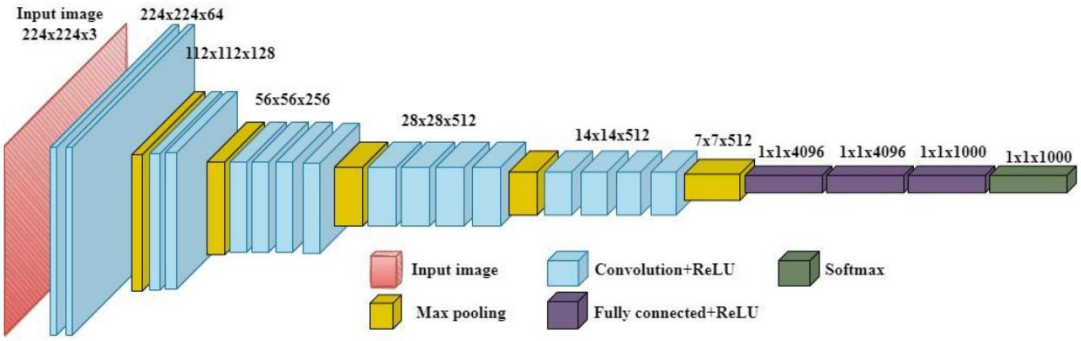
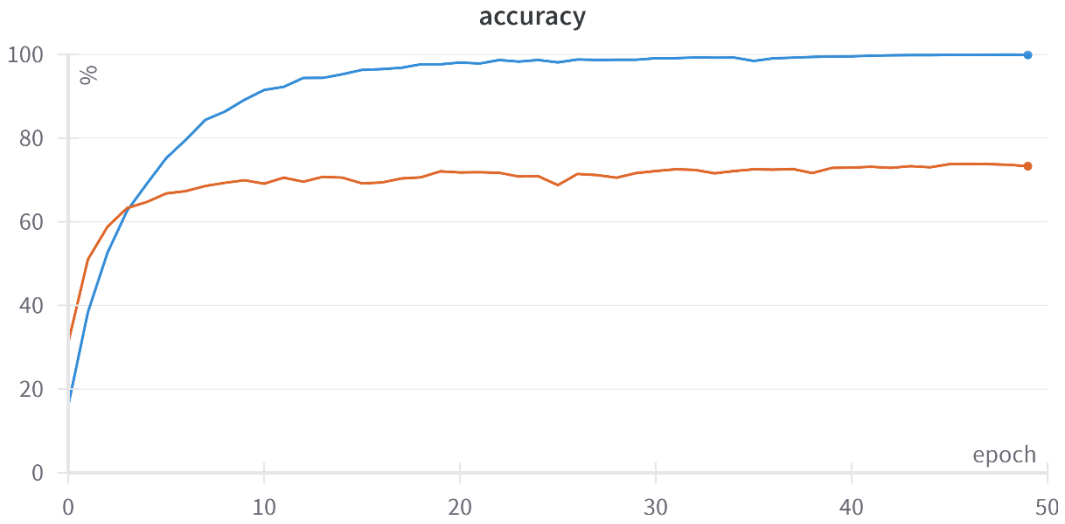
Report

