An Exploration of Machine Learning Methods on SVHN Dataset

Hongjie Fang
Department of Computer Science and Engineering
Shanghai Jiao Tong University
galaxies@sjtu.edu.cn
Student ID: 518030910150

Abstract

In this paper, we explore various machine learning methods on SVHN dataset, including logistic regression, support vector machine, neural networks such as AlexNet, VGG and ResNet. We also study the effects of histogram of oriented gradient feature extraction methods, the influence of adding lasso loss term and ridge loss term into the loss function, as well as the function of kernel methods. A great number of experiments is conducted on SVHN dataset to verify the hypotheses that we propose. What's more, we conduct further researches on the generative models such as variational auto-encoders. All source codes are available at https://github.com/Galaxies99/SVHN-playground.

1. Classification: Logistic Regression

1.1. Brief Introduction

In logistic regression, we assume $y_i \mid X_i \sim B(p_i)$, *i.e.*, $P(y_i = 1 \mid X_i) = p_i$ and $P(y_i = 0 \mid X_i) = 1 - p_i$. Then we assume $\text{logit}(p_i)$ can be estimated using simple linear model $X_i^T\beta$, that is,

$$\text{logit}(p_i) = \log \frac{p_i}{1-p_i} = X_i^{\mathrm{T}}\beta$$

where β is the parameter. Therefore, we can derive that

$$p_i = \text{sigmoid}(X_i^{\mathrm{T}}\beta) = \frac{1}{1+e^{-X_i^{\mathrm{T}}\beta}}$$

where sigmoid is the inverse function of logit. Assume y_i^* is the ground-truth label. The probability for correct estimation of sample (X_i, y_i^*) is calculated as follows.

$$\begin{aligned} P(y_i = y_i^* \mid X_i) &= p_i^{y_i^*}(1-p_i)^{1-y_i^*} \\ &= \left(\frac{1}{1+e^{-X_i^{\mathrm{T}}\beta}}\right)^{y_i^*}\left(\frac{e^{-X_i^{\mathrm{T}}\beta}}{1+e^{-X_i^{\mathrm{T}}\beta}}\right)^{1-y_i^*} \\ &= \frac{e^{y_i^* X_i^{\mathrm{T}}\beta}}{1+e^{X_i^{\mathrm{T}}\beta}} \end{aligned}$$

The likelihood of labels of all samples being correctly estimated is given as

$$P(\beta) = \prod_{i=1}^{n} P(y_i = y_i^* \mid X_i) = \prod_{i=1}^{n} \frac{e^{y_i^* X_i^{\mathrm{T}}\beta}}{1+e^{X_i^{\mathrm{T}}\beta}}$$

Then, we take logarithm for convenience, which is the log-likelihood function.

$$\log P(\beta) = \sum_{i=1}^{n}\left[y_i^* X_i^{\mathrm{T}}\beta - \log\left(1+e^{X_i^{\mathrm{T}}\beta}\right)\right]$$

The loss function of the logistic regression of samples is the negation of the log-likelihood function. For a mini-batch $\mathcal{B}$, the loss function can be viewed as the average probability of all samples in the batch, that is,

$$L = \frac{1}{|\mathcal{B}|}\sum_{(X_i, y_i^*)\in\mathcal{B}}\left[-y_i^* X_i^{\mathrm{T}}\beta + \log\left(1+e^{X_i^{\mathrm{T}}\beta}\right)\right]$$

Here, the average factor $1/|\mathcal{B}|$ is used to reduce the influence of the batch size on loss, as well as decoupling the effect of batch size and learning rate. We can use a mini-batch stochastic gradient descent algorithm, as well as Adagrad[5], Adadelta[22], Adam[11], AdamW[15] and other optimization algorithms, to solve this optimization problem and find the parameter β.

For Lasso loss, we need to add a regularization term $\lambda\|\beta\|_{l_1}$ in the loss function, as shown below.

$$L_{lasso} = L + \frac{|\mathcal{B}|}{|\mathcal{D}|}\cdot\lambda\|\beta\|_{l_1}$$

where $\mathcal{D}$ is the whole training dataset, and the coefficient $|\mathcal{B}|/|\mathcal{D}|$ aims to normalize the regularization term to sample-level, λ is the balancing coefficient.

For ridge loss, we also need to add a similar regularization term $\lambda\|\beta\|^2$ in the loss function, as shown below.

$$L_{ridge} = L + \frac{|\mathcal{B}|}{|\mathcal{D}|}\cdot\lambda\|\beta\|_{l_2}$$

where λ is the balancing coefficient.

Feat. Method	**Feat. Dim.**	**Accuracy** (optional best epoch)							
		Learning Rate (Batch Size = 128)				**Batch Size** (Learning Rate = 10^{-3})			
		10^{-2}	10^{-3}	10^{-4}	10^{-5}	64	128	256	512
RGB	3072	0.2642(4)	0.3004(3)	0.3016(25)	0.2840(50)	0.3142	0.3004	0.3024	0.3019
Gray	1024	0.2702(2)	0.2490(19)	0.2318(30)	0.1963(49)	0.2531	0.2490	0.2475	0.2478
RGB-HOG	8100	0.6765(50)	0.8141(8)	0.8145(49)	0.7992(50)	0.8069	0.8141	0.8140	0.8147
Gray-HOG	8100	**0.7277**(50)	**0.8169**(4)	**0.8177**(40)	**0.8021**(50)	**0.8162**	**0.8169**	**0.8168**	**0.8172**

Table 1. Accuracy of using logistic regression based on different feature representations on the testing set, and the effect of different learning rates and batch sizes on results (the bold numbers are the best feature representation, while the blue numbers mean the top-2 hyper-parameter that achieve the best performance) ("Feat." is the abbreviation of feature, while "Dim." is the abbreviation of dimension.)

1.2. Classification: From Binary to Categorical

The original logistic regression is a binary classifier. If we want to perform a classification with C classes, *i.e.*, a categorical classification task, we may need C separate logistic regression models. The i-th model predicts the probability that the given image belongs to class i.

