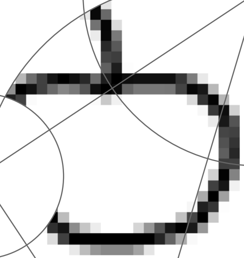
QuickDraw Data: large CNNs and GANs

Applied Machine Learning - Final Project

Marie Cornelius Hansen, Kasper Hede Nielsen
and Martin Langgård Ravn
Niels Bohr Institute



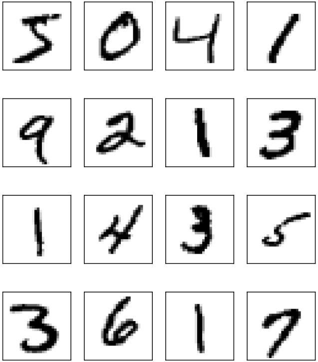
Introduction

- Classification on large dataset with many categories
- Investigate double descent behavior of large neural networks
- Make our own drawings (with GANs!)

[https://quickdraw.withgoogle.com/]

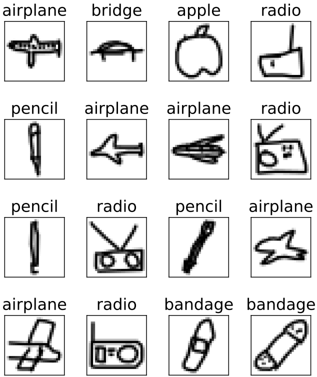
QuickDraw dataset

Advanced version of MNIST



10 categories
Low variance

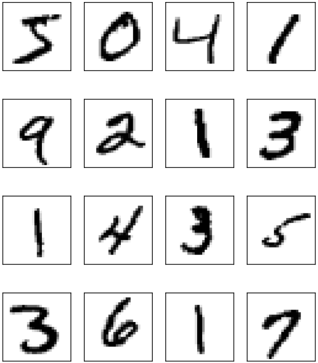
$\rightarrow$

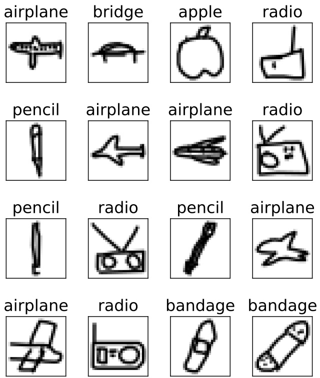


345 categories
High variance

QuickDraw dataset

Advanced version of MNIST





10 categories
Low variance

345 categories
High variance

36 GB of 50,426,266 images!

Choice of model

Train on 50 categories with 2000 drawings of each:

- LightGBM $\rightarrow$ 62.10% test accuracy

Choice of model

Train on 50 categories with 2000 drawings of each:

- LightGBM $\rightarrow$ 62.10% test accuracy
- Fully connected neural network $\rightarrow$ 41.28% test accuracy

Choice of model

Train on 50 categories with 2000 drawings of each:

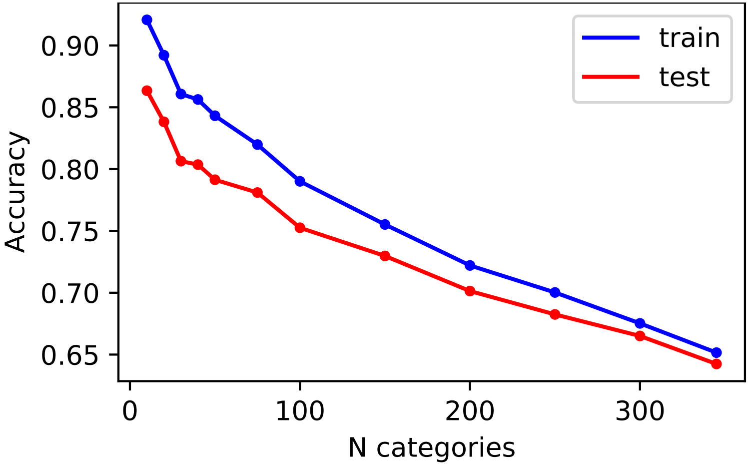
- LightGBM $\rightarrow$ 62.10% test accuracy
- Fully connected neural network $\rightarrow$ 41.28% test accuracy
- Convolutional neural network $\rightarrow$ 72.03% test accuracy

Convolutional network with Tensorflow

Layer	Kernel	filters	Output shape
Input			[28,28]
Conv2D	$[3,3]$	k	[28,28,k]
Downsampling	[2,2]		[14,14,k]
Conv2D	$[3,3]$	$2*k$	[14,14,2*k]
Downsampling	[2,2]		[7,7,2*k]
Conv2D	$[3,3]$	$4*k$	[7,7,4*k]
Downsampling	[2,2]		[3,3,4*k]
Conv2D	$[3,3]$	$8*k$	[3,3,8*k]
Downsampling	[2,2]		[1,1,8*k]
Dense		$8*k$	[8*k]
Output			[n]

