Leong Shi Yin, Vanessa

MINST Data Anomaly Detection

Task: In this project, I aim to identify anomaly in the popular MINST handwritten dataset, where digits 0 are labelled as normal data while digits 1-9 are labelled as anomaly data.

This report details the pre-processing and model building/validation approach that is used across all models explored (see Section 1 and 2), other methods experimented along the way (see Section 3) and final model (see Section 4). To ensure reproducibility of results, seeds were set to control for randomness. Two approaches were explored – **Anomaly Detection** where the model is only trained on normal data and **Classification** where the model is trained on both anomaly and normal data.

1. Data Pre-processing & Feature Engineering

1.1 Normalization

Given that MINST dataset is grey-scaled ranging from 0 to 255 and is of 28 x 28 pixels, all values of the pixels are being normalized to range from 0 to 1. Normalising is crucial in ensuring that all features are learnt at the same speed because gradient descent and updates of weights is proportional to the magnitude of the input. Separately, the labels class 0 and 1 were one-hot encoded.

1.2 Data Augmentation

With slightly over 6000 MINST labelled data, data augmentation is done to expand upon the training data. This is to ensure that when digits in MINST images are drawn in different rotation, size, position, the model would have been trained on similar images. Data Augmentation applied includes rotation, zoom, shifting images horizontally and vertically.

1.3 Splitting data

Using Stratified Random Split based on the labels, the data was split into different ratios based on the methods used. For anomaly detection method, the data was first split into the following ratio: 90% Training & Validation Data, 10% Testing Data. The normal labelled Training & Validation split is further split into 90% Normal Train and 10% Normal Validation. The remaining anomaly labelled Training Data is used for validation of model with the 10% normal data from the split above. For classification, the data was split into the following ratio: 70% Training Data, 15% Validation Data, 15% Testing Data. To ensure comparability of results across different models, a seed is set for the splitting of data.

2. Model Building and Validation

In model building, to accurately evaluate a model's architecture, the model is run 3 times with different weights initialized. They are then individually evaluated on the Testing Data and AUC-ROC score averaged across the 3 iterations to determine the overall performance of the model. This ensures that a model's performance was not affected by different initialization of weights for comparison across different architectures.

2.1 Choice of Activation Function – ReLu & SoftMax

ReLu is chosen to prevent vanishing gradient problem faced from sigmoid and tanh, thus speeding up training. In the last layer of the fully connected layers, SoftMax is used to compute the probabilities of normal and anomaly class for the given data.

2.2 Choice of Optimizers – Adam

In order to reach global minima, optimizers are usually used to guide the updates of model weights. While local minima are often of concern, another problem face is pathological curvature. This results in the model training for longer period to reach global minima as it oscillates around the ravine of the loss function multiple times. This occurs as it only computes the gradient and neglects the curvature of the gradient. As such, momentum is used together with gradient descent approaches to approximate the curvature of gradient. Hence, the following models will use Adam Optimizer as it adapts learning rate of each weight, unlike Stochastic Gradient Descent that uses a single learning rate for all weights updates. Having tested SGD and RMSProp, Adam is able to converge the fastest achieving higher AUC.

2.3 Dropout/Regularizes & Early Stopping

Differing rates of dropout was used to prevent overfit of the model. Dropout helped to prevent the neurons from co-adapting, this enables each neuron to capture most information. As neural network trains complex models, overfitting problem is prevalent, dropout significantly improved results. In certain models, L1 regularisation was used that reduces the parameters to zero, simplifying the model.

Early stopping is also used by monitoring the validation loss. A patience of 5 is set for all models, allowing training to persist for 5 more epochs in order to be confident that that is indeed the minimum validation loss. Upon early stopping, the weights of the best model giving lowest validation loss will be returned. Validation loss is used instead of validation accuracy due to the nature of anomaly detection where there is a significant class imbalance in the data, thus affecting the accuracy rate.

3. Models Explored & Evaluation of Models

Two main techniques were used **Autoencoder** and **Convolutional Neural Networks** (CNN)**.** Autoencoder has been used for two purposes, first as a classifier and second to determine the reconstruction errors for anomaly data. CNN was used as a binary classifier.

3.1 Autoencoder

3.1.1 Autoencoder for Classification

Autoencoder aims to reconstruct its inputs to learn a lower dimension representation layer. For classification, the lower dimension layer is used to make predictions of the normal and anomaly data. This is achieved by using the weights of the encoder layers obtained from the training and adding a fully connected layer to build the neural network for classification of normal and anomaly data.

➔ **Denoising Autoencoder & Results**

In Denoising Autoencoder approach, the input images were corrupted with Gaussian noise before being fed into the autoencoder network that aims to output the original (noise-free) images. Additionally, to corrupt the images, masking noise was introduced to the network by applying drop out.[1] Input images corrupted with noise will allow for more powerful representation ability given the variances in the images. Thus, this allows robustness and generalization of model to prevent overfitting of the encoder layer.

- **Model 0:** When this encoder was used with the fully connected layers to classify normal and anomaly data using SoftMax activation function. This achieved an average 0.999062 on test data but 0.55410 on Kaggle test

Rationale for why classification autoencoders did not perform well

A possible rationale for the poor performance of autoencoders for classification is that the feature extraction is trained separately from the classification task. Hence, the reduced dimension representation layer is not adjusted to the classification done in the fully connected layer. This result

[1] https://ieeexplore-ieee-org.libproxy1.nus.edu.sg/document/7407967

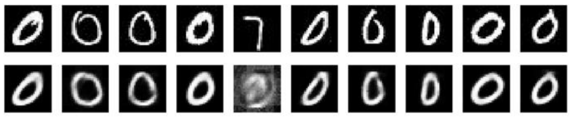
in a lack of representation required to distinguish the categories. [2]As such, this motivated the next approach of using the reconstruction errors of autoencoder to identify anomaly and normal data.

3.1.2 Autoencoder Reconstruction Errors

By training autoencoders on normal data alone, it will be able to reconstruct normal images with low error rate, while for the case of anomaly images, high error rate. In order to check for error rates, Mean Square Error (MSE) is used.

➔ **Shallow Autoencoder**

- **Model 1**: To start off, a shallow fully connected autoencoder was used. This comprise of a single Dense layer each in encoding and decoding. This achieved an average AUC of 0.99905 on testing dataset and 0.98522 on Kaggle test case.
- **Model 2**: In order to prevent overfitting, L1 regularisation was used, resulting in an AUC of 0.99901 on testing data and 0.98957 on Kaggle. The model performed better on Kaggle testcase, which shows that the regularization helped to reduce overfitting.



➔ **Deep Autoencoder**

- **Model 3:** With increased depth in the autoencoder, it learned more complex features from the images. Hence, six dense layers were used in total for encoder and decoder for symmetry. This achieved an average AUC of 0.96879 which happens to be lower than the Shallow Encoder models. However, it performed poorly on Kaggle test data, this could be attributed to the larger number of parameters and layers comparative to the training size. This results in the model overlearning, for example, the model would learn that the shape of 0 to be a continuously joint circle but end up misclassifying 0 digits that are disjointed.

➔ **Denoising Autoencoder**

- **Model 4:** As the Shallow Encoder performed better, denoising autoencoder is attempted to provide robustness to the model. Gaussian noise is added to the normal training data. This achieved an average AUC of 0.99515 on testing dataset. The amount of noise introduced to the images might have affected the model's ability to learn the features of a normal data.

3.1.3 Autoencoder with Convolution Layers

Instead of using fully connected layers in the autoencoder, I proceeded to try out using convolution and pooling layers instead to extract local features.

- **Model 5:** 3 pairs of convolution-pooling layers were used with dropout to control overfitting. This achieved an average AUC of 0.99329 on testing data and 0.99828 on Kaggle test. This result seems promising and the spatial features are deemed to be important in identifying normal data. Hence, motivating the subsequent variants of replacing the max pooling layer and introducing smaller stacked convolution layers.

[2] https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

➔ **Max Pooling Layer vs Learnable Pooling Layer**

Pooling layers are used to down sample the feature maps in order to make the network more robust to differences in location of features on the input data. An alternative to Max Pooling is to use a Convolution Layer of different strides to down sample the feature maps, resulting in a learnable layer.

- **Model 6:** Convolution layers of 5x5 filter with 2 strides is used as the learnable pooling layer. This achieved an average AUC of 0.98701 on testing dataset.

➔ **Stacked Convolution Layers**

With reference to VGG16, it was seen that by stacking convolutional layers with smaller filters allowed the network to learn more complex features as it introduced more depth and non-linearity.

- **Model 7:** Motivated by VGG16, this model used multiple stacks of 3x3 convolutional filters. This achieved an average AUC of 0.99640 on testing data and 0.98396 on Kaggle test.

Model	Description	Average AUC Test Data	Kaggle Results
1	Shallow Autoencoder	0.99905	0.98522
2	Shallow Autoencoder w L1 Regularization	0.99901	0.98957
3	Deep Autoencoder	0.96879	-
4	Denoising Autoencoder	0.99515	-
5	Six Convo(3x3)-Max Pooling Autoencoder	0.99329	0.99828
6	Six Convo(5x5)-Learnable Pool Autoencoder	0.98701	-
7	Six Stacked(3x3)-Learnable Pool Autoencoder	0.99640	0.98396

3.3 Convolution NN

Convolution NN is used to train a classifier to predict the probability of a class being normal or anomaly. Similar architectures used in the convolution & pooling layers in autoencoder is used below as well.

3.3.1 Convolution Layers

- **Model 1**: To start off, a basic model was trained with two pairs of convolutions (3x3 filters) & pooling layers with dropout. This achieved 0.99085 AUC score on the testing set.
- **Model 2**: Motivated by VGG16, the next model was built using multiple stacks of 5x5 convolutional filters. This achieved 0.99304 AUC score on testing set and 0.85899 on Kaggle. The use of stacked layers increased the depth of the network, allowing for more complex features to be learnt.
- **Model 3:** The next model was built using smaller filters – multiple stacks of 3x3 convolutional filters. This achieved 0.99486 AUC score on testing set. We thus see an increase in performance which result in subsequent models using multiple stacks of 3x3 convolutional filters. Having submitted this result, it achieved a 0.86655 on Kaggle test data.

3.3.2 Max Pooling Layer vs Learnable Pooling Layer

- **Model 4:** With all else similar to model 3, model 4 was built to replace max pooling layers with learnable pooling layers. Hence Convolution layers with 2 strides were used. This achieved 0.98809 AUC-ROC score on the validation dataset.

3.3.3 Batch Normalization

Batch Normalization applied to the hidden layers will ensure that the variances and differences in distribution of the input image will not affect the model largely, thus allowing features to be learnt equally.

- **Model 5:** Comparing to Model 4, Model 5 uses the same architecture but includes batch normalization after each layer. This achieved 0.99267 AUC-ROC score on testing set. Although this performed slightly lower than Model 3 on the test data, it gave a better performance of 0. 94650 on Kaggle test data.

Rationale for why CNN Classification did not perform as well

A possible rationale for the lower performance in CNN classification model is because of the large variation in the anomaly class that comprise of digits from 1 to 9. With that, the locally learnt features in convolution layers may not be able to do a good job as there would have been too much variances in the features. As a result, a digit 9 will be more similar to the digit 0 as compared to the digit 7. Additionally, due to the nature of anomaly detection problem, the problem of class imbalance would have contributed to its poor performance due to the lack of training on anomaly class.

Model	Description	Average AUC Test Data	Kaggle Results
1	Two Convo(5x5)-Max Pooling Pairs	0.99085	-
2	Two Stacked (5x5) Conv-Max Pooling	0.99304	0.85899
3	Two Stacked (3x3) Conv-Max Pooling	0.99486	0.86655
4	Two Stacked (3x3) Conv-Learnable Pool	0.98809	-
5	Two Stacked (3x3) Conv-Learnