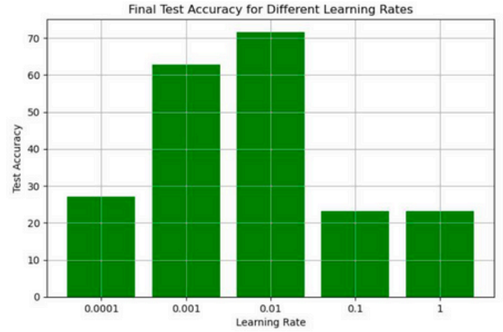


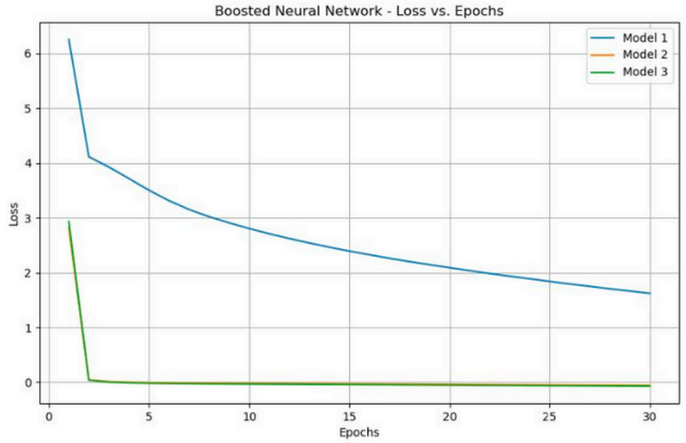



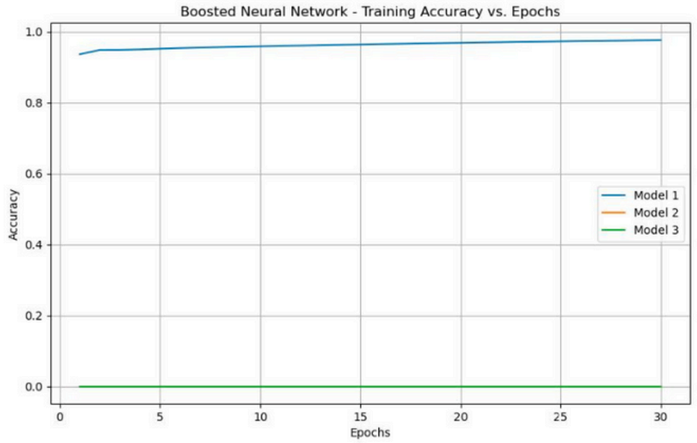



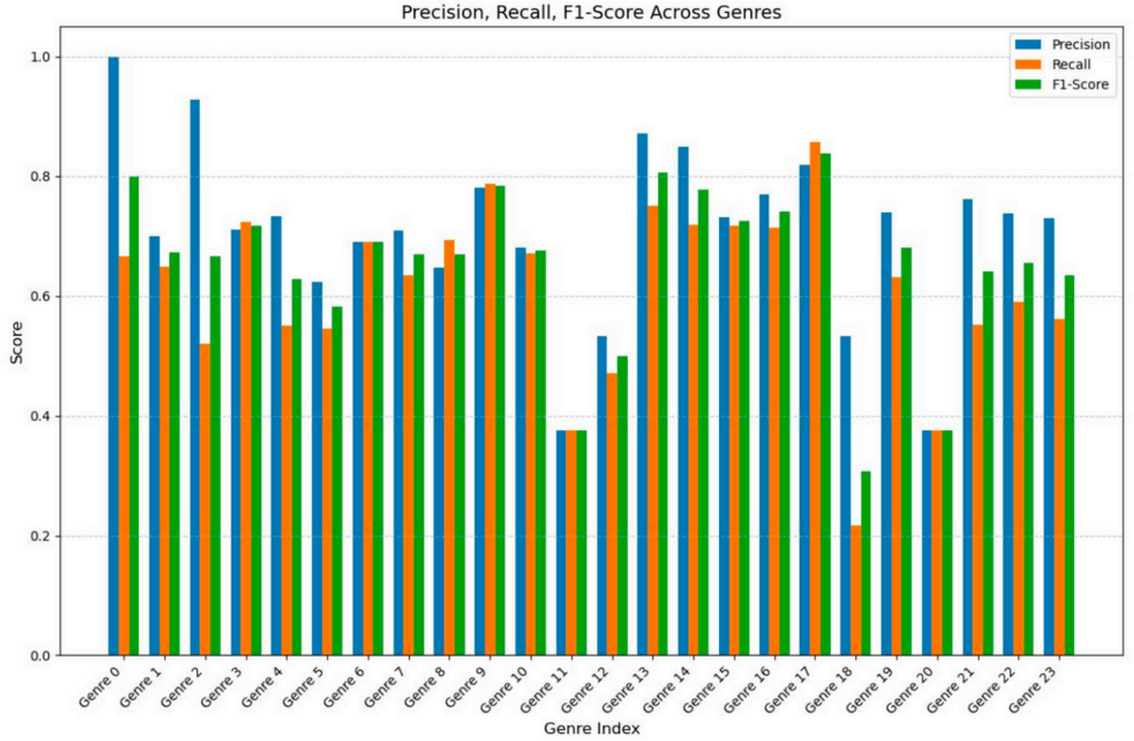




*The model evaluates multi-label classification by checking if predicted genres match any associated with the artist. For the architecture [512, 256] and