An important issue is that we should balance the numbers of positive samples and negative samples in the batch for every logistic regression model for binary classification. Actually, the number of samples belonging to different categories is balanced. Therefore, we just need to multiply the loss of positive samples by C for balance consideration, *i.e.*,

$$L = \frac{C\sum_{s\in\mathcal{B}_{pos}} L(s;\beta) + \sum_{s\in\mathcal{B}_{neg}} L(s;\beta)}{C\left|\mathcal{B}_{pos}\right| + \left|\mathcal{B}_{neg}\right|}$$

1.3. Histogram of Oriented Gradient

Histogram of oriented gradient (HOG) [3] feature is a feature descriptor used in computer vision and image processing for object detection. It is able to extract the effective features of the image, while discarding those unimportant features to improve the performances of many feature-based algorithms. The pipeline of HOG feature extraction can be concluded as follows.

1. Calculate the gradient on horizontal and vertical axis, namely g_x and g_y. Then, the amplitude g and angle θ of the gradient is calculated using formula below.

$$g = \sqrt{g_x^2 + g_y^2}, \quad \theta = \arctan\frac{g_y}{g_x}$$

   For colored image, use one with maximum magnitude.

2. Divide the image into cells of size $c \times c$. Construct a histogram in each cell based on the category of the angle with magnitude as value. There are totally d categories of angle dividing 180 degrees into several fans.

3. Use a block of $b \times b$ cells to normalize the histogram, to reduce the effect of irrelevant factors like lightness.

4. Combine the features together and concatenate into a long feature vector, which has the size of $(n/c - b + 1)^2 \cdot d$ where n is the original image size.

We implement the HOG features by ourselves and visualize the results after HOG extraction on gray-scale images as shown in Fig. 1.

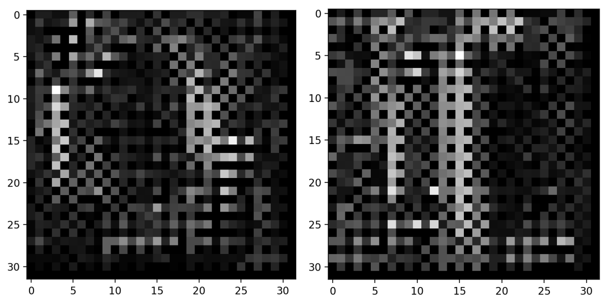

Figure 1. The Gray-HOG feature of digit 0 and 1

1.4. Experiments: Features and Hyper-parameters

We want to find an effective feature representation method. Therefore, we compare the RGB feature of each pixel of the image, the gray-scale feature of each pixel of the image and the HOG feature applied in RGB image (RGB-HOG) along with the HOG feature applied in the gray-scale image (Gray-HOG) that is introduced in Section 1.3.

The dataset we used for evaluation is the Street View House Number (SVHN) dataset[16]. For feature comparation, we set the block size b to 2, the cell size c to 2 and the angle categories d to 9 in RGB-HOG and Gray-HOG. We also set different learning rates $(10^{-2}, 10^{-3}, 10^{-4}, 10^{-5})$ and differenet batch sizes $(64, 128, 256, 512)$ to study the effects of these hyper-parameters. We train our models using AdamW optimizer[15] (the weight decay parameter is set to 0.0 to avoid introducing regularization term into loss) for 50 epochs, and the results are shown in Tab. 1.

From the results, we can observe that HOG features outperform other features by a large margin, and the Gray-HOG feature representation performs a little better than HOG features. Therefore, we can conclude that HOG feature representations, especially the Gray-HOG feature representation, extract effective features that can be easily used in classification.

About the learning rate, 10^{-2} is too big for the model to converge, thus causing unsatisfactory results. The results

Learning Rate Scheduling Method	**Accuracy**	**Best Epoch**
Constant learning rate, 10^{-3}	0.8169	**4**
Constant learning rate, 10^{-4}	**0.8177**	40
Learning rate step decay, $10^{-3} \xrightarrow{5} 10^{-4}$	0.8173	12
Learning rate exponential decay, $10^{-3} \cdot \gamma^n$, where $\gamma = 0.75$	0.8168	16

Table 2. The results of learning rate decay method

of learning rate 10^{-4} and 10^{-5} sometimes may outperform 10^{-3}, but the convergence speeds are relatively slower than using learning rate of 10^{-3}.

About the batch size, from the result we can see that, the influence of batch size on accuracy is not obvious. Therefore, we choose a reasonable batch size of 128 in the following experiments for speed and performance considerations.

Therefore, **in the following experiments about logistic regression, we will use Gray-HOG features representation, and a learning rate of 10^{-3} with a batch size of 128 to achieve the relatively great speed and performances**.

1.5. Experiments: Balancing Samples

We state that we can balance the positive and negative samples in loss in Section. 1.2. But is it necessary, or effective? Let us do some experiments to verify the conclusion.

We use the Gray-HOG with block size $b = 2$, cell size $c = 2$ and angle categorry $d = 9$ as an example. The learning rate is set to 10^{-3} and the batch size is set to 128. We use AdamW optimizer[15] with weight decay term 0.0 for 20 epochs only, since it is enough for models to convergence according to Tab. 1. The results are shown in Tab. 3.

Method	**Loss Settings**	**Accuracy**
Gray-HOG $(2, 2, 9)$	without balancing	0.4030
	with balancing	**0.8169**

Table 3. The results of whether balancing samples matters

From results we can conclude that, balancing positive and negative samples in loss brings a large performance gain on accuracy. This is because we treat the categorical classification task as binary classification task, thus causing the inbalance between positive samples (a class of samples) and negative samples (other classes of samples). If we don't balance the samples in loss, every binary classifier may tends to say "no" since there are more negative samples, without learning knowledge from the features. Therefore, **balancing samples in loss is necessary and effective**.

1.6. Experiments: Learning Rate Decay

From the previous experiments, we find out that learning rate 10^{-4} performs a little better than learning rate of 10^{-3}, but it may result in slow convergence. Therefore, we design an experiments to verify that a descending learning rate can make the model to converge faster and smoother.