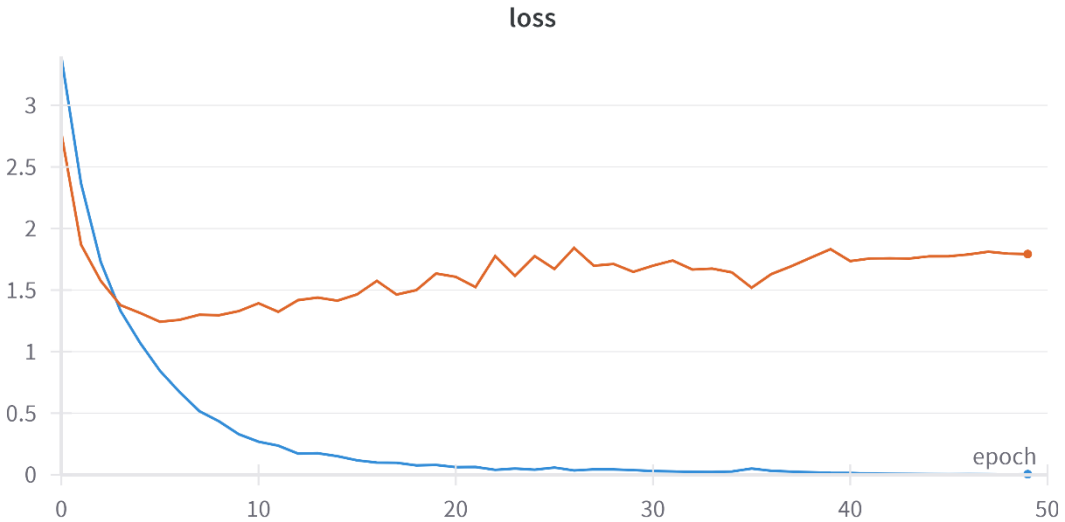
- Model Architecture

  Vgg19



- Validation Results





橘色線皆為 validation 的表現，其中 accuracy 最高可到 73.85%，loss 的話則是在訓練沒多久後即降不下來

- Problems and Solutions
  1. 想用 wandb 紀錄訓練過程的結果，可是 init()後仍然沒有紀錄，原來是沒有 log 起來
  2. 當時在 kaggle 上有訓練好一個 model，下載下來後在本地跑 validation 試試看，結果準確率個位數，原來問題出在 kaggle 上與本地端兩者回傳之 os.listdir 順序不一樣，導致 index 對不到，解決辦法便是直接給 list，規定順序就是如此，就不會有問題了
  3. 單張照片下去預測時遇到 mat1 and mat2 shapes cannot be multiplied，原來是 model 需要第四個維度(為 data 數)，因此我們做 reshape 再送進 model 即可運行
  4. Test 的 Dataframe 中有一欄為 index，在 to_csv 中會多一個欄位，導致出來的 csv 會與要求不同，因此我們手動刪掉該欄即可
- Methods to improve accuracy

```python
# Normalize 和 totensor
train_transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

val_transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

test_transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

標準化處理 image，使 model 更容易收斂

```python
model = models.vgg19(weights='DEFAULT')
```

當初先使用的是 vgg16，但 vgg19 表現更勝過他(vgg16 的 accuracy 約為 50 幾%，但換成 vgg19 就升高到 70%左右)

```python
# 計算自定義 weight (先計算每個 class 數量) - 實作 penalized loss
class_num = []
for c in cls_list :
    p = os.path.join(train_path , c)
    num = len(os.listdir(p))
    class_num.append(num)
weights = []
for i in range(len(class_num)) :
    weights.append(sum(class_num) / class_num[i])
class_weights = torch.FloatTensor(weights).to(device)