a learning rate of 0.01, the SNN achieved a test accuracy of* ***73.13%.***

# Key results: Boosted Neural Network

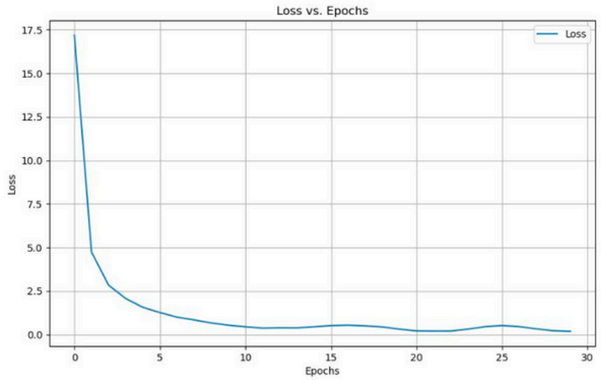


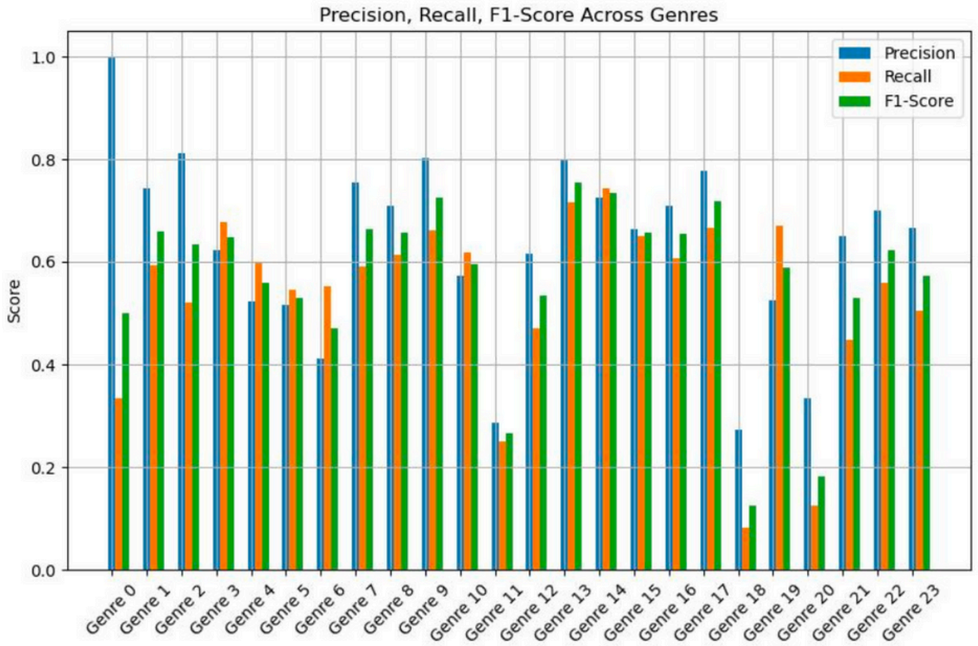



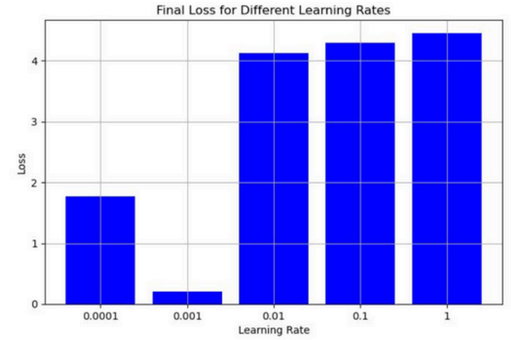

*BNN achieved a test accuracy of 73.25% using the same evaluation method as SNN.*