In this experiment, we use the same experimental settings as introduced in Section 1.5, except that training for 50 epochs. Then, we apply multi-step learning rate scheduler and exponential learning rate scheduler to the training process respectively. For both learning rate schedulers, the initial learning rate is 10^{-2} for faster convergence. For multi-step learning rate schedulers, and after 5 epochs, the learning rate will drop to 10^{-3} for smoother convergence. For exponential learning rate schedulers, after each epoch, the learning rate will be multiplied by a fixed constant 0.75. The results of the experiment is shown in Tab. 2

From the results we can observe that, the learning rate scheduler can make the model to converge faster and smoother, but the performance gain is very limited, or even worse than the original one. Therefore, we do not apply the learning rate decay method introduced before in further experiments.

1.7. Experiments: HOG Parameters

We explore the effects of HOG parameters in this subsection. We set different block sizes b, cell sizes c and angle directions d to explore the performance difference between different parameters.

We use the same experimental settings as introduced in Section 1.5. We test the performance of several (b, c, d) tuples, and the results are shown in Tab. 4.

Parameter			**Feature Dimension**	**Accuracy**
b	c	d		
1	2	9	2304	0.7985
2			8100	0.8169
4			24336	**0.8273**
2	1	9	34596	0.8001
	2		8100	0.8169
	4		1764	**0.8243**
2	2	5	4500	0.8168
		9	8100	**0.8169**
		18	16200	0.8138

Table 4. The results of different Gray-HOG parameters

From the results we can observe that, the values of b and c affect the accuracy greatly, while the value of d only slightly affect the accuracy. Here are our explanations.

1. When c becomes larger, *i.e.*, the cell size becomes larger, every cell can capture high-level features, which

Method	Accuracy	Lasso Loss Accuracy					
		$\lambda = 10^{-1}$	$\lambda = 10^{-2}$	$\lambda = 10^{-3}$	$\lambda = 10^{-4}$	$\lambda = 10^{-5}$	$\lambda = 10^{-6}$
Gray-HOG (2,2,9)	0.8169	0.5250	0.7695	0.8113	0.8184	0.8165	**0.8186**
Gray-HOG (2,4,9)	0.8243	0.5422	0.7686	0.8121	0.8246	**0.8252**	0.8250
Gray-HOG (4,2,9)	0.8273	0.4526	0.7360	0.8112	**0.8278**	0.8277	0.8277

Table 5. The results of Lasso loss on different λ

Method	Accuracy	Ridge Loss Accuracy					
		$\lambda = 10^{-1}$	$\lambda = 10^{-2}$	$\lambda = 10^{-3}$	$\lambda = 10^{-4}$	$\lambda = 10^{-5}$	$\lambda = 10^{-6}$
Gray-HOG (2,2,9)	0.8169	0.8092	**0.8190**	0.8169	0.8163	0.8184	0.8172
Gray-HOG (2,4,9)	0.8243	0.8036	0.8228	**0.8258**	0.8248	0.8257	0.8258
Gray-HOG (4,2,9)	0.8273	0.8156	0.8278	**0.8284**	0.8279	0.8283	0.8282

Table 6. The results of ridge loss on different λ

are usually well-quality features that can lead to better accuracy. Therefore, the accuracy increases when c increases. On the other hand, c can not be too large, or many features may be coupled in one feature dimension, thus results in worse performance. **Hence, selecting parameter c is basically adapting the model to the dimensionality of the features**.

2. When b becomes larger, *i.e.*, the normalization range becomes larger, we can reduce the impact of the light more effectively. However, the increasing of b also brings the increasing of feature dimensions, which may result in lower training speed and convergence speed. **Hence, selecting parameter d is basically evaluating a trade-off between speed and accuracy**.

3. When d becomes larger, there is more accurate information in the histogram of each cell. However, the accuracy does not matter that much. **Therefore, d hardly affects the accuracy**.

According to our analyses, parameters $(4, 4, 9)$ should perform better than the previous models, since it can detect the feature effectively while reducing the effects of the background. Now, let us train a Gray-HOG model of parameters $(4, 4, 9)$ to verify our hypothesis.

However, out of our expectation, the model only reaches an accuracy of 0.8205, which is less than the accuracy of models on parameter $(2, 4, 9)$ and $(4, 2, 9)$. Actually, if we observe the training process, the accuracy is still increasing, which means the model has not converge yet. Then, if we train the models to 50 epochs, we can see that the accuracy becomes 0.8290, which means the HOG feature under these parameters are well-quality representations, but it may converge slower due to the increasing of feature dimension.

1.8. Experiments: Extra Training Dataset

The SVHN Dataset[16] also provides extra training set, with 531,131 less difficult samples. In this experiment, we will test the performance if we train the model on both training and extra set, which contains 640,388 samples in total.

The experimental settings are the same as introduced in Section 1.5. We evaluate the lasso and ridge loss on different parameters of Gray-HOG features, and the results are shown in Tab. 7.

Method	Training Settings	Accuracy
Gray-HOG $(2, 2, 9)$	without extra set	0.8169
	with extra set	**0.8356**
Gray-HOG $(2, 4, 9)$	without extra set	0.8243
	with extra set	**0.8405**
Gray-HOG $(4, 2, 9)$	without extra set	0.8273
	with extra set	**0.8470**

Table 7. The results of whether we use the extra set

From results we can know that, training with the extra set can improve the performances to some extent (about 5% improvement of accuracy), which seems quite obviously. But from the experiment we can know that, no matter how "easy" the figure is to human, like the samples in the extra set which are "less difficult", it may contribute to the performance gain of the model, since it may has the features not found in other samples.

1.9. Experiments: Lasso Loss and Ridge Loss

Lasso loss and ridge loss are loss regularization terms, which is used to penalize the model parameters. The lasso loss can leads to sparse feature, while the ridge loss can assure that there is no dominant features. Actually, adding suitable regularization terms to loss function can prevent the model from overfitting. Now, let us conduct several experiments to verify the conclusion.

The experimental settings are the same as introduced in Section 1.5, except that we change the training epoch to 30 since regularization terms may result in slower convergence. We evaluate the lasso and ridge loss on different parameters of Gray-HOG features, and we choose λ from 10^{-1} to 10^{-6}. The results are shown in Tab. 5 and Tab. 6.