# 套入 loss 的 function 來改變計算權重
loss = nn.CrossEntropyLoss(weight = class_weights)
optimizer = optim.SGD(model.parameters(), lr, momentum=0.9,weight_decay=5e-4)
# optimizer = optim.Adam(model.parameters(), lr=1e-3, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)
```

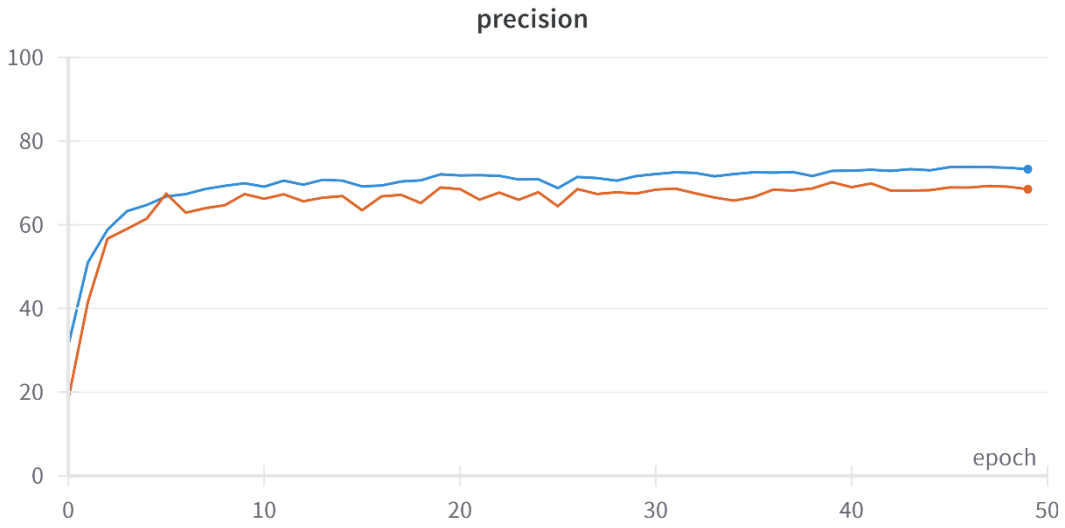
因為 dataset 屬於 imbalanced dataset，因此我們進行 penalized loss 的動

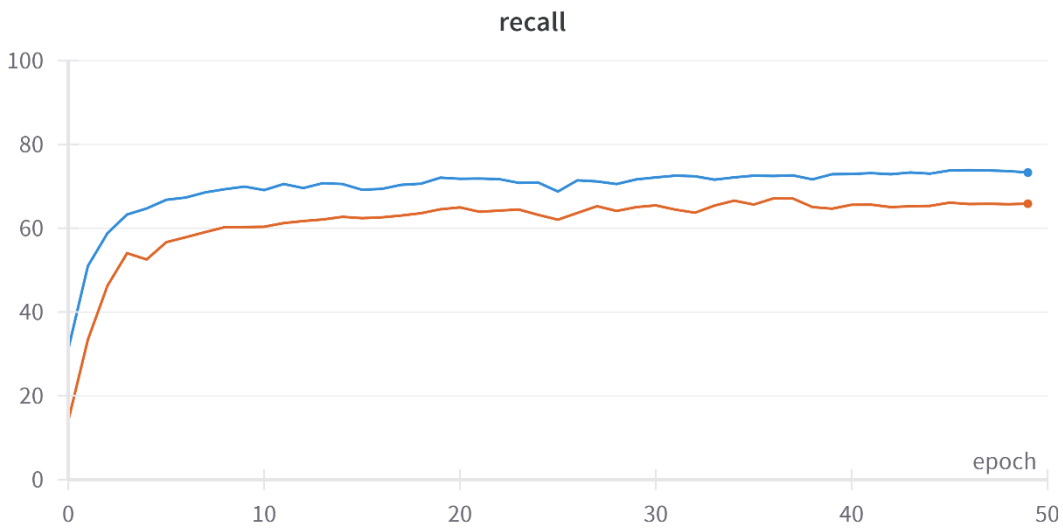
作，讓少樣本的 class 權重高一點，並把他放入 loss function 進行改權重。除此之外，我還試過使用 Adam 當作 optimizer，但在此 dataset 會卡在 local minimum

```
# 實作 learning rate 隨著 epoch 改變
def adjust_learning_rate(optimizer, epoch):
    if epoch <= 20:
        lr = 0.001
    elif epoch <= 40:
        lr = 0.001
    else:
        lr = 0.0001
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr

for epoch in range(epochs):
    # train
    train_epoch_loss = 0.0

    train_class_correct = list(0. for i in range(len(classes)))
    train_class_total = list(0. for i in range(len(classes)))

    # 每個 epoch 前都去看看要不要調 learning rate
    adjust_learning_rate(optimizer, epoch)
```

這邊我進行定義 adjust_learning_rate，讓訓練前期能大膽點前進，在後期能慢慢收斂下來，避免使用過大的 learning rate 使他震盪

- Evaluation metrics intro





這邊我們使用 Precision-Recall 來當作指標，並分別計算 macro 與 micro 出來，並畫成 line chart(橘線為 macro、藍線為 micro)