Video Defect Classification on the QV Pipe Dataset

Santosh Thoduka
Department of Computer Science,
Hochschule Bonn-Rhein-Sieg
53757 Sankt Augustin, Germany
Email: santosh.thoduka@h-brs.de

***Abstract*—This report describes the method, training and inference process for the VideoPipe Challenge on Video Defect Classification. The QV Pipe dataset consists of short video clips recorded inside underground pipes, and is labelled with one or more classes; the classes refer to pipe defects that are present in the video clip (16) or the nominal class for clips with no defects present. Since multiple defects can be present in a video clip, we treat the task as a multi-label classification problem, with 17 classes. We train a ResNet-18 model, initialized by weights trained on the ImageNet dataset, with the addition of a transformer encoder layer for final classification. During training, 5 frames are sampled randomly from a video clip, and is provided as input to the model. During inference, 5 frames are sampled 5 times for a given video clip, and the mean output is used as the final prediction for that clip. Several image augmentation methods are used, and attempts are made to counter the heavy imbalance of classes in the data. Our best performing submission was a combination of results from three previous submissions, by averaging the scores for each video clip across the three submissions. The code can be found on Github[1].**

I. Introduction

The VideoPipe Challenge on Video Defect Classification is concerned with classifying defects that appear in videos in urban pipes. This report describes the method, training and inference process to solve the task. Submission of results on the test set are done through the Codalab website.

II. Dataset

The QV Pipe dataset consists of 6399 training videos and 3202 test videos, each with a duration between 0.7 and 385.2 seconds[2].

A. Training and Validation Split

We split the provided training set into 6000 videos for training and 399 videos for validation, so that performance on the validation set can be used to validate models and methods before submission of results to the Codalab leaderboard. For some initial submissions, we used 4000 training videos and 2399 validation videos, and for some final submissions we extended the training for 1 or 2 epochs with the full training set. No additional or external data was used, with the exception of using weights for a ResNet-18 model trained on the ImageNet dataset (as described in Sec. III).

[1]https://github.com/sthoduka/qvpipe_classification
[2]https://videopipe.github.io/qvpipe/

B. Class Distribution

The class distribution is imbalanced; therefore we experimented with balancing the distribution in a few different ways.

1) Class 0 contains the most number of samples, and also is the only class that occurs by itself in all videos. Presumably this means it is the nominal class. For the first 60 epochs, we remove 1050 samples of class 0.
2) For 3 epochs we only train with the poorly performing classes: 6, 8, 11, 13, 14 and 15

C. Sampling

For training, we randomly sample 5 frames from each video for a given epoch. Therefore, for each epoch, a different set of 5 frames are sampled from a given video. We also experimented with sampling up to 20 frames per clip, but did not see much of an improvement.

D. Augmentation

We first resize each frame to have a height of 300 pixels, perform random cropping to size 300x300 and finally resize the image to 224x224 pixels. During inference, the process is the same, except that we center crop the image to 300x300 instead. During training, additional augmentation such as `RandomAdjustSharpness`, `RandomAutocontrast`, `RandomHorizontalFlip`, `RandomVerticalFlip`, `RandomErasing`, `GaussianBlur`, `RandomRotation` and `RandomAffine` are performed. The pixel values are then normalized to have mean = $[0.485, 0.456, 0.406]$ and standard deviation = $[0.229, 0.224, 0.225]$.

III. Method

As the base model, we use a ResNet-18 model which was pretrained on the ImageNet dataset (obtained from the pytorch repositories[3]). The last classification layer is removed, and we use a TransformerEncoder with 4 heads and 8 layers, followed by a linear layer for classification (referred to as `TModel7`). The ResNet-18 weights are unfrozen and allowed to train with the same learning rate as the rest of the network.

For some earlier models, we also experimented with two linear layers with Dropout (referred to as `TModel2`) instead of the TransformerEncoder, though there was no significant difference in results. In both cases, the final layer has 17 outputs, one for each defect class and the nominal class.

[3]https://pytorch.org/vision/0.9/models.html#classification

Since we sample 5 frames per video clip, the output from the ResNet-18 model is a feature vector of size 512 for each frame. For `TModel7`, this is treated as the input sequence of tokens, with an additional class token. The output corresponding to the class token is passed through a linear layer for classification. In the case of `TModel2`, the features from all frames are concatenated before passing it through the remaining network. That is, each video results in a feature vector of size $512 \times 5 = 2560$.

1) Training Process: We use a batch size of 64 for training, using Stochastic Gradient Descent as the optimizer, with momentum of 0.9 and a weight decay of $1 \times e^{-4}$. We use a One Cycle learning rate scheduler[1] with two phases, with a maximum learning rate of 1.0. Since the task is a multi-label problem and the class distribution is imbalanced, we use focal loss[2] as the loss function (i.e. each output of the final layer is expected to be a value between 0 and 1, with 1 as the positive class). The labels are represented as a multi-hot encoded vector with the positive class represented by 1, with the remaining classes set to 0. The mean average precision (mAP) on the validation set is monitored through the training process, and the model with the best mAP is chosen for producing the results with the test set.

2) Inference Process: During inference, no data augmentation is used and 5 frames are sampled per video just as in the training step. However, we also perform the sampling 5 times for each video clip, resulting in 5 output logit vectors for each video clip (i.e. we sample 5 frames 5 times for each clip, performing a forward pass 5 times to get the output logits). The final scores for a video are calculated by applying the sigmoid operation on the mean of the 5 outputs.

For our final submission, which achieved an mAP of 59.91 on the test set, we combined the results from three different training runs (submissions 6, 7 and 9, each of which individually had mAPs of 58.0, 56.47 and 54.04) by averaging the softmax scores for each video clip from the three models. Some details of the three models are listed below:

1) learning rate of 0.1, weight decay of $1 \times e^{-5}$, MultiStep learning rate scheduler, no rotation or affine transforms for augmentation, `TModel2`, binary cross entropy loss (without focal loss)
2) learning rate of 0.0001, trained for 355 epochs, One cycle LR scheduler, no rotation or affine transforms, `TModel2`, focal loss
3) parameters as described in this report using `TModel7` (with training of 3 epochs using only poorly performing classes, and 2 epochs with the full training dataset)

IV. Conclusion

This report describes our approach for the Track on Video Defect Classification of the VideoPipe challenge, including the use of the dataset, learning model used, training and inference process and data augmentation. A ResNet-18 network was used as the base-model (pretrained on ImageNet), with two variants used for the final classification layers. Focal loss was used due to the imbalanced dataset. Only 5 frames were sampled randomly per video, though for inference this process was performed 5 times. The best performing result was a combination of the outputs from three separately trained models.

References

[1] L. N. Smith and N. Topin, "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. International Society for Optics and Photonics, 2019, p. 1100612.

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2980–2988.