Regularizations		**Original Parameters**				**Abs. of Parameters**			**# of Parameters**
Type	λ	**Max**	**Min**	**Mean**	**Std**	**Max**	**Mean**	**Std**	**less than 10^{-3}**
Lasso	10^{-1}	1.0914	-0.9680	-0.0012	0.0489	1.0914	0.0047	0.0487	**16848**
Lasso	10^{-2}	0.6317	-0.5798	-0.0077	0.0885	0.6317	0.0338	0.0822	**10026**
Lasso	10^{-3}	1.9924	-1.9322	-0.0183	0.2709	1.9924	0.1221	0.2425	**2651**
Lasso	10^{-4}	2.6163	-3.1506	-0.0328	0.4836	3.1506	0.3194	0.3645	**405**
Lasso	10^{-5}	2.7190	-3.3475	-0.0378	0.5612	3.3475	0.4151	0.3796	**35**
Lasso	10^{-6}	2.5678	-3.1157	-0.0366	0.5348	3.1157	0.3991	0.3578	**32**
None	-	3.2002	-4.1483	-0.0426	0.6736	**4.1483**	0.5068	0.4457	**18**
Ridge	10^{-6}	2.6021	-3.1371	-0.0378	0.5438	**3.1371**	0.4068	0.3628	26
Ridge	10^{-5}	2.7146	-3.3615	-0.0376	0.5653	**3.3615**	0.4228	0.3771	35
Ridge	10^{-4}	2.4258	-2.9346	-0.0359	0.5142	**2.9346**	0.3837	0.3441	24
Ridge	10^{-3}	2.6075	-3.0443	-0.0356	0.5327	**3.0443**	0.3991	0.3546	32
Ridge	10^{-2}	2.0341	-1.7719	-0.0233	0.3607	**2.0341**	0.2665	0.2442	48
Ridge	10^{-1}	0.7686	-0.7188	-0.0118	0.1229	**0.7686**	0.0824	0.0919	249

Table 8. The results of Lasso loss on different λ

From results we can observe that, adding suitable regularization term will improve the performance of logistic regression, but only by a very small margin (approximate 0.1%). In our experiment, lasso loss with $\lambda = 10^{-4}$ and ridge loss with $\lambda = 10^{-3}$ is relatively optimal comparing with other parameters. **In conclusion, the regularization term indeed can prevent the model from overfitting, but concerning the task and the logistic regression model, the improvement is limited.**

Let us have a deep insight into the regularization term by analyzing the parameters before and after regularization. We choose the Gray-HOG with parameters $(2, 4, 9)$ as an example. The results are shown in Tab. 8.

From the results we can see that, the number of parameters whose values are less than 10^{-3} becomes more and more as λ in Lasso loss increasing, which verifies the conclusion that Lasso loss will generate sparse parameters, as illustrated when $\lambda = 10^{-1}$, approximate 95.51% of the parameters are less than 10^{-3}. **Hence, lasso loss encourages sparse features.**

From the results we can also observe that, the maximum of the absolute values of parameters are decreasing as λ in ridge loss increasing, which verifies the conclusion that ridge loss will eliminate dominant features, as illustrated when $\lambda = 10^{-1}$, every parameter is within range $[-1, 1]$ which is relatively balanced. **Hence, ridge loss encourages no dominant features.**

2. Classification: Support Vector Machine

2.1. Brief Introduction

Support vector machines (SVM)[2] are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis developed at AT&T Bell Lab. by Vladimir Vapnik with colleagues.

The optimization objective of support vector machine is

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\left\|w\right\|^2 \\ \text{s.t.} \quad & \forall i, y_i(w^T X_i + b) \geq 1 \end{aligned}$$

Use the Lagrange multipliers with the Karush-Kuhn-Tucker conditions, the objective can be transformed into the following formula, which is the loss function of support vector machines.

$$L(w, b, \alpha) = \frac{1}{2}\left\|w\right\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(w^T X_i + b) - 1\right]$$

Solving the optimization needs two steps. First, we fix α and learn w, b to minimize $L(w, b, \alpha)$. By plugging the first-order Karush-Kuhn-Tucker conditions into the objective, we can derive the following optimizations which only includes variable α.

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j X_i^{\mathrm{T}} X_j$$

with the restrictions of $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.

Since this is a quadratic programming problem, we can use the conventional algorithm to solve it. The data quantity is too large for the algorithm to perform efficiently. Sequential minimal optimization method[17] can be used to solve the optimization problem in an efficient way. Notice that, the optimization process should follow the Karush-Kuhn-Tucker condition, that is,

$$\begin{cases} \alpha_i \geq 0 \\ y_i(w^{\mathrm{T}} X_i + b) - 1 \geq 0 \\ \alpha_i \left[y_i(w^{\mathrm{T}} X_i + b) - 1\right] \geq 0 \end{cases}$$

Theoretically, we can use any support vector (X_i, y_i) to solve b from the property $y_i(w^{\mathrm{T}}X_i + b) = 1$ of support vectors. More robustly, if we have s support vectors, we can calculate b as follows.

$$b = \frac{1}{s}\sum_{i=1}^{n}[\alpha_i \geq 0]\left(\frac{1}{y_i} - w^{\mathrm{T}}X_i\right)$$

Therefore, we have solved the optimization problem of support vector machines. For inference, we just need to use support vectors with $\alpha_i \geq 0$.

2.2. Experiments: Kernel Methods

SVM is a very suitable vehicle for applying the kernel method. We can apply different kernels such as RBF kernel and polynomial kernel to SVM, and we only to slightly modify $\langle X_i, X_j\rangle$ to the kernel function $K(X_i, X_j)$ in the derivation and inference of SVM. We conduct experiments to see the performance results of various kernel methods on SVHN dataset[16].

We use the `SVC` in `sklearn` package to implement the SVM. Here we use the linear kernel function, polynomial kernel function, RBF kernel function and sigmoid kernel function as examples to conduct our experiments. We only train our SVM models on 10000 samples of our training set for time considerations. The features of the samples have been extracted using Gray-HOG method with parameters of $(b, c, d) = (2, 4, 9)$ introduced in Section. 1.7. The results is shown in Tab. 9.