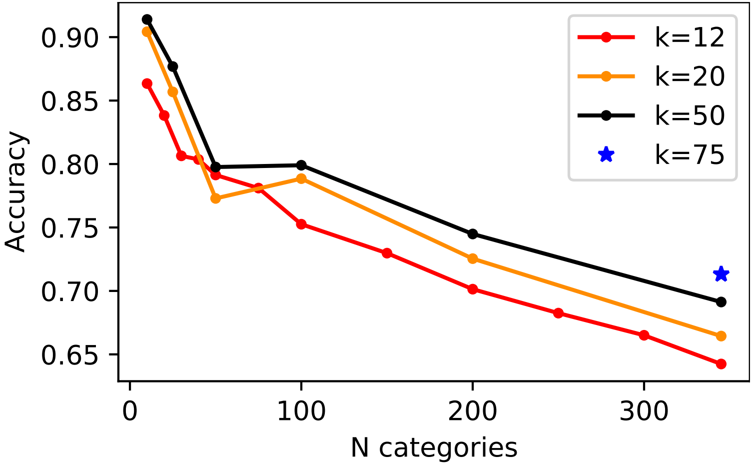
Dependence on number of categories

k=12, 8000 training points per category



Dependence on number of categories

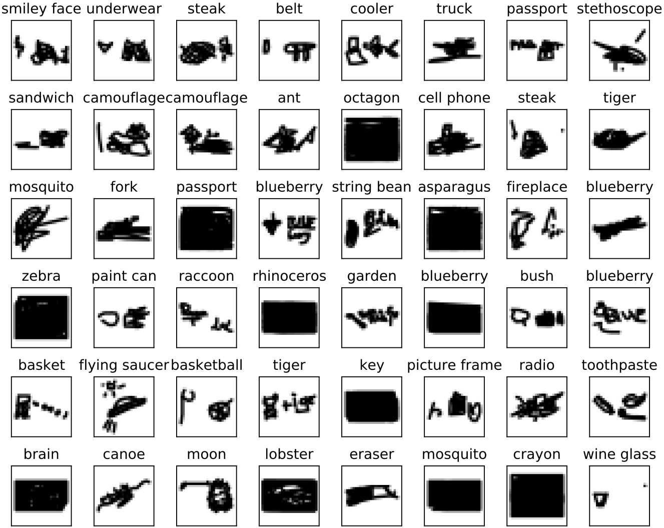
8000 images per category in training, 4000 in validation and test



Categories get mixed up

Category 1	Category 2	% wrong
Cake	Birthday cake	0.26425
School bus	Bus	0.23875
Hurricane	Tornado	0.22675
Motorbike	Bicycle	0.202
Octagon	Hexagon	0.18625
Mug	Coffee cup	0.17425
Violin	Guitar	0.168
Mug	Cup	0.16025
Stereo	Radio	0.148
Hockey stick	Golf club	0.14225
Truck	Pickup truck	0.139
Cello	Violin	0.12675
Paint can	Bucket	0.12675
Smiley face	Face	0.12575
⋮	⋮	⋮

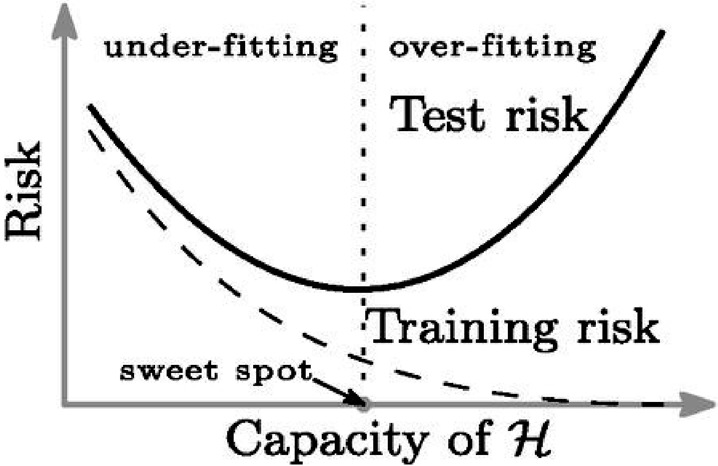
Worst pictures



Comparison to other works

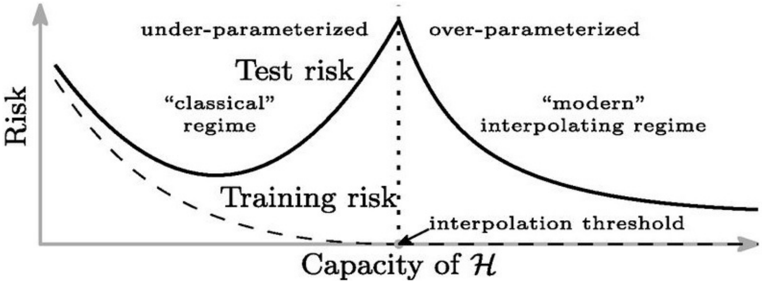
- [Lamb et al, 2020] get 87.25% accuracy on 10 categories
- [Kabakus, 2020] gets 89.53% accuracy on 10 categories
- **[Xu et al, 2020] get an accuracy of up to 74.22% on all 345 categories with Inception V3 [Szegedy et al, 2015] (25 million parameters). Used only 1000 training images per category.**
- By changing to stroke (coordinate and time) based images and using MGT [Xu et al, 2020] improves speed by a factor of 3 and get an accuracy of 72.80%

Introduction to Double Descent (DD)



