Mar 29, 2024

AI6126 – Advanced Computer Vision Project # 1

Reinelle Jan Bugnot
G2304329L

I. Model Architecture

For this project, I implemented the **SE-ResNext50_32x4d**, which is an image classification model with Squeeze-and-Excitation channel attention and features: ReLU activations, single layer 7x7 convolution with pooling, 1x1 convolution shortcut downsample, grouped 3x3 bottleneck convolutions and is trained on ImageNet-1k in Apache Gluon using Bag-of-Tricks based recipes [1].

The architecture is a relatively small and simple model with 50 layers and a cardinality of 32 (number of groups for the grouped convolutions), where each group has 4 channels. Fig. 1 shows an illustration of a simple block of this network.

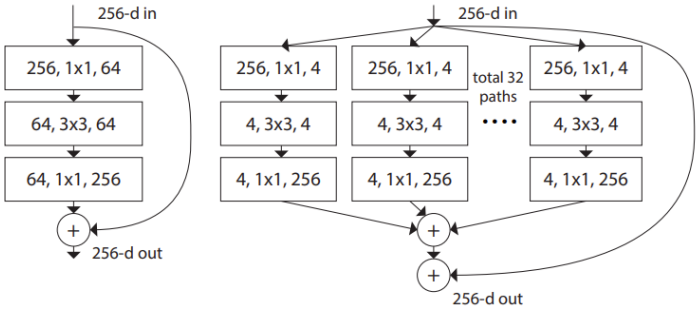

Fig. 1. Left: A block of ResNet. Right: A block of ResNeXt with Cardinality of 32 and 4 channels for each group [1].

SE-ResNeXt50_32x4d also takes advantage of “squeeze-and-excitation” (SE) blocks as shown in Fig. 2., to perform dynamic channel-wise feature recalibration.

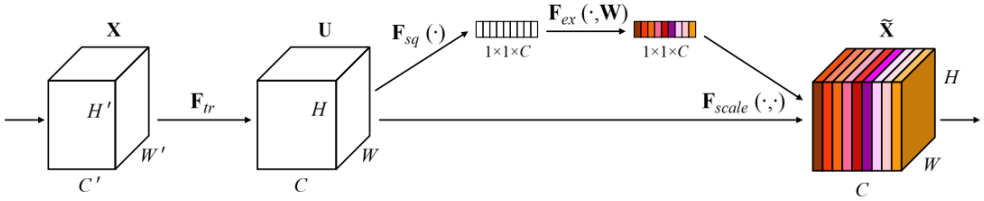

Fig. 2. Squeeze and Excitation Block [2]

For this project, I froze all the layers of the model except for the final bottleneck block, and then added a 512x128x26 fully connected network to serve as my prediction head. The 26 output nodes corresponds to the 26 attribute labels, which will be divided and matched into their corresponding 6 categories later in the pipeline. With this, I have a total of **17,242,394 trainable parameters**.

II. Loss Function

The Loss Function used for the project is a weighted cross-entropy loss with label smoothing. The 26 output logits of the model is sliced into 6 categories whose slice lengths are [7, 3, 3, 4, 6, 3], which are the number of classes per category in the problem. The weights used for the cross-entropy loss are tensors of the same length, whose values are calculated as follows:

$$w_i = \frac{5000}{220N_i}$$

where w_i is the weight for class i for all i from 1 to 26, and N_i is the total population of class i in the training set. This weight strategy therefore gives more weight to classes with low frequency and vice versa, helping to mitigate the effects of the class imbalance observed in the dataset.

III. Results

Fig. 3 shows the training and validation curves generated. Big emphasis was placed on regularizing the model in order to bring the training and validation curves closer together and solve my initial overfitting issue, where the training accuracy converges to near 95+% while my validation is stuck at 60%.

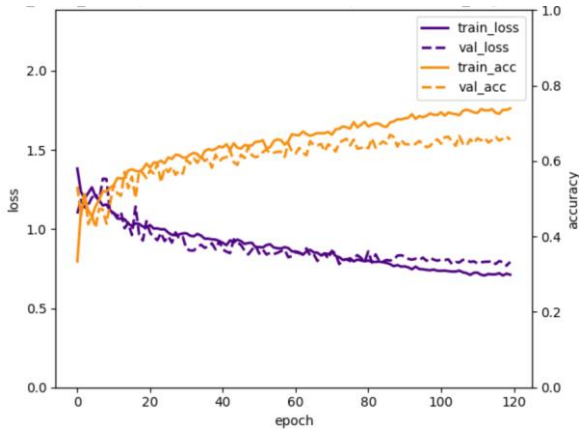

Fig. 3. Training and Validation Curves of Best Model

Interestingly, my highest validation accuracy for this particular experiment was around 66.3%. Yet, on model inference, the predictions achieved 70.02% accuracy on the test set, as shown in Fig. 4, which initially landed me on top 9 of the leaderboards; but is now pushed back to top 16 as of Mar 29, 2024. For me, this proves the power of properly regularizing your model in order to strengthen its generalization capabilities. The training specifications that I implemented to get this result in shown in Table I.

