Practical Work in AI
Pushing Baselines for Anomaly Detection on MVTec

Christian Michael Krickl
Johannes Kepler University Linz
k11942694@students.jku.at

Abstract

We perform experiments on the MVTec Anomaly Detection dataset with classic out-of-distribution (OOD) models, including: One Class SVM, Isolation Forest, Kernel Density Estimation, Local Outlier Factor and Elliptic Envelope. Various pretrained ResNet and ViT configurations are used as a preprocessing step on the images to overcome scaling problems with the high dimensional input. This is done by extracting multiple hidden layer activations of ResNet/ViT and using those as input for the OOD models. In addition to that, we experiment with data augmentation and do a simple hyperparameter search.

Averaged across all dataset objects, we achieve an AUROC of 0.970 and gain insights into why some augmentation techniques are counterproductive.

1. Introduction

1.1. Project Description

The MVTec AD dataset [1] consists of images from 15 different objects, whereas, each image shows either a good or faulty object (scratched, dislocated, etc). The task is to train an OOD model for each object only on its good images, with the prospect to separate good and faulty images in the test set.

We approach this anomaly detection task by reducing the high dimensionality of the images with pretrained ResNet/ViT models and use those extracted features to train One Class SVM, Isolation Forest, Kernel Density Estimation, Local Outlier Factor and Elliptic Envelope.

1.2. Motivation

Detecting outliers in the MVTec AD dataset is a rather solved task with the leading models having an AUROC performance of 0.997 [11]. We would like to find out what performance we can reach with traditional OOD approaches and gather further insights during the process, such as augmentation effectiveness and training/inference time.

2. Pipeline

2.1. Dataset

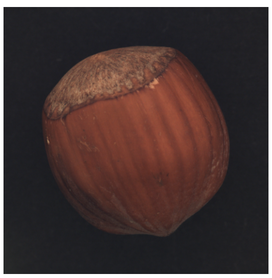
All experiments are performed on each object separately, which has the benefit that we can optimize the OOD model for each distribution.

Object	**Original**	**Augmented**
bottle	209	1254
cable	224	1344
capsule	219	1314
carpet	280	1680
grid	264	1584
hazelnut	391	2346
leather	245	1470
metal_nut	220	1320
pill	267	1602
screw	320	1920
tile	230	1380
toothbrush	60	360
transistor	213	1278
wood	247	1482
zipper	240	1440

Table 1. Training set sample size.

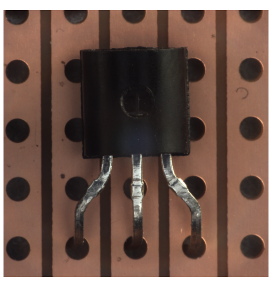
We augment the training set with the image transformations *rotate 90/180/270* and *flip horizontally/vertical*. Giving us an original to fabricated samples ratio of 1:5.

Early experiments with other image transformations (such as *resize* or *rotate+crop*) have shown to be counterproductive. The reason for this is that the images are taken in a very controlled environment and are expected to always be in the same lighting, angle and size. Furthermore, we will see in Table 6 which objects benefit from our applied *rotate* and *flip* augmentation because the orientation does not account for being a faulty object (e.g. *hazelnut* in Figure 1a/1b), while other objects depend on being oriented in a specific way (e.g. *transistor* in Figure 2a/2b)



(a) Example of good *hazelnut*.

(b) Example of good *hazelnut*, rotation does not account for being faulty.



(a) Example of good *transistor*.

(b) Example of faulty *transistor*, a rotated transistor is considered misplaced.

All experiments are performed on either the augmented or the original training data. As seen in Table 1, the number of training samples is between 60 and 2346, giving us the ability to see the effectiveness of OOD models with regard to training set size.

2.2. Feature Extraction

Because the OOD models do not scale well with high dimensional input data, we use various configurations of pretrained ResNet [8] and Vision Transformer (ViT) [5] to reduce the features to a lower dimensionality.

Architecture	**Initialization**	**Layer**
ResNet18	random imagenet1k_v1	layer1 layer2 layer3 layer4
ResNet50	random imagenet1k_v2	layer1 layer2 layer3 layer4
ViT-B/16	random imagenet1k_v1	encoder_layer_0 encoder_layer_1 encoder_layer_2 encoder_layer_3
ViT-B/32	random imagenet1k_v1	encoder_layer_0 encoder_layer_1 encoder_layer_2 encoder_layer_3

Table 2. Image feature extraction model configurations.