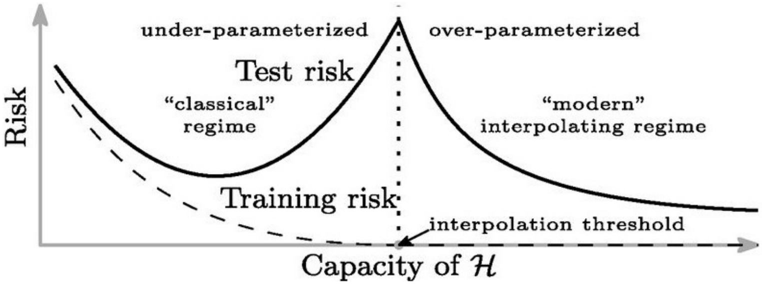
[https://arxiv.org/abs/1812.11118]

Introduction to Double Descent (DD)



[https://arxiv.org/abs/1812.11118]

Introduction to Double Descent (DD)

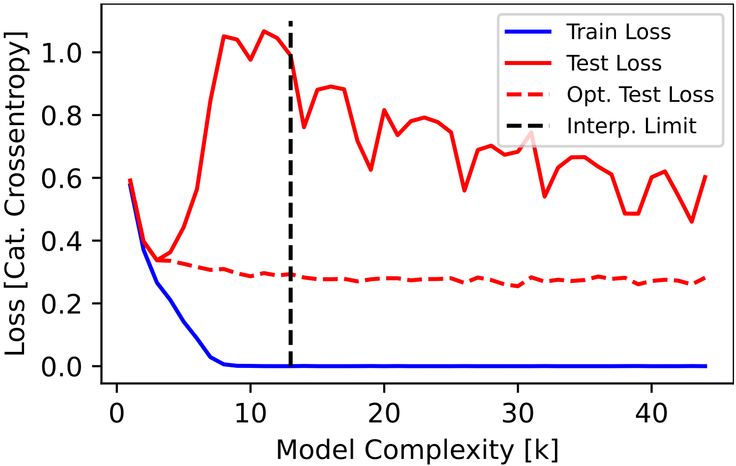


- L2-norm: $b\sqrt{\sum_i a_i^2}$
- BatchNormalization: Mean = 0, RMSE = 1
- Nothing at all
- Other L-norms

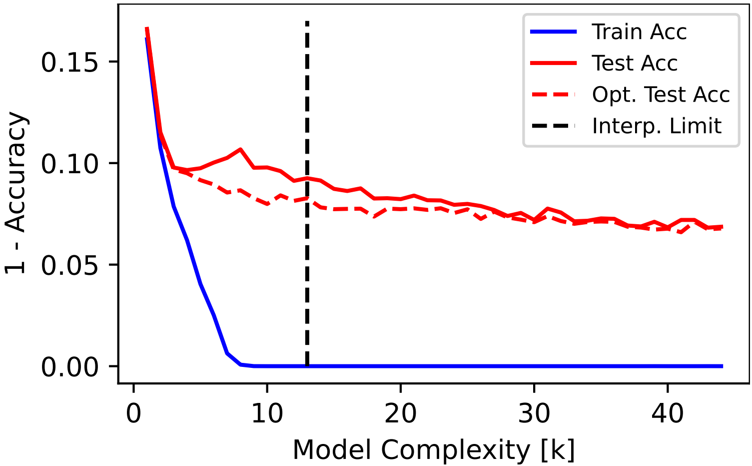
DD on QuickDraw

- Family of CNNs (depend on width parameter k)
- 10 classes (8000 train, 2000 test)
- Fixed learning rate
- 200 epochs
- $k \in$ range $[1, 45]$

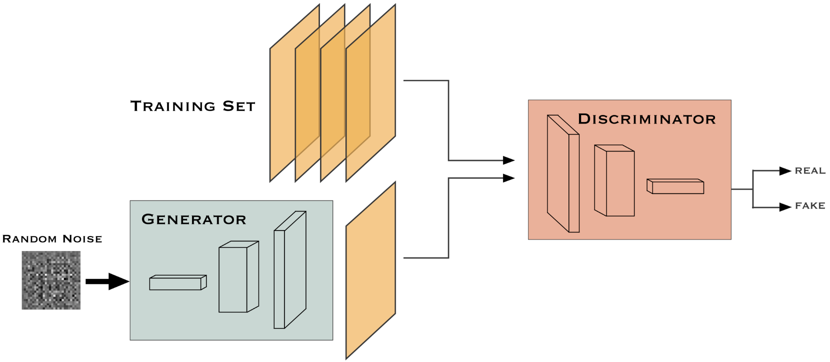
DD - BatchNorm loss



DD - BatchNorm accuracy



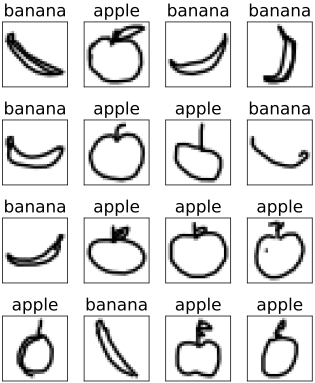
GANs



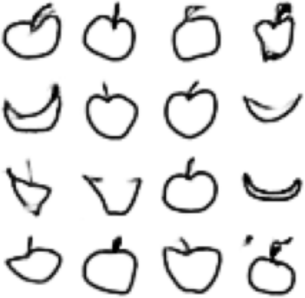
[https://towardsdatascience.com/image-generation-in-10-minutes-with-generative-adversarial-networks-c2afc56bfa3b]

We use CNN for both generator and discriminator.