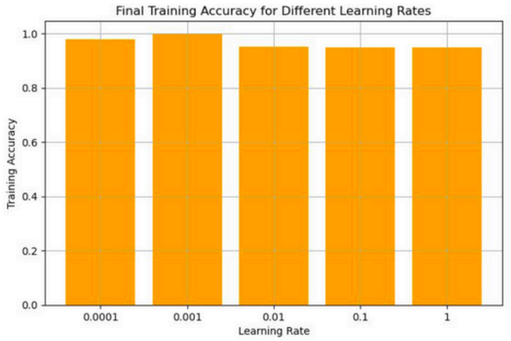
# Key results: Adam Optimizer Method

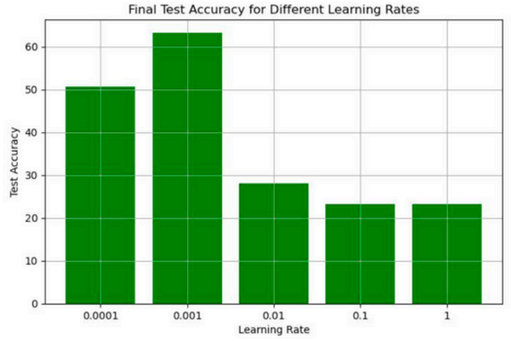






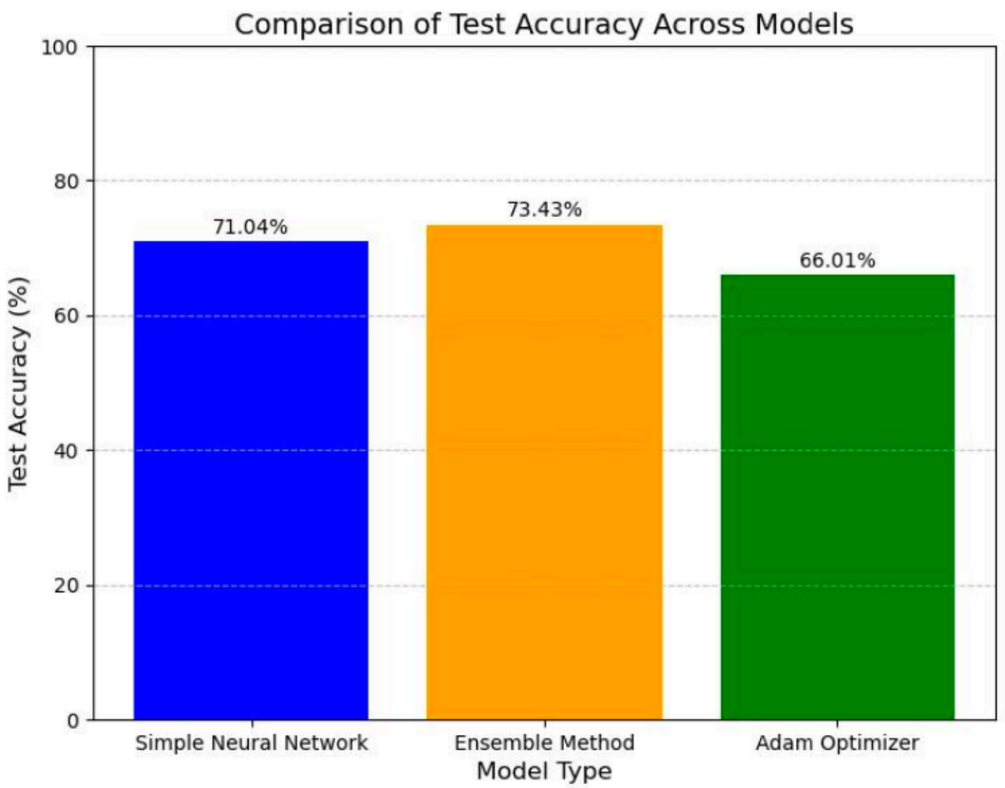



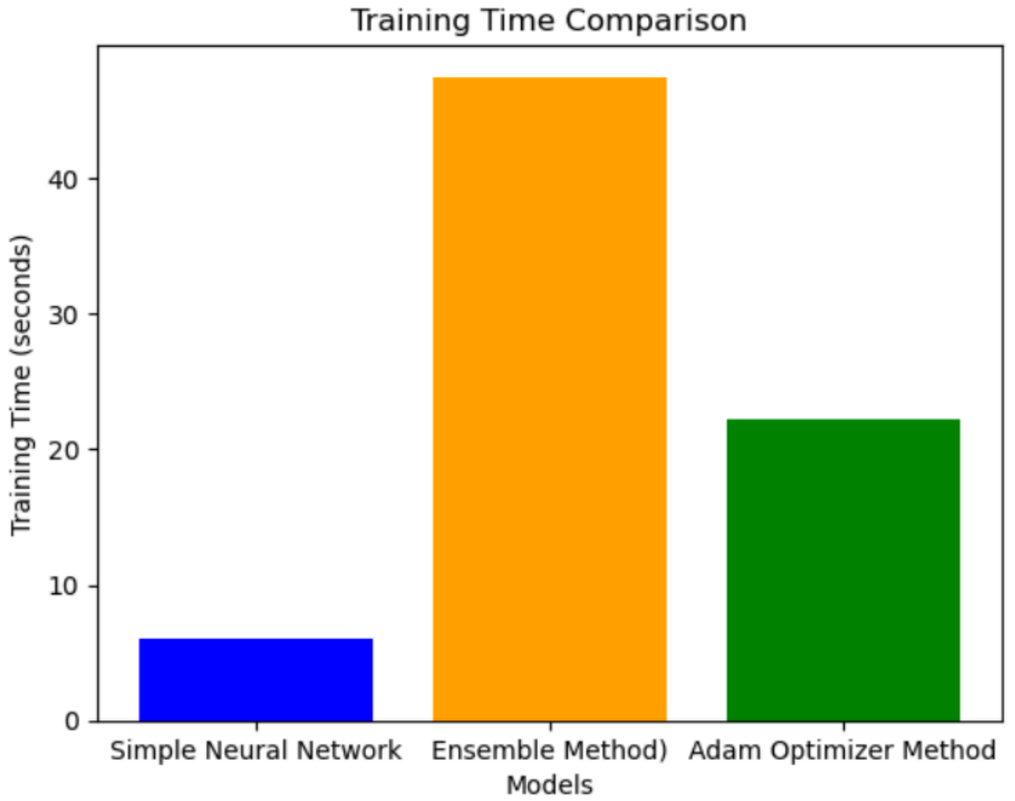




*Adam Optimizer Method achieved a test accuracy of 66.55% using the same evaluation method as SNN. The Adam model exhibits higher variance in precision and recall, indicating less consistent performance across