Efficient CIFAR-100 Modelling

Timofei Zavileiskii

Abstract

This paper proposes 2 small Deep Models to classify images and to generate samples from CIFAR-100 dataset. The models are constrained to be within 100,000 parameters and 10,000 optimisation steps for classifier, and be within 1,000,000 parameters and 50,000 optimisation steps for image generator. Models use bottlenecks and depthwise separable convolutions to reduce number of parameters, to prioritise depth, and maximise performance on such a small scale.

Part 1: Classification

1 Network Architecture

The architecture is based on the Convolutional Neural Network, introduced by AlexNet [10], which allows to efficiently use the spatial information by applying the same convolutional filter onto every pixel of the image.

The key idea behind the network is to prioritise depth to keep the number of parameters low while achieving sufficient number of layers, as in practice it was shown that depth is required for high performance [14, p417].

Proposed architecture combines ideas of residual layers from ResNet [**ResNet**], the Depthwise Separable Convolutions, introduced by Xception Network [1], and efficiently used by MobileNet to shrink the size of a classifier [4], and applying global average to turn 2D channels into scalars like InceptionNet [15]. Residual layers introduce a skip connection which adds result to the input $x = x + f(x, W_i)$, where W_i are parameters determined by optimisation algorithm. Depth-wise separable convolutions separate 3x3 convolution into 2 layers: 3x3 filters applied only to single layer, and 1x1 pointwise convolutions, which combine features together. The block utilises ReLU activation function $f(x) = \max(0, x)$ [2], and 2D Batch Normalisations [7]. Batch normalisation shifts and scales data according $x_{norm} = \frac{x-\mu}{\sigma+\epsilon} * \gamma + \beta$ where epsilon is smoothing term to avoid division by 0, and mean μ and standard deviation σ are updated based on input in the end of each mini-batch, and β, γ are learnable parameters.

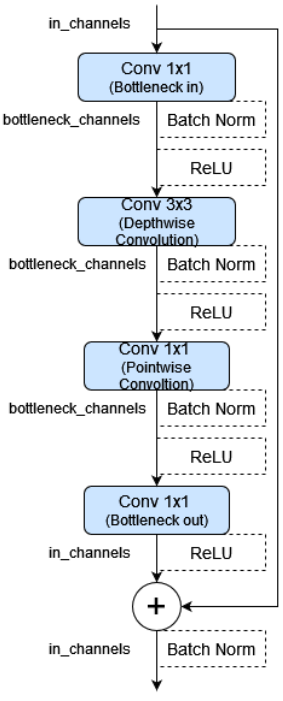


Figure 1: Architecture of the Residual Block

The block first applies bottleneck 1x1 layer to decease number of channels, then applies depthwise separable convolution and finally another 1x1 convolution to increase the number of channels to the initial number to add the output to the input (fig 1).

There are also small Convolutional channel Expansion blocks, which double the number of channels, followed by a Batch Normilisation and a ReLU (fig 3).

The final model is on fig 2, starting with a 5x5 convolution, followed by mix of Convolutional Expansions, Residual Blocks, and 2 max pooling layers. It ends with an average pool, reducing 8x8 channels into scalars, ending with a fully connected layer.

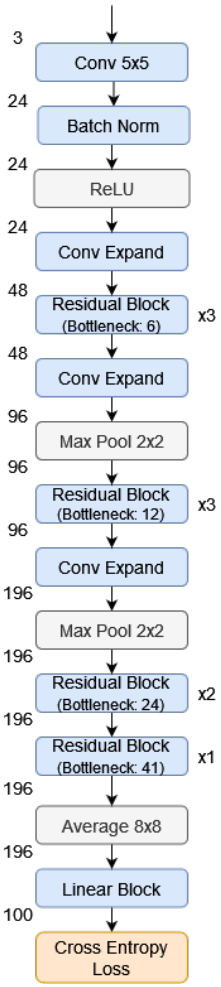


Figure 2: Architecture of the Whole Model

Initial training parameters used conservative defaults with little manual tuning. It is done using the Adam optimiser [9], which uses constant sized steps in the direction of the gradient as it tends to be less sensitive to hyperparameters than SGD, making for an easier initial estimate. The learning rate was set to 0.005, weight decay to 0.00005 as a simple form of l2 regularisation, and batch size to 256, and exponential weight decay at 0.98. Only image augmentation was random horizontal flip with probability 0.5.

