Homework 2 Machine Learning Andrea Morelli 1845525

Multiclass Image Classification

The problem

Multiclass image classification is the task of training an algorithm to classify images into one of several predefined classes. The algorithm must be able to accurately distinguish between multiple classes of objects, each of which may have complex and varied visual characteristics. In addition, the algorithm must be able to generalize to new, unseen images, which may contain variations in appearance or background. Multiclass image classification is an important problem in computer vision, with applications in different domains.

The dataset

The dataset is from Kaggle and contains images of characters from the "One Piece" anime and manga series.
The dataset is balanced, with approximately 650 images for each of the 18 classes.
The dataset has been augmented by rotating and inverting in color the original images, resulting in a total of approximately 11,700 images.
The images in the dataset will be used to train and test the image classification algorithms.
The images are in the folders named with the corresponding name of the class..
I created an annotation file that is then used to index the path and label of those images.
The dataset is splitted in training and testing set of respectively 80% and 20%.
Once the dataset was organized and split into training and testing sets, I was ready to train and evaluate my image classification models.

Tecnologies Used

I used the pytorch framework to implement and train my models.Pytorch is a popular machine learning library that provides a range of tools and libraries for training and evaluating deep learning models,it is intrinsically built together with autograd (useful for the backpropagation) and is similar to numpy.
In addition to pyorch, I also used the weights and biases library to track and visualize the training and evaluation of my models. Weights and biases is a cloud-based platform that allows me to log and analyze the results of my experiments, including metrics. This helped me to understand how the models were performing and identify any patterns or trends in the data

The metrics

To evaluate the performance of the models, i used a range of metrics during both the training and testing phases. During training, I tracked the loss of each batch to understand how well the model was learning and to identify any potential issues.. During testing, I used a variety of metrics to assess the accuracy and reliability of my models. These included overall accuracy, which measures the percentage of images that were correctly classified, as well as more detailed metrics such as F1 score, precision, and recall. Here i describe how those metrics are built and what they represent.

Overall accuracy: This is the percentage of images that were correctly classified. It is calculated as the number of correct predictions divided by the total number of predictions.
Overall accuracy = (number of correct predictions) / (total number of predictions)

F1 score: This is a measure of the balance between precision and recall. It is calculated as a combination of precision and recall.
F1 score = 2 * (precision * recall) / (precision + recall)

Precision: This is the proportion of true positive predictions among all positive predictions. It is calculated as the number of true positive predictions divided by the total number of positive predictions.
Precision = (number of true positive predictions) / (total number of positive predictions)

Recall: This is the proportion of true positive predictions among all actual positive examples. It is calculated as the number of true positive predictions divided by the total number of actual positive examples.
Recall = (number of true positive predictions) / (total number of actual positive examples)

for the precision recall and the f1 score i used the macro option but since the dataset is balanced the accuracy is a good metric for the performances of the models.

The models
1)Mlp
Multi-Layer Perceptron (mlp): The mlp is a type of neural network that consists of a series of fully-connected (fc) layers, with each layer containing a set of neurons that are connected to all the neurons in the previous layer. the mlp has six fc layers in total, with the final layer producing an output of size 18, corresponding to the number of classes in the dataset. Between each fc layer, there is a ReLU activation function, which introduces non-linearity to the model and allows it to learn more complex patterns in the data.

2)Resnet
ResNet: ResNet (short for Residual Network) is a type of convolutional neural network (CNN) that was introduced in 2015 and has achieved state-of-the-art results on a number of image classification benchmarks. One of the key innovations of ResNet is the use of skip connections, which allow the model to learn residual functions that are added to the outputs of earlier layers. This can help reduce the vanishing gradient problem and enable the model to learn deeper networks without suffering from degradation. In my implementation, iused ResNet18, which is a variant of ResNet that has 18 layers.

I used three different versions of ResNet18: (1) a pre-trained model with the final fc layer fine-tuned, (2) a pre-trained model with all layers fine-tuned, and (3) a model trained from scratch.

3)MobileNet
MobileNet: MobileNet is a type of CNN that was designed to be lightweight and efficient, making it suitable for use on mobile devices. It uses depthwise separable convolutions, which can significantly reduce the number of parameters and computational complexity compared to standard convolutions. MobileNet is available in several versions, including MobileNetV1 and MobileNetV3. In my implementation, i used MobileNetV3, which has improved accuracy and efficiency compared to earlier versions. I used a pre-trained MobileNetV3 model with all layers fine-tuned.