In Table 2 the layer references the node name in the PyTorch Vision [13] implementation, i.e. the *ViT encoder_layer_0* is the output from the first *Transformer Encoder* block, which includes all tokens. The initialization of the model weights are also sourced from the PyTorch Vision project, whereas *imagenet1k_v1* and *imagenet1k_v2* reference models which were trained on ImageNet [4] and the *random* initialization uses a distribution according to the default values of the PyTorch implementation.

The extracted features are used as training data for the OOD models. Note that the feature extraction models are used only as a preprocessing step and not trained in any way. We use the fact that these models have an architecture suited for images and that they may have learned an internal representation which is useful for other natural pictures.

2.3. OOD Models

Our code is based on the scikit-learn [14] implementations. We also perform a basic hyperparameter search, which is based on common values used for the respective model, those values are documented in Table 3.

2.3.1 One Class SVM

One Class SVM [17] uses a non-linear transformation to map the data into a higher dimensional space and maximizes the margin between the normal instances and the decision boundary, modeling the underlying data distribution.

2.3.2 Isolation Forest

Isolation Forest [12] utilizes Random Forest [9] by randomly selecting features and split values between their maximum and minimum. The method measures the path length required to isolate each sample, shorter path lengths indicate anomalies.

2.3.3 Kernel Density

Kernel Density [15] employs density estimation by utilizing Ball Tree or KD Tree for efficient lookup. It can use different kernels for a smooth estimation.

2.3.4 Local Outlier Factor

Local Outlier Factor [2] computes a score for each observation, indicating the degree of abnormality based on the local density deviation from its neighbors. The density information is obtained from the n-nearest neighbors, which is a tunable hyperparameter.

2.3.5 Eliptic Envelope

Eliptic Envelope [16] assumes the data comes from a know distribution (e.g. Gaussian) and fits a covariance estimate, creating an ellipse around the central data points.

Most scikit-learn implementations worked as expected, however, for Elliptic Envelope the memory complexity was too high for our machine with 32GB RAM. As a workaround we use Principal Component Analysis (PCA) [6] to reduce the dimensionality of the extracted features even further and introduce it as hyperparameter. PCA is in theory not ideal for this job because we eliminate dimensions with low variance, which might be highly relevant for outlier detection. It was quite surprising that the best performing experiment for *toothbrush* is archived with this approach.

Model	**Hyper-parameter**	**Values**
One Class SVM	kernel	linear rbf sigmoid poly3, poly5, poly8, poly13, poly21, poly34, poly55
Isolation Forest	n_estimators	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
Kernel Density	bandwidth	0.5 1.0 scott silverman
	kernel	gaussian tophat epanechnikov exponential linear cosine
Local Outlier Factor	n_neighbors	2, 4, 6, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64
Elliptic Envelope	PCA components	32, 64, 128, 256

Table 3. Evaluated hyperparameters.