genres. This reflects the Adam model's tendency to converge faster but with reduced accuracy compared to the SNN.*

# Comparison between all 3 models



# Limitations & Potential Improvements

# Limitations and improvements

## Imbalanced Dataset:

### Genre Imbalance Issue:

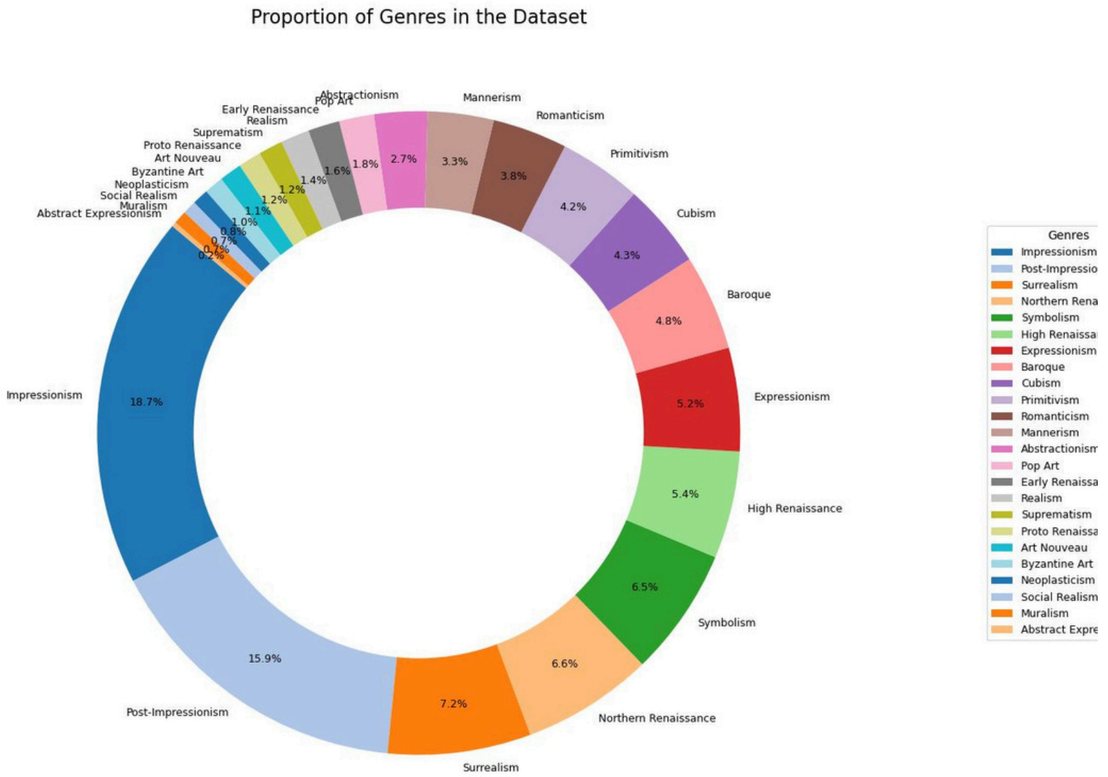
- Genres like Impressionism dominate the dataset, while others (e.g., Abstract Expressionism) are underrepresented.

