Art style Classification

ids: 1)322530080 2)318375318 3)209307727

Table of contents

1) *Introduction*

The goal of this project is to classify paintings into their genres. The dataset that was used for this task was the WikiArt dataset, which consists of ~80,000 paintings with 27 different art genres.
The project addresses a classification problem. Relevant features identified for this analysis include:

1. Colors present in the painting
2. The painting tools used
3. The shades within the artwork
4. The overall structure and composition
5. The artist of the painting
6. Objects depicted in the painting

The wikiArt data set:
The data set contains 80020 unique images from 1119 different artists in 27 styles.
The data set is organized in a folder structure where each folder represents the genres which serve as the labels of the data set.
The distribution of images across these genres is imbalanced, with some genres containing significantly more images than others.
An interesting aspect of this dataset is the subjectivity in genre classification. While each image is labeled with a single genre, some images exhibit characteristics that could place them into multiple genres. For example, an artwork labeled as "Impressionism" might also share visual elements with "Post-Impressionism." This inherent overlap highlights the complexity of artwork classification and suggests that models trained on this dataset might require sophisticated architectures strategies to handle such ambiguities effectively.

The genres that are classified by the model
1. Abstract Expressionism
2. Art Nouveau Modern
3. Baroque
4. Cubism
5. Expressionism
6. Impressionism
7. Naive Art Primitivism
8. Northern Renaissance
9. Post Impressionism
10. Realism
11. Rococo
12. Romanticism
13. Symbolism

2) *Pre-processing*

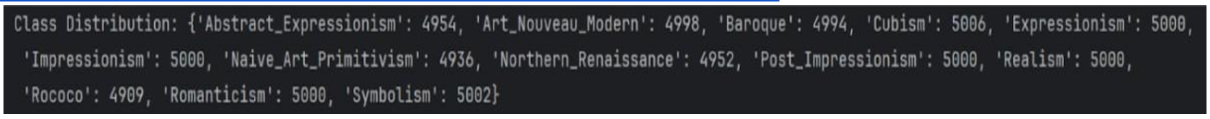
The dataset initially had a great variability in the number of paintings across genres. So, a preprocessing approach was implemented to ensure balanced class distributions suitable for the classification tasks.
Genres containing fewer than 2,000 images were excluded from the dataset due to insufficient sample size, which could lead to unreliable model training and evaluation.

For genres with image counts ranging between 2,000 and 5,000, data augmentation techniques were implemented to artificially increase the sample size while preserving the integrity of the original images.
Augmentation methods included geometric transformations, such as horizontal flipping and rotations (90°, 180°, and 270°), as well as photometric adjustments, including modifications to brightness and contrast.
These transformations were designed to overcome the variability without compromising the defining visual features of each genre.

Genres with more than 5,000 images were down sampled to a uniform size of 5,000 images per class.
This step was implemented to prevent overrepresentation of certain genres, which could otherwise bias the model toward classes with larger sample sizes.
The down sampling process was performed by randomly selecting a subset of images, ensuring that the resulting distribution remained representative of the original data.

Finally, all images were resized to a fixed dimension of 256x256 pixels to standardize input dimensions, facilitating efficient computation and compatibility with convolutional neural network architectures.

The image distribution after the preprocessing part:

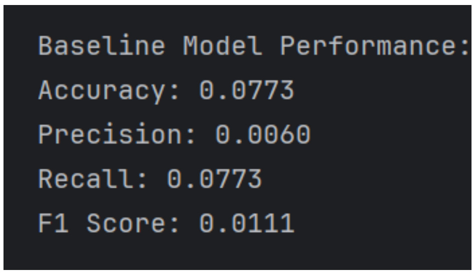


```
Class Distribution: {'Abstract_Expressionism': 4954, 'Art_Nouveau_Modern': 4998, 'Baroque': 4994, 'Cubism': 5006, 'Expressionism': 5000,
 'Impressionism': 5000, 'Naive_Art_Primitivism': 4936, 'Northern_Renaissance': 4952, 'Post_Impressionism': 5000, 'Realism': 5000,
 'Rococo': 4909, 'Romanticism': 5000, 'Symbolism': 5002}
```

3) ***Baseline model***

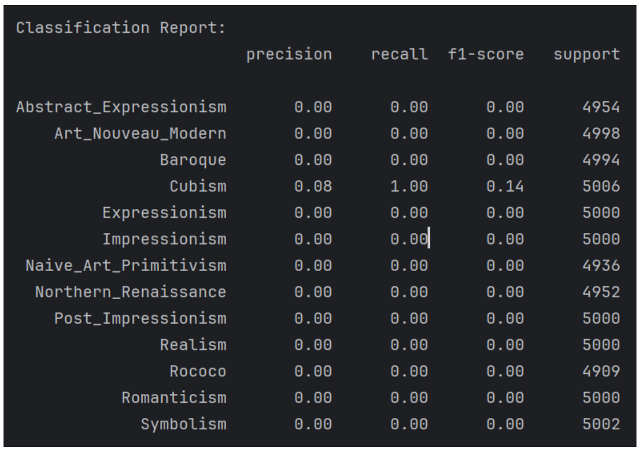
The baseline model predicts the majority class in the art style classification problem. This simplistic approach assumes that the class with the highest frequency in the dataset will always be chosen.
While straightforward, this model is expected to perform poorly in imbalanced datasets, achieving high accuracy only when the majority class dominates.
For evaluation, accuracy, precision, and recall are calculated to quantify the model's performance.

The evaluation of the baseline model using the sklearn. metrics classification functions:



```
Baseline Model Performance:
Accuracy: 0.0773
Precision: 0.0060
Recall: 0.0773
F1 Score: 0.0111
```

The classification report of the sklearn metrics_classificatio library across the classes we had:



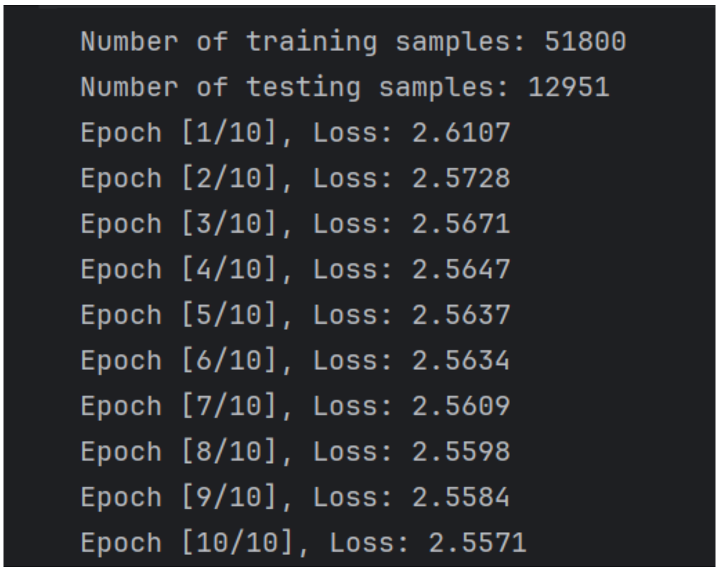
```
Classification Report:
                         precision    recall  f1-score   support

Abstract_Expressionism       0.00      0.00      0.00      4954
     Art_Nouveau_Modern       0.00      0.00      0.00      4998
                Baroque       0.00      0.00      0.00      4994
                 Cubism       0.08      1.00      0.14      5006
          Expressionism       0.00      0.00      0.00      5000
          Impressionism       0.00      0.00      0.00      5000
  Naive_Art_Primitivism       0.00      0.00      0.00      4936
   Northern_Renaissance       0.00      0.00      0.00      4952
     Post_Impressionism       0.00      0.00      0.00      5000
                Realism       0.00      0.00      0.00      5000
                 Rococo       0.00      0.00      0.00      4909
            Romanticism       0.00      0.00      0.00      5000
              Symbolism       0.00      0.00      0.00      5002
```

The results align with the expected values, as the model predicts the majority class for each image. Given that the model operates with 13 distinct classes, the expected accuracy is approximately 113\frac{1}{13}131 (~7%), consistent with the observed outcomes.