GANs images



(a) Real drawings



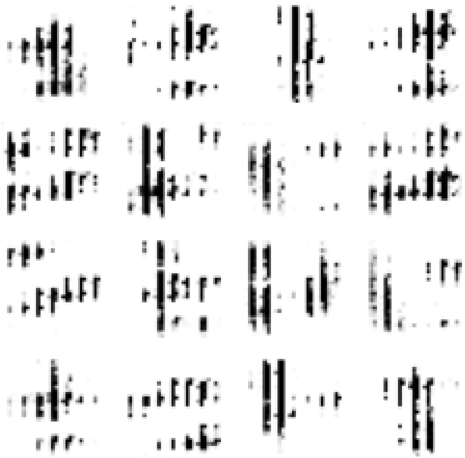
(b) Generated drawings

GANs optimization

- Changing number of filters
- Sensitive to learning rates
- Generator vs. discriminator - the discriminator often wins (dropout layers needed)
- Easier for generator to produce images that look real with 1 class as input
  - Faster: fewer epochs needed
  - Simpler: no confusion between classes

GANs GIF

(See proper GIFs in attachments. Here apples+bananas).

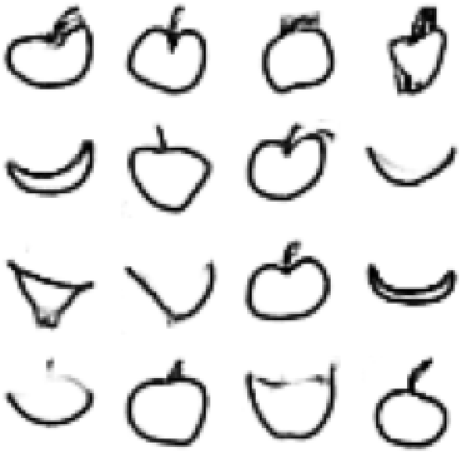


GANs GIF

(See proper GIFs in attachments. Here apples+bananas).

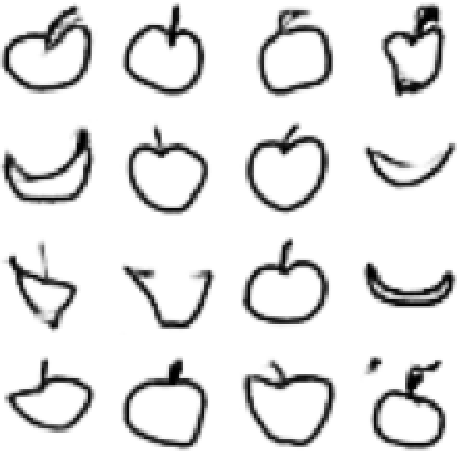
GANs GIF

(See proper GIFs in attachments. Here apples+bananas).



GANs GIF

(See proper GIFs in attachments. Here apples+bananas).



Conclusion and outlook

Classification:

- Up to 71.3% accuracy on 345 categories
- There are problems in the data: wrong classes drawn, unfinished drawings etc.

Double descent curve:

- Bigger models are better
- Use early stopping!

GANs:

- It is difficult but possible
- Many GANs that generate one class each is better than one GAN that can generate all classes.

Conclusion and outlook

Classification:

- Up to 71.3% accuracy on 345 categories
- There are problems in the data: wrong classes drawn, unfinished drawings etc.

Double descent curve:

- Bigger models are better
- Use early stopping!

GANs:

- It is difficult but possible
- Many GANs that generate one class each is better than one GAN that can generate all classes.

Outlook:

- Bigger models, more data, more GPUs
- Is multiclass GANs possible?
- Would GANs work better if we removed ”bad” data?

References

QuickDraw Data:
https://github.com/googlecreativelab/quickdraw-dataset
Tensorflow GANs algorithm:
https://www.tensorflow.org/tutorials/generative/dcgan
Wouter Bulten GANs algorithm: https://www.wouterbulten.nl/blog/tech/getting-started-with-generative-adversarial-networks/
GANs figure: https://towardsdatascience.com/image-generation-in-10-minutes-with-generative-adversarial-networks-c2
Articles:
Xu et al., CoRR 2020 https://arxiv.org/pdf/2001.02600.pdf
Xu et al. CoRR 2019 https://arxiv.org/pdf/1912.11258.pdf
Kabakus, International Congress on Human-Computer Interaction 2020 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9152911
Lamb et al. CoRR 2020 https://arxiv.org/pdf/1912.11570.pdf
Belkin et. al (2019) https://arxiv.org/abs/1812.11118
Nakkiran et. al (2019) https://arxiv.org/abs/1912.02292

Thanks for listening!

Questions?

Appendix

Data preprocessing

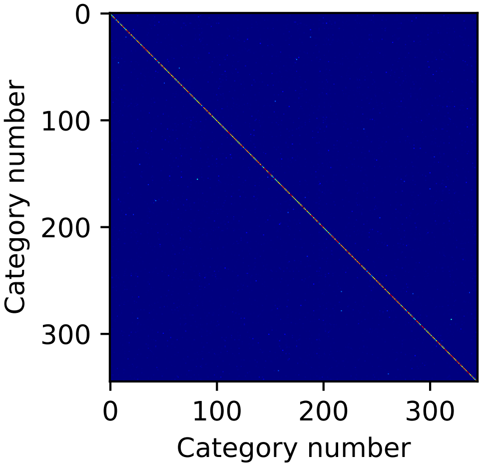
- Every picture is a 28x28 grayscale bitmap
- Every pixel is a numpy numpy.float64 between 0 and 255
- We converted all pixels to numpy.int8 between -128 and 127 to reduce data memory consumption by a factor of 8
- We worked on GPU(CUDA) on own computers + Google Colab

Full details of CNN model

Layer	Kernel	filters	Padding	Activation	Output shape
Input					[28,28]
Conv2D	$[3,3]$	k	same	relu	[28,28,k]
Downsampling	[2,2]			max	[14,14,k]
Conv2D	$[3,3]$	$2*k$	same	relu	[14,14,2*k]
Downsampling	[2,2]			max	[7,7,2*k]
Conv2D	$[3,3]$	$4*k$	same	relu	[7,7,4*k]
Downsampling	[2,2]			max	[3,3,4*k]
Conv2D	$[3,3]$	$8*k$	same	relu	[3,3,8*k]
Downsampling	[2,2]			max	[1,1,8*k]
Dense		$8*k$		relu	[8*k]
Output				softmax	[n]

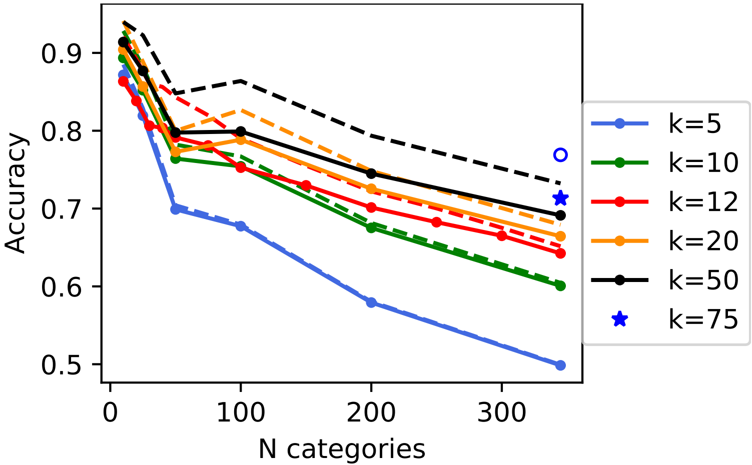
Optimizer: Adam with learning rate 0.001
Loss: (Sparse) categorical cross entropy
Trained on 8000 images from each category with early stopping
Test and validation sets consisted of 4000 images from each category

Confusion matrix



Most data points are in the diagonal. Some off-diagonal entries are large.

Full results from CNN classification



Full results from CNN classification

Solid lines are test scores, dashed lines are training scores. The blue star is the test accuracy for a k=75 network and the circle is the training accuracy.
From this figure it is clear that for small k, the network does not seem to overfit very much before stopping. As the networks increase in size, the difference between test and training loss increases, but the overall accuracy of both also increases meaning the networks classify with better accuracy.

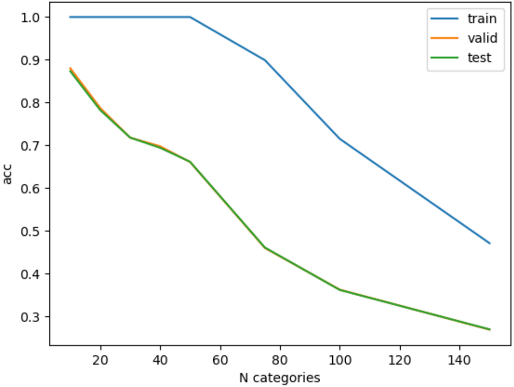
The 5% worst drawings

The 5% worst drawings

All test drawings have been sorted from least to most confusing for the trained neural network (k=12), with the most confusing being the worst. 48 drawings from around the 5% worst percentile are shown on the previous slide. We judge these drawings as being good enough to guess, since they all resemble their labels. Thus less than 5% of the wrongly classified drawings by the trained neural network can be attributed to bad drawings. The accuracy of the model is therefore not affected too much by bad drawings.

LightGBM - dependency on number of categories

2000 training points per category



The accuracy falls off heavily for larger numbers of categories, and can thus not be used for classification with 345 categories.

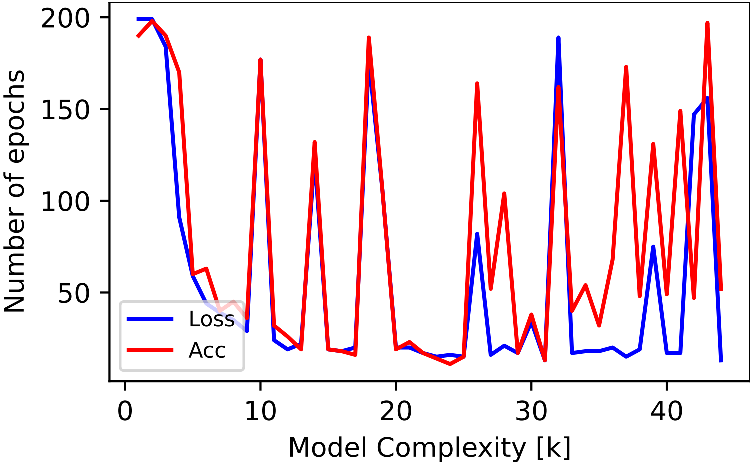
Quality of data and network

- Some of the drawings are just black
- Bad data is not necessarily bad
- Drawings not finished since the Google algorithm guesses while drawing, and you can no longer draw when the correct class is guessed
- Some categories are way too similar i.e. cake and birthday cake
- Network is translational invariant but not rotational or mirror invariant: some drawn images might be mirrored or rotated compared to the majority causing them not to be recognized