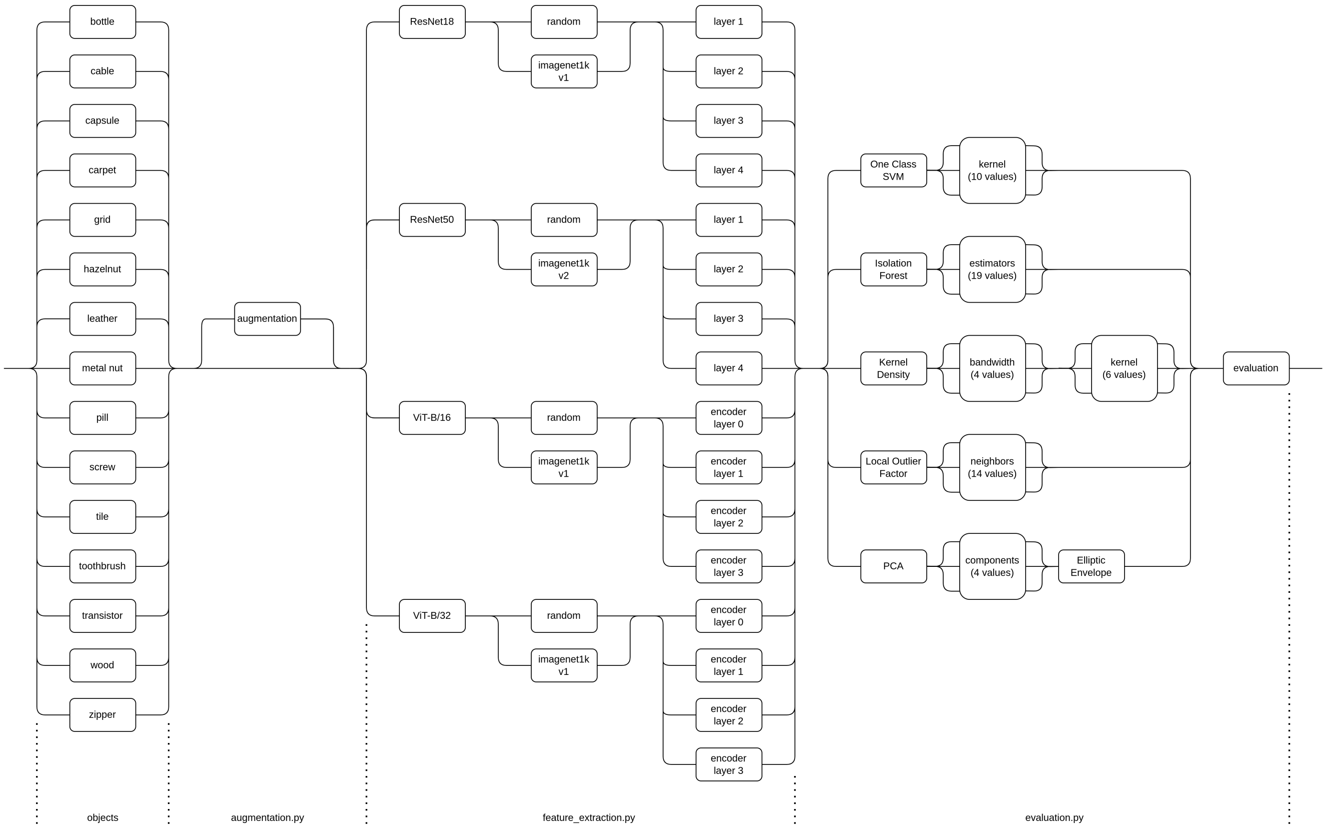


Figure 3. Possible experiment combinations.

3. Execution

Python was used as programming language for this task, including common libraries such as NumPy [7], Pillow [3], scikit-learn [14], torchvision [13] and Matplotlib [10]. The code is structured in a way that the augmentation, image feature extraction and training parts can be run separately. In addition to that, the training is made fault tolerant and will resume after a crash in an efficient manner. We can see in Figure 3 all possible experiment combinations and how the implementation is split into multiple Python scripts.

The augmented images, extracted features and evaluation results are cached on disk to make crash recovery backups and improve computation speed. Most engineering effort went into the correct extraction of the vision model layers and this fault tolerance.

3.1. Reproducibility

We ran our experiments on a PC with an AMD Ryzen 5600X and 32GB of memory. Due to bugfixes overall execution took about 2 weeks, however, raw compute time for augmentation is about 1 hour, feature extraction around 2 days and training/evaluation 9.85 days. To reproduce the results, 380GB of persistent storage is required.

Code and results as JSON/CSV/SQLite can be found here:
https://github.com/m1ckey/jku-practical-work

Results as Google Sheet:
https://docs.google.com/spreadsheets/d/16narg8QGXiT-fdJuZVnhQ9EA33XIcGu66Z39z9bWwdk/edit

4. Results

From the total 68160 experiments 378 failed due to crashes, all caused by Local Outlier Factor segmentation faults. Nevertheless, we reach an average AUROC of **0.970** and perfectly solve *bottle*, *screw* and *tile*.

Object	AUROC
bottle	1.000
cable	0.942
capsule	0.917
carpet	0.945
grid	0.962
hazelnut	0.993
leather	0.992
metal_nut	0.935
pill	0.945
screw	1.000
tile	1.000
toothbrush	0.994
transistor	0.947
wood	0.998
zipper	0.975
average	**0.970**

Table 4. Best AUROC performance for each object.