4) ***Basic Soft-Max model:***

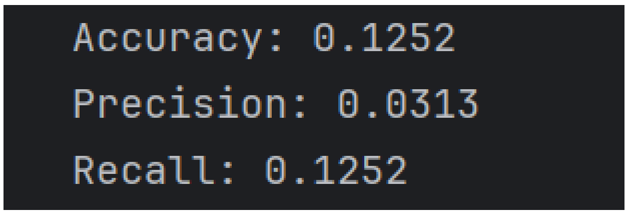
To address the multi-class classification problem, a SoftMax regression model was implemented as the first advanced approach.
SoftMax is an extension of logistic regression designed for multi-class classification, enabling predictions across multiple classes simultaneously.
Compared to the baseline model, which predicts the majority class, the SoftMax model achieved improved accuracy and recall due to its ability to account for class probabilities. However, the model's performance remains limited by its simplicity and lack of advanced optimization.
The results, as shown below, indicate that while the SoftMax model outperforms the baseline, there is significant room for improvement with more complex and better-trained models.

The training part of the soft max model - 10 epochs and the loss is decreased slowly:



```
Number of training samples: 51800
Number of testing samples: 12951
Epoch [1/10], Loss: 2.6107
Epoch [2/10], Loss: 2.5728
Epoch [3/10], Loss: 2.5671
Epoch [4/10], Loss: 2.5647
Epoch [5/10], Loss: 2.5637
Epoch [6/10], Loss: 2.5634
Epoch [7/10], Loss: 2.5609
Epoch [8/10], Loss: 2.5598
Epoch [9/10], Loss: 2.5584
Epoch [10/10], Loss: 2.5571
```

The evaluation of soft-max model):

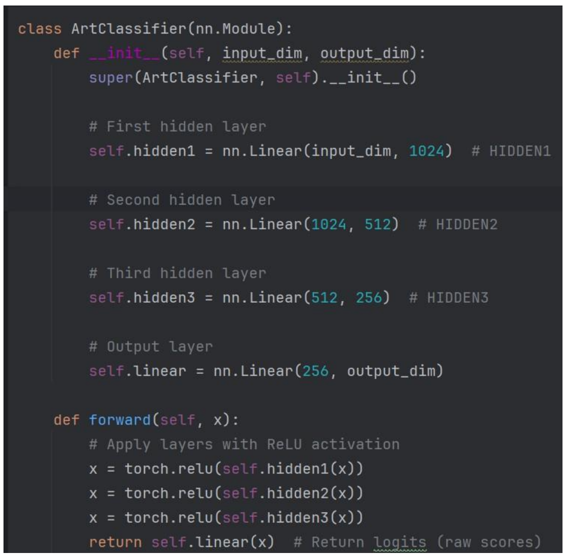


```
Accuracy: 0.1252
Precision: 0.0313
Recall: 0.1252
```

5) *Fully connected NN:*

This section presents the implementation of a neural network designed for multi-class art style classification.
The first model architecture that was tried is shown as three fully connected hidden layers.
The network reduces the dimensionality through the layers (1024 → 512 → 256 neurons), allowing a step by step feature extraction.
Each hidden layer is followed by a ReLU activation function, for non-linearity.
The input layer takes flattened RGB images (64×64×3 = 12,288 dimensions), while the output layer produces logits corresponding to the distinct art style categories.
Model architecture more specifically:

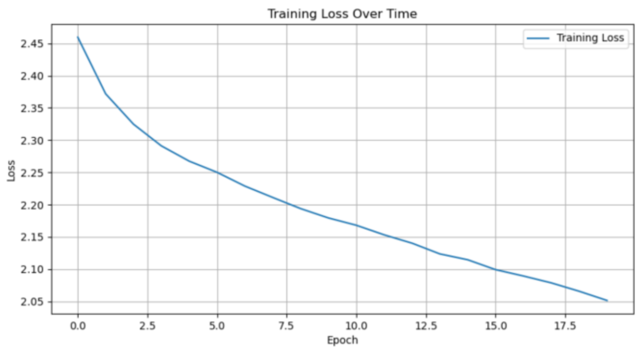
- Input Layer: 12,288 dimensions (flattened 64×64×3 RGB images)
- Hidden Layer 1: 1024 neurons with ReLU activation
- Hidden Layer 2: 512 neurons with ReLU activation
- Hidden Layer 3: 256 neurons with ReLU activation
- Output Layer: 13 neurons (corresponding to art style classes)



```python
class ArtClassifier(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(ArtClassifier, self).__init__()

        # First hidden layer
        self.hidden1 = nn.Linear(input_dim, 1024)  # HIDDEN1

        # Second hidden layer
        self.hidden2 = nn.Linear(1024, 512)  # HIDDEN2

        # Third hidden layer
        self.hidden3 = nn.Linear(512, 256)  # HIDDEN3

        # Output layer
        self.linear = nn.Linear(256, output_dim)

    def forward(self, x):
        # Apply layers with ReLU activation
        x = torch.relu(self.hidden1(x))
        x = torch.relu(self.hidden2(x))
        x = torch.relu(self.hidden3(x))
        return self.linear(x)  # Return logits (raw scores)
```

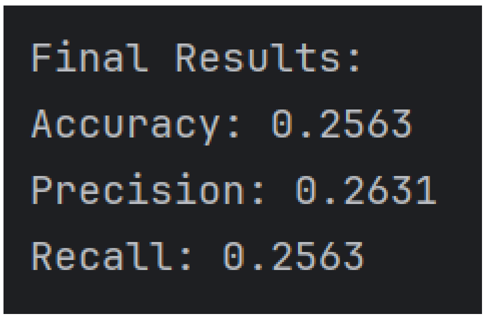
This implementation serves as an initial model to which more layers will be added in the next steps.

After running this model for 20 epochs this plot was received:



The plot shows the training loss decreasing steadily over time as the number of epochs increases. This indicates that the model is learning and improving its performance during training, as it minimizes the error between predictions and true labels.

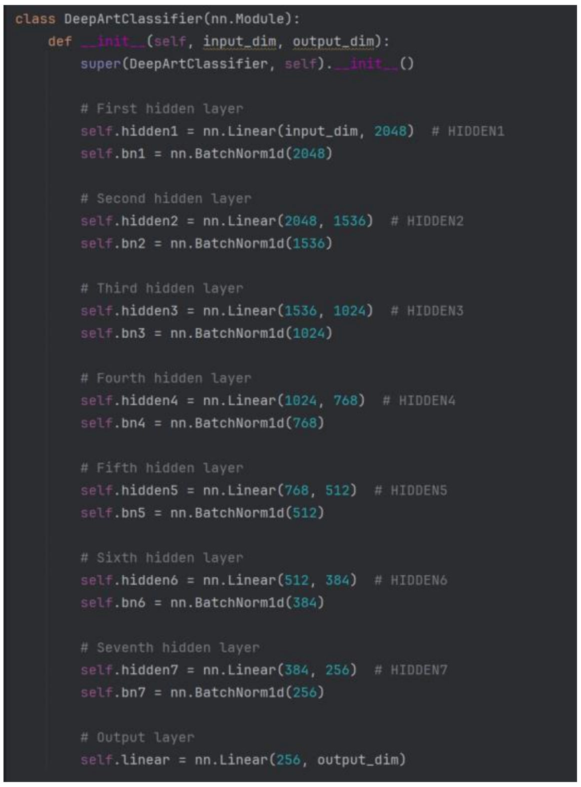
The evaluation of the test results is as follows:



```
Final Results:
Accuracy: 0.2563
Precision: 0.2631
Recall: 0.2563
```

Since the accuracy, although better than the models mentioned above, is still relatively low, a deeper network was implemented with 7 hidden layers instead of 3, and a higher image resolution was used (compared to 64x64 in the previous model).