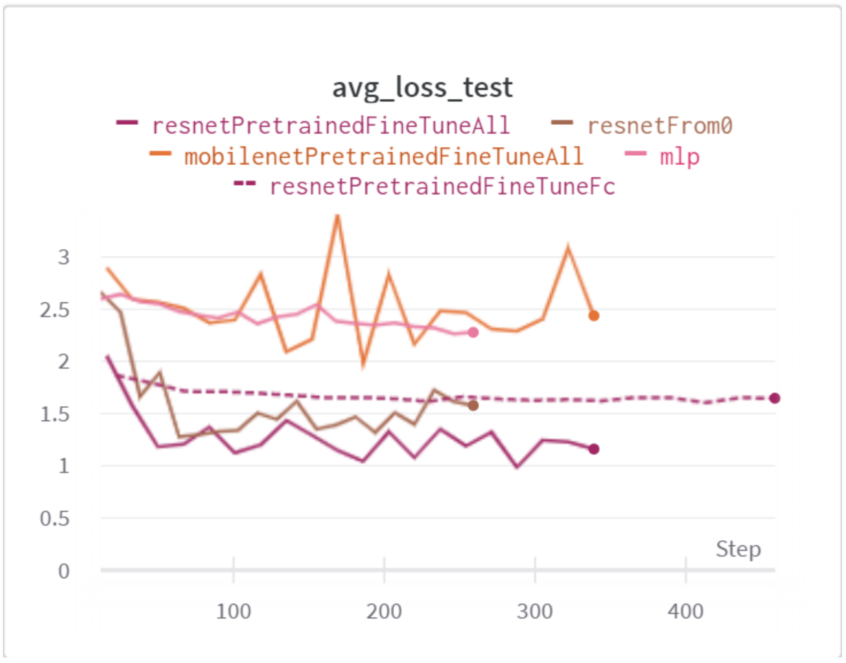
The results

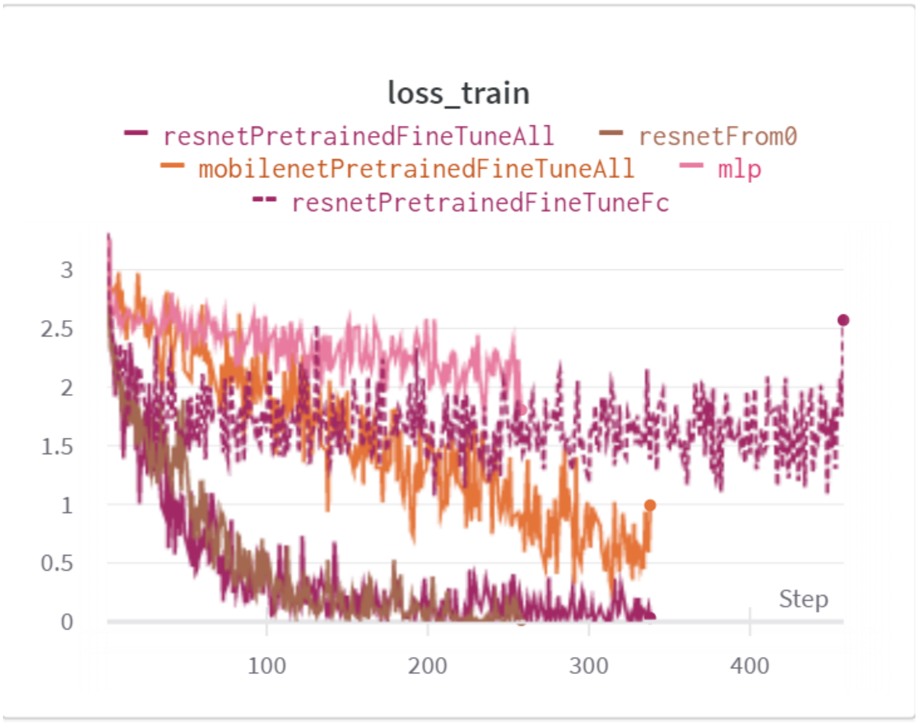
I used different configurations of the training.

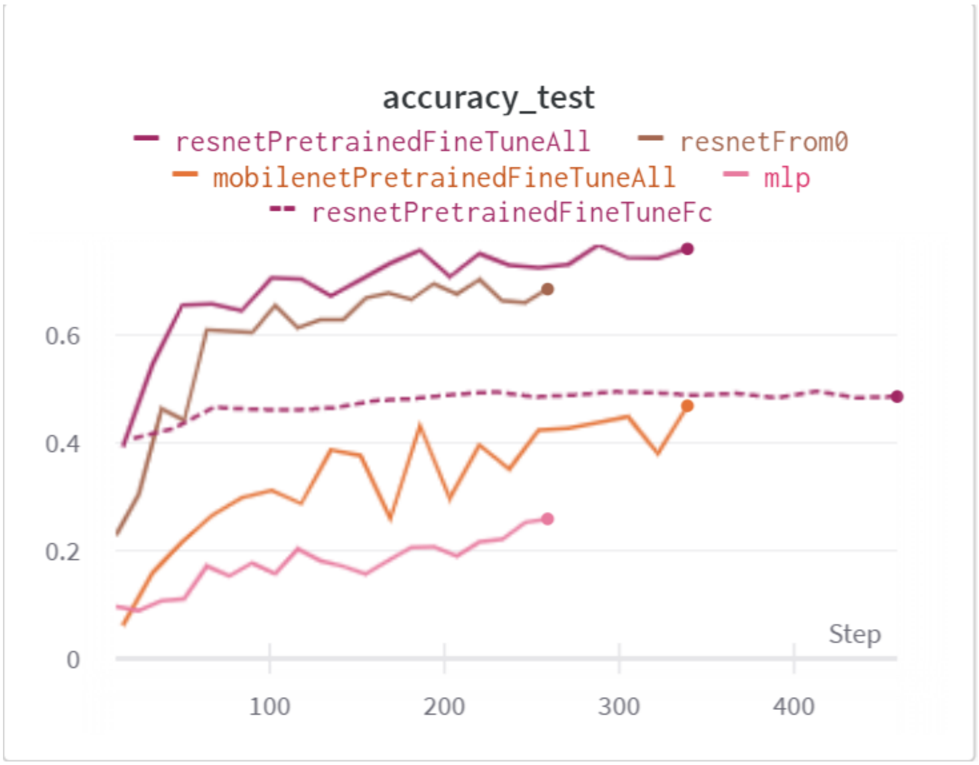
1)In the first configurations I omitted the f1 score, the precision score and the recall score. In the train part i logged in weights and biases every 20 batches the loss of the batch, while in the testing phase i logged the average loss and the accuracy. I used the Adam optimizer without declaring the learning rate so that it could optimize it automatically. In this way it uses momentum what helps with performances, going faster in the minimum and without getting stuck to bad local minimums.
I used in this first configuration a resize of the images to 50 px in width and in height.

This as I will describe later is a really important factor that impacts the performances of the models.

Those were the results :







The pre-trained and fine-tuned ResNet models may have performed better than the other models because they were able to take advantage of the knowledge learned by a model trained on a large dataset, and then fine-tune that knowledge to fit the specific characteristics of the new dataset. The ResNet architecture may have also contributed to their good performance, as it is designed to be very deep and to use skip connections, which can be well-suited for image classification tasks.

The ResNet model that I trained from scratch may have performed less well than the pre-trained and fine-tuned ResNet models because it did not have the benefit of being initialized with weights that were already learned on a large dataset. This means that the model had to learn all the features from scratch, which can be more challenging.

The ResNet model that was pre-trained and fine-tuned only on the fully-connected (fc) layer may have performed less well than the model that I fine-tuned with all layers because it was not able to take full advantage of the knowledge learned by the pre-trained model. By only fine-tuning the fc layer, the model may not have been able to effectively adjust the weights of the earlier layers to fit the specific characteristics of the new one piece dataset .
The general features of the pretrained are not further optimized for the specific dataset so there may be a loss in accuracy because of that.

MobileNet: The MobileNet model may have performed less well than the other models because it uses a different architecture that may not have been as well-suited for the image classification task. MobileNet architectures are designed to be efficient and lightweight, which can make them well-suited for tasks that require fast inference, but may not be as effective for tasks that require high accuracy.

The mlp model may have performed less well than the other models because it does not use convolutional layers, which are well-suited for processing image data. mlps are fully-connected networks that do not have the ability to learn local features from the input data, which can make them less effective for tasks that require understanding the spatial relationships in the data, such as image classification. Also the model was significantly smaller then all the above networks so that could also have been a problem.

As we can see there is some overfitting that does not allow the models to perform better (the loss of the test set is a lot higher than the one of the train set). I also tried to use weight decay as a form of regularization but did not obtain reasonable results. I also tried a custom fixed learning rate and in this way i reached the same performances but I reached the same maximum performances more slowly.
I also tried dropout just in the mlp but i was underfitting with the same architecture.

2)
What really changed the performances of the network was to have an initial resize bigger in this case 200 px both in height and in width.
In this second configurations i used 32 as batchsize, bigger resize(200x200), weight decay, variable learning rate with the adam optimizer .

The following are the performances of the models with this second configuration:

The performances were really better as we can see. In this configuration i also added the f1 score, the precision and the recall score that were omitted in the previous configuration.