Kernel	Accuracy
Linear	0.7815
Polynomial with degree of 2	0.8379
Polynomial with degree of 3	0.8476
Polynomial with degree of 5	**0.8551**
Polynomial with degree of 10	0.8209
Sigmoid	0.8039
RBF	0.8387

Table 9. The results of SVM with different kernel methods

From the results we can observe that the polynomial kernel with degree of 5 outperforms other kernel-based SVMs. Actually, the performance of polynomial kernels reaches the relatively optimal point when degree is 5. **Generally, the kernel-based methods can outperform the original SVM with the linear kernel by relatively large margins.**

2.3. Experiments: Regularizations

In this experiments, we choose RBF kernel to test the effect of regularization terms. We use the same experimental setttings as introduced in Section 2.2. The results are shown in Tab. 10.

From the results we can draw the same conclusion as introduced in Section 1.9. If the regularization coefficient is too large, it may do harm to the model performance, *e.g.* here when $\lambda = 100$, the accuracy suddenly drops to a very low value comparing to other accuracy. **In conclusion, the proper regularization coefficient can improve the performance of the models by a limited margin.**

Regularization Parameter	Accuracy
$\lambda = 100$	0.3796
$\lambda = 10$	0.7944
$\lambda = 1$	0.8387
$\lambda = 0.1$	**0.8455**
$\lambda = 0.01$	0.8454

Table 10. The results of SVM of RBF kernel with different regularization parameters

3. Classification: Deep Neural Network

3.1. Brief Introduction

Deep neural network, especially deep convolution neural network, has played a more and more important role in image processing, feature extraction and image classification since AlexNet[13] made a huge performance improvement on ImageNet dataset[4]. Then, VGG series network[19], GoogLeNet[21]and ResNet series network[7] are proposed in the following years. Let us quickly review the networks that we used in the project first.

LeNet[14] LeNet was first proposed by LeCun to perform document recognition task. It is a small network, but it includes almost all of the important modules of today's network, such as convolutional block, activation function, full connected layers and pooling layers. LeNet on SVHN dataset[16] has the following structure:

- (5×5 convolution without padding $\rightarrow$ ReLU $\rightarrow$ 2×2 max pooling layer) $\times 2$;
- flatten $\rightarrow$ (fully connected layer $\rightarrow$ ReLU) $\times 2$, with input channels $400, 120$ and output channels $120, 84$ respectively $\rightarrow$ fully connected layer with input channels of 84 and output channels of 10.

In the following network description, we use Conv(c, k, s, p) to represent a convolution layer with the output channel of c, the kernel size of k, the stride of s and the padding of p. We use ConvBN(c, k, s, p) to represent a convolution layer followed by a batch normalization layer[10] in it. ReLU, Sigmoid and Tanh are activation functions. We use FC(c) to describe a fully connected layer with the ouput channel of c. We use MaxPool(k, s), AvgPool(k, s) to represent the max pooling layer and the average pooling layer of the kernel size of k and the stride of s respectively. Dropout(p) stands for a dropout layer of probability p.

AlexNet[13] AlexNet was the first model to make a major breakthrough on the ImageNet dataset. Its original architecture takes a 224×224 image as the input, but SVHN dataset includes image of size 32×32. Therefore, we make a few modifications to AlexNet. Our revised AlexNet has the following architecture:

- Conv(64, 3, 2, 1) $\to$ ReLU $\to$ MaxPool(3, 2) $\to$ Conv(192, 3, 1, 1) $\to$ ReLU $\to$ MaxPool(3, 2) $\to$ Conv(384, 3, 1, 1) $\to$ ReLU $\to$ (Conv(256, 3, 1, 1) $\to$ ReLU) $\times 2$ $\to$ MaxPool(3, 2);
- Feature Flatten $\to$ (Dropout(0.5) $\to$ FC(4096) $\to$ ReLU) $\times 2$ $\to$ FC(10).

VGG[19] VGGs have improved the performance of AlexNet on ImageNet dataset further. VGG series networks share the same main architecture as follows.

- (Conv(64, 3, 1, 1) $\to$ ReLU) $\times a$ + MaxPool(2, 2);
- (Conv(128, 3, 1, 1) $\to$ ReLU) $\times b$ + MaxPool(2, 2);
- (Conv(256, 3, 1, 1) $\to$ ReLU) $\times c$ + MaxPool(2, 2);
- (Conv(512, 3, 1, 1) $\to$ ReLU) $\times d$ + MaxPool(2, 2);
- (Conv(512, 3, 1, 1) $\to$ ReLU) $\times e$ + MaxPool(2, 2);
- Feature Flatten $\to$ (Dropout(0.4) $\to$ FC(512) $\to$ ReLU) $\times 2$ $\to$ FC(10).

For VGG-11, parameters a, b, c, d, e are set to $1, 1, 2, 2, 2$ respectively. For VGG-13, parameters a, b, c, d, e are set to $2, 2, 2, 2, 2$ respectively. For VGG-16, parameters a, b, c, d, e are set to $2, 2, 3, 3, 3$ respectively. For VGG-19, parameters a, b, c, d, e are set to $2, 2, 4, 4, 4$ respectively. For VGG-x-BN where $x \in \{11, 13, 16, 19\}$, just replace the Conv blocks with ConvBN block in the network.

GoogLeNet[21] GoogLeNet is based on the inception module, which utilizes 1×1, 3×3, 5×5 convolution blocks and a max pooling layers to extract features. For details of the architecture, refer to the original paper [21].

ResNet[7] ResNet is based on the residual basic blocks and the residual bottleneck blocks, which includes a residual path and a shortcut path. For details of the ResNet architecture, refer to the original paper[7].

3.2. Experiments: Model Performance

In this section, we will evaluate the models introduced before. We will train each model for 50 epochs, with AdamW optimizer[15] and a constant learning rate of 10^{-3}, only on the training set of SVHN dataset[16] (without extra set). Then, we evaluate their performances on the testing set. The results are shown in Tab. 11.

Model	Accuracy	Best Epoch
LeNet	0.8894	18
AlexNet	0.8875	23
VGG-11	0.9292	30
VGG-11-BN	0.9426	8
VGG-13	0.9358	39
VGG-13-BN	0.9503	35
VGG-16	0.1959	1
VGG-16-BN	0.9537	27
VGG-19	0.1959	1
VGG-19-BN	0.9549	32
GoogLeNet	**0.9592**	42
ResNet-18	0.9532	37
ResNet-34	0.9548	49
ResNet-50	0.9516	38
ResNet-101	0.9537	37
ResNet-152	0.9490	35

Table 11. The results of the deep neural networks

In Tab. 11 we can make the observations and explanations listed as follows.

1. For LeNet and AlexNet, they outperforms the previous methods such as SVM and logistic regressions, which shows that they have actually learnt the features not extracted by HOG. **Their accuracy rate illustrates the effectiveness of deep neural networks comparing to traditional feature extraction method such as HOG.**

2. For VGG networks, VGG-19 has the strongest ability followed by VGG-16, VGG-13 and VGG-11. Adding batch normalization layer[10] can eliminate the differences between samples and improve the model performances. **Therefore, theoretically, the accuracy of VGG networks should follow**

   - VGG-19 $>$ VGG-16 $>$ VGG-13 $>$ VGG-11;
   - VGG-x-BN $>$ VGG-x for $x \in \{11, 13, 16, 19\}$.

   The experiment results support our second conclusion. However, we observe a strange phenomenon: the VGG-16 and VGG-19 performs extremely bad. To explain the situation, we conduct several ablation studies in the Section 3.3.

3. For ResNets, ResNet-152 has the strongest ability followed by ResNet-101, ResNet-50, ResNet-34 and ResNet-18. Therefore, theoretically, the accuracy of ResNets should follow that ResNet-152 $>$ ResNet-101 $>$ ResNet-50 $>$ ResNet-34 $>$ ResNet-18. **However, the experiment results show that ResNet variants all have similar accuracy. This is because that the**

task is too easy for ResNets to becomes overfitted, hence the accuracy increases slowly after 10 epochs, especially for large networks like ResNet-152.

4. Comparing the architecture of all the models, the GoogLeNet has a deeper architecture and a larger inception field, which makes it perform better than VGG networks. The ResNets is a combination of lots of shallow networks, which makes it stonger than the previous networks. Therefore, theoretically, the performance ranking from highest to lowest should be ResNets, GoogLeNet and VGG. However, the experiment results show that ResNets might not be the most strongest models, which is because **this task is not so difficult to let ResNets play to their strengths**.

3.3. Experiments: Why does VGG Perform Bad?

In the previous section, we observe that VGG-16 and VGG-19 perform unusually bad on the task. We are curious about what leads to this phenomenon. Therefore, we conduct more experiments on VGG networks.

We design several experiments on different hyper-parameters of VGG networks. Finally we found out that it is the high learning rate that causes the extremely bad results. The results of experiments on learning rate are shown in Tab. 12.

Models	**Accuracy**	
	lr 10^{-3}	**lr** 10^{-4}
VGG-11	0.9292	**0.9299**
VGG-11-BN	**0.9426**	0.9259
VGG-13	0.9358	**0.9397**
VGG-13-BN	**0.9503**	0.9374
VGG-16	0.1959	**0.9430**
VGG-16-BN	**0.9537**	0.9387
VGG-19	0.1959	**0.9425**
VGG-19-BN	**0.9549**	0.9408

Table 12. The results of the VGG networks with different learning rate, where “lr” stands for “learning rate”

Therefore, the high learning rate causes that some complex VGG networks without batch normalization are unable to converge, which results in the unusual performance. If we dive deeper into the feature level, for batches that consists of very different values of features (*e.g.*, a batch of samples whose values are too large, and a batch of samples whose values are too small), the models without batch normalization can hardly learn from those batches. **Only when learning rate is very small, *e.g.*,** 10^{-4}**, the model can learn the features very slowly.**

However, for those networks with batch normalization layers, the low learning rate may weaken the performance since it slows down the speed to convergence of the models, which explains that in the models with batch normalization layers, high learning rate performs better than low learning rate in Tab. 12.

3.4. Experiments: What do the DNNs focus on?

In this experiments, we want to explore what feature do DNNs focus on to achieve the satisfactory performance introduced in Section 3.3? We use the Grad-CAM[18] implemented in PyTorch-Grad-CAM[6] to visualize the attention of the ResNet-18 models. The results are shown in Fig. 2.

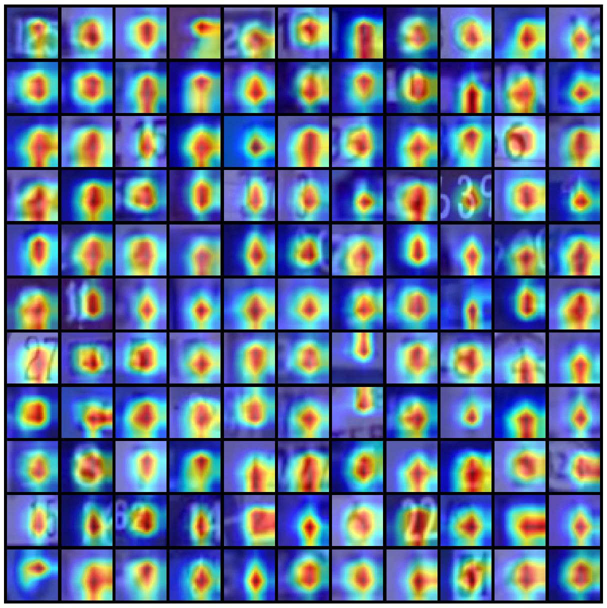

Figure 2. The attention of the ResNet-18 Model

From the results we can observe that the ResNet-18 model usually focuses on the obvious feature of the number, such as the corners of the numbers, *e.g.*, the corners of 3 and 2. By focusing on the special feature of digits, DNNs are able to make accurate classifications. **Hence, DNNs can capture features more effectively by themselves, *i.e.*, they do not need humans to tell them here is the correct feature, like logistic regression and SVM based on Gray-HOG features.**

4. Generation: Variational Auto-encoders

4.1. Brief Introduction

Variational Auto-encoder (VAE) [12] is a series of unsupervised generative models first proposed in the last ten years, which adapts an encoder-decoder architecture, while making strong assumptions concerning the distribution of the latent variables h. For convenience, we assume that $q_\phi(h|X)$ is the notation of the encoder network, and $p_\theta(X|h)$ is the notation of the decoder network.

