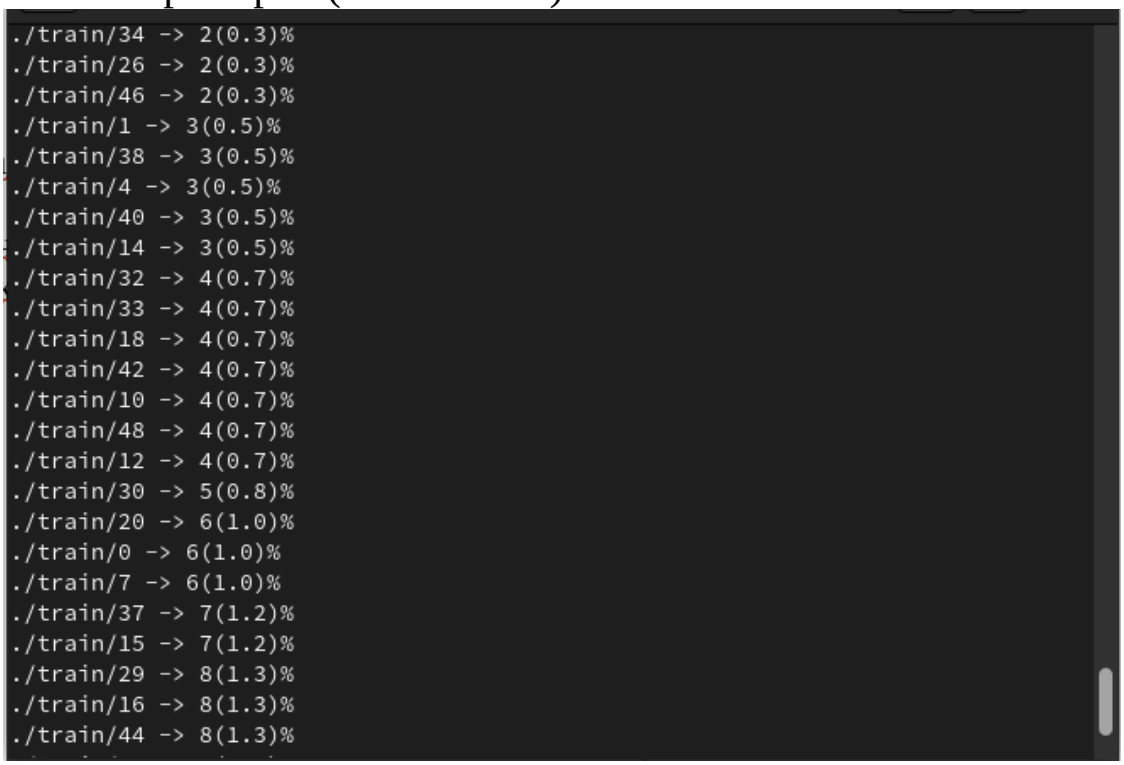
Исследование на тему: **Классификация птиц**

Проблемы, выявленные на этапе ручного анализа датасета:

1. Дисбаланс классов: некоторые классы(34, 26, 46, 1) имеют очень мало примеров (2-3 объекта)

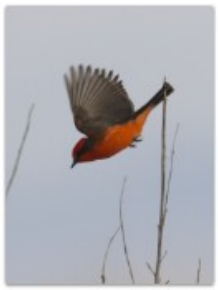


(результат получен через скрипт birds_classification/class_freq.py)

2. Неправильная разметка (почти в каждом классе есть неправильный пример данных)

00.jpg

01.jpg



02.jpg

03.jpg

(Изображения из класса №1)

3. Изображения птиц имеют очень вариативный фон, который занимает значительную часть изображения, и будет снижать качество распознавания

00.jpg

01.jpg

02.jpg

03.jpg

Предполагаемое решение найденных на первом этапе проблем:

- Для уменьшения вариативности изображения можно применить предобученный детектор, который сможет локализовать изображение птиц (идея вдохновлена статьёй[i])
- Когда изображение птиц станет менее вариативным, мы сможем найти похожие изображения с помощью векторизации картинки (Embedding) и метода кластеризации. Это должно улучшить ситуацию с неправильной разметкой *
- Дисбаланс классов попробуем исправить с помощью аугментаций и oversampling метода[ii]

* Из-за больших временных потерь на отладку алгоритма и маленького кол-ва изображений разметка была исправлена вручную.