### Impact on Performance:

- Imbalance leads to lower Precision, Recall, and F1-Scores for minority genres.

### Solutions to Improve Performance:

- Data Augmentation: Increase samples for minority genres by generating synthetic data.
- Balanced Sampling: Ensure models train on a more balanced subset of data to improve generalization across all genres.



# Limitations and improvements

## Imperfect Feature Extraction with ResNet50:

### Pretrained ResNet50 Limitations:

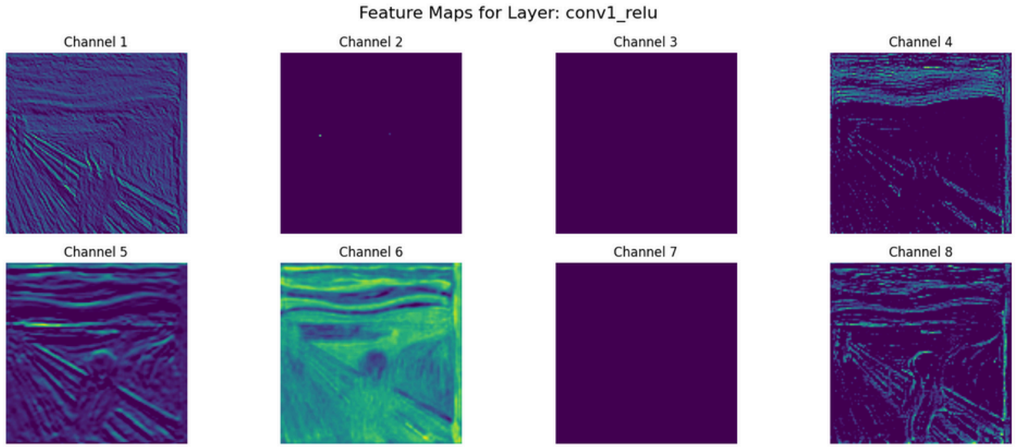
- ResNet50, trained on ImageNet, excels at general object recognition.
- It may struggle to capture the nuanced stylistic details needed for art genre classification.

### Feature Mismatch:

- Extracted features may not fully align with the specific requirements of art classification tasks.

### Performance Improvement Strategies:

- Fine-Tuning: Retrain ResNet50 on domain-specific art datasets to better capture stylistic features.
- Attention Mechanisms: Integrate attention techniques to help the model focus on important stylistic details.

# Conclusion

# Conclusion

**Research Implications:**

- Demonstrates the potential of machine learning for art genre classification.
- Shows that Boosted Neural Networks (BNN) provide the best accuracy (73.43%) through ensemble learning, while Simple Neural Networks (SNN) offer efficiency and practicality.

**Future Directions:**

- Address Dataset Imbalance: Use balanced sampling or data augmentation to improve performance for underrepresented genres.
- Domain-Specific Fine-Tuning: Adapt ResNet50 to art datasets for better feature extraction.

**Broader Impact:**

- Machine learning can support art education, digital curation, and the preservation of cultural heritage by enabling efficient and accurate art classification.