DD general

- Number of categories = 10, Number of epochs = 200, Adam learning rate = 0.001, Number of images pr category = 8000
- Tried different learning rates. The one used seem to work well on all network sizes. Used a fixed rate to stay unbiased.
- Tried different values for l2, settled on 0.00001 (started with 0.01 - too large value where l2 became more important than the actual loss)
- Could have tried larger k, takes a long time and the point of DD was already made (each run of DD takes about 8 hours on GPU)
- The l2 regularization used on all convolutional layers(same l2 parameter), and batch normalization used after all convolutional layers
- All DD curves generated with following categories: rainbow, lighthouse, giraffe, eyeglasses, tooth, teapot, sock, camouflage, cow, couch
- BatchNormalization seemed to give the best accuracy.

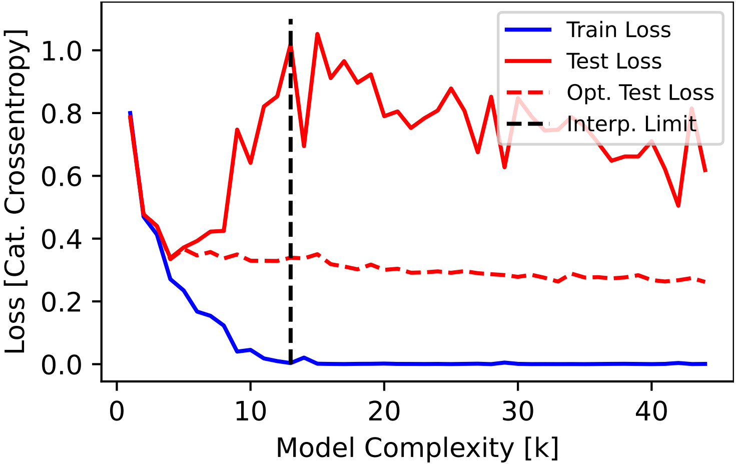
DD - BatchNorm Best Epoch



DD - BatchNorm Best Epoch

The figure shows which epoch is the best, judged by the test loss(lowest) or test accuracy(highest) during the 200 training epochs. For small networks more epochs are better. Bigger models converge(overfit) quickly, but each epoch is much slower to train.

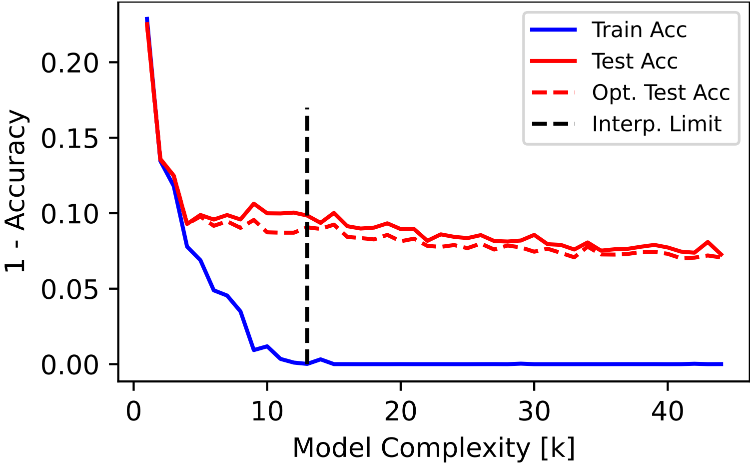
DD - None Loss


Train Loss
Test Loss
Opt. Test Loss
Interp. Limit
Loss [Cat. Crossentropy]
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Model Complexity [k]

DD - None Loss

A DD curve where no additional regularization/normalization is included. The double descent behavior is still present which might seem surprising at first.However, train loss not zero, goes from 0.001 to 0.0001 and so on, and thus improvements can still be made.

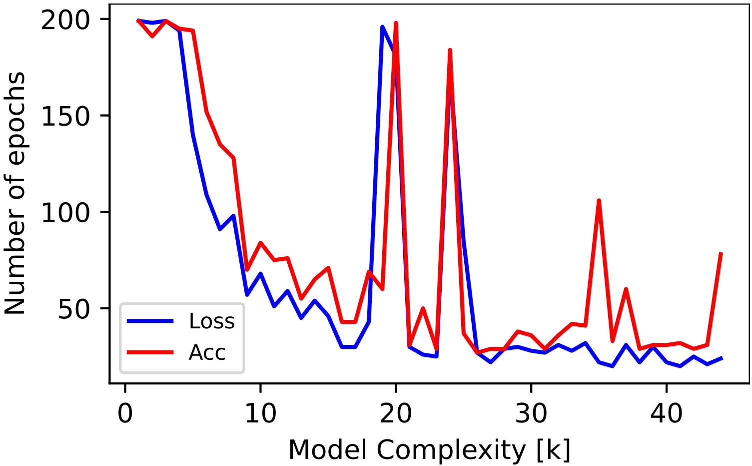
DD - None Accuracy



DD - None Accuracy

The accuracy also improves with higher k, however not as good as when the batch normalization is used.

