Analytics Vidhya: Game of deep learning computer vision hackathon

This document will cover the following topics:

1. A brief on the approach, which you used to solve the problem.
2. Which data-preprocessing ideas really worked? How did you discover them?
3. Mention the pre-trained models with links used for building the model.
4. How does your final model look like? How did you reach it?
5. What are the key takeaways from the challenge, if any?
6. According to you, what are the 5 things a participant must focus on while solving such problems?

Now let's discuss the details in subsequent sections.

A) Problem solving approach

The problem given was to identify different types of ships from the pictures. The training set contained around 6.2K images across all the ship categories while the test set was 2.6K.

Fastai library provides a very convenient way for training image classifier models using CNN. I will be using this library to a great extent.

From deep learning context, this amount of images for training is very less. Hence we will heavily depend upon different transformation techniques as well as weights of pre-trained models. Data transformation techniques refer to tweaking of same image using various approaches such as horizontal flipping, random rotation, zooming, lighting, warping, padding etc. to create different versions of same image such that while training at each epoch, our model gets different images as if we have much more images to give the model to learn. After the transformation, next step was to fix a particular image size to the whole data set.

Now, we are ready to create CNN model with pre-trained weights from resnet/ densenet architectures. While training the model, I also used `mixup` technique in some models. Details of this paper can be found here. The main idea behind this novel technique is stated below.

`mixup()` refer to a pretty aberrant feature which seems very un-familiar to us (human), however, this technique works better for computers. The theory behind is like this:

- We do not train directly on the raw data images instead the model is trained on mixes of images i.e. we add 2 or more images to combine a single picture by this: `new_image = t * image1 + (1-t) * image2` (not necessary to take only 2 images). By using this same technique, targets are changed as well `new_target = t * target1 + (1-t) * target2` ; where t is a float between 0 and 1.
- One thing to note that, the mixup model may perform better than the regular one but when you compare the training/validation losses, the mixup model has loss far greater than the regular one (although accuracy seems better in mixup model). This is because the mixup model predictions are less confident about the target. i.e. when we do prediction through our normal model, model seems pretty confident about the target (one target probability prediction will be close to 1 and others will be close to 0). When we predict through mixup model, the probabilities values of the targets will more likely to be close to one another.

After the model is created, we are now ready for training. First step is to find the correct learning rate, so that we can quickly train the model and we do not have do any guess work. Then, I trained the model for few epochs (with weight decay to not to overfit the model). Now it's time to unfreeze the entire model

weights and train the initial layers using very low learning rate and last layers using a high learning rate. This is because the initial layers are weights coming from pre-trained model (imagenet)which are responsible for finding particular pattern (e.g. corners, sides, shapes etc.) and we should not be changing them to a greater extent (as changing them will greatly affect our model's final score). The last layers, however, are responsible for finding patterns to differentiate ships. As the pre-trained model may/may not have parameters for finding exact ship category as per the problem (because the original pre-trained model may not have different ships as different categories), so we need to train the last few layers (at a higher learning rate) to arrive at correct weights according to our problem.
Now, we have trained initial and last layers at differentiated learning rates. Hence we are now confident that we have desired weights for the initial layers (after unfreezing and training). The next step was to continue training (last few layers) for few more cycles after freezing the weights of initial layers. This was done by manually freezing the initial layers and continually training the last layers.
At this point, we now have a model which can differentiate ships to a greater extent.
For the prediction of the test set, I used augmentation/ transformation technique again on the test data as well. Here we take the average of our regular predictions (with a weight beta) with the average of predictions obtained through augmented versions of the training set (with a weight 1-beta).

B) Pre-processing ideas

The pre-processing steps mainly include image transformation techniques and resizing.

- Horizontal flipping of images didn't give same level score as without flipping.
- Zoomed a higher bit (1.22X) than the fastai default as the ships seemed to cover less portion of the whole image.
- Vertical flipping was not done (as these pictures are not taken from far above like satellite images)
- Warping was set to zero (distorting the ship images might change the category of prediction resulting in lower score)
- Maximum rotation of the image was set to 10 degree (default) along with other default like maximum lighting (0.22).
- Reflection padding was used (fastai default)
- All the images were squished and final size was fixed to 299 X 299 or 484 X 484 or 599 X 599 according to different models.
- Batch size was also fixed according to the deeper network and higher resolution images.

C) Pre-trained models used

I have used 4 pre-trained models for training different models

i) Resnet101 ("https://download.pytorch.org/models/resnet152.pth")
ii) Resnet152 ("https://download.pytorch.org/models/resnet152.pth")
iii) Densenet161 ("https://download.pytorch.org/models/densenet161.pth")
iv) Densenet169 ("https://download.pytorch.org/models/ densenet169.pth")

D) Final model

Final model output was as a result of voting of 22 different model outputs created using different pre-trained models (above stated 4 models), different image sizes and with or without mixup technique.

All these 22 models have F1 score greater than 0.975 (public score), however, none single model reached 0.99 score. The final score (public) of greater than 0.99 reached after taking a voting of predictions done by all the 22 models.

E) Key takeaways

- Mixup technique proved to be very useful and the score of these models were better than regular models.
- Data augmentation is a must for any type of computer vision problems to achieve state of the art scores.
- Different pre-trained models works differently for any computer vision problems. It is always useful to try which pre-trained model is giving better result or for best results ensemble the different models trained using different pre-trained architectures.
- The higher resolution images always (mostly) improves scores (may be a very little) but before training the model on very high resolution images, doing cost-benefit analysis will help (to see that if benefit score improvement till 2/3/4/ decimals will outweigh computational cost)
- Training the data using 22 different models seem very computationally intensive as well as time consuming. But for these small type of small datasets, it is a very useful technique. (Although I was able to achieve same level of score (private score) up to 4 decimal places using voting of only 9 models, but the public score was less)

F) 5 things participants must focus

- Always set aside validation set to check the progress (with seed for reproducibility).
- Focus on augmentation techniques because mastering this allows how and when to apply such techniques to easily achieve state of the art scores.
- Initially do not focus on creating highly convoluted networks and/or high image resolutions, instead focus on basic parameters like setting correct learning rate, weight decay, different augmentation techniques etc. Steadily move for more deep networks and higher resolutions for fine-tuning and improving scores further.
- Be comfortable with the codes.
- Be ready to interpret the model (answer question like why the model predicted this particular image this particular image category)

Submitted by:

Narendra Sahu