COMP9444 Final write up

Jia (Andrew) Wu, Yifan He

August 2021

Contents

1 Architecture

1.1 Overall architecture

Since the problem is an image classification task, we used a convolutional neural network (CNN) architecture, which can easily capture the complex features in images and requires far fewer parameters than a dense network. The overall architecture of convolutional blocks followed by fully connected layers was inspired by VGG-Net [2].

Our model contains four 2D-convolutional and maxpooling blocks with seven convolutional layers and four max pooling layers in total. This is followed by an adaptive average pooling layer and three fully connected layers. All convolutional and fully connected layers were succeeded by a batch normalisation layer and ReLU activation function. The `BatchNorm` layers sped up convergence and prevented any exploding/vanishing gradient problems. Dropout was applied in the first and second fully connected layers with a keep probability of 0.5, which significantly reduced overfitting. The probability of 0.5 was found to be optimal, with lower/higher probabilities both yielding lower validation accuracies. The full list of layers and their parameters are given in Table 1.

Layer	**Dimensions**
`Conv`	64
`MaxPool`	
`Conv`	128
`Conv`	128
`MaxPool`	
`Conv`	192
`Conv`	256
`MaxPool`	
`Conv`	256
`Conv`	192
`AdaptiveAvgPool`	(5,5)
`FC`	(4800, 2048)
`FC`	(2048, 256)
`FC`	(256, 14)

Table 1: Neural network architecture, BatchNorm, ReLU and dropout layers excluded for brevity. The **dimensions** column refers to the number of filters for convolutional layers, the output window size for the AdaptiveAveragePool layer, and an ordered pair of (`in_features`, `out_features`) for fully connected layers. Kernel size, stride and padding are ommitted as they are identical for each layer (see below).

1.2 Convolutional layers

All convolutional layers used 3×3 filters with a stride and padding of 1. We found that the combination of small strides and filters yielded better results than larger filters and strides, most likely because these were able to better capture fine features, especially at the first few layers. Each maxpooling layer reduced the window size by a factor of 2 with a kernel size of 2 and a stride of 2.

As with most CNN architectures, we increased the number of filters for the deeper convolutional layers since features become increasingly complex deeper in the network.

1.3 Adapted Average Pooling Layer

The adaptive average pooling layer was used to reduce the window to a fixed size of 5×5 and was inspired by the VGG-Net architecture [2]. It was critical that this window was small since we found that the majority of the model memory went towards storing the parameters in the first fully connected layer, which contained the most parameters. A fixed window size also removed the need to change the dimensions of the first fully connected layer with each new model, increasing the efficiency of the design process. Most importantly, the addition of this layer boosted validation accuracy, perhaps due to the smooth extraction of features facilitated by average pooling in comparison to max pooling.

1.4 Design process

In terms of the design process, we began with with a modified version of LeNet [1] with two convolutional and maxpooling layers each and three fully connected layers. This yielded a decent validation accuracy of 82%, so we increased the number of convolutional layers to five, hypothesising that a deeper network yield better accuracy. With batch normalisation and dropout to induce regularisation, the five layer convolutional model achieved a validation accuracy of $91 - 92\%$. Since training accuracy was significantly higher than validation accuracy, we added a greater variety of image transformations to reduce overfitting. This significantly boosted the validation accuracy to $94 - 95\%$, indicating our goal of reducing overfitting had been achieved. Finally, we added two additional convolutional layers and tuned the hyperparameters to arrive at our best model, which achieved $95 - 96\%$ validation accuracy.

2 Loss and Optimizer

The `CrossEntropyLoss` class, which combines a log softmax activation function with the negative log-likelihood loss, was used in training our model since the problem involves multiclass classification. The optimizer used was ADAM, with the default parameters, with the exception of the learning rate which was changed to 0.0006 (see section on hyperparameters). We also tried using Stochastic Gradient Descent, however, the convergence and validation accuracy were always poorer than ADAM, even after hyperparameter tuning and applying Nesterov momentum.