The highest test accuracy was 58.14% at 9945 steps and with 79.34% train accuracy. It shows that the model is able to fit onto data, but the large gap between the test and train error suggests applying more regularisation techniques to decrease the train test error gap, and achieve higher performance.

1.1 Regularisation

To decrease the test error gap, several techniques were used and maximum test and train accuracies are presented. For simplicity, standard deviation is not considered

- Increasing weight decay to 0.00015: test accuracy - 43.26, train accuracy - 52.94
- Adding dropout of 0.4% before last 2 channel expansion blocks: test accuracy - 49.55, train accuracy - 67.27
- Augmenting data by horizontal flip with p=0.5, random vertical flip with p=0.5, adding Gaussian noise with varience 0.015 with probability 0.7, applying random crop with probability 0.7: test accuracy - 55.40, train accuracy - 62.04
- Augmenting data like before but instead of Gaussian adding random spacial and colour jitter: test accuracy - 52.74, train accuracy - 58.38

Techniques were able to decrease the gap between the test and train errors but resulted in lower test accuracy, which likely suggests that the model is not expressive enough and achieved test accuracy was the maximal for the architecture within allowed optimisation steps.

So no additional regularisation beyond techniques from section 1 will be used for the final model.

2 Final Model

The network has 99,703 parameters. The maximal test accuracy was 0.582 with 0.027 std, attained at step 10,000. The train loss was at 0.596, resulting in train accuracy 0.815 with 0.021 std. This creates a large gap between test and train errors of 21.9%. The gap shows that the model was memorising the test data, making it improve on the train set but not on actual data distribution. The results are slightly different from section 1 as model was retrained.

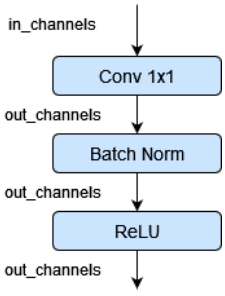


Figure 3: Channel Expansion Block

3 Limitations

There are more techniques which can be used to make a small classifier. SqueezeNet introduced fire modules, which instead of using depthwise separable convolutions to reduce parameter count, combines filters of 1x1 and 3x3 sizes, which may make the model more expressive [6].

The method for hyperparameter tuning can be improved by introducing 3rd validation set, used to test performance of the model with the best hyperparameters. It ensures that hyperparameters were the best not only for the test set but in general for the data distribution [14, p133].

Finally, given small size of the network, resulting in very short training time, automated methods can be used to tune the architecture. Other optimisers can be tried like SGD, as well as other learning rate schedules. More advanced methods for tuning architecture for small models are possible like a genetic approach, using selection, crossover and mutation to choose a broad range of hyperparameters [11].

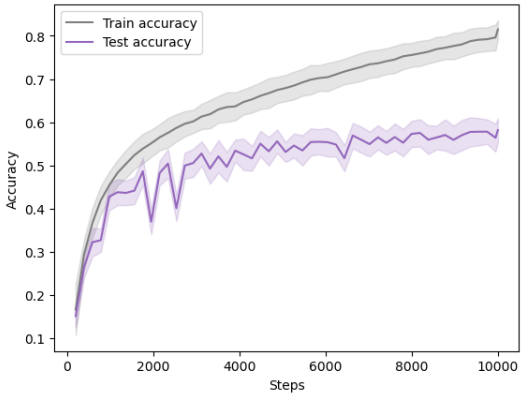


Figure 4: Training Graph

Part 2: Generative model

4 Diffusion Model

The chosen architecture was based on the denoising diffusion probabilistic models [3], as it considered to be easier to train due to it not suffering from mode collapse.

