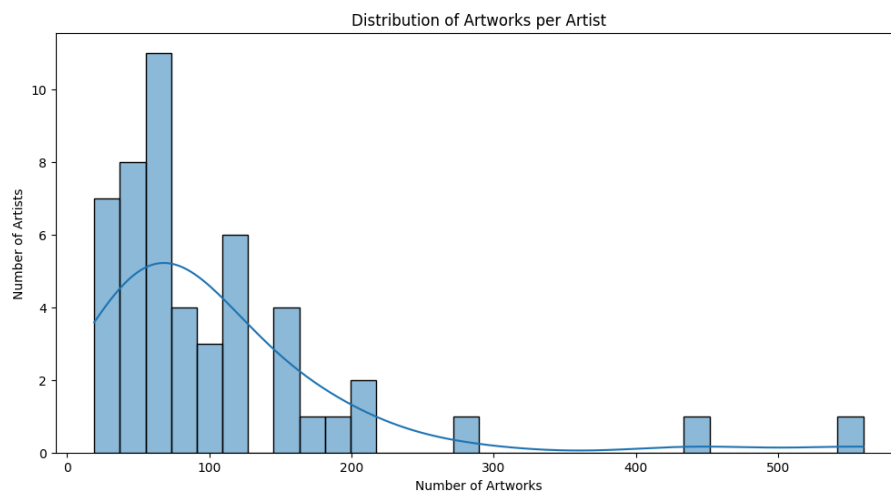


(b)

First, I checked whether the given data was evenly distributed among the painters.

...	artist
Vincent van Gogh	560
Edgar Degas	443
Pablo Picasso	274
Pierre-Auguste Renoir	210
Albrecht Dürer	201
Paul Gauguin	195
Francisco Goya	176
Rembrandt	161
Marc Chagall	156
Titian	153
Alfred Sisley	147
Paul Klee	124
Amedeo Modigliani	121
Rene Magritte	121
Andy Warhol	118
Sandro Botticelli	111
Henri Matisse	111
Mikhail Vrubel	106
Hieronymus Bosch	105
Leonardo da Vinci	96
Salvador Dalí	88
Peter Paul Rubens	83
Kazimir Malevich	82
Pieter Bruegel	79
Frida Kahlo	72
Diego Velázquez	71
Andrei Rublev	70
Joan Miro	69
Raphael	68
Gustav Klimt	65
Giotto di Bondone	64
Jan van Eyck	61
Camille Pissarro	58
Henri de Toulouse-Lautrec	57
El Greco	57
Claude Monet	55

Vasiliy Kandinskiy	53
Edouard Manet	51
Piet Mondrian	51
Henri Rousseau	48
Diego Rivera	47
William Turner	41
Edvard Munch	39
Gustave Courbet	37
Caravaggio	30
Michelangelo	30
Georges Seurat	29
Paul Cezanne	28
Jackson Pollock	20
Eugene Delacroix	19



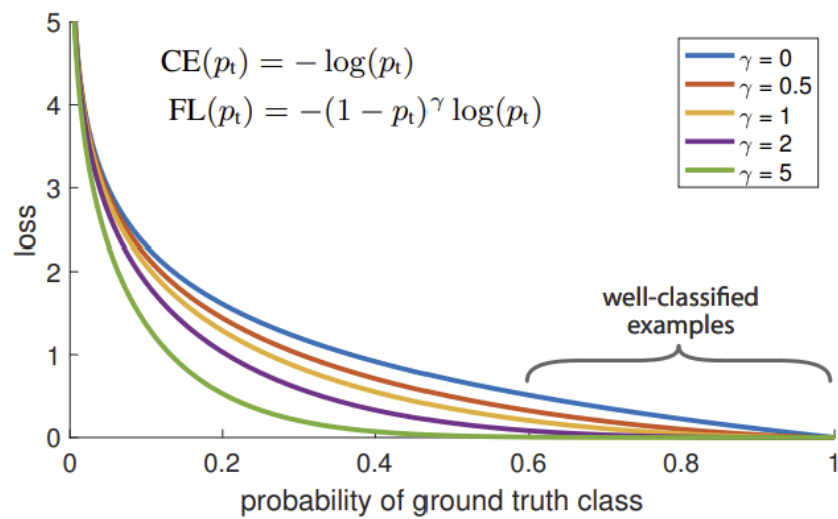
This assignment is not a simple classification task, but a more complex task of distinguishing between artists' works. In previous assignments, SVM or Multinomial/Multiclass Logistic Regression alone had limitations, and simple Multi-Layer Perceptrons posed risks of runtime errors in the early stages due to an excessive number of parameters, or overfitting to specific artists due to imbalanced training data.

In particular, there was a concern that the problem experienced with the 1-vs-All model in Assignment 2 might recur. At that time, due to data imbalance between the '1' label and the remaining labels, the model simply learned to predict 'not 1', resulting in a Validation Accuracy of 90% but very low Test Accuracy.

Against this background, we adopted a CNN model. Our classification target is not simple datasets like MNIST, Fashion-MNIST, or CIFAR-10, but rather artistic works where we need to capture and distinguish the unique painting style and artistic characteristics of each artist.

Therefore, the key challenge was how to address the severe disparity in the number of data samples across different artists.

To solve this severe data imbalance problem, we initially considered a method of categorizing training data by label and applying more data augmentation to categories with insufficient data. However, with the difference between the artist with the most works (550 pieces) and the artist with the fewest works (19 pieces) reaching approximately 29 times, augmentation alone had its limitations. While investigating solutions, we discovered Focal Loss, which was proposed by Facebook AI Research (FAIR) to address class imbalance problems.



$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

When p_t is low, $(1-p_t)$ is close to 1 and $-\log(p_t)$ is also large, so it incurs a large loss. On the other hand, when p_t is high, the loss value is lower than that of a general CE (100 times lower loss than the general CE in the case of $p_t=0.99$, 1000 times lower loss in the case of $p_t=0.999$), so Focal Loss can focus on labels that are difficult to classify.

```

class FocalLoss(nn.Module):
    def __init__(self, alpha=0.25, gamma=2.0, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        ce_loss = F.cross_entropy(inputs, targets, reduction='none')
        pt = torch.exp(-ce_loss)
        focal_loss = self.alpha * ((1 - pt) ** self.gamma) * ce_loss

        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        else:
            return focal_loss

```

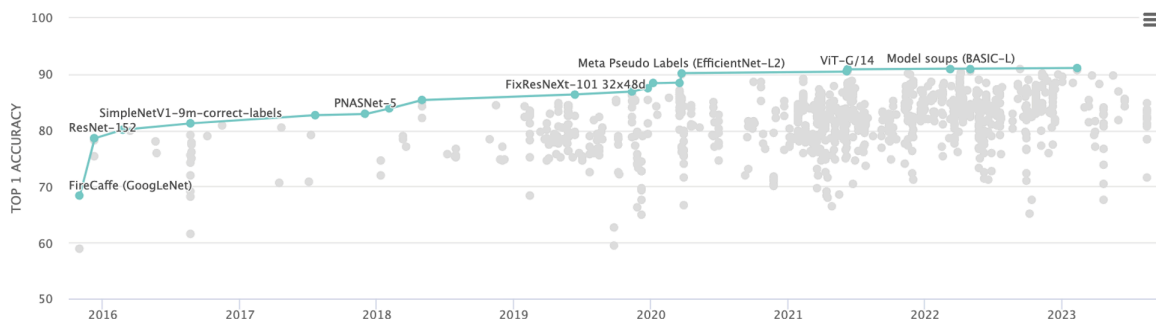
Alpha and gamma were set to 0.25 and 2.0, which are the default values set by FAIR.