VAEs are based on the maximization of the log-

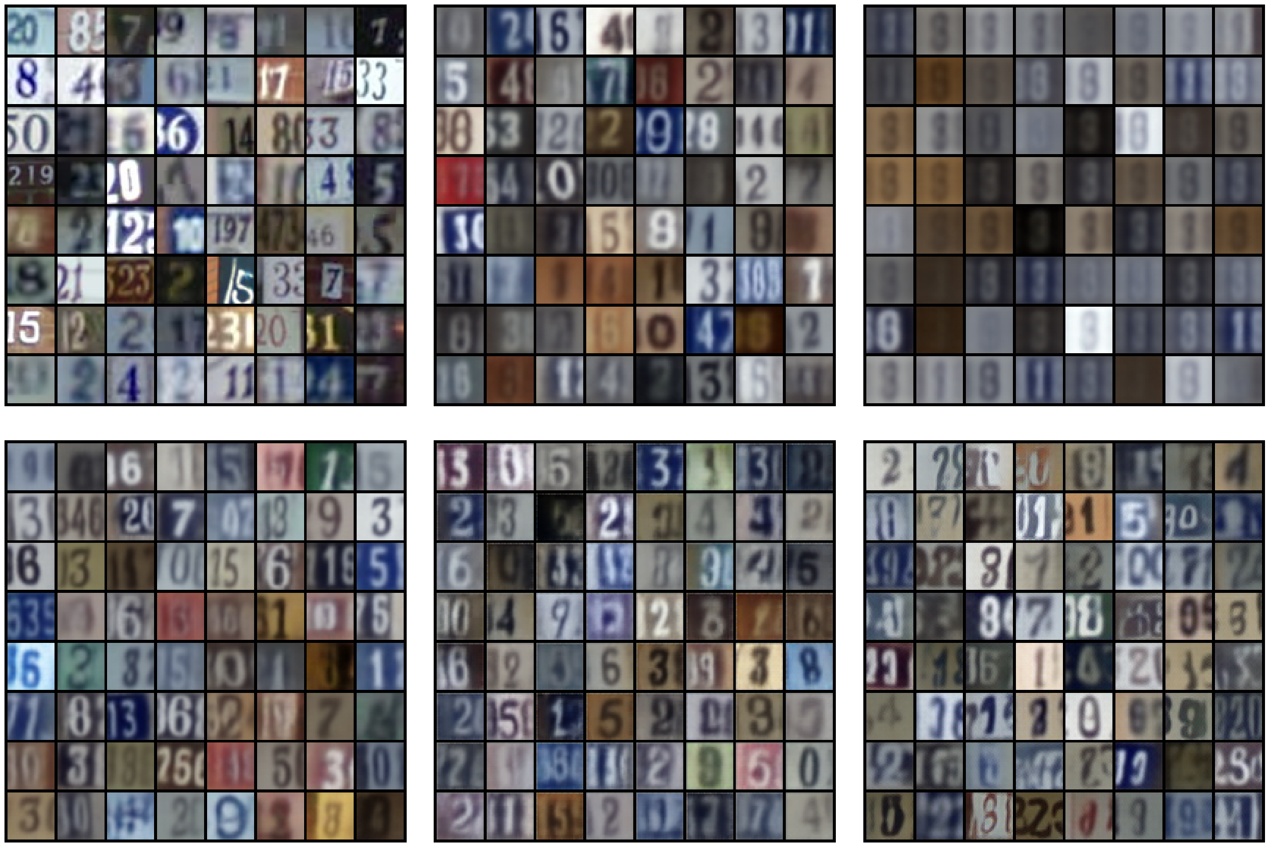

Figure 3. The results of image generation using variational autoencoder and its variants. The top-left one is the original image in dataset; the top-middle one is the image generated by VAE[12]; the top-right one is the image generated by β-VAE[8]; the bottom-left one is the image generated by disentangled β-VAE[1]; the bottom-middle one is the image generated by MSSIM-VAE[20]; the bottom-right one is the image generated by DFC-VAE[9]. All models are trained on the full SVHN dataset[16] (training set and extra set) after 20 epochs.

likelihood $\mathcal{L}$ of all observations, which is shown as follows.

$$\mathcal{L} = \log p_\theta(X) = \log\left(E_{h\sim q_\phi(h|X)}\left[\frac{p_\theta(h,X)}{q_\phi(h|X)}\right]\right)$$

Unfortunately, it is impractical to calculate it directly. Recall the Jensen's inequality as applied to the probability distributions, which states that when f is a concave function, $f(E[X]) \geq E\left[f(X)\right]$. Applying Jensen's inequality to the formula above gives us a lower bound of the log-likelihood as follows, which is also known as Evidence Lower Bound (ELBO).

$$\mathcal{L} \geq E_{h\sim q_\phi(h|X)}\left[\log p_\theta(h,X)\right] - E_{h\sim q_\phi(h|X)}\left[q_\phi(h|X)\right]$$

We can choose a family of variational distributions, *i.e.*, a parameterization of a distribution of the latent variables h, such as the expectations are computable, *e.g.*, the standard Gaussian distributions $N(0, I)$. Then, we can maximize the ELBO in order to maximize $\mathcal{L}$, which gives us the VAE objective $\mathcal{L}'$. The form of ELBO can be further divided into

$$\begin{aligned}\mathcal{L}' &= E_{h\sim q_\phi(h|X)}\left[\log p_\theta(h,X)\right] - E_{h\sim q_\phi(h|X)}\left[q_\phi(h|X)\right]\\ &= E_{h\sim q_\phi(h|X)}\left[\log\frac{p_\theta(X|h)p(h)}{q_\phi(h|X)}\right]\\ &= E_{h\sim q_\phi(h|X)}\left[\log p_\theta(X|h)\right] - KL(q_\phi(h|X)\mid p(h))\end{aligned}$$

where $E_{h\sim q_\phi(h|X)}\left[\log p_\theta(X|h)\right]$ measures the quality of reconstruction, while $KL(q_\phi(h|X)\mid p(h))$ measures the similarity between the posterior $q_\phi(h|X)$ and the given distribution $p(h)$. Notice here h is not determined by p_θ, hence we use $p(h)$ instead of $p_\theta(h)$ to represent its distribution.