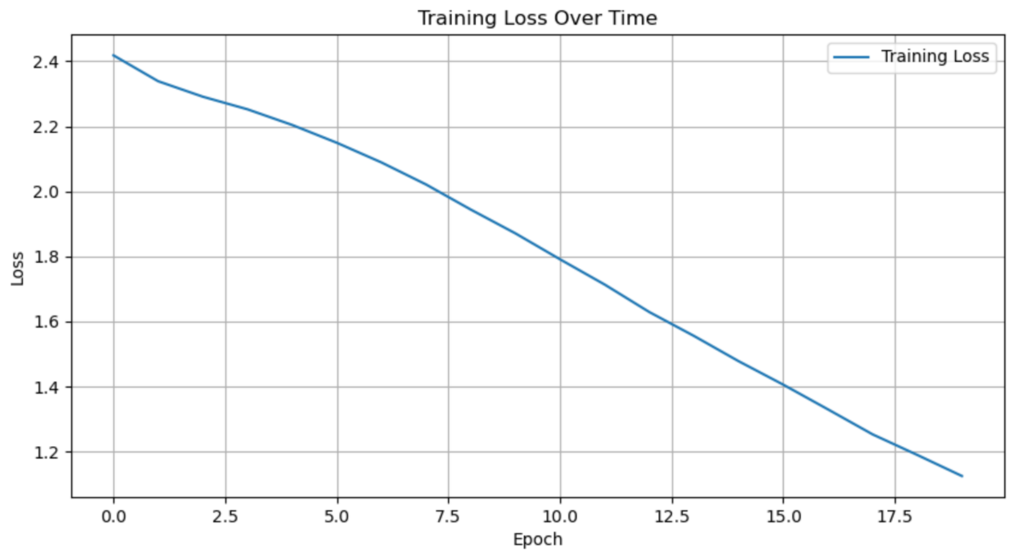
The architecture of the model that was implemented at this time is as follows:



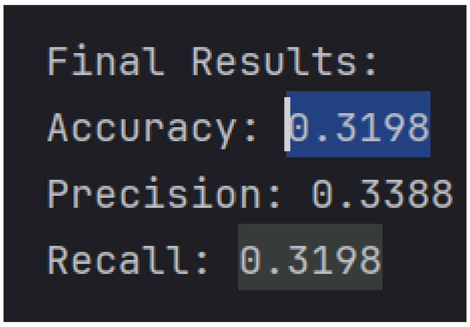
```
class DeepArtClassifier(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DeepArtClassifier, self).__init__()

        # First hidden layer
        self.hidden1 = nn.Linear(input_dim, 2048)  # HIDDEN1
        self.bn1 = nn.BatchNorm1d(2048)

        # Second hidden layer
        self.hidden2 = nn.Linear(2048, 1536)  # HIDDEN2
        self.bn2 = nn.BatchNorm1d(1536)

        # Third hidden layer
        self.hidden3 = nn.Linear(1536, 1024)  # HIDDEN3
        self.bn3 = nn.BatchNorm1d(1024)

        # Fourth hidden layer
        self.hidden4 = nn.Linear(1024, 768)  # HIDDEN4
        self.bn4 = nn.BatchNorm1d(768)

        # Fifth hidden layer
        self.hidden5 = nn.Linear(768, 512)  # HIDDEN5
        self.bn5 = nn.BatchNorm1d(512)

        # Sixth hidden layer
        self.hidden6 = nn.Linear(512, 384)  # HIDDEN6
        self.bn6 = nn.BatchNorm1d(384)

        # Seventh hidden layer
        self.hidden7 = nn.Linear(384, 256)  # HIDDEN7
        self.bn7 = nn.BatchNorm1d(256)

        # Output layer
        self.linear = nn.Linear(256, output_dim)
```

Batch normalization was added to this deeper network because of the increased number of layers (from 3 to 7), the network becomes more prone to training instability, so batch normalization helps stabilize the training process and allows better gradient flow through these deeper layers.

After running this model for 20 epochs we received this plot:

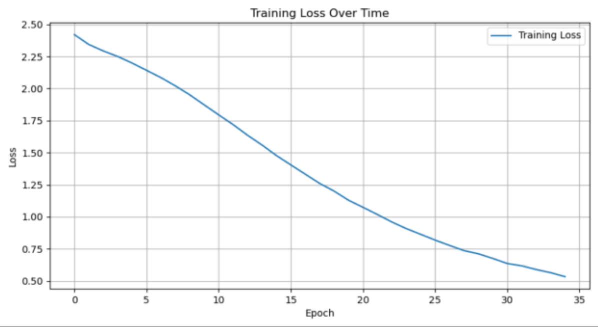


The loss consistently decreases, indicating the model's predictions align more closely with the actual labels during training. This model demonstrated slightly improved performance compared to the previous one, achieving an accuracy of 0.3198 versus 0.256 from the earlier version
The evaluation of the test part of the data is shown as follows:

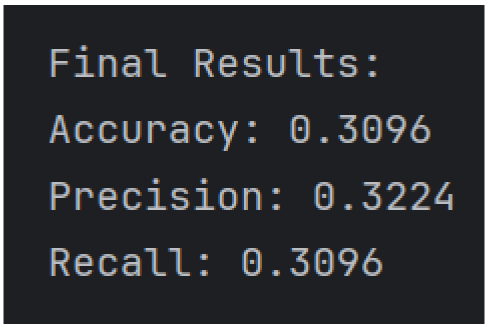


```
Final Results:
Accuracy: 0.3198
Precision: 0.3388
Recall: 0.3198
```

Because the loss seemed to get even lower the model was trained again with 35 epochs.
After running this model for 35 epochs the plot that was received is as follows:

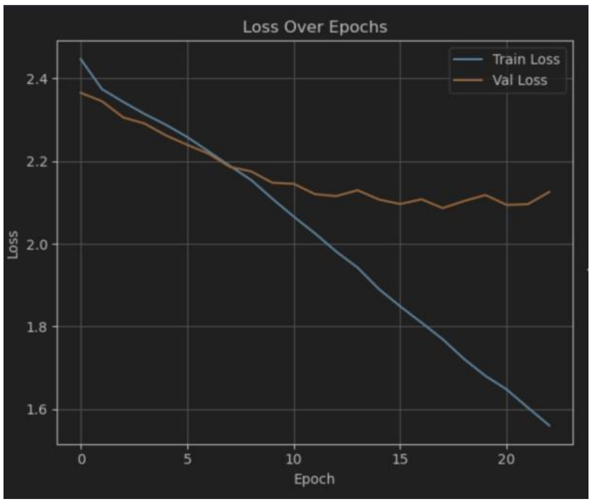


The evaluation of the test is as follows:

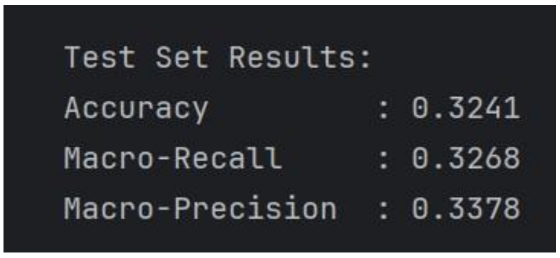


```
Final Results:
Accuracy: 0.3096
Precision: 0.3224
Recall: 0.3096
```

The accuracy seemed to get lower, so an early stopping was added with 5 as a patients parameter.
after running this model with an early stopping this plot was received:



The model was stopped after 23 epochs implying the overfitting mentioned above and as expected the accuracy of the model and the other metrics was improved:



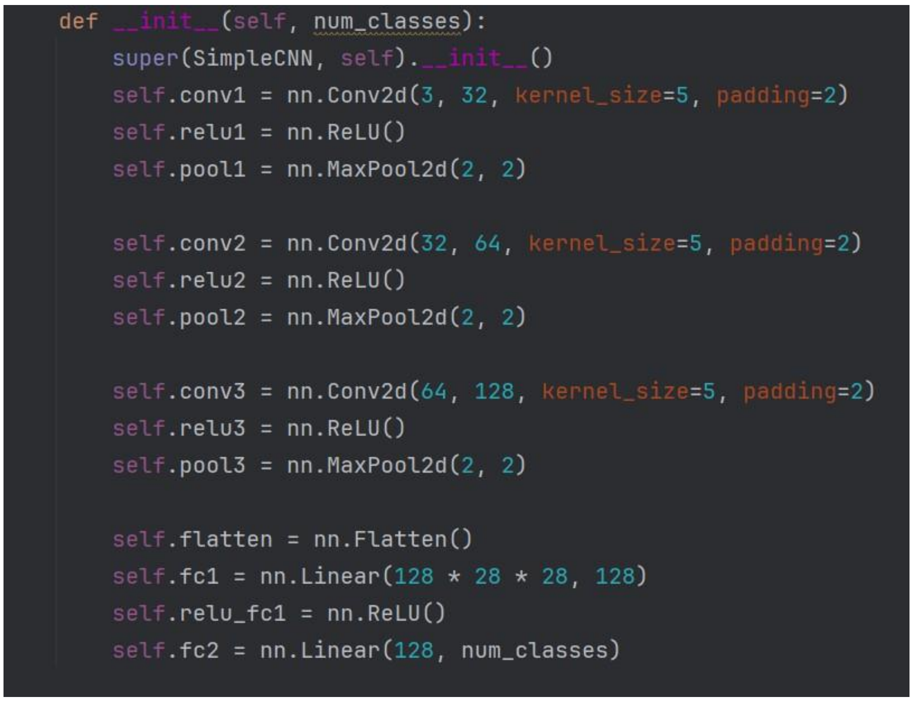
```
Test Set Results:
Accuracy          : 0.3241
Macro-Recall      : 0.3268
Macro-Precision   : 0.3378
```

Which represents the best accuracy received so far.