Построение алгоритма обучения:

1. Выполнена подготовка компонентов PyTorch, которые необходимы для обучения
2. Для разделения данных на train и valid выборки используется стратификация(чтобы сохранить распределение классов), а потом настраивается объект PyTorch Sampler, который позволяет выполнить oversampling для датасета(это позволит модели учитывать миноритарные классы при обучении)
3. Создаём объект класса из числа обученных моделей PyTorch, замораживаем веса сети и заменяем последний линейный слой, который и будем обучать.

Функция потерь:

В качестве стартовой функции потерь была выбрана функция кросс энтропии, которая в большинстве случаев применяется в задачах многоклассовой классификации.

Разделение датасета:

Исходный набор данных был поделён на две части в соотношении 70/30 с стратификацией

Метрики качества:

В качестве метрики для обучения был выбран F1 score и accuracy. F1 позволяет нам более объективно оценить качество, в отличии от accuracy, а также она позволяет одновременно оптимизировать precision и recall.

Эксперименты:

Исходные настройки:

lr: [0.1, 0.01, 0.001],
scheduler: Exponential (gamma = [0.7, 0.8, 0.9])
optimizer: Adam (default)
Аугментации: HorizontalFlip
Кол-во эпох: 15
Batch size: 32

Перебор архитектур:

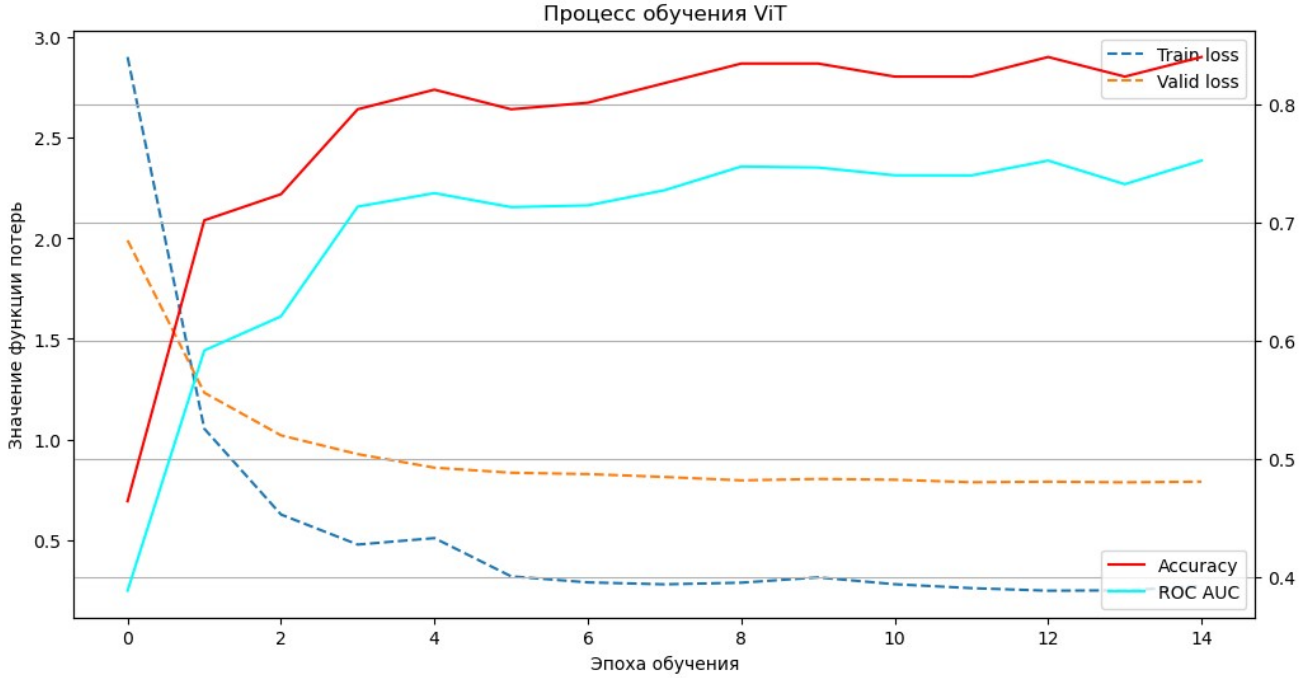
Для тестов возьмём ResNet(тк она обладает хорошими характеристиками скорости и качества), EfficientNet(её качество обычно выше чем у предыдущей модели, и размер меньше) и Visual Transformer(обладает механизмом внимания, что должно дать преимущество при выделении деталей на изображении)