Table I. Winning Parameters

Parameter	Value
epochs	120
batch size	128
cosine annealing	True
learning rate	0.005

learning rate warmup	True
weight decay	0.0008
dropout	0.5
label smoothing	0.01
data augmentation	True

#	SCORE	FILENAME	SUBMISSION DATE	SIZE (BYTES)	STATUS	✔	
1	0.6305627377	prediction.zip	03/27/2024 05:09:32	211676	Finished		+
2	0.6259856015	prediction.zip	03/27/2024 07:48:34	215382	Finished		+
3	0.6438846953	prediction.zip	03/27/2024 11:43:25	224642	Finished		+
4	0.6330593329	prediction.zip	03/27/2024 13:33:17	230576	Finished		+
5	0.7001765455	prediction.zip	03/27/2024 15:04:29	232927	Finished	✔	+
6	0.6957052584	prediction.zip	03/28/2024 03:52:17	246401	Finished		+

Fig. 4. Test Accuracy

IV. Specs of Training Machine

I fully utilized the SCSE GPU Cluster to train my models, using the two QOS assigned to me: q_amsai and q_dmsai. The available hardware information are as follows:

Table II. Hardware Specifications

	CPU	GPU	memory	MaxWall
q_amsai	7	1	12G	8 Hrs
q_dmsai	10	1	30G	8 Hrs

My training strategy is to use the smaller q_amsai for parameter tuning, and then using q_dmsai for deep training with large epochs for submission.

V. Implemented Strategies

Learning Rate Warmup + Cosine Annealing

For learning rate scheduling, I implemented an initial learning rate warmup where, for the first 5 epochs, the learning rate is gradually increased from 0 to the initial learning rate (of 0.005), and then cosine annealing is performed for epochs 6+. Fig. 5 illustrates the learning rate strategy implemented for this experiment.

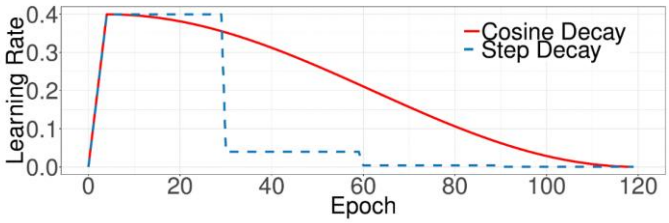

Fig. 5. Learning Rate Warmup + Cosine Annealing [3]

Tuning the Batch Size

In this project, I've experimented with different batch sizes and found that the model performs well on batch sizes of size 128 and 256. The best performing model generated used a batch size of 128.

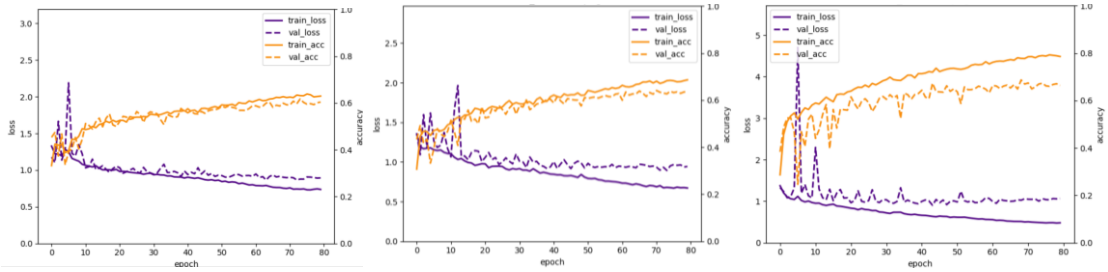

Fig. 6. Experimenting with Different Batch Sizes: 64 (left), 128 (center), and 256 (right)

Data Augmentations

Fig. 7 shows the data augmentations implemented in this project. The images are first resized to a standard 224x224 image dimension to fit the input dimension of our model. Afterwards, center-cropping was performed in order to minimize the "backgrounds" of certain images, effectively zooming into the clothes that are usually in the center of the image. The images are then resized back to 224x224.