For loss function L on a mini-batch $\mathcal{B}$, we can simply take the negation of the VAE objective $\mathcal{L}'$, that is,

$$\begin{aligned}L = &-\frac{1}{|\mathcal{B}|}\sum_{X\in\mathcal{B}} E_{h\sim q_\phi(h|X)}\left[\log p_\theta(X|h)\right]\\ &+\frac{|\mathcal{B}|}{|\mathcal{D}|}KL(q_\phi(h|X)\mid p(h))\end{aligned}$$

where $\mathcal{D}$ is the whole training dataset, and the coefficient $|\mathcal{B}|\,/\,|\mathcal{D}|$ aims to normalize the KL divergence term to sample-level.

For VAE implementation, we need to slightly modify the network architecture. We can modify the output of the auto-encoder to mean μ and log-variance $\log\sigma$. Therefore, we assume that $h \sim N(\mu, \sigma^2)$ so that we can take samples from the distribution and send it into the decoder. The decoder network will reconstruct images using the samples drawn from the distribution.

We have explained that the first term is the reconstruction loss, and the second term is the KL divergence between the posterior distribution $q_\phi(h|X)$ and our desired distribution $p(h)$. Here we assume that $h \sim p(h) = N(0, I)$. For KL divergence, we can use the output $\mu, \log\sigma$ of auto-encoder to calculate it directly as follows.

$$KL(q_\phi(h|X) \mid p(h)) = \frac{1}{2}\left(\mu^2(X) + \sigma(X) - \log\sigma(X) - 1\right)$$

For reconstruction loss, we can use the MSE loss or MSSIM loss[20] between input and reconstructed images.

4.2. Variants of VAEs

In this section, we also discuss some variants of variational auto-encoders, listed as follows.

- **β-VAE**[8]: The loss function of β-VAE is modified into the following form.

$$L = -\frac{1}{|\mathcal{B}|}\sum_{X\in\mathcal{B}} E_{h\sim q_\phi(h|X)}\left[\log p_\theta(X|h)\right] + \frac{|\mathcal{B}|}{|\mathcal{D}|}\cdot\beta\cdot KL(q_\phi(h|X) \mid p(h))$$

  where β is a hyper-parameter.

- **Disentangled β-VAE**[1]: The loss function of disentangled β-VAE is modified into the following form.

$$L = -\frac{1}{|\mathcal{B}|}\sum_{X\in\mathcal{B}} E_{h\sim q_\phi(h|X)}\left[\log p_\theta(X|h)\right] + \frac{|\mathcal{B}|}{|\mathcal{D}|}\cdot\beta\cdot |KL(q_\phi(h|X) \mid p(h)) - C|$$

  where C is an annealing constant, that is,

$$C = \min\left(\frac{iter}{iter_{stop}}, 1\right)\cdot C_{max}$$

  where $iter$ is the number of iteration, $iter_{max}$ is the maximum iteration that stops the annealing process, and C_{max} is a constant which is the maximum of C.

- **MSSIM-VAE**[20]: The reconstruction loss is replaced with the MSSIM loss.

- **DFC-VAE**[9]: The reconstruction loss is replaced with the MSE loss between original images and reconstructed images, as well as the feature loss between features of original images and reconstructed images, which is extracted by VGG-19 model[19].

4.3. Experiment: Image Generation

We conduct the experiments of image generation on SVHN dataset[16]. We have implemented VAE and its variants introduced in Sec. 4.2. For all the VAE models, we use the same architecture as follows:

- **Encoder**: 5 Convolution layers with batch normalization layer and leaky ReLU activation function of $32, 64, 128, 256, 512$ channels respectively, with kernel sizes of 3, strides of 2 and paddings of 1. Two fully connected layers are used to obtain μ and $\log\sigma$ from the extracted features of the last convolution layer.

- **Decoder**: 5 De-convolution layers with batch normalization layer and leaky ReLU activation function of $256, 128, 64, 32, 32$ channels respectively, with kernel sizes of 3, strides of 2, paddings of 1 and output paddings of 1. A convolution block of 3 channels with the kernel size of 3, the stride of 1, the padding of 1 is used to reconstruct the images.

We use the AdamW optimizer[15] with learning rate of 10^{-3} and an exponential learning rate scheduler with $\gamma = 0.95$ to train the variational auto-encoders and its variants for 50 epochs. The results of the generated images is illustrated in Fig. 3.

From the results we can observe that, VAE is able to generate well-quality results, while β-VAE can only generate similar results of digit 3 and 8. Disentangled β-VAE, MSSIM-VAE and DFC-VAE are able to generate great-quality images with lots of details such as background.

5. Conclusion

In this paper, we have explored various machine learning methods on SVHN dataset[16], such as logistic regression, support vector machine[2], deep neural networks and so on. We have studied the traditional machine learning methods based on HOG features[3], while the neural networks based on the features extracted by convolution layers. We also visualize the features for better illustrations. A great number of experiments are conducted to verify the conclusion on the textbook as well as the hypotheses we propose.

Besides traditional classification task on SVHN dataset, we also study the generative models on SVHN dataset including variational auto-encoders[12] and its variants [8, 1, 20, 9] by illustrating the generated images and comparing their qualities to show the effectiveness of the model.

In conclusion, this paper explores a great number of machine learning methods by conducting lots of meaningful experiments to verify the conclusions and hypotheses. All source codes and experiments' configuration files are available at https://github.com/Galaxies99/SVHN-playground.

Acknowledgements

I would like to express my deepest gratitude to Prof. Quanshi Zhang for the detailed introduction and explanation on machine learning algorithms, including traditional and contemporary methods.

I would like to thank the teaching assistants for answering our questions and providing us with supports.

References

[1] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *arXiv preprint arXiv:1804.03599*, 2018. 9, 10

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 5, 10

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005. 2, 10

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 1

[6] Jacob Gildenblat and contributors. Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`, 2021. 8

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6, 7

[8] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016. 9, 10

[9] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE, 2017. 9, 10

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 6, 7

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 8, 9, 10

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 6, 7

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[15] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *a*, 2018. 1, 2, 3, 7, 10

[16] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *b*, 2011. 2, 4, 6, 7, 9, 10

[17] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998. 5

[18] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 8

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6, 7, 10

[20] Jake Snell, Karl Ridgeway, Renjie Liao, Brett D Roads, Michael C Mozer, and Richard S Zemel. Learning to generate images with perceptual similarity metrics. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4277–4281. IEEE, 2017. 9, 10

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 6, 7

[22] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 1