The key idea is to use the a Markov Chain where Gaussian noise ϵ is gradually applied to an image. Take T being maximum number of timesteps, with x_T representing pure noise, and $x_{t+1} = \sqrt{1-\beta_1} * x_t + \sqrt{\beta_1} * \epsilon$ where x_t is image at timestep t. We can compute coefficients $\alpha_t = 1 - \beta_t$ and $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$ The model can learn predicting noise ϵ at time-step t given diffused image. When sampling an image, random noise is latent space and we can use model's noise predictions to reverse the diffusion process, $x_{t-1} = 1/\sqrt{\alpha_t} * (x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha_t}}} * \epsilon_\theta) + \epsilon_t * z$, where ϵ_θ is noise predicted by the model, ϵ_t is, in this case, standard deviation at t, and z is Gaussian noise.

Diffusion schedule determines β_t values and significantly influences performance of the network and quality of the produced images. Linear schedule from the original paper along sigmoid schedule from [8], were tested with 50,000 steps at learning rate: 0.003, exponential weight decay: 0.995, batch size: 128. Sigmoid scheduling resulted in FID score of 75.19 and Linear Schedule in FID of 90.52, so sigmoid schedule was chosen in the future.

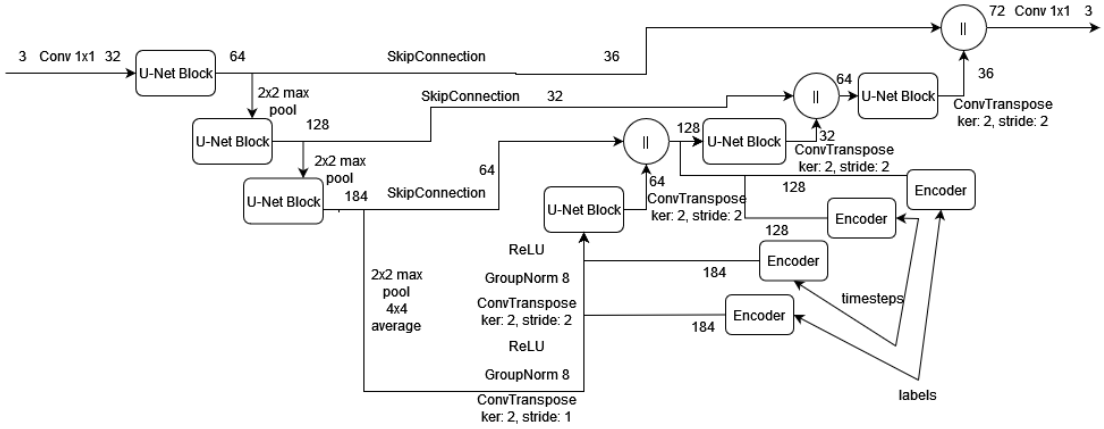
To interpolate between 2 images, interpolations of their latent vectors are upsampled by the model. Label of the closer image is used to condition the sampling of interpolation.

The implementation used code from HuggingFace blog post [5], Medium article on diffusion models [12], and Durham University Deep Learning practicals.

5 U-Net Architecture

In the Diffusion Model, U-Net, proposed in [13], is used to predict the noise from a diffused image. This is conditional architecture which takes into account image labels and timestep. The U-net consists of smaller parts: encoders, residual blocks with depthwise separable convolutions, u-net blocks which have 6 repeated residual block, and skip connection which reduce of number of channels. The full architecture can be see on the diagram below. Numbers represent number of channels in a connection and the $\|$ represents concatenation of channels. The key idea is to have layers at different levels and channel resolutions to be able to consider both global and local features.