Model	Learning rate					
	0.1		0.01		1e-3	
	ACC	F1	ACC	F1	ACC	F1
ResNet101(0.7)	60.77	48.37	60.22	48.41	48.62	41.37
ResNet101(0.8)	58.56	44.87	65.19	50.76	53.59	45.03
ResNet101(0.9)	53.04	40.53	61.88	46.93	58.01	44.75
EfficientNetB7(0.7)	64.09	53.87	67.4	56.89	50.83	45.4
EfficientNetB7(0.8)	62.43	51.28	67.96	56.43	53.04	48.07
EfficientNetB7(0.9)	62.98	50.77	69.06	57.22	58.56	51.82
Visual Trans(0.7)	78.45	63.21	81.77	69.74	83.98	**75.22**
Visual Trans(0.8)	76.24	59.03	81.22	69.89	83.43	72.95
Visual Trans(0.9)	74.03	57.51	80.66	69.18	**84.53**	73.74

* В скобочках указывается gamma для Exponential scheduler

По результатам эксперимента лучше всего себя показала архитектура Visual Transformer, как и предполагалось механизм внимания помог ей лучше справиться с задачей.

После обучения ViT архитектуры были построены графики обучения и на нём есть некоторые признаки переобучения модели (большие различия в функции потерь train и valid, valid loss вышел на постоянное значение):



Попробуем исправить ситуацию с помощью аугментации исходных данных:

Исходная конфигурация:

Модель: ViT
lr: 0.001
scheduler: ExponentialLR(gamma=0.7)
epoch: 15
batch_size: 32

Виды аугментаций:

HorizontalFlip(),
Rotate(0-15 градусов),
GaussianBlur(ядро 5),
RandomBrightnessContrast(изм. яркости = 0.2, изм. Контраста = 0.2),

Будем контролировать качество модели, а также разницу между значениями функции ошибок на тренировке и валидации. Переберём все комбинации данных аугментаций(кроме горизонтального разворота), при этом аугментации не применяются к валидационной выборке

HorizontalFlip	Rotate	GaussianBlur	Brightness	Train Valid diff	Accuracy	F1 score
+	-	-	-	-0.5212	**83.98**	**75.22**
+	+	-	-	-0.5006	83.43	75.21
+	+	+	-	-0.4848	81.22	72.92
+	+	+	+	0.0597	75.69	67.24
+	-	+	+	-0.0397	74.59	67.43
+	+	-	+	**-0.006**	75.14	67.57
+	-	-	+	-0.0291	78.45	69.94
+	-	+	-	-0.4952	83.43	74.85

Отсюда видно, что аугментации не смогли улучшить качество модели, поэтому дальше будем использовать только горизонтальный разворот

Смена функции потерь:

Попробуем сменить функцию потерь на logit adjactment loss, которая была специально разработана[iii] для задач с дисбалансом классов:

$$\ell(y, f(x)) = -\log \frac{e^{f_y(x)+\tau\cdot\log\pi_y}}{\sum_{y'\in[L]} e^{f_{y'}(x)+\tau\cdot\log\pi_{y'}}} = \log\left[1+\sum_{y'\neq y}\left(\frac{\pi_{y'}}{\pi_y}\right)^{\tau}\cdot e^{(f_{y'}(x)-f_y(x))}\right].$$

Аналогично реализации авторов была написана функция потерь для PyTorch и теперь можно провести некоторые эксперименты с ней.

Аугментации: HorizontalFlip
Oversampling убран из обучения, тк loss уже учитывает неравномерное распределение классов

τ	Accuracy	F1 score
0.6	82.87	73.38
0.8	**83.43**	74.55
1.0	81.77	**75.66**
1.2	76.8	70.88
1.5	67.96	65.02

Результаты похожи на результаты экспериментов авторов, т.е лучшая метрика получилась при τ = 1.

Авторы также говорят об эффективности применения аугментаций вместе с этой функцией потерь. Повторим предыдущий эксперимент на новой функции потерь

Исходная конфигурация:

Модель: ViT
lr: 0.001
scheduler: ExponentialLR(gamma=0.7)
epoch: 15
batch_size: 32

Виды аугментаций:

HorizontalFlip(),
Rotate(0-15 градусов),
GaussianBlur(ядро 5),
RandomBrightnessContrast(изм. яркости = 0.2, изм. Контраста = 0.2),

HorizontalFlip	Rotate	GaussianBlur	Brightness	Train Valid diff	Accuracy	F1 score
+	-	-	-	-0.4757	**81.77**	**75.66**
+	+	-	-	-0.4475	81.22	74.5
+	+	+	-	-0.4237	80.66	71.38
+	+	+	+	0.1138	74.03	65.02
+	-	+	+	0.1272	78.45	69.73
+	+	-	+	**0.0683**	74.03	67.11
+	-	-	+	0.097	73.48	65.84
+	-	+	-	-0.4421	79.56	71.53

Результаты получились такими же как и в прошлый раз, поэтому далее будет использоваться только горизонтальный разворот

Использование альтернативной модели

В результате предыдущего сравнения была подтверждена экспериментально эффективность модели ViT. Ключевой особенностью трансформеров является механизм внимания, которым не обладали другие модели, участвовавшие в сравнении. В результате поисков в сторону применения данного метода в задачах fine-grained image classification, была найдена архитектура, которая по результатам работы авторов[iv] дала хорошую точность в задаче классификации болезней растения. По статье авторов я реализовал аналогичную архитектуру с некоторыми поправками:

- В статье выход Attention блока после Softmax имеет один канал, в моей реализации Softmax применяется поканально для каждого пиксела.

- Также как указали авторы статьи можно добавить нелинейность для улучшения выразительности механизма внимания между свёртками 3x3 и 1x1, что и было сделано.

Исходная конфигурация:

lr = 0.001
batch_size = 32
weight_decay = 0.005
optimizer = Adam(defaul settings)
ExponentialLR(gamma = 0.7)
loss = CrossEntropyLoss(без весов, но с OVERSAMPLING)

Аугментации:

HorizontalFlip
Rotate(15 градусов)

Model name	Accuracy	F1 score	Train valid diff
ResNet50	62.43	45.9	-1.3024
ResNet101	63.54	49.17	-1.208
ResNet152	**66.3**	**54.78**	-1.2215
EfficientNetB0	56.91	44.97	-1.2581
EfficientNetB2	**66.3**	54.44	-1.2071
EfficientNetB5	60.22	47.03	**-1.1475**
EfficientNetB7	61.88	47.31	-1.2852

По этим результатам можно увидеть, что данный подход дал очень сильное переобучение моделей, несмотря на аугментации и регуляризацию, а также разница функции потерь между обучением и валидацией увеличилась.

Такой эффект происходит скорее всего из-за большого линейного слоя, который находится в конце. До этого в моделях был avg_pooling, который ограничивал линейный слой размерами 2048xКол-во классов, а в текущей реализации автор говорил о преобразовании квадратной матрицы выходного слоя основной модели в вектор, который имеет размерность 25600xКол-во класов, а тк линейные слои склонны к переобучению, и размер слоя прямо на это влияет, то и данная архитектура не может нормально обучаться на таком маленьком наборе данных.

Аналог альтернативной модели

Чтобы решить проблему с большим линейным слоем в конце и не нарушать кардинальным образом структуру сети я нашёл альтернативный способ применения механизма внимания, который встраивается внутрь сети[v].

Для обучения применялась разная скорость для backbone сети и новых блоков. Также протестирован вариант с неизменяющимися параметрами предобученной модели(freeze)

Исходная конфигурация:

lr = 0.001
classifier_lr = 1e-4
backbone_lr = 1e-5
batch_size = 32
weight_decay = 0.005
optimizer = Adam(defaul settings)
ExponentialLR(gamma = 0.7)
loss = CrossEntropyLoss(без весов, но с OVERSAMPLING)

Модель	Accuracy	F1 score	Train - valid diff
ResNet152 (freeze)	2.21	0.47	-0.0098
ResNet152	1.66	0.35	-0.0235
EfficientNetB2 (freeze)	3.31	1.79	-0.0293
EfficientNetB2	3.87	2.38	-0.0238

По результатам данного эксперимента можно сделать вывод, что встраивание блоков подобного типа невозможно без полного переобучения сети. Я также пробовал изменять скорость обучения, кол-во блоков и их позиции, но просто дообучить модель под данную задачу не получается.

Выводы:

В данном исследовании я провёл серию различных экспериментов и изучил статьи других авторов на тему *Fine-Grained image classification*.

- Лучше всех себя показала модель, основанная на трансформере(ViT)
- Эффективными методами для борьбы с дисбалансом классов оказались: смена loss функции на ту, которая учитывает дисбаланс классов и oversampling.
- Аугментации слабо влияли на проблему с переобучением модели.

Для улучшения качества данных методов нужно добавить больше данных.

[i] Bird Species Classification using Transfer Learning with Multistage Training
[ii] Address class imbalance easily with Pytorch
[iii] Long-Tail Learning via Logit Adjustment
[iv] Fine-Grained Image Classification for Crop Disease Based on Attention Mechanism
[v] CBAM: Convolutional Block Attention Module