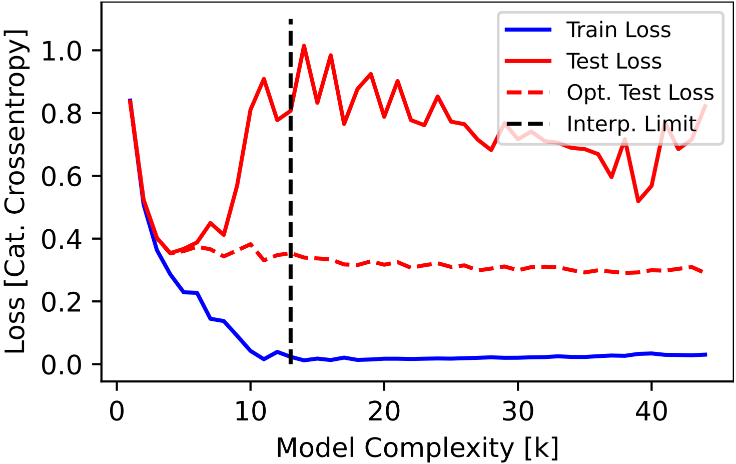
DD - None Best Epoch



DD - None Best Epoch

A similar picture as for batch normalization, for small models it is best to use many epochs, but as the models get bigger, it converges very quickly.

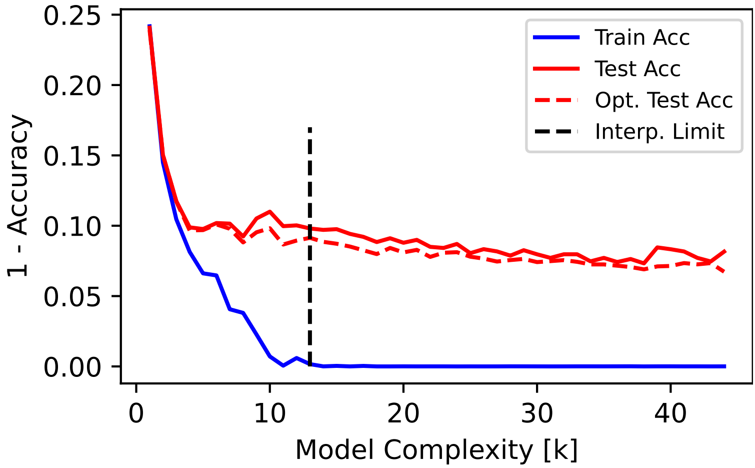
DD - L2 Loss


Train Loss
Test Loss
Opt. Test Loss
Interp. Limit
Loss [Cat. Crossentropy]
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Model Complexity [k]

DD - L2 Loss

A clear double descent curve, but where we can see the train loss start to increase for large k. This is because the model gets more and more trainable parameters, but the loss is penalized for non-zero parameters through the l2 regularization. We also see the test loss increase for very large k, but using early stopping the model slowly gets better for larger and larger k.

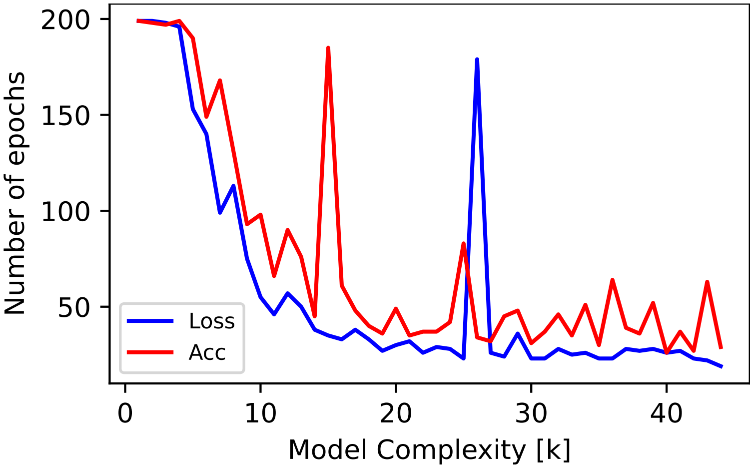
DD - L2 Accuracy



DD - L2 Accuracy

The accuracy improves with higher k. The accuracy is better than the model where no regularization is used, but slightly worse than the model with batch normalization.