4.1. Runtime

As shown in Figure 4, we observe an approximately linear increase in training time with respect to training data dimensionality for all OOD models. From Table 5 we can additionally see that, with an average training time of 0.255 seconds, the best performing model in this regard is Isolation Forest. Elliptic Envelope has the best inference performance, however, this is not quite representative because it works with the heavily downprojected PCA output.

Sample size time complexity in Figure 4 is rather inconclusive.

4.2. Interpretation

From table 6 we can see which objects benefit from augmentation, which dependends on whether or not the transformations interfere with the controlled environment, in which the images are taken. Not being *rotate/flip* robust explains why *metal nut* (flip changes direction of teeth), *pill* (always horizontal), *screw* (flip changes the direction of thread), *transistor* (rotation makes it misplaced) and *zipper* (always vertical) do not benefit from augmentation. For *tile* there also exists an experiment with perfect AUROC that uses augmentation.

Model	Train time [s]	Test time [s]
Isolation Forest	0.255	1.885
Local Outlier Factor	2.761	1.126
Kernel Density	4.638	9.111
Elliptic Envelope	4.720	0.001
One Class SVM	34.769	2.452

Table 5. Average compute time.

ResNet18 and ResNet50 seem to be most suitable as feature extraction models and almost always use the non-random initialization. Furthermore, layer3 provides the best abstraction on average. When looking at the top-10 models (see Google Sheet / CSV) we observe that the ViT models seem to be more effective with a random initialization.

The most successful OOD model across all objects is Local Outlier Factor, being involved in 10 out of 15 best performing experiments (including the tie in *screw* and *tile*).

AUROC performance varies from 0.917 to 1.0. No definite link to what causes the worse performance for some objects could be established. A subjective guess is that our setup does not work well for defects which cover only a small image area. This would align with the worse performance observed for *capsule* (scratch), *carpet* (hole) and *pill* (crack). A major drawback of this solution is the inability to do anomaly segmentation on the original image.

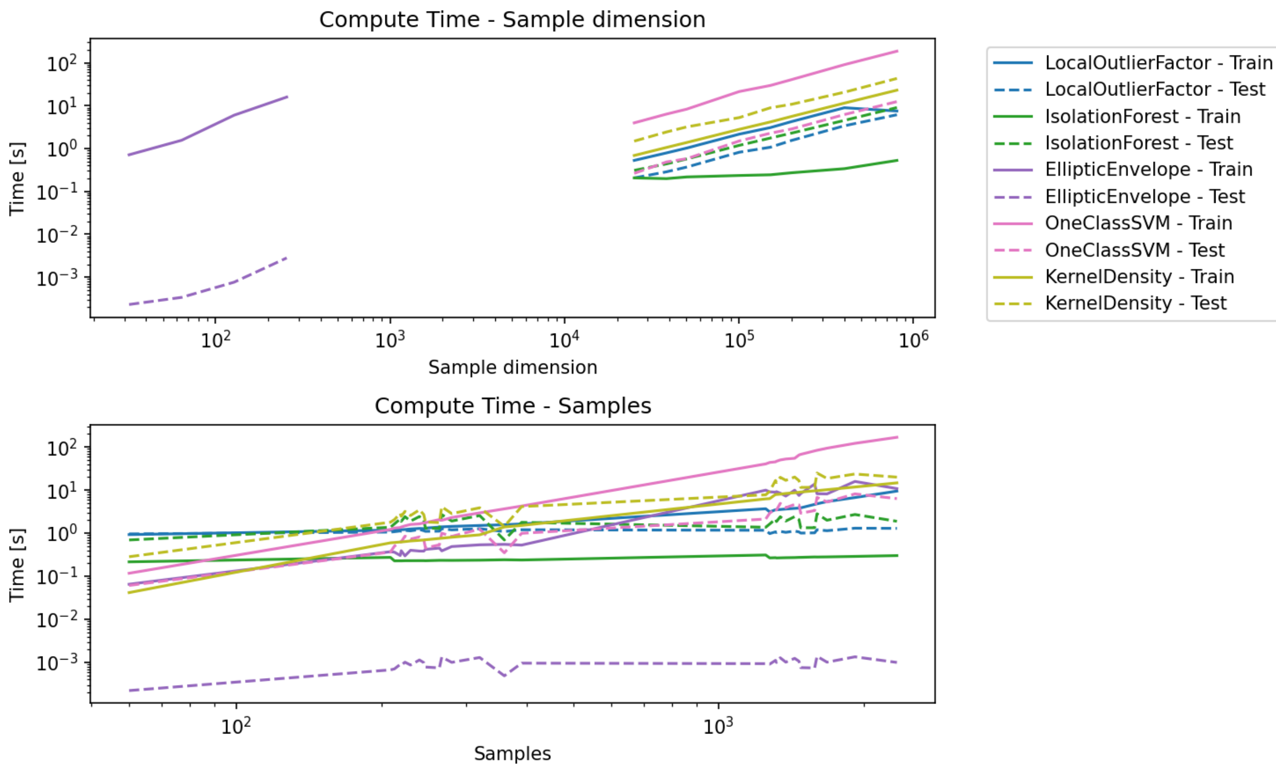


Figure 4. Average compute time by train data dimensionality and sample size.

object	augmented	feature extraction architecture	feature extraction initialization	feature extraction layer	model	hyperparameter	auroc	train time	test time
bottle	TRUE	resnet18	imagenet1k_v1	layer4	LocalOutlierFactor	n_neighbors: 4	1.000	0.619	0.218
cable	TRUE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 4	0.942	1.442	0.435
capsule	TRUE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 12	0.917	1.335	0.418
carpet	TRUE	resnet18	imagenet1k_v1	layer3	IsolationForest	n_estimators: 600	0.945	0.397	1.071
grid	TRUE	vit_base_16	random	layer2	OneClassSVM	kernel: poly5	0.962	72.939	3.376
hazelnut	TRUE	resnet50	imagenet1k_v2	layer3	LocalOutlierFactor	n_neighbors: 6	0.993	15.418	2.274
leather	TRUE	resnet50	imagenet1k_v2	layer4	LocalOutlierFactor	n_neighbors: 20	0.992	3.294	0.995
metal nut	FALSE	resnet50	imagenet1k_v2	layer3	OneClassSVM	kernel: poly8	0.935	1.829	0.773
pill	FALSE	resnet50	imagenet1k_v2	layer2	LocalOutlierFactor	n_neighbors: 2	0.945	3.604	3.214
screw	FALSE	vit_base_32	random	layer0	OneClassSVM	kernel: rbf	1.000	0.441	0.615
tile	FALSE	resnet50	imagenet1k_v2	layer3	IsolationForest	n_estimators: 800	1.000	0.586	5.878
toothbrush	TRUE	resnet50	random	layer2	EllipticEnvelope	pca_n: 32	0.994	0.124	0.000
transistor	FALSE	resnet50	imagenet1k_v2	layer3	LocalOutlierFactor	n_neighbors: 4	0.947	1.578	1.449
wood	TRUE	resnet50	imagenet1k_v2	layer4	OneClassSVM	kernel: rbf	0.998	54.235	3.461
zipper	FALSE	resnet18	imagenet1k_v1	layer3	LocalOutlierFactor	n_neighbors: 8	0.975	0.374	0.395

Table 6. Best performing experiments for each object.

5. Conclusion

We achieved a surprisingly good detection AUROC of **0.970** with our approach. The best performing models have a fast training and inference time, however, we needed an extensive search with 68160 experiments to find those, which makes it in total not very computationally efficient. We can confidently conclude that augmentation helps to mitigate the small dataset size, as long as it does not interfere with the object specific controlled environment assumptions. Furthermore, Local Outlier Factor with ResNet as feature extractor is best suited for this specific approach and yields best average AUROC performance and good training/inference times.

References

[1] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9584–9592, 2019. 1

[2] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, may 2000. 3

[3] Alex Clark. Pillow (pil fork) documentation, 2015. 4

[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. 2

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 2

[6] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. 3

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. 4

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[9] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995. 3

[10] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007. 4

[11] Jeeho Hyun, Sangyun Kim, Giyoung Jeon, Seung Hwan Kim, Kyunghoon Bae, and Byung Jun Kang. Reconpatch : Contrastive patch representation learning for industrial anomaly detection, 2023. 1

[12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. 3

[13] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. 2, 4

[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, nov 2011. 3, 4

[15] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956. 3

[16] Peter J. Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999. 3

[17] Bernhard Schölkopf, John Platt, John Shawe-Taylor, Alexander Smola, and Robert Williamson. Estimating support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 07 2001. 3