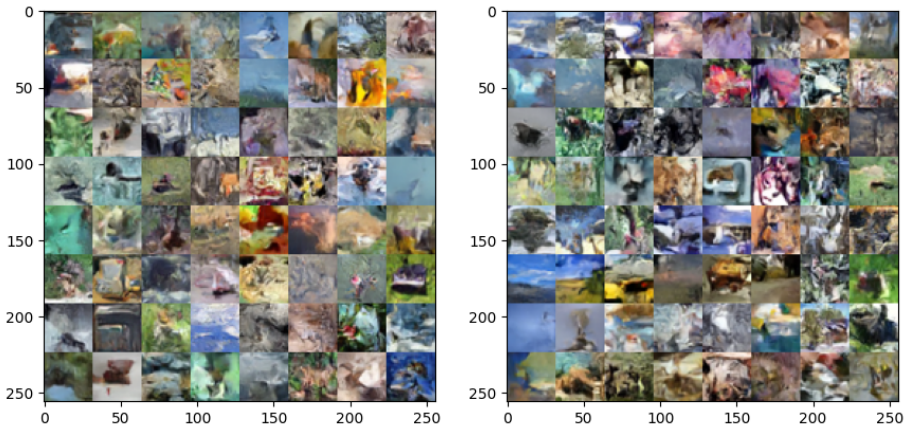
Conditionals were injected by passing labels and timestep into encoders, and multiplying x by label encoding and adding time encoding. The method was from the minimal diffusion implementation [16].



6 Results

The network has 996,863 parameters and achieves a FID of 58.00. The model was trained for 50,000 optimisation steps at same parameters as in section 4, except the batch size was set to 256. It was trained on the entire dataset.

The results look mostly unrealistic but most images look crisp. A batch of non-cherry picked samples is on the left and interpolants between points in the latent space are present on the right (interpolation goes horizontally):



7 Limitations

A lot of images do not look realistic likely due to small size and simplicity of the architecture, which is unable to capture all patterns in real world data. There are a lot of possible improvements, as an example original DDPM paper used transformers [17] and sinusoidal time embedding with them, swish activation function.

There are also more realistic ways to find interpolations between images [18]. And at last, the method included only limited and manual hyperparameter tuning due to long training times of the model. It can be automated given more time on model training, and trying other sampling strategies and schedules.

References

[1] Francois Chollet. "Xception: Deep Learning With Depthwise Separable Convolutions". In: (2017), pp. 1251–1258.

[2] Kunihiko Fukushima. "Cognitron: A self-organizing multilayered neural network". In: *Biological cybernetics* 20.3-4 (1975), pp. 121–136.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851.

[4] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[5] Kashif Rasul Niels Rogge. *The Annotated Diffusion Model*. 2022. URL: `https://huggingface.co/blog/annotated-diffusion`.

[6] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[7] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 448–456. URL: `https://proceedings.mlr.press/v37/ioffe15.html`.

[8] Allan Jabri, David Fleet, and Ting Chen. "Scalable adaptive computation for iterative generation". In: *arXiv preprint arXiv:2212.11972* (2022).

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.

[11] Benteng Ma et al. "Autonomous deep learning: A genetic DCNN designer for image classification". In: *Neurocomputing* 379 (2020), pp. 152–161.

[12] DZ. *Intro to Diffusion Model — Part 4*. Article on implementation of diffusion models. 2023. URL: `https://dzdata.medium.com/intro-to-diffusion-model-part-4-62bd94bd93fd`.

[13] Thomas Brox Olaf Ronneberger Philipp Fischer. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, pp. 234–241.

[14] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[15] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[16] TeaPearce. $Conditional_Diffusion_MNIST$. GitHub repository. 2023. URL: `https://github.com/TeaPearce/Conditional_Diffusion_MNIST`.

[17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[18] Zhaoyuan Yang et al. "Impus: Image morphing with perceptually-uniform sampling using diffusion models". In: *arXiv preprint arXiv:2311.06792* (2023).