6)More complex models

6.1) convolutional neural network:

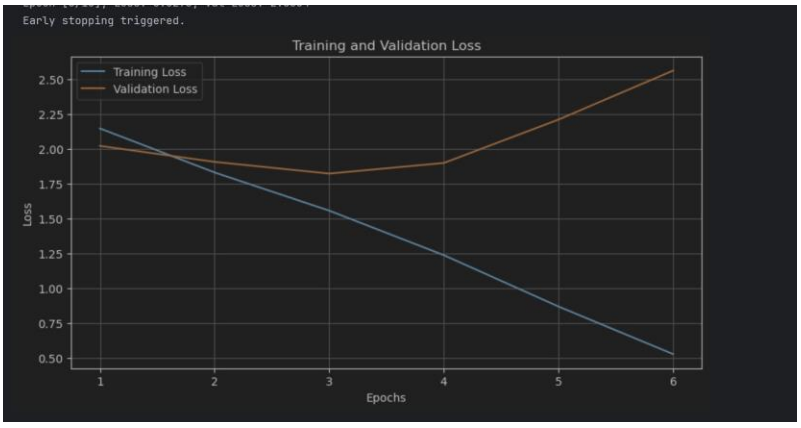
After evaluating the fully connected neural network, a standard convolutional neural network (CNN) was implemented to detect the art style of the images. Building on concepts from coursework, the model's core strength lies in its ability to identify optimal filters that capture distinctive patterns in the images.
The initial model architecture was structured as follows:



```
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=5, padding=2)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=5, padding=2)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2, 2)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=5, padding=2)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(2, 2)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 28 * 28, 128)
        self.relu_fc1 = nn.ReLU()
        self.fc2 = nn.Linear(128, num_classes)
```

The model was trained using the Adam optimizer and Cross Entropy Loss to monitor learning performance.

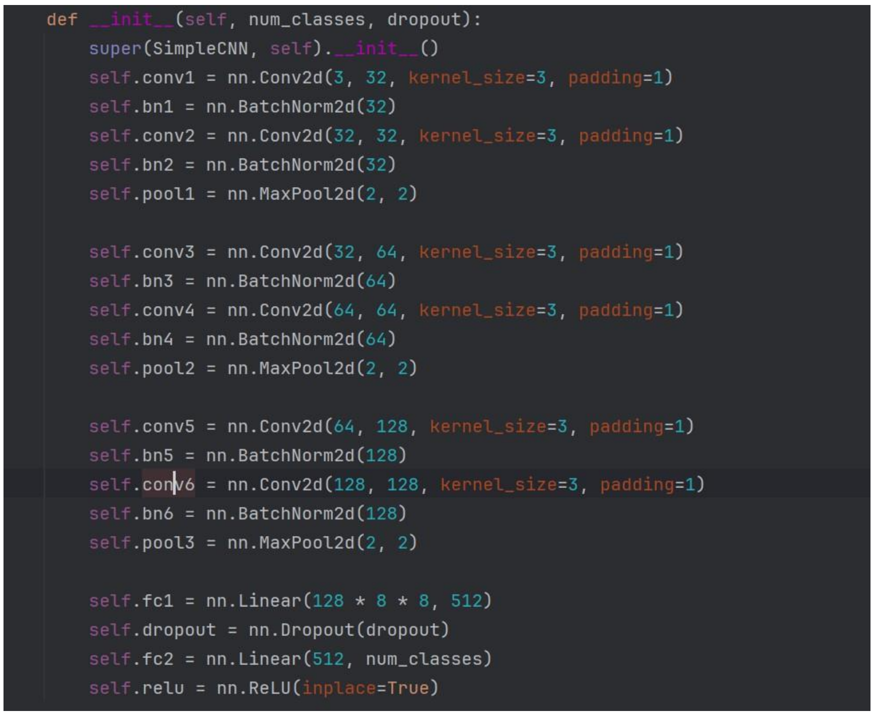
Testing the network revealed the following plot:



```
Early stopping triggered.
```

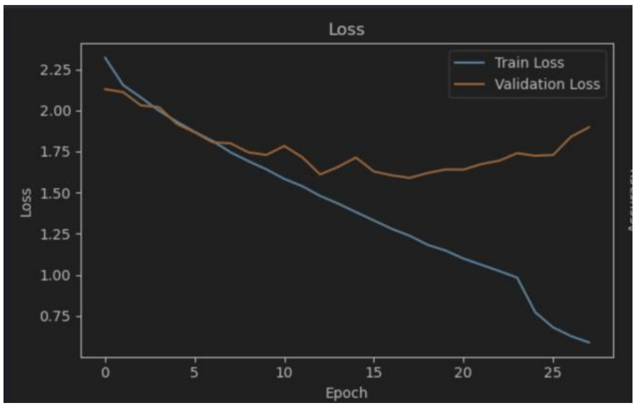
The growing gap between the training and validation losses, reflected in a decreasing training loss and increasing validation loss, indicates signs of overfitting. To address this issue, the model's depth was increased by adding additional convolutional layers. An early stopping mechanism was also implemented to prevent overfitting and halt training once improvements ceased.

The model that was tested next had 6 convolutional layers and was designed as follows: The architecture of the next model tested, which used 6 convolutional layers, is detailed below:

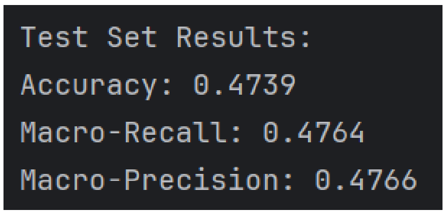


```
def __init__(self, num_classes, dropout):
    super(SimpleCNN, self).__init__()
    self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
    self.bn1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
    self.bn2 = nn.BatchNorm2d(32)
    self.pool1 = nn.MaxPool2d(2, 2)

    self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
    self.bn3 = nn.BatchNorm2d(64)
    self.conv4 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
    self.bn4 = nn.BatchNorm2d(64)
    self.pool2 = nn.MaxPool2d(2, 2)

    self.conv5 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
    self.bn5 = nn.BatchNorm2d(128)
    self.conv6 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
    self.bn6 = nn.BatchNorm2d(128)
    self.pool3 = nn.MaxPool2d(2, 2)

    self.fc1 = nn.Linear(128 * 8 * 8, 512)
    self.dropout = nn.Dropout(dropout)
    self.fc2 = nn.Linear(512, num_classes)
    self.relu = nn.ReLU(inplace=True)
```

As mentioned, an early stopping was added too with patience value of 10. After training this model this plot was received:

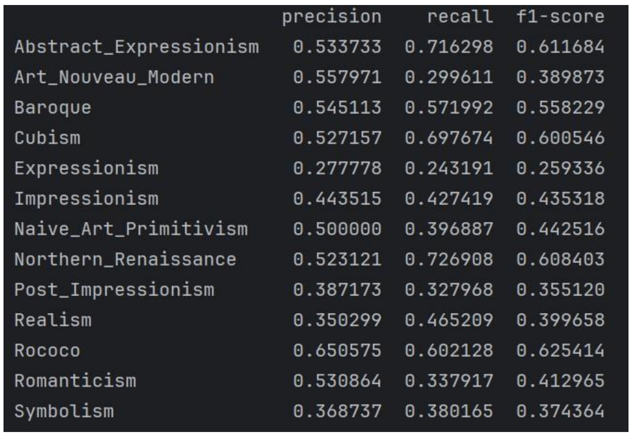


The plot highlights an improvement in the overfitting issue observed earlier. After testing the model, the following evaluation results were obtained:



```
Test Set Results:
Accuracy: 0.4739
Macro-Recall: 0.4764
Macro-Precision: 0.4766
```

This evaluation results highlight the CNN architecture's performance improvement over the fully connected neural network and all previously tested models. The term "macro" in the image refers to an overall evaluation across all classes. Additionally, separate evaluations were conducted for each class to better understand which classes are easier for the model to classify:

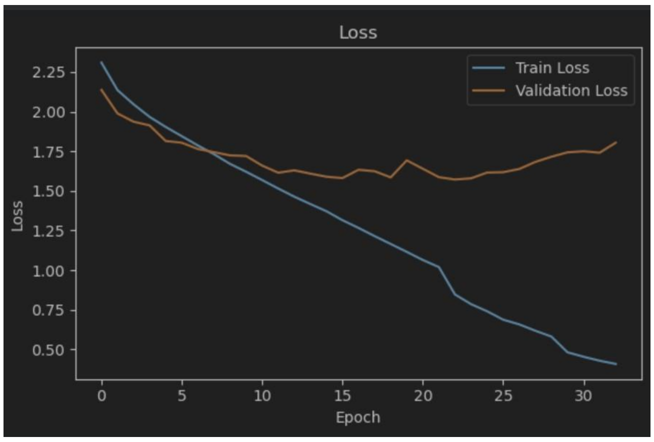


```
                        precision    recall  f1-score
Abstract_Expressionism   0.533733  0.716298  0.611684
Art_Nouveau_Modern       0.557971  0.299611  0.389873
Baroque                  0.545113  0.571992  0.558229
Cubism                   0.527157  0.697674  0.600546
Expressionism            0.277778  0.243191  0.259336
Impressionism            0.443515  0.427419  0.435318
Naive_Art_Primitivism    0.500000  0.396887  0.442516
Northern_Renaissance     0.523121  0.726908  0.608403
Post_Impressionism       0.387173  0.327968  0.355120
Realism                  0.350299  0.465209  0.399658
Rococo                   0.650575  0.602128  0.625414
Romanticism              0.530864  0.337917  0.412965
Symbolism                0.368737  0.380165  0.374364
```

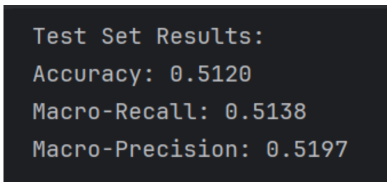
This table shows that the model performs well on some classes like "Cubism" and "Rococo," while struggling with others such as "Expressionism" and "Symbolism."

To improve the model's accuracy and reduce overfitting, the patience setting was lowered from 10 to 5. Also, the number of epochs that the code could perform was raised to 100 instead of 50.

With these modifications the plot of the losses is shown as follows:

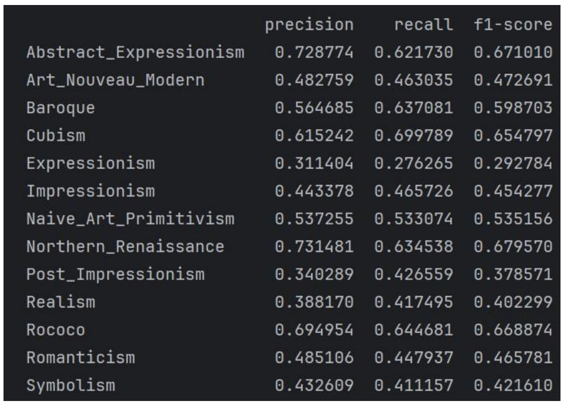


And an improvement was made in the testing part:



```
Test Set Results:
Accuracy: 0.5120
Macro-Recall: 0.5138
Macro-Precision: 0.5197
```

The micro evaluation of each class:



```
                         precision    recall  f1-score
Abstract_Expressionism    0.728774  0.621730  0.671010
Art_Nouveau_Modern        0.482759  0.463035  0.472691
Baroque                   0.564685  0.637081  0.598703
Cubism                    0.615242  0.699789  0.654797
Expressionism             0.311404  0.276265  0.292784
Impressionism             0.443378  0.465726  0.454277
Naive_Art_Primitivism     0.537255  0.533074  0.535156
Northern_Renaissance      0.731481  0.634538  0.679570
Post_Impressionism        0.340289  0.426559  0.378571
Realism                   0.388170  0.417495  0.402299
Rococo                    0.694954  0.644681  0.668874
Romanticism               0.485106  0.447937  0.465781
Symbolism                 0.432609  0.411157  0.421610
```

Comparing this table to the earlier one, it is evident that the model's performance has improved. Precision, recall, and F1-scores for most classes are higher, indicating better classification across multiple categories. For example, classes like "Abstract_Expressionism" and "Cubism" show improvements in F1-scores, reflecting better balancing between precision and recall. Additionally, previously struggling classes such as "Expressionism" and "Post_Impressionism" also demonstrate slight progress, even though they are still harder to classify compared to other classes. These improvements suggest that the adjustments made to the model were effective.

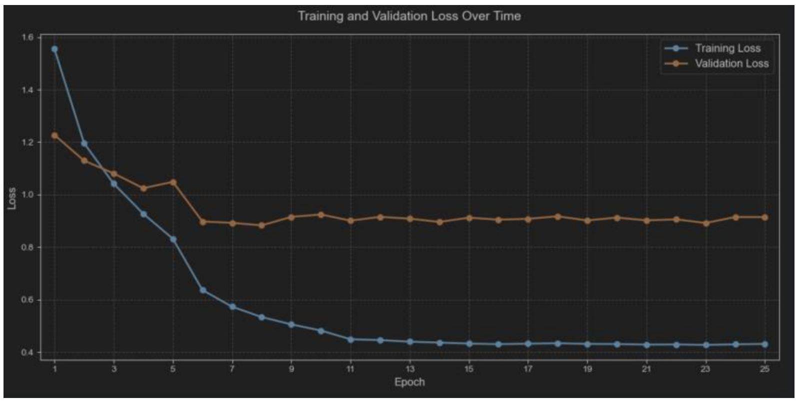
6.2) Using a pre-trained models for feature extractions:

6.2.1)Resnet network:

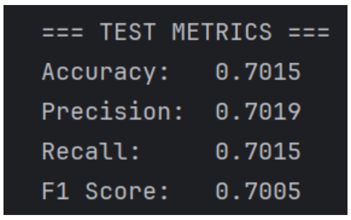
This section presents the implementation of a convolutional neural network (CNN) based on the ResNet18 architecture.
ResNet is a deep neural network designed to handle very deep layers without losing information.
It uses skip connections, which let data skip some layers, making training easier and more effective. ResNet18 has 18 layers, including convolutional layers, batch normalization, and ReLU activations.
It is built with small blocks that learn features while keeping important details through shortcuts.
The final layer is fully connected and outputs predictions after reducing features with global average pooling. This design helps it learn complex patterns while staying efficient.
The model that was implemented for the art style classification problem uses the ResNet18 network, which is fine-tuned by replacing the fully connected output layer to match the number of target classes.
The last fully connected layer of the model is replaced with:

1. A dropout layer with a rate of 0.5 to reduce overfitting.
2. A fully connected layer that outputs logits equal to the number of target classes.

The model was trained using the cross-entropy loss function.
Optimization is performed with the Adam optimizer, initialized with a learning rate of 0.0001.
A learning rate scheduler reduces the learning rate by a factor of 0.1 every 5 epochs.
Training proceeds for 25 epochs, with performance monitored on the validation set after each epoch.
The loss plot that was received after running this model is as follows:

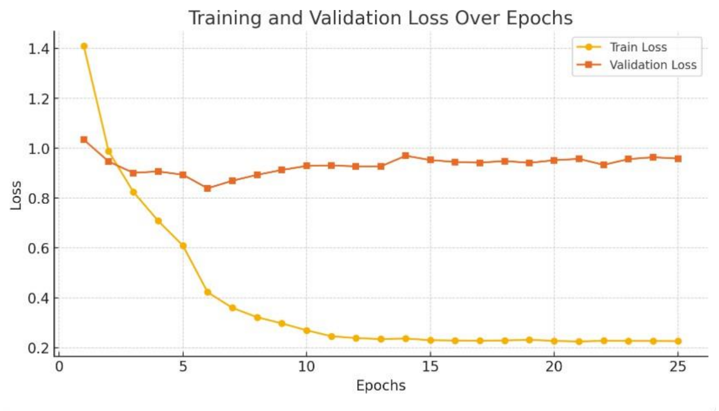


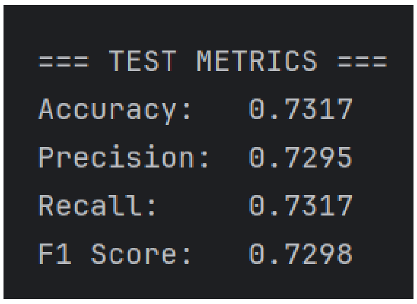
The plot shows both training and validation loss trajectories across 25 epochs of model training. The training loss (blue line) demonstrates a consistent decrease from 1.55 to 0.43, particularly sharp in the first 6 epochs, indicating effective model learning. Meanwhile, the validation loss (orange line) initially drops but plateaus around 0.90 after epoch 6, with the growing gap between the two lines suggesting potential overfitting in the later stages of training. Despite this overfitting pattern, this model configuration achieved our highest test accuracy until this part
with huge improvement from the regular CNN architecture as shown here:



```
=== TEST METRICS ===
Accuracy:   0.7015
Precision:  0.7019
Recall:     0.7015
F1 Score:   0.7005
```

After using the resent18 for the art genre classification problem and receiving the result, which was the highest until this point, the next attempt that was tried for our data was a more robust model - resnet50
this model uses 50 layers with same techniques as described above for the resenet18 model.
An assumption was made that using a deeper network may increase the accuracy of the model and as follows these plots and evaluations were received:





```
=== TEST METRICS ===
Accuracy:   0.7317
Precision:  0.7295
Recall:     0.7317
F1 Score:   0.7298
```

This accuracy represents the beset accuracy that was received for this classification problem and shows a high improvement over the initial models that was designed.

6.2.2) EfficientNet:

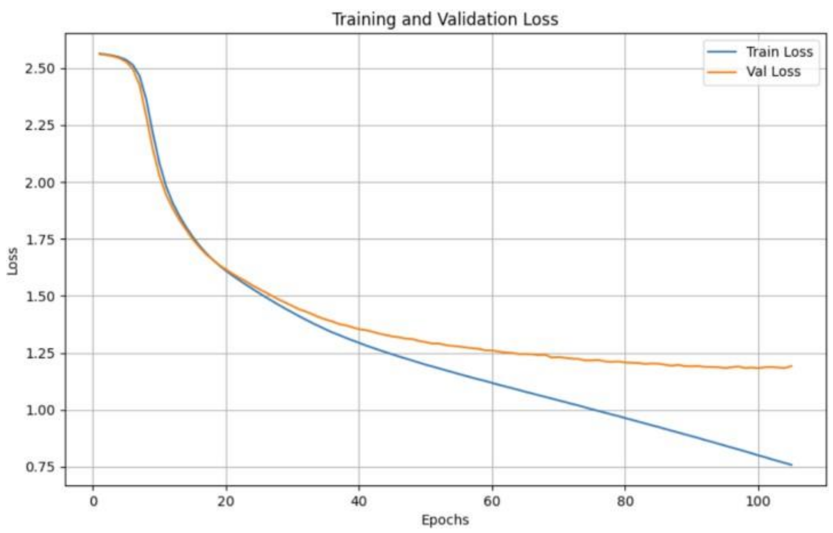
This section presents the implementation of a feedforward neural network based on extracted EfficientNet-B0 features. This network is designed to efficiently process pre-extracted features while maintaining good classification performance. It uses a deep architecture with multiple fully connected layers and ReLU activations that help learn complex patterns while preventing vanishing gradients. The network processes the 1280-dimensional feature vectors from EfficientNet-B0 through multiple layers of transformations before making final predictions.

The model that was implemented for the classification problem uses a deep neural architecture, which is structured with multiple fully connected layers:

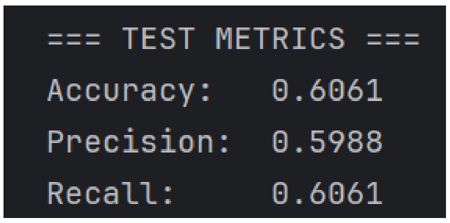
1. An input layer that handles the 1280-dimensional EfficientNet-B0 features
2. Two hidden layers with 256 neurons each
3. A third hidden layer with 128 neurons
4. A final output layer matching the number of target classes

The model was trained using the cross-entropy loss function. Optimization is performed with the SGD optimizer, initialized with a learning rate of 1e-4, momentum of 0.9, and weight decay of 1e-5. Training proceeded for 105 epochs, with performance monitored on the validation set after each epoch and early stopping patience of 5 epochs.

The loss plot that was received after running this model shows:



The plot shows both training and validation loss trajectories across 105 epochs of model training. The training loss (blue line) demonstrates a consistent decrease from 2.5 to 0.75, with particularly rapid improvement in the first 20 epochs, indicating effective model learning. The validation loss (orange line) follows a similar pattern, dropping significantly in the early epochs and stabilizing around 1.2 after epoch 60. The model achieved moderate classification performance with test accuracy of 0.6061, precision of 0.5988, and recall of 0.6061.



```
=== TEST METRICS ===
Accuracy:   0.6061
Precision:  0.5988
Recall:     0.6061
```

6.3) Hyper-network:

HyperNetwork with ResNet50 Feature Extractor:

As the process of learning and knowing more advanced architecture came forward, the idea of hypernetwork model was suggested due to its novel strategy: learning the weights in a different NN and feeding those weights and biases to the base NN. Moreover, this idea was never tested on the wikiart dataset, and it was an opportunity to do something that had never been done.
After learning and reading more about hypernetwork , it was time to implement and test this idea.
This section presents the implementation of dynamic neural network architecture that combines a ResNet50 feature extractor with a hypernetwork.
The hypernetwork generates weights dynamically based on a learnable latent vector, allowing for adaptive feature processing.

The model architecture consists of three main components:

1. Feature Extractor:
   - Uses pretrained ResNet50
   - Extracts 2048-dimensional feature vectors from input images
2. HyperNetwork Component:
   - Takes a 64-dimensional learnable latent vector as input
   - Contains an embedding network with two fully connected layers (64→128→256)
   - Generates weights (2048×512) and biases (512) dynamically
   - Uses ReLU activations between layers
   - The generated weights are used to transform the ResNet features
3. Classification Head:
   - Processes the transformed 512-dimensional features
   - Contains two fully connected layers (512→256→num_classes)
   - Includes dropout (0.5) for regularization
   - ReLU activation after the first layer

The model was trained using the following configuration:

- Loss Function: Cross-Entropy Loss
- Optimizer: Adam with two parameter groups:
- Base parameters: learning rate of 1e-4
- Latent vector: learning rate of 1e-3
- Learning Rate Scheduler: ReduceLROnPlateau with patience of 3 epochs
- Training proceeded with early stopping based on validation performance

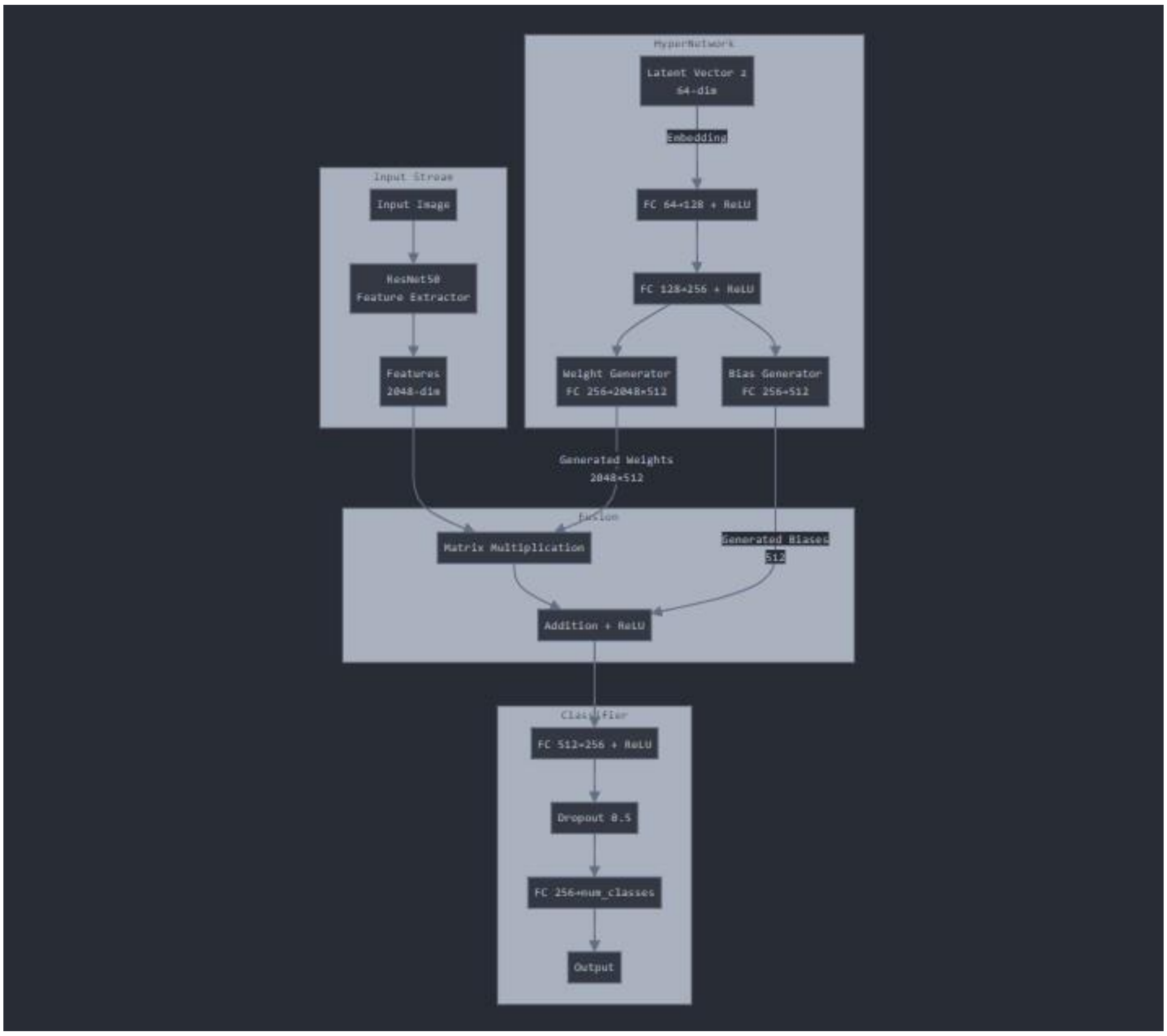
Training Results:

- Training spanned 11 complete epochs.
- Best validation accuracy: 72.39% (achieved at epoch 10)
- Final test metrics:
- Accuracy: 71.69%
- Precision: 0.7167
- Recall: 0.7169

The training progression shows:

- Training loss decreased significantly from 1.3295 to 0.0189
- Model showed consistent improvement in validation accuracy up to epoch 10
- Performance plateaued around epoch 11, with validation metrics slightly declining

The architecture demonstrates competitive performance on the art style classification task, achieving over 71% accuracy on the test set. The hypernetwork component provides an interesting approach to dynamic weight generation, though the increasing validation loss suggests room for improved regularization strategies.



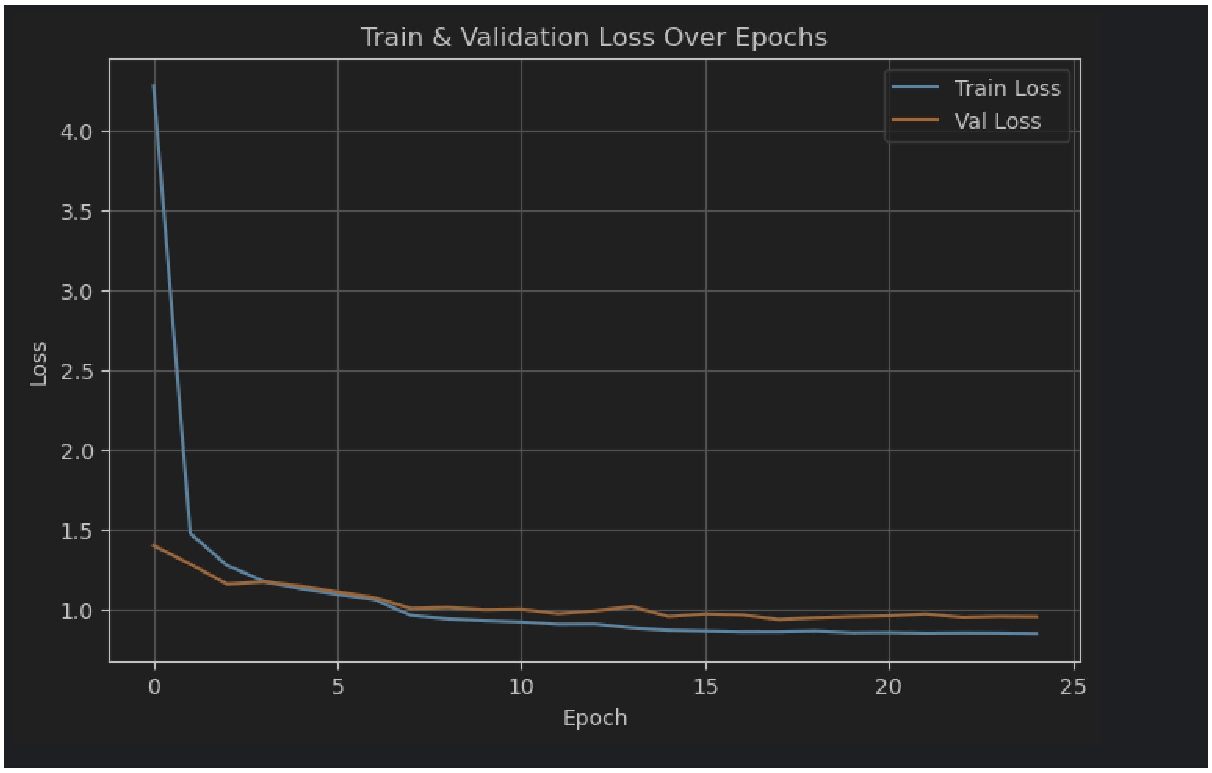
7)*Another attempt - 5 classes classification:*

After conducting experiments with the models mentioned above to classify 13 distinct art genres, a decision was made to simplify the task by testing the models on a subset of 5 art genres. The selected genres for this focused evaluation are: **Abstract Expressionism, Cubism, Impressionism, Rococo, and Northern Renaissance.** The models evaluated for this classification task include:

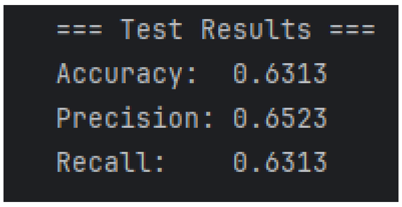
1. Two CNN architectures of varying depths.
2. A pre-trained ResNet-50 model with fine-tuning applied.

This approach aims to assess the performance and adaptability of the models on a more defined and less complex classification problem.

The initial model developed for this task was structured with three convolutional layers, including batch normalization, dropout, and max pooling layers. The resulting performance of this model:



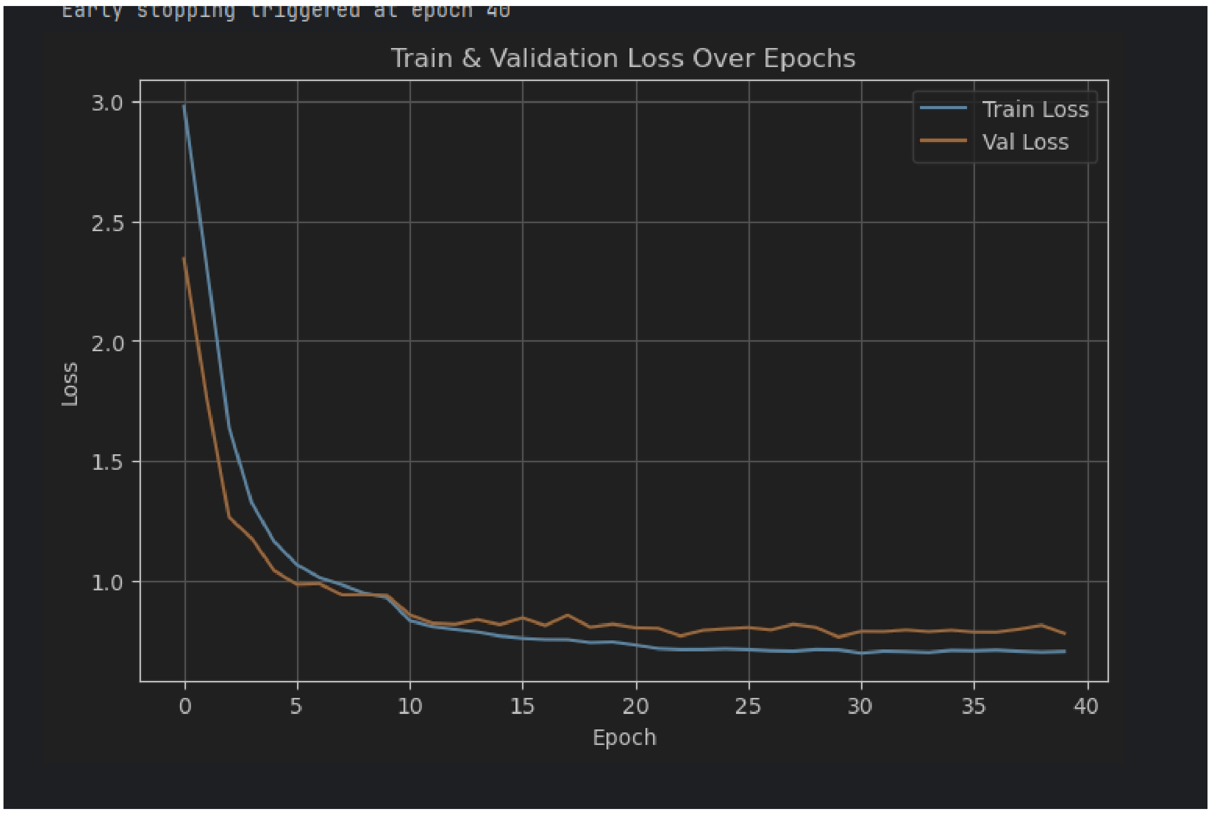
The following metrics were obtained on the test set:



```
=== Test Results ===
Accuracy:  0.6313
Precision: 0.6523
Recall:    0.6313
```

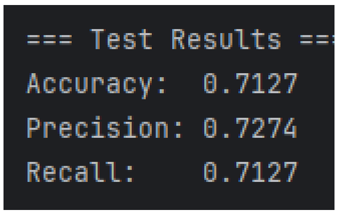
The unexpectedly high evaluation results achieved with such a simple model prompted the development of a more robust architecture, incorporating additional convolutional layers.

The next step was designing a model with 5 conv layers with the same logic that is mentioned above after the running of this model this plot was received:



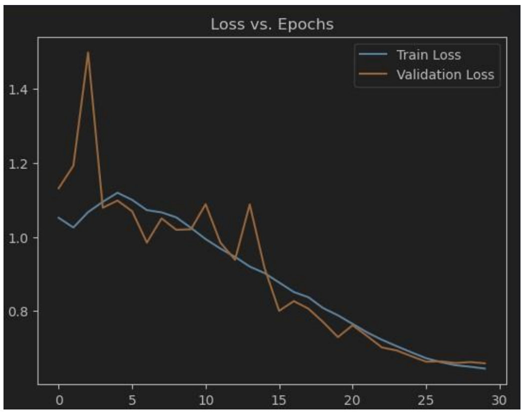
```
Early stopping triggered at epoch 40
```

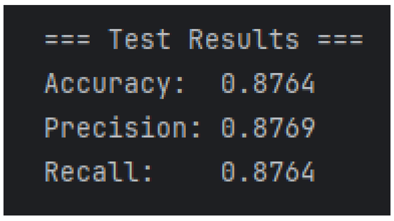
The following metrics were obtained on the test set:



```
=== Test Results ==
Accuracy:  0.7127
Precision: 0.7274
Recall:    0.7127
```

With the more complex model showing better results, it was clear there was still room for improvement. To take it a step further, a model using the ResNet-50 feature extractor was implemented, and here's what it achieved:





```
=== Test Results ===
Accuracy:  0.8764
Precision: 0.8769
Recall:    0.8764
```

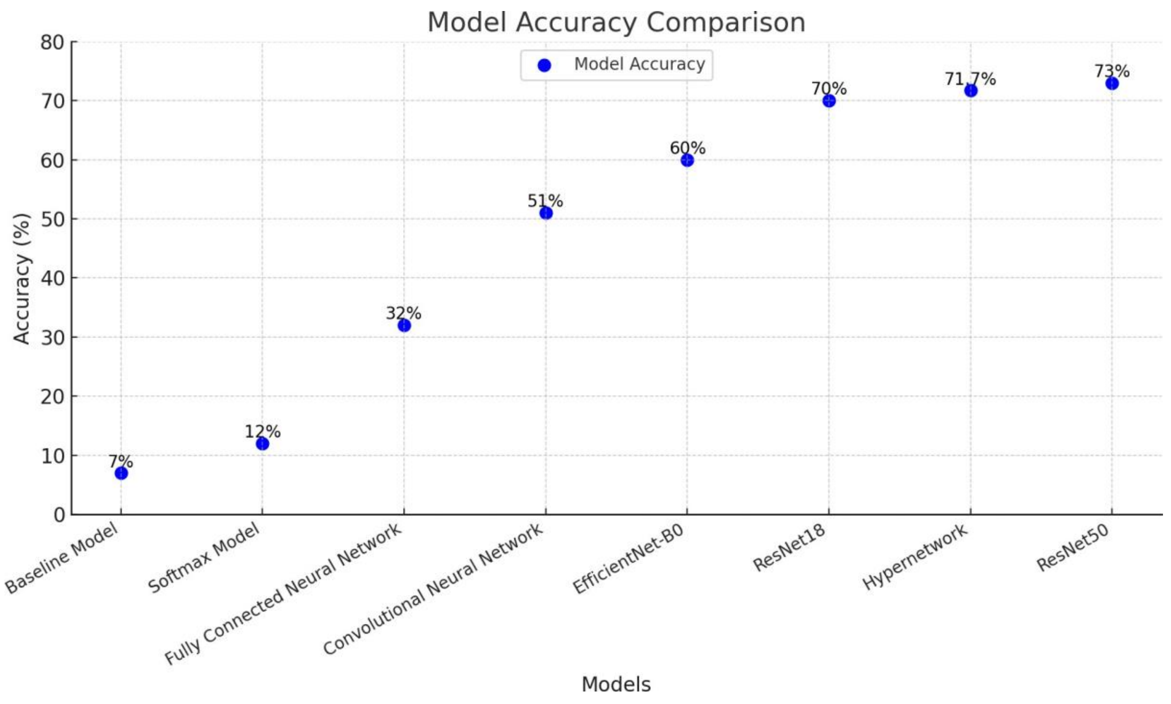
8) ***Summary:***

The project focused on classifying paintings into their art genres using the WikiArt dataset, which consists of approximately 80,000 paintings across 27 art genres.

Several models were developed to tackle the classification task. The baseline model, which predicted the majority class, achieved an accuracy of approximately 7%, reflecting the limitations of simplistic approaches.
Progressing to more advanced architectures, models like SoftMax regression, fully connected neural networks, convolutional neural networks (CNNs), and pre-trained architectures (ResNet and EfficientNet) feature extraction that fed to FNN were implemented.
In these models, techniques like data augmentation, dropout regularization, and learning rate scheduling were implemented.

Moving forward to more sophisticated models demonstrated significant improvements.
Pre-trained networks like ResNet18 and EfficientNet-B0 were fine-tuned for the classification task, with ResNet18 achieving a significant boost in accuracy.
The resnet50 showed an improvement over the resnet18 and the evaluation on the test with that model was 73% which preforms the best accuracy that was received during the progress.
Another attempt of designing a hypernetwork that will generate weights and biases was tried using the resnet50 as a feature extractor but received a lower accuracy score than the basic resent50 architecture with adjustments to fit to out dataset.

The accuracy received by each model during the process shows as follows:


Model Accuracy Comparison
● Model Accuracy
Accuracy (%)
Models
Baseline Model
Softmax Model
Fully Connected Neural Network
Convolutional Neural Network
EfficientNet-B0
ResNet18
Hypernetwork
ResNet50