The resnet could reach a maximum dimension of something like 250 x 250 px i resized everything to 200 x 200 and i reached really good results.

The performances compared to the previous configuration are better especially in the resnet architecture.
I suppose that the main reason is that the resnet model is very deep and is able to find really good features useful for the classification task and giving images that have a greater resolution could improve the finding of those good features. Also The details are a lot more defined and having inserted in the dataset also images with inverted colors, the model has to have a good grasp of the contours of the images to

be able to classify the images and a greater resolution helps the model in that regard.

The performances are somewhat impacted because there is the need of dealing with more pixels but using a gpu runtime with cuda allows to deal with this increase in the number of parameters.

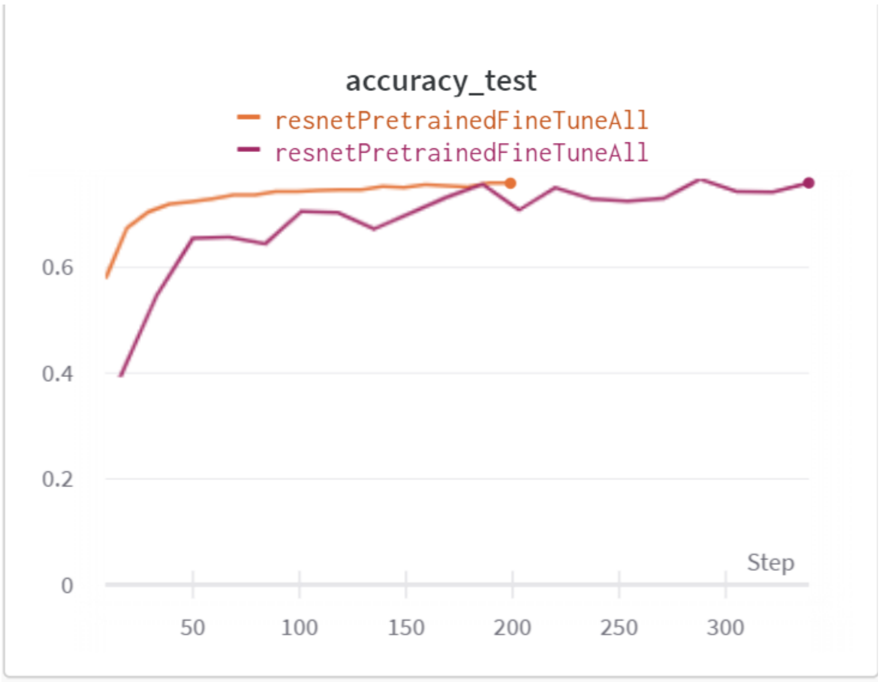
The models reach their maximum performances in approximately the same time with respect to the first configuration and that was a strange thing i noticed that i did not expect to happen.
I thought that having less parameters to optimize would allow to reach the optimal performances faster but that is not the case maybe because the optimizer is learning optimizing the learning rate to learn similarly in percentage to the model with a smaller input size image.

Different Hyperparameters

The biggest differences in the results as we saw was in the initial dimension of the image after the resize.Passing from 50 50 to 200 200 the performances were really impacted.

On the contrary adding weight decay or changing for example the learning rate would not change the performances.
A thing that i found being really helpful was increase the batch size.
The performances would improve by a really small margin but the maximum performances in the test set would be achieved faster.
Increasing the batch size allows to have a better estimate of the entire dataset loss so the gradient could go more easily to a minimum but there could be a problem of being stuck in a bad local minimum in this way since there is not a “noise” in the gradient that allows to escape minimums.
Also its more intensive to calculate everything on a bigger batch so parallelize with gpu and cuda its helpful to not degrade the performances and have approximately the same time of training.

here is a graph with the differences between a batch of 64(the upgraph) and the 32 batch size:



everything is smooter using a bigger batch size but the final performances are approximately the same.

Conclusions

In this report, I explored the problem of multiclass image classification using a Kaggle dataset of One Piece characters images. I used several different models and technologies, including multi-layer perceptrons (MLPs), ResNet, and MobileNet, and evaluated their performance using a variety of metrics, including loss, accuracy, F1 score, precision, and recall. My results showed that all of the models achieved good performance, with the Resnet pretrained and from scratch models achieving the highest scores. However, there is still room for improvement, and I believe that further experimentation with different hyperparameters, optimizers,models and other techniques such as data augmentation could yield even better results. There are many models and approaches that could be tried for example the new vision transformers architecture with data augmentation, and I believe that continued exploration of these options could lead to even more promising results.