3 Image transformations

3.1 Training and Testing

Various data transformations were used to optimise model training and perform data augmentation to reduce overfitting, with the full list given in Table 2. Firstly, the Grayscale transformation was applied to reduce the three channels of each image to a single channel, speeding up training and decreasing model size. The final transformation applied to each image was ToTensor, which converted the images into Torch tensors and scaled pixel values to numbers between 0.0 and 1.0, providing a boost in training speed and reducing the issue of exploding gradients. The Grayscale and ToTensor transforms were applied during both training and testing to ensure consistency of inputs. However, no other transforms were applied during validation and testing, since the model must classify the original images.

3.2 Data Augmentation During Training

The transformations performed during training were selected to artificially enlarge the dataset in ways that enhanced the model's ability to generalise. The same model trained with only the GrayScale and ToTensor transforms applied during training yielded a validation accuracy that was anywhere between $2 - 5\%$ lower than when the full range of transformations was applied. When applying the transformations, the training accuracy only exceeded the training accuracy by around 1%, compared to over 5% without the transformations, suggesting a much higher degree of overfitting without the transformations. Experimenting with several combinations, it was found that the order of the transformations had minor effects on the validation accuracy. The RandomResizedCrop transform ensured the model was exposed to each character in a variety of positions during training as well as reducing background noise from other objects and people not belonging to the 14 classes. However, to ensure the crop did not exclude the character itself, the lower bound on the scale factor was increased significantly from 0.08 to 0.5.

Next, the spatial transformations RandomAffine, RandomPerspective and RandomHorizontalFlip were applied to ensure the model could correctly predict the character regardless of the character's orientation in the image, up to a reasonable level. For example, excessive rotation yielded suboptimal results (likely because few images in the dataset contained characters at rotated excessive angles), so the range of angles was limited to between -15 and 15 degrees.

Finally, the RandomAutocontrast and RandomAdjustSharpness transformations ensured the model's prediction would not be significantly affected by the level of lighting or the blurriness of the image. Experimentally, it was found that random increasing the sharpness boosted validation accuracy by around 2%, while decreasing the sharpness reduced validation accuracy.

Training	**Testing**
`Grayscale`	`Grayscale`
`RandomResizedCrop((64, 64), scale=(0.5, 1.0))`	`ToTensor`
`RandomPerspective(p=0.2)`	
`RandomAffine(degrees=(-15, 15), translate=(0.0, 0.5))`	
`RandomHorizontalFlip()`	
`RandomAutocontrast()`	
`RandomAdjustSharpness(sharpness_factor=2)`	
`ToTensor`	

Table 2: Table of Image transformations applied during training/testing, in the order that they were applied in the model. Only arguments different to their default value are displayed for brevity, in accordance with the code.

4 Hyperparameters

The model was trained for 100 epochs at a learning rate of 0.0006 with a batch size of 64. Training for more epochs resulted in overfitting, with little to negative improvement in validation accuracy, while the smaller learning rate tended to yield slightly better results than the default ADAM learning rate of 0.001. Larger batcher sizes such as $128, 256$ tended to yield lower validation accuracies and provided insignificant speed boosts. No weight decay was used as experimentation with a range of lambda parameters $(10^{-4}, 5 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-6})$ all resulted in poorer validation accuracy. This was likely because dropout and the image transformations already provided sufficient regularisation, and with weight decay, the model suffered an increase in bias and received an insignificant reduction to variance in return.

5 Validation set/other overfitting

An $80\% - 20\%$ split was used in separating the original dataset into a training set and validation set. The validation set was used to compare the accuracy of different models, particularly during hyperparameter tuning. The model with the best validation accuracy was then retrained on the entire original dataset before submission to ensure it was exposed to the largest possible amount of data.

References

[1] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324.

[2] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.