Afterwards, standard augmentation techniques were implemented such as horizontal flipping, color jitter, random affine, and normalization.

```
# Transform Parameters
train_transform = transforms.Compose([
    transforms.Resize([224, 224]), # Resize
    transforms.CenterCrop([200, 160]), # Crop Center (to eliminate white background)
    transforms.Resize([224, 224]), # Resize Again
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.3),
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.5, 1.5), shear=15),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Fig. 7. Data Augmentations

Weight Decay

Weight decay is a regularization technique used in machine learning to prevent overfitting. It works by adding a penalty term, usually the L2 norm of the weights (all the weights of the model), to the loss function [4]. Fig. 8 shows sample weight decay experiments done during the tuning process. The best model results was achieved with a weight decay of **wd=0.0008**.

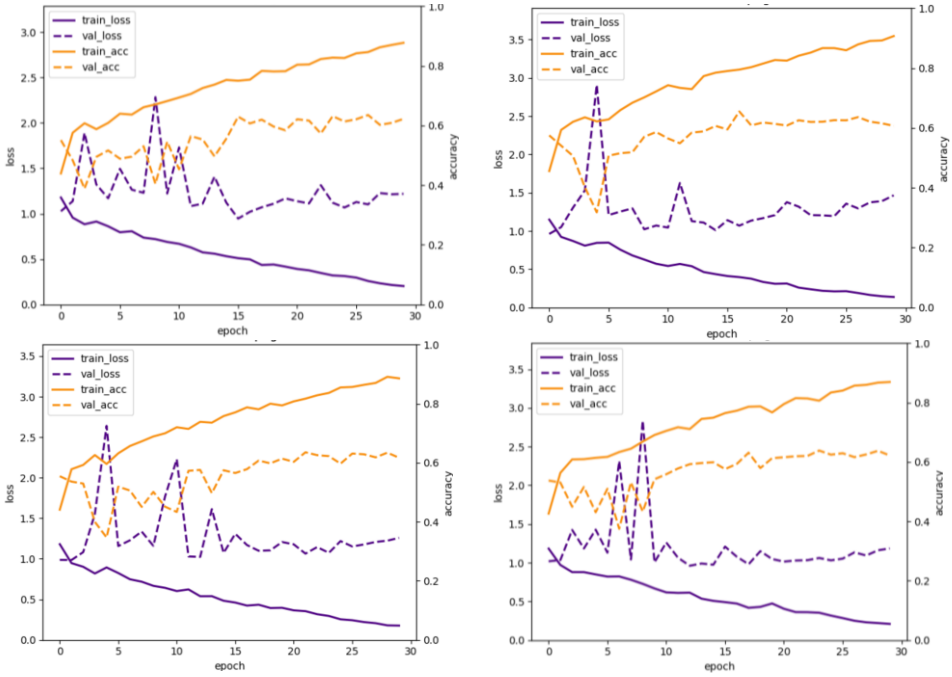

Fig. 8. Weight Decay Experiments: wd=0.0003 (top left), wd=0.0005 (top right), wd=0.0008 (bottom left), and wd=0.0095 (bottom right)

Label Smoothing

Label smoothing is a regularization technique used in machine learning, especially for classification tasks. It softens the "hardness" of one-hot encoded labels, improving model training and generalization by introducing noise for the labels using the following formula:

$$y_{\text{true}} = y_{\text{true}} * (1.0 - \varepsilon) + 0.5 * \varepsilon$$

Label smoothing is a very potent regularization technique. For this project, a small epsilon ε value of **0.01** was used and it was able to bump up the performance of the model by quite a bit [5].

Dropout

Lastly, some dropout layers were introduced to the fully-connected layers of the prediction head to add extra regularization. A **dropout of 0.5** was found to be best. Fig. 9 shows the experiments implemented to tune this parameter.

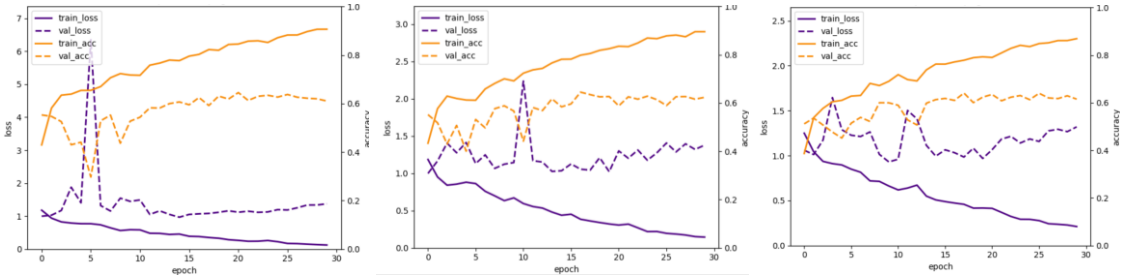

Fig. 9. Dropout Experiments at 30 epochs: dropout = 0.2 (left), dropout = 0.3 (center), and dropout = 0.5 (right).

References

[1] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated Residual Transformations for Deep Neural Networks,” arXiv preprint arXiv:1611.05431, Nov. 2016.

[2] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks,” arXiv preprint arXiv:1709.01507, Sep. 2017

[3] Y. Jeong, “Are You Sure That You Can Implement Image Classification Networks?,” Towards AI, March 25, 2022. [Online]

[4] S. Yang, “Deep learning basics — weight decay,” Analytics Vidhya, Medium, Sep. 3, 2020. [Online]

[5] A. Rosebrock, “Label smoothing with Keras, TensorFlow, and Deep Learning,” PyImageSearch, Dec. 30, 2019. [Online]