DD - L2 Best Epoch



DD - L2 Best Epoch

Again for small models the last epochs are the best, whereas for large models it overfits after quite few epochs

GAN code

- It is very time-consuming to train GANs, which gave us struggles in optimizing the models
- Balancing the two different networks is very difficult. We have investigated the loss change during the epochs and made sure that the discriminator loss would not be too small, which would mean the generator would have no chance at tricking the discriminator. This we have done by changing the hyperparameters of the model multiple times (learning rates and dropout rates), but it was difficult to control.
- The model for the generator and discriminator is inspired by a Tensorflow MNIST GAN guide (see references). The QuickDraw dataset images have the same image size as MNIST and thus this specific generator also works in this case.
- We also tried different numbers of filters for the two models.
- We normalize the colors to [-1,1] since it works better for the activation functions + it makes it possible to avoid the black on white problem (It is better to generate images with white on black, due to the values of black being 0 normally)

GANs: generator model

Layer	Kernel	Strides	Filters	Activation	Output shape
Input					[100]
Dense			12544		[12544]
BatchNorm.				LReLU	[12544]
Reshape					[7,7,256]
Conv2DTra.	[5,5]	[1,1]	128		[7,7,256]
BatchNorm.				LReLU	[7,7,128]
Conv2DTra.	[5,5]	[2,2]	64		[14,14,64]
BatchNorm.				LReLU	[14,14,64]
Conv2DTra.	[5,5]	[2,2]	1		[28,28,1]

Padding = same
Adam learning rate = 0.001
LReLU = LeakyReLU
Idea for improvement: Is UpSampling2D better than Conv2DTranspose?

GANs: discriminator model

Layer	Kernel	Strides	Filters	Activation	Output shape
Conv2D	[5,5]	2	64	LReLU	[14,14,64]
Dropout			0.3		[14,14,64]
Conv2D	[5,5]	2	64	LReLU	[7,7,128]
Dropout			0.3		[7,7,128]
Flatten					[6271]
Dense			1	Softmax	[1]

Padding = same

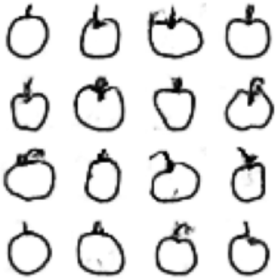
Adam learning rate = 0.001

GANs 1 vs 2 classes

The loss entropy is binary, and hence the discriminator does not predict the specific class but only whether the image is fake or real.
This means that the generated images will not necessarily have the same distribution in classes as the training input of the original images.
The hypothesis is, that it is easier for the generator to produce images of 1 class instead of 2, even though the input is 2 classes.
In the following we compare two GANs: (a) has only 1 class as input (apples), (b) has 2 classes as input (apples and bananas)

GANs 1 vs 2 classes

Epoch 50: the GAN with only apples look real, the GAN with apples and bananas does not look real

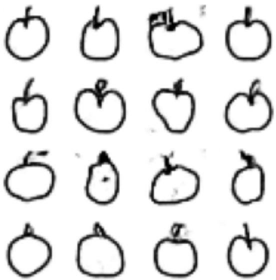


(a) Apples

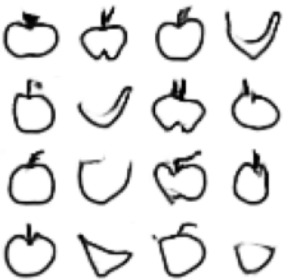
(b) Apples and bananas

GANs 1 vs 2 classes

Epoch 100: the GAN with only apples didn't change much, the GAN with apples and bananas is still not there



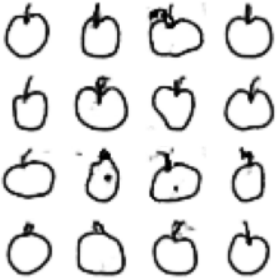
(a) Apples



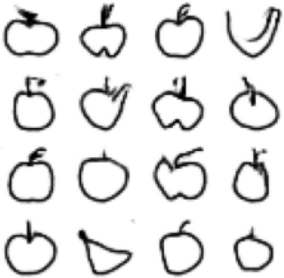
(b) Apples and bananas

GANs 1 vs 2 classes

Epoch 150: the GAN with only apples is the same, the GAN with apples and bananas has made bananas into apples



(a) Apples

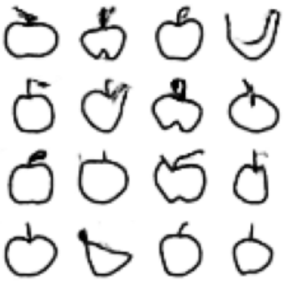


(b) Apples and bananas

GANs 1 vs 2 classes

Epoch 200: the GAN with only apples is the same, the GAN with apples and bananas have more apples now but is still not finished

(a) Apples



(b) Apples and bananas

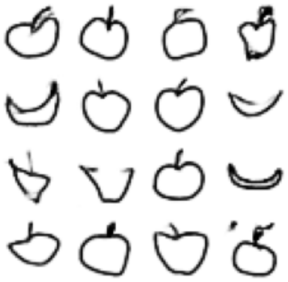
GANs 1 vs 2 classes

Conclusion:

- The GAN works faster (takes fewer epochs) with only one as class as input for the generator to produce images that look real
- Many images start out looking like bananas but end up as apples. The generator is not punished for not making both classes, hence it is simpler to only produce one class even though the input are two classes.
- The shapes are quite alike (half circle in bottom), but the apple is overall simpler, since it is round

GANs - apples and stars

We also compare a GAN of apples and stars to the one of apples and bananas. (GIF uploaded on Absalon)
Epoch 200:



(a) Apples and bananas

(b) Apples and stars

GANs - apples and stars

- Apples and stars do not look good after 200 epochs
- Apples and stars are more different than apples and bananas, thus it is more difficult to draw something
- The apples are round whereas stars are straight (and pointy), the generator merges the classes during training
- The model could probably be tuned for drawing these two classes, but it demonstrates the difficulty with GANs. The model works well for apples and bananas, but not for apples and stars although the situations seem very similar.