First, we compute cross entropy, which is then used to inversely compute the predicted probability (pt) of the correct class. (pt = exp(-ce_loss) is mathematically equivalent to the softmax probability for the correct class.)

Then, using $(1 - pt)^\gamma$ as a dynamic weight, we assign high weights to difficult-to-predict samples and low weights to easy samples. This automatically focuses on minority classes that are difficult to learn in class imbalanced situations.

At first, I was wondering if I should show each artist's paintings, learning the artist's unique brushstrokes and such in the initial layers, and then classifying them using the accumulated data in the final layer. I knew I needed a CNN, but I took the time to get a feel for how it works (and how CNN learns image features) by classifying FashionMNIST using a simple CNN example. (This process is contained in Pratiec.ipynb.)

One thing I learned through this process is that the combinations that can be configured through CNNs are endless, so it was nearly impossible to create a model while considering the optimal number of layers, learning rate, and many other factors one by one with less than a week left until the assignment was due.



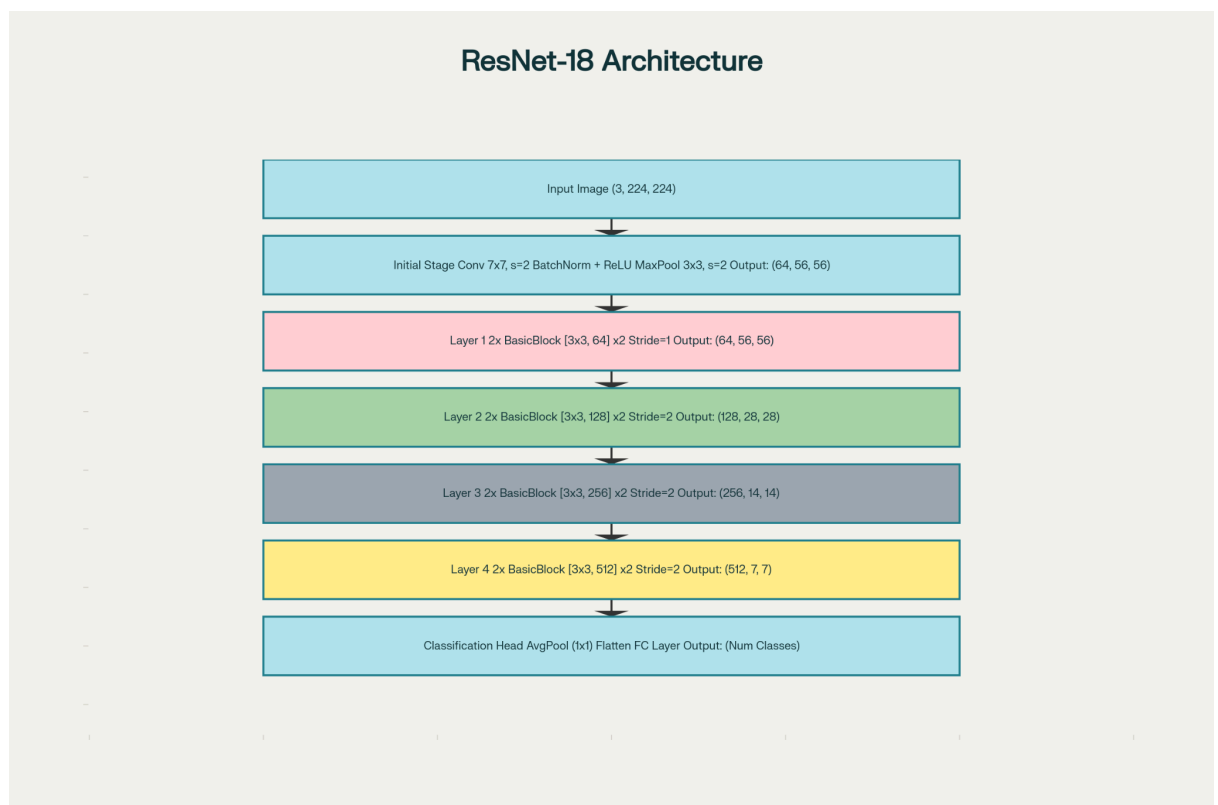
The ipynb file provided in the tutorial session contained useful advice. While it would be ideal to reference high-performance models employing state-of-the-art techniques, it was necessary to consider the risk that excessively complex models with deep layers might

become biased toward artists with large numbers of works, such as Van Gogh, given the limited training dataset of 5,300 samples. Additionally, there was a practical constraint that more complex models would require acquiring additional advanced learning techniques.

Taking these circumstances into account, I explored models that are compatible with Focal Loss while demonstrating stable generalization capabilities on small training datasets. The available resources consisted of Colab's A100/L4 GPUs and a training dataset of 5,300 samples, and I sought to select a model that could be understood and implemented without difficulty, rather than an overly complex one.

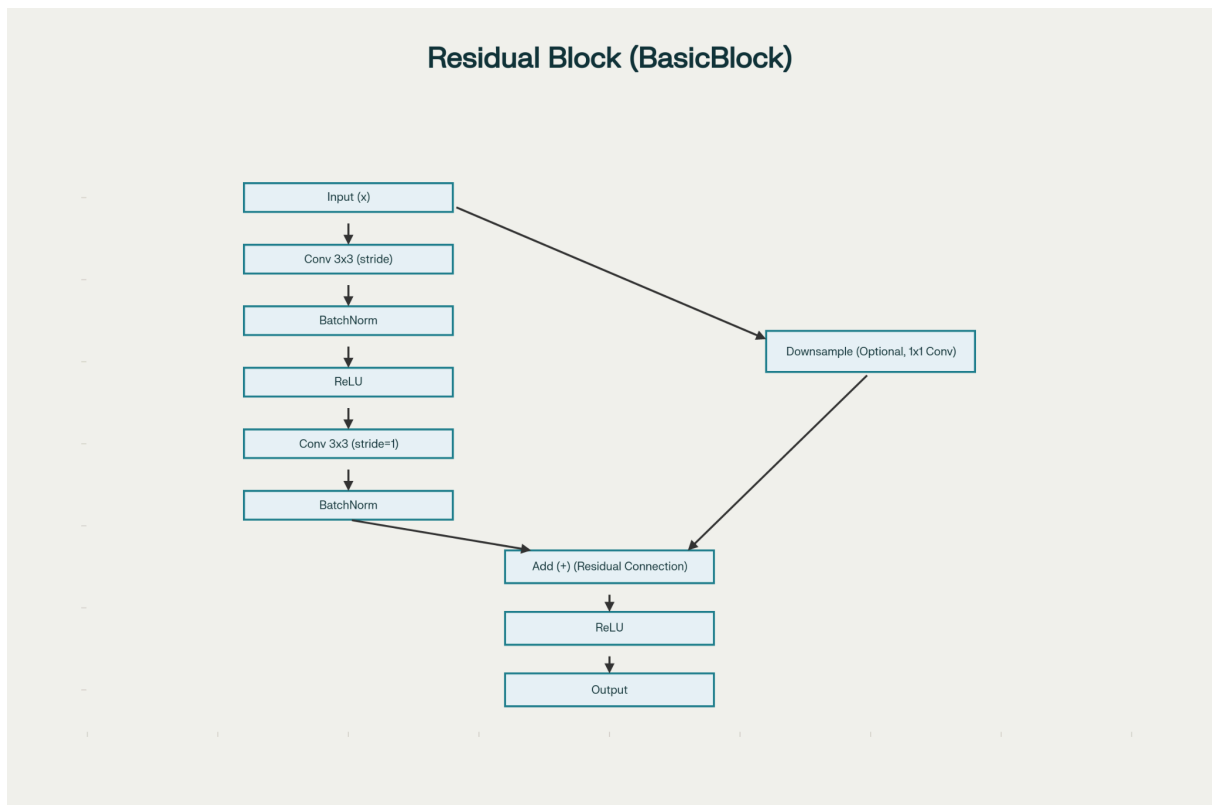
Consequently, I explored benchmark models suitable for the current situation (addressing class imbalance through Focal Loss, limited data of 5,300 samples, and the Long Tail Problem).

ResNet18 was selected. There are many ResNet models, but ResNet-18 was chosen because, with a limited dataset of approximately 5,400 data points, if the model is too deep and has too many parameters (more than ResNet-50), overfitting can easily occur, where the model memorizes the data itself rather than learning its general characteristics.



ResNet18 first reduces the amount of computation by quickly reducing the large image (224x224) to 56x56 in the Initial Stage and extracts features into 64 channels.

Let's learn about BasicBlock, which goes from Layer 1 to 4.



```
import torch
import torch.nn as nn

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(BasicBlock, self).__init__()

        self.conv1 = nn.Conv2d(
            in_channels,
            out_channels,
            kernel_size=3,
            stride=stride,
            padding=1,
            bias=False
        )
        self.bn1 = nn.BatchNorm2d(out_channels)

        self.conv2 = nn.Conv2d(
            out_channels,
            out_channels,
            kernel_size=3,
            stride=1,
            padding=1,
            bias=False
        )
        self.bn2 = nn.BatchNorm2d(out_channels)
```

```

        self.relu = nn.ReLU(inplace=True)

        self.downsample = downsample

    def forward(self, x):
        """
        Forward pass with skip connection
        """
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out

```

To understand complex features like an artist's style, a model requires deep layers. However, as the network deepens, the vanishing gradient problem often occurs. The Residual Block in ResNet18 prevents this issue through the following structure:

1. Two Paths When input x is received, the data flows along two paths:

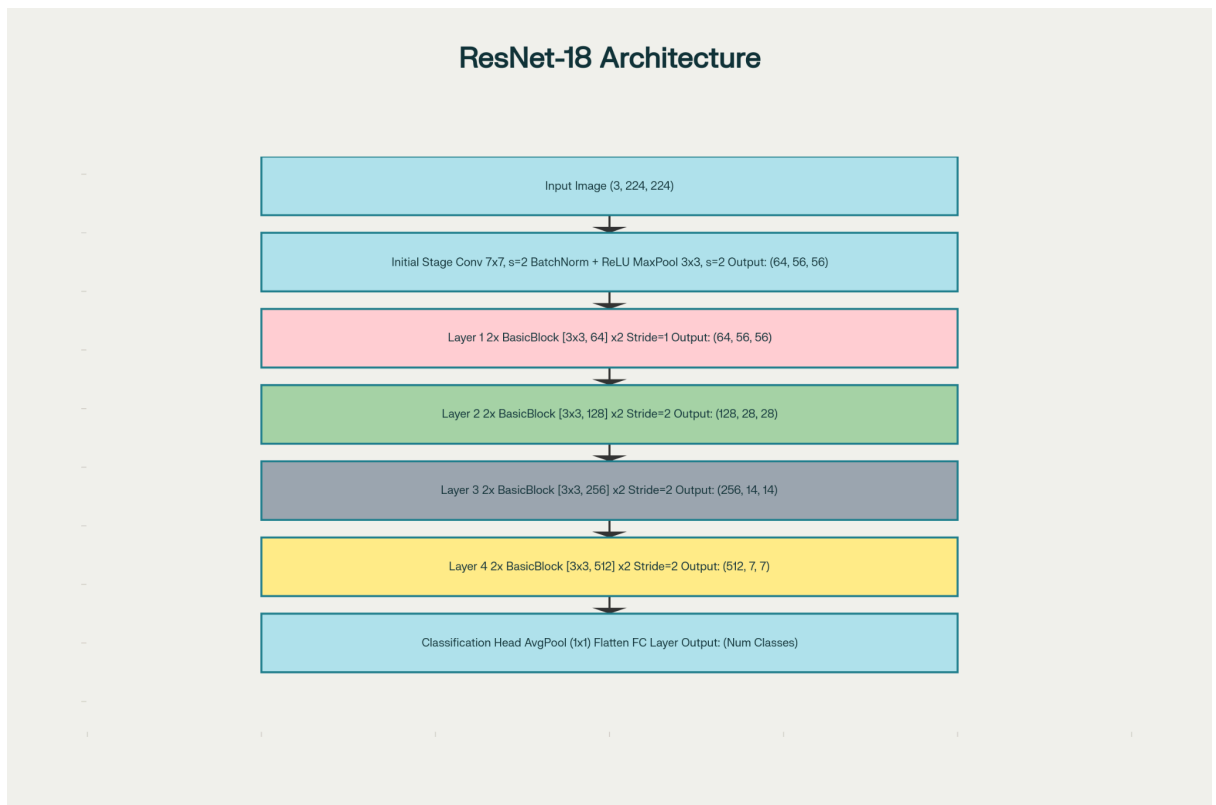
Main Path: Passes through $\text{conv1} \rightarrow \text{bn1} \rightarrow \text{relu} \rightarrow \text{conv2} \rightarrow \text{bn2}$ to calculate the transformed features ("out").

Shortcut Path: Transmits the input x directly (Identity Mapping). Downsampling is applied only when necessary to match dimensions.

2. This corresponds to the $\text{out} += \text{identity}$ operation. It embodies the core concept that the network should learn the "residual" (difference between input and output) rather than the perfect mapping function itself.

3. Final Activation ReLU is applied immediately after the addition. This introduces non-linearity before passing the information to the next block.

Thanks to this structure, the network preserves the gradient flow even in deep layers, ensuring input information is effectively propagated.



Layer 1: BasicBlock (stride = 1) * 2 → Extract features while maintaining resolution.

Layer 2~4: BasicBlock (stride = 2) * 2, Double Channel.

→ Double the number of channels while halving the width and height, gradually increasing the complexity of the learned patterns.

Finally, AvgPool calculates the average value of each feature map, flattens it, and outputs probability values equal to the number of classes.

Additionally, training was conducted using CustomResNet18, which added regularization to prevent overfitting, and Dropout was added to CustomResNet18.

```

Calculating mean and standard deviation for the training dataset...
Calculated Mean: tensor([0.5027, 0.4420, 0.3677])
Calculated Std: tensor([0.2103, 0.1947, 0.1761])
'/content/drive/MyDrive/MLassignment3/test.csv' file successfully loaded for test set.

Train dataset size: 5311 images
Test dataset size: 600 images
Number of classes (artists) in training data: 50
Classes (artists) in training data: ['Albrecht Du rer', 'Alfred Sisley', 'Amedeo Modigliani', 'Andrei Rublev', 'Andy Warhol', 'Camille Pissarro', 'Caravaggio', 'Claude Monet', '
Train DataLoader has 166 batches.
Test DataLoader has 19 batches.
  
```

It took a very long time because I had to mount each image on Google Drive, convert it into a tensor, and then generalize it. The mean and std were saved through WanDB.

```

run = wandb.init(project="kaggle-artist-classification", job_type="preprocessing")
wandb.config.update({
    "dataset_mean": calculated_mean.tolist(), # 텐서를 리스트로 변환
    "dataset_std": calculated_std.tolist(),
    "image_size": 224,
    "batch_size": 32
})

artifact = wandb.Artifact('dataset-stats', type='dataset-metadata')
metadata = {
    "mean": calculated_mean.tolist(),
    "std": calculated_std.tolist()
}
import json
with open("stats.json", "w") as f:
    json.dump(metadata, f)
artifact.add_file("stats.json")
run.log_artifact(artifact)

print("✅ Mean/Std stats saved to WandB!")
run.finish()

```

```

final_train_transforms = transforms.Compose([

    transforms.Resize((256, 256)),

    transforms.CenterCrop((224, 224)),

    transforms.RandomHorizontalFlip(), # Example augmentation

    transforms.ToTensor(),

    transforms.Normalize(mean=calculated_mean, std=calculated_std)

])

# Transforms for validation/test data (no augmentation, just consistent preprocessing)

final_test_transforms = transforms.Compose([

    transforms.Resize((256, 256)),

    transforms.CenterCrop((224, 224)),

    transforms.ToTensor(),

    transforms.Normalize(mean=calculated_mean, std=calculated_std)

])

```

Since our goal is to implement a model with excellent generalization ability using a trainset with a LongTail distribution, we should not evaluate the model's performance simply by accuracy.

```
def custom_macro_f1(y_true, y_pred, num_classes):
    f1s = []
    for c in range(num_classes):
        tp = ((y_pred==c) & (y_true==c)).sum().item()
        fp = ((y_pred==c) & (y_true!=c)).sum().item()
        fn = ((y_pred!=c) & (y_true==c)).sum().item()
        precision = tp/(tp+fp) if tp+fp>0 else 0
        recall = tp/(tp+fn) if tp+fn>0 else 0
        f1 = (2*precision*recall)/(precision+recall) if
precision+recall>0 else 0
        f1s.append(f1)
    return sum(f1s)/len(f1s)
```

Macro f1 will be a good measure to evaluate generalization ability by treating each class equally regardless of the class distribution.

```
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3,
weight_decay=1e-4)
```

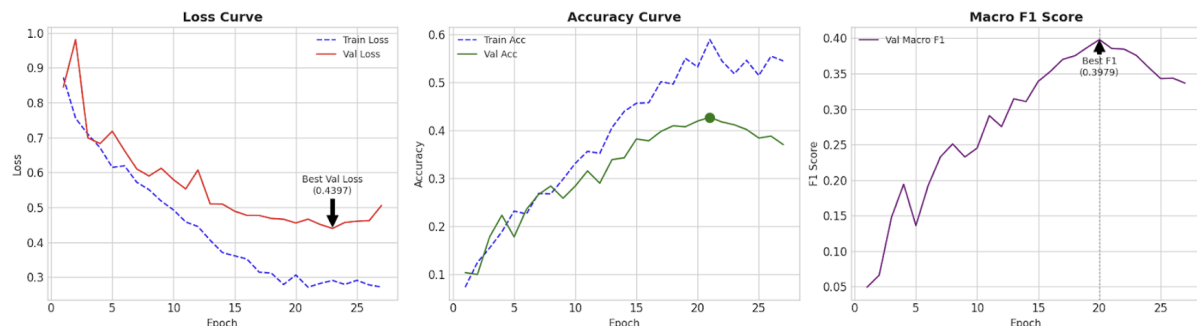
I used AdamW and its default lr, which are suitable for small datasets and to prevent overfitting through sophisticated weight decay.

Now it's time to learn multiple times and make adjustments while submitting to Kaggle.

First Try: [2023036299_ML2_assignment3.ipynb](#)

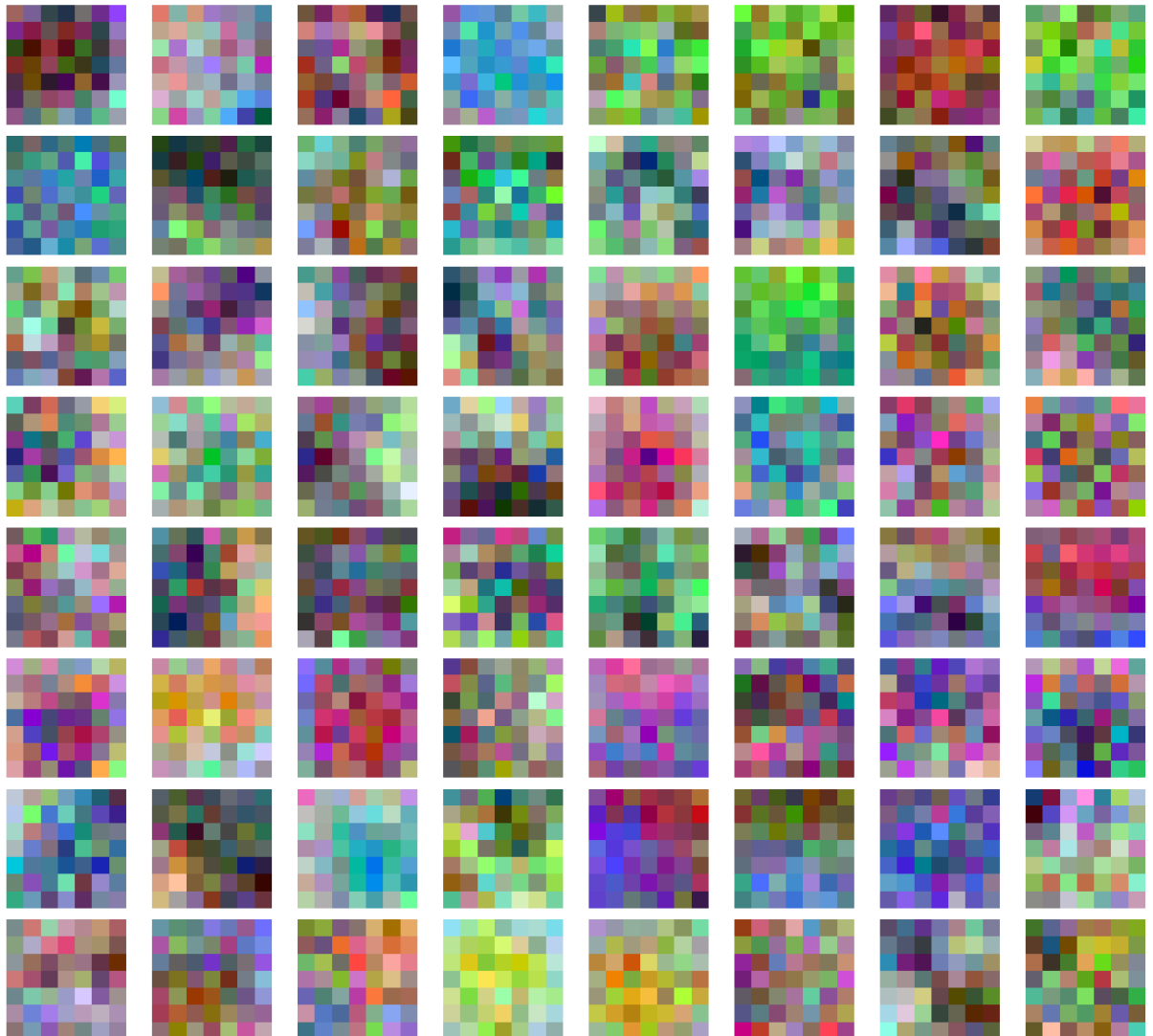
(Other experimental data are included in a zip file in ipynb format. However, readability may be slightly reduced.)

The first training was ResNet18 with dropout + Epoch 50, and the Early Stop technique (stopping if there is no improvement in Macro F1 for 7 Epochs) was applied.



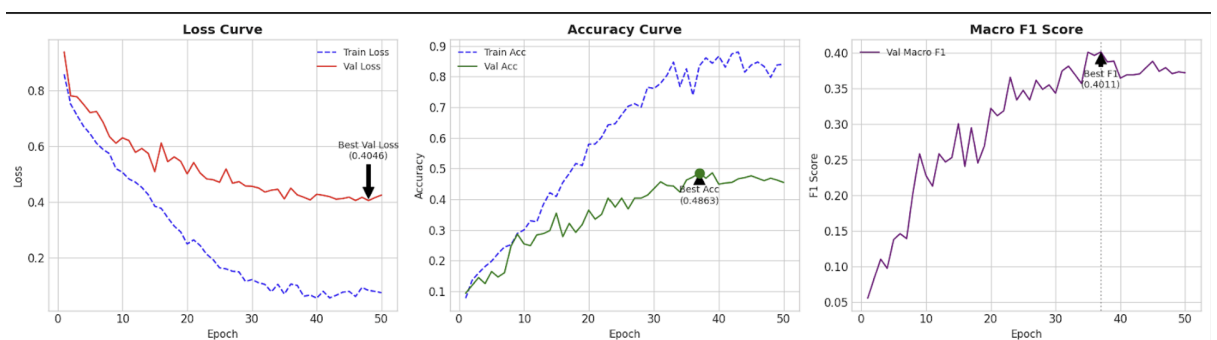
Submission and Description		Public Score ⓘ
<div> <div>✓</div> <div> submission.csv Complete · 17s ago · First Submission </div> </div>	0.34310	

Filters of the First Convolutional Layer (conv1)



Second Try:

The data was already small and very unbalanced, so I boldly abandoned the Early Stop logic because I thought that underfitting would occur if I went with the Early Stop.



Filters of the First Convolutional Layer (conv1)



All
Successful
Selected
Errors
Recent ▾

Submission and Description
Public Score ⓘ
Select

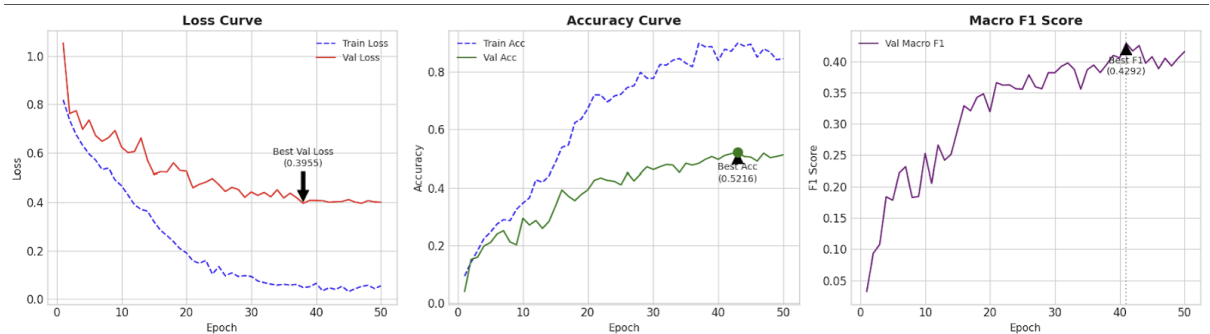
submission.csv
 Complete · 18s ago · second submission

0.47800
☐

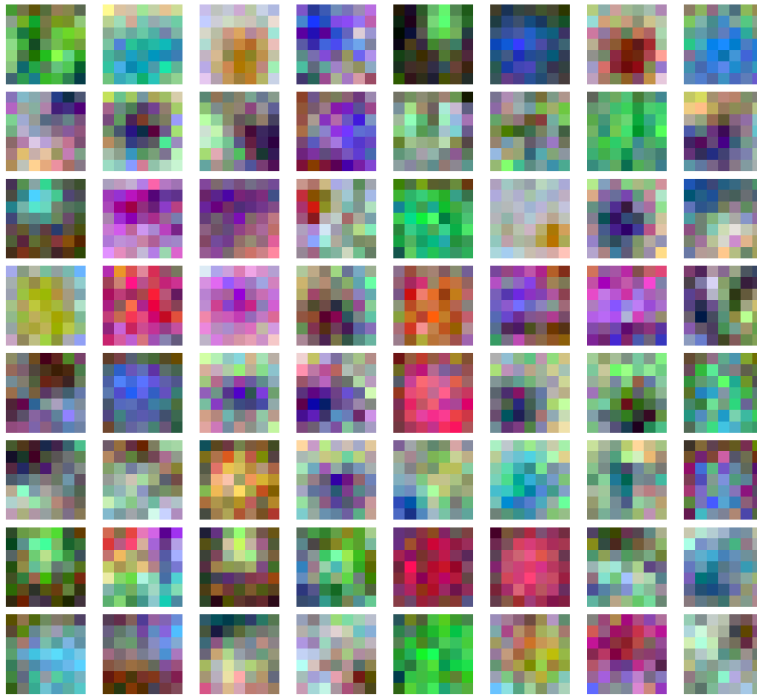
It definitely shows better performance just by doing the set 50 times.

Third Try: Basic RES18 with out Early Stop

Third, I turned off both dropout and early stop.



Filters of the First Convolutional Layer (conv1)



Submission and Description		Public Score	Select
	submission_third.csv Complete · 20s ago	0.48973	

Accuracy actually increased by about 0.01. However, the more important Macro F1 achieved a significant improvement, reaching 0.4292. It seems likely that randomly turning dropout on and off may have hindered the model's generalization.

Fourth Try: Custom ResNet18 and Epochs 100

```
Epoch 62: train loss 0.0634, acc 0.8234; val loss 0.4258, acc 0.5137, macro F1 0.4541
-> Best Macro F1 updated: 0.4541
Epoch 63: train loss 0.0650, acc 0.8265; val loss 0.4371, acc 0.4627, macro F1 0.3684
Epoch 64: train loss 0.0447, acc 0.8484; val loss 0.4033, acc 0.5000, macro F1 0.3996
Epoch 65: train loss 0.0640, acc 0.8046; val loss 0.4255, acc 0.4941, macro F1 0.4120
Epoch 66: train loss 0.0704, acc 0.8355; val loss 0.4453, acc 0.4667, macro F1 0.3792
Epoch 67: train loss 0.0584, acc 0.9254; val loss 0.4146, acc 0.5118, macro F1 0.4405
Epoch 68: train loss 0.0825, acc 0.8307; val loss 0.4567, acc 0.4510, macro F1 0.3819
Epoch 69: train loss 0.0871, acc 0.8030; val loss 0.4383, acc 0.4627, macro F1 0.3774
Epoch 70: train loss 0.0725, acc 0.8140; val loss 0.4868, acc 0.4333, macro F1 0.3814
Epoch 71: train loss 0.0516, acc 0.8621; val loss 0.4793, acc 0.4431, macro F1 0.3703
Epoch 72: train loss 0.0750, acc 0.8019; val loss 0.4834, acc 0.4314, macro F1 0.3554
Epoch 73: train loss 0.0559, acc 0.8500; val loss 0.5253, acc 0.4353, macro F1 0.3397
Epoch 74: train loss 0.0714, acc 0.8202; val loss 0.5113, acc 0.4333, macro F1 0.3536
Epoch 75: train loss 0.0680, acc 0.7998; val loss 0.4834, acc 0.4373, macro F1 0.3779
Epoch 76: train loss 0.0553, acc 0.8517; val loss 0.5015, acc 0.4451, macro F1 0.3919
Epoch 77: train loss 0.0948, acc 0.8044; val loss 0.5405, acc 0.3882, macro F1 0.2993
Epoch 78: train loss 0.0932, acc 0.8367; val loss 0.4963, acc 0.4000, macro F1 0.3568
Epoch 79: train loss 0.0910, acc 0.8269; val loss 0.4593, acc 0.4745, macro F1 0.3887
Epoch 80: train loss 0.0733, acc 0.8469; val loss 0.5216, acc 0.4412, macro F1 0.3500
Epoch 81: train loss 0.1149, acc 0.7761; val loss 0.5183, acc 0.4353, macro F1 0.3384
Epoch 82: train loss 0.0684, acc 0.7840; val loss 0.5240, acc 0.4157, macro F1 0.3188
Epoch 83: train loss 0.0740, acc 0.8213; val loss 0.5487, acc 0.3725, macro F1 0.3311
Epoch 84: train loss 0.0804, acc 0.8065; val loss 0.5011, acc 0.4255, macro F1 0.3445
Epoch 85: train loss 0.0755, acc 0.8330; val loss 0.5170, acc 0.4490, macro F1 0.3533
Epoch 86: train loss 0.0794, acc 0.8075; val loss 0.4653, acc 0.4745, macro F1 0.4041
Epoch 87: train loss 0.0656, acc 0.8275; val loss 0.5000, acc 0.4322, macro F1 0.3500
```

(The model in question was unable to be visualized due to poor version management.)

Increasing it too much seems pointless. Out of 100 epochs, the best Macro F1 record came from 62, which is closer to 50.

Filters of the First Convolutional Layer (conv1)



submission_fourth.csv

Complete · 2m ago

0.38709



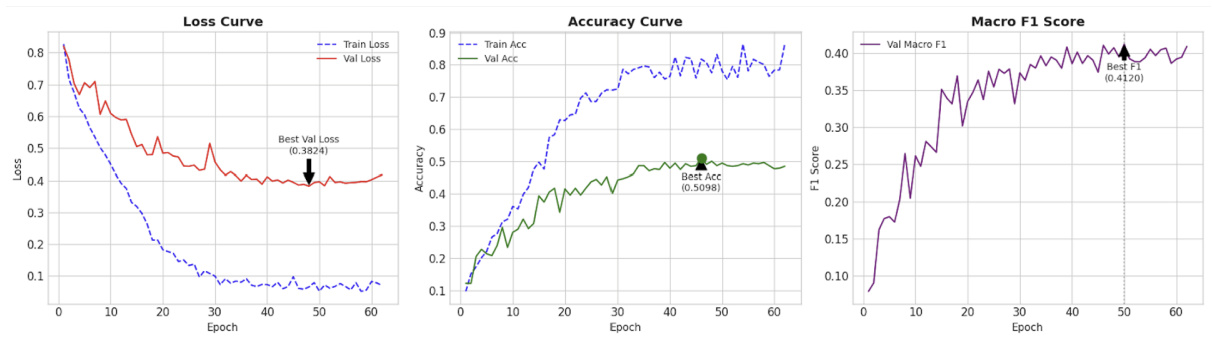
```
if best_state_dict is not None:
    model.load_state_dict(best_state_dict)
    print(f"Loaded best model with Macro F1:
{best_macro_f1:.4f}")
```

Due to the nature of the learning logic, the model at the time of the epoch with the best macro_f1 (which improved significantly to 0.45) was loaded and evaluated, and train_public came out the lowest.

Fifth Try: ResNet18 + BEST EPOCH(62) + Mix Up probability (0.4)

From this point on, I started mixing up a few more. The deadline for the assignment was approaching, and I still wanted to improve it a bit more. For epoch = 100, I increased the probability of mixup occurrence to 0.4, with 62 being the highest Marco F1 achieved at the time.

```
if torch.rand(1).item() < 0.40:
    images, y_a, y_b, lam = mixup_data(images, labels)
    outputs = model(images)
    loss = mixup_criterion(criterion, outputs, y_a, y_b, lam)
else:
    outputs = model(images)
    loss = criterion(outputs, labels)
```



Submission and Description		Public Score 📄	Select
	submission_5.csv Complete · 13s ago	0.46334	<input type="checkbox"/>



Sixth Try: ResNet18 + Custom mixup_probs + epoch = 62
 The probability of mix-up differs by class.

```
from collections import Counter

labels = torch.tensor([train_subset.dataset.samples[i][1] for i in train_subset.indices], dtype=torch.long)
class_counts = Counter(labels.tolist())

max_count = max(class_counts.values())
min_count = min(class_counts.values())

class_mixup_probs = {}
for c, count in class_counts.items():
    prob = 0.3 + 0.4 * (1 - (count - min_count) / (max_count - min_count))
    class_mixup_probs[c] = prob

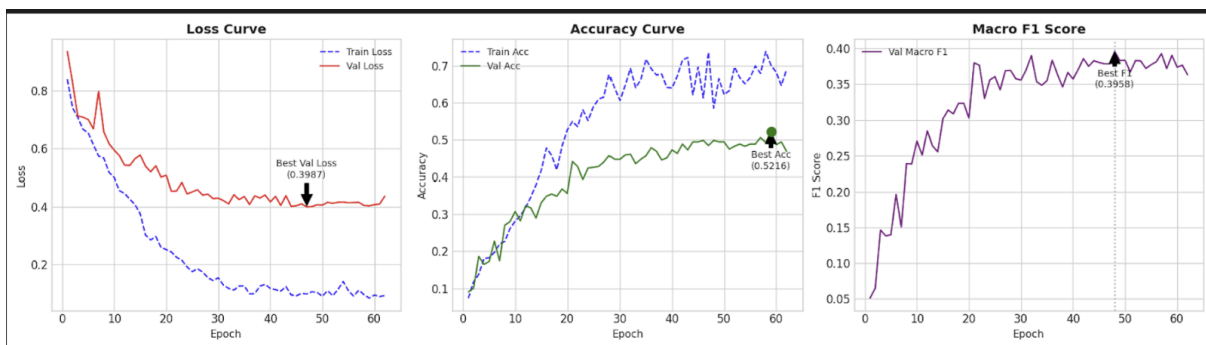
print("클래스별 Mixup 확률:")
for c in sorted(class_mixup_probs.keys()):
    print(f"Class {c}: {class_mixup_probs[c]:.3f} (samples: {class_counts[c]})")
```

However, the above probabilities are a fixed class-probability dictionary. In actual epoch-based learning, the class composition varies for each batch, requiring dynamic adjustments.

```
for epoch in range(num_epochs):
    model.train()
    trn_loss, trn_correct, total = 0, 0, 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        batch_mixup_probs = [class_mixup_probs[int(l)] for l in labels.cpu()]
        mean_mixup_prob = sum(batch_mixup_probs) / len(batch_mixup_probs)
```



submission_6.csv

0.47507

The F1 score is low, but the public score is not bad.

Filters of the First Convolutional Layer (conv1)



Seven Try: Multicrop + ResNet18 + EPOCH = 50 + Custom mixup_probs

```
import torch
import torchvision.transforms as transforms
from torch.utils.data import Dataset
from PIL import Image

class MultiCropDataset(Dataset):
```

```

    def __init__(self, base_dataset, class_counts, threshold=50,
n_crops=3):
        self.base_dataset = base_dataset
        self.class_counts = class_counts
        self.threshold = threshold
        self.n_crops = n_crops

        # Minority Class's Multi-Crop Transform
        self.multi_crop_transform = transforms.Compose([
            transforms.RandomResizedCrop(224, scale=(0.6, 1.0)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(15),
            transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean=calculated_mean,
std=calculated_std)
        ])

        # Majority Class Transform
        self.normal_transform = transforms.Compose([
            transforms.Resize((256, 256)),
            transforms.CenterCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=calculated_mean,
std=calculated_std)
        ])

    def __len__(self):
        return len(self.base_dataset)

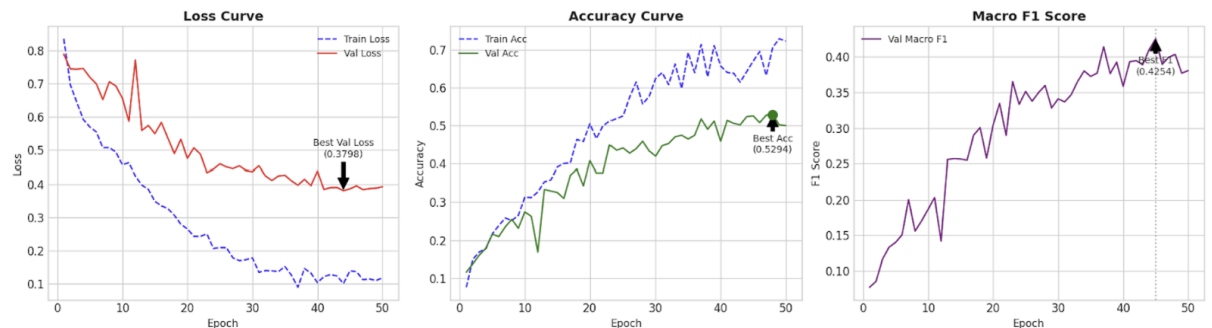
    def __getitem__(self, idx):
        img_path, label = self.base_dataset.samples[idx]
        image = Image.open(img_path).convert('RGB')

        if self.class_counts[label] < self.threshold:
            crop_idx = torch.randint(0, self.n_crops, (1,)).item()
            transformed_image = self.multi_crop_transform(image)
        else:
            transformed_image = self.normal_transform(image)

```

```
return transformed_image, label
```

We added the MultiCrop feature to increase the opportunities for models to learn from works by a small number of artists with limited data.



submission_7.csv

Complete · 41s ago

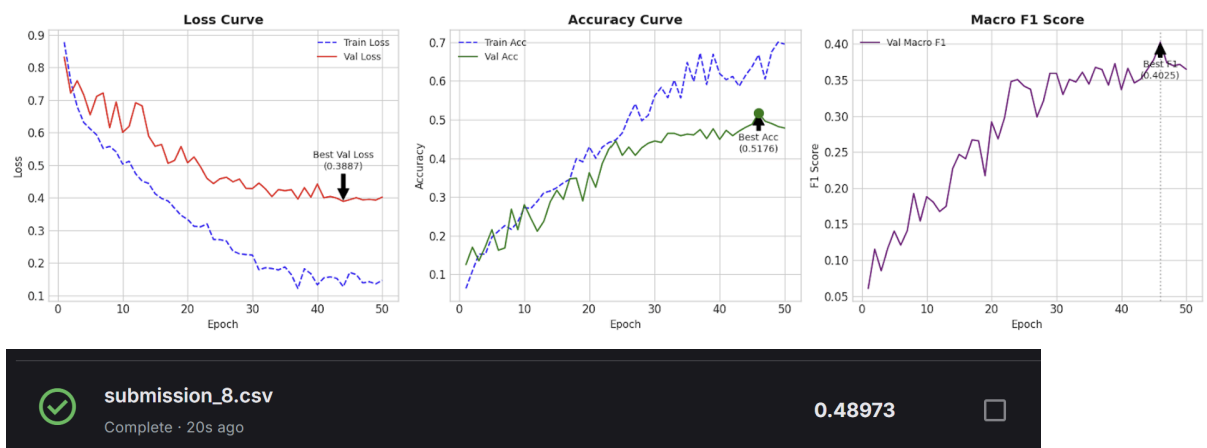
0.51906

As a result, we finally broke through the 0.5 barrier.

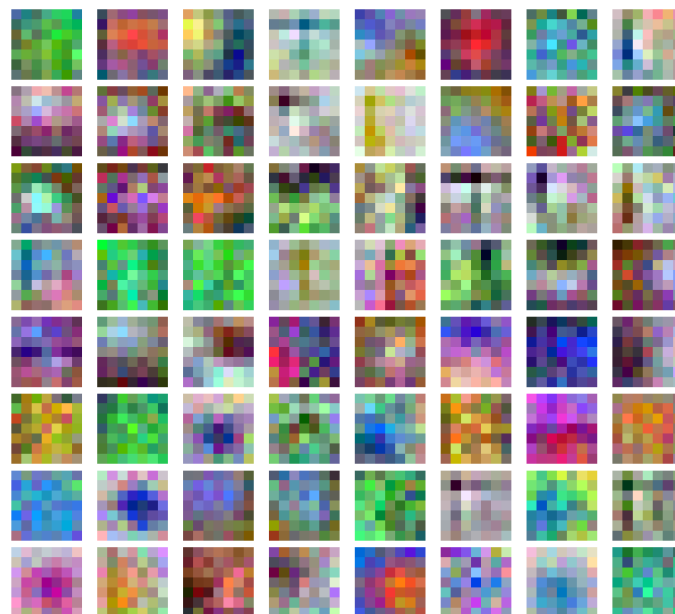
Filters of the First Convolutional Layer (conv1)



8: CustomResNET18 + Multicrop+ Epoch = 50 + Custom mixup_probs
There is a Last Try.



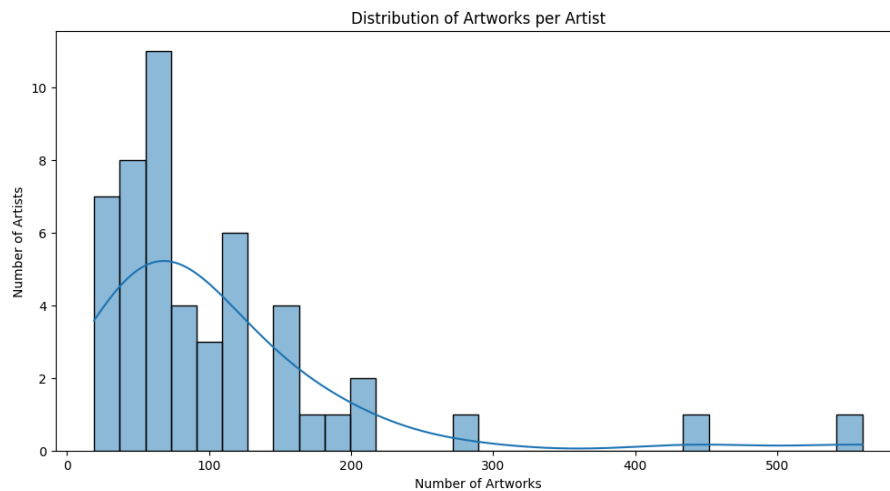
Filters of the First Convolutional Layer (conv1)



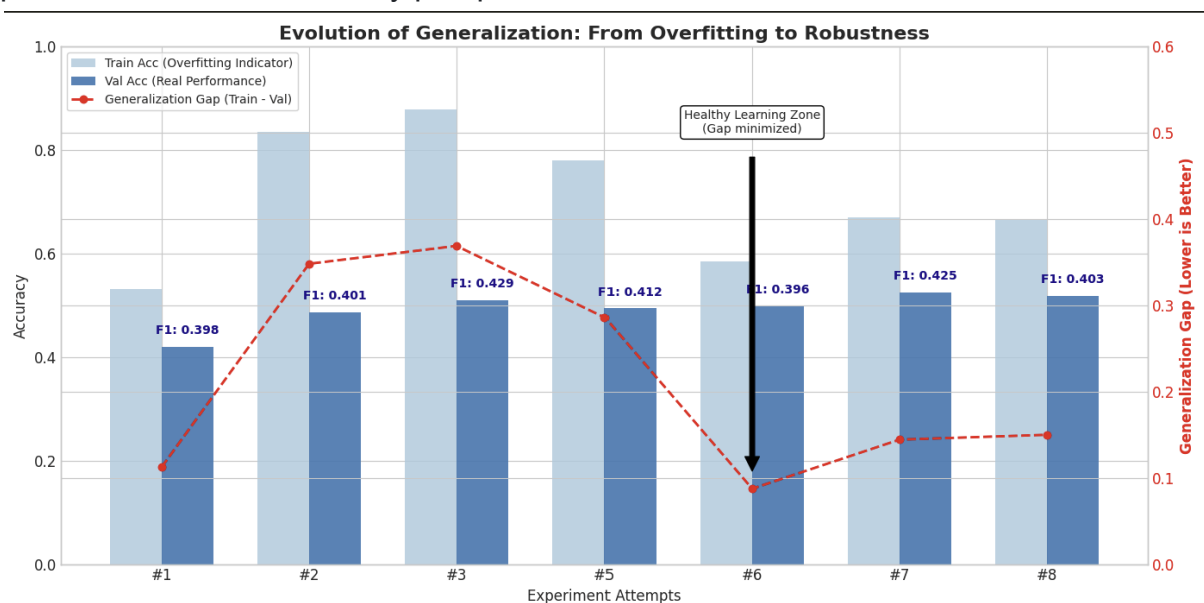
For private submissions, I plan to submit the 4th model, which had the highest macro f1 (0.4541) among all attempts, although its public acc was low, and the 7th model, which had the highest public acc (0.51906) and a respectable macro f1 (0.4254). Judging from the discrepancy between the public acc and the best macro f1 of each model from the training process, the test data also seems to be unbalanced. In this case, it seems more appropriate to submit a CSV for private evaluation based on macro f1, which is an indicator of generalization ability.

✓	submission_7.csv Complete · 4h ago	0.51906	✓
✓	submission_6.csv Complete · 4h ago	0.47507	□
✓	submission_5.csv Complete · 5h ago	0.46334	□
✓	submission_fourth.csv Complete · 1d ago	0.38709	✓

What impressed me most about this project was the power of deep learning, where the model autonomously learns patterns in the data that I didn't fully understand and produces meaningful results. I was able to observe performance improvements in a short training time, which allowed me to experience the practical potential of deep learning firsthand.



However, I reflect critically on how I became overly focused on finding the balance between 'Overall Accuracy' and 'Tail Class Generalization' during the research process, which narrowed my perspective.



- 1: ResNet18 + DropOut(0.5) + Epoch(50 + Early Stop
- 2: ResNet18 + DropOut(0.5) + Epoch(50)
- 3: ResNet18 + Epoch(50)
- 4: ResNet18 + DropOut(0.5) + Epoch(100)
- 5: ResNet18 + Epoch(62) + MixUP with 0.4 + Epoch(62)
- 6: ResNet18 + Mixup with Custom probs + Epoch(50)
- 7: ResNet18 + MultiCrop + Mixup with Custom probs + Epoch(50)
- 8: ResNet18 + DropOut(0.5) + MultiCrop + Mixup with Custom probs + Epoch(50)

1. Phase 1: Overfitting Detection (Attempts #1-3)

Initial experiments exhibited severe overfitting on the limited dataset of 5,400 images. Training Accuracy reached 90% while Validation Accuracy remained around 50%, resulting in approximately 40%p generalization gap.

Attempt #1: ResNet18 + Dropout 0.5 + Early Stopping

- Training terminated prematurely due to early stopping trigger
- Dropout 0.5 alone proved insufficient to prevent data memorization

Attempt #2-3: Dropout Removal and Adjustment

- Attempt #3 without Dropout achieved the highest F1 Score of 0.429
- However, generalization gap remained at a critical level

Key Finding: Excessive regularization on small datasets can degrade performance, indicating the need for fundamental solutions

2. Phase 2: The Specialist Model - An Anomaly (Attempt #4)

Configuration: ResNet18 + Dropout 0.5 + Epoch 100

- **Macro F1: 0.450 (Highest)**
- **Accuracy: 0.39 (Lowest)**

Analysis of Characteristics

This model represents an intriguing case demonstrating extreme trade-off between accuracy and F1 Score:

1. **Long-tail Class Learning:** Extended training of 100 epochs allowed the model to focus on rare classes in later stages
2. **Dual Nature of Dropout:** Reduced overconfidence in majority classes while improving minority class prediction capability
3. **Strategic Significance:** A valid approach when F1 Score is prioritized in class imbalanced problems

Conclusion: Provides insights as a "minority class specialist" model for imbalanced datasets

3. Phase 3: Introduction of Regularization Techniques (Attempts #5-6)

The learning dynamics changed dramatically after introducing Mixup augmentation. Training Accuracy dropped from 90% to 70% while the generalization gap significantly decreased.

Attempt #5: Mixup Alpha 0.4

- Confirmed healthy learning curve with substantially reduced train-validation gap
- Model transitioned from image memorization to feature learning

Attempt #6: Mixup with Custom Probabilities

- **Val Acc > Train Acc** reversal phenomenon observed
- Achieved the most robust generalization performance despite relatively lower F1 Score of 0.396
- Generalization gap minimized to approximately 0.1

Key Achievement: Secured genuine generalization capability through Mixup

4. Phase 4: Reaching Optimal Balance (Attempts #7-8)

Attempt #7: MultiCrop + Mixup Custom (Submitted Model)

- **F1 Score: 0.425 (2nd highest)**
- **Generalization Gap: ~0.15**

Achieved optimal balance through synergy between MultiCrop augmentation and Mixup regularization:

- MultiCrop: Increased data information through diverse viewing angles
- Mixup: Prevented overfitting while maintaining generalization
- Secured both stable loss curves and high F1 Score, making it **the most practical model**

Attempt #8: Side Effects of Excessive Regularization

- Applied additional Dropout 0.5
- F1 Score decreased to 0.403
- Signs of underfitting due to overlapping regularization from both Mixup and Dropout

Lesson Learned: Excess is as bad as deficiency in regularization - proper balance is essential

Summary

Experimental Evolution

1. **Phase 1-2:** Diagnosed overfitting and confirmed effects of extended training
2. **Phase 3:** Secured generalization performance through Mixup introduction
3. **Phase 4:** Discovered optimal combination of data augmentation and regularization

While being absorbed in this trade-off, I failed to sufficiently experiment with other important hyperparameters that could enhance overall model performance, such as Batch Size, Kernel Size, and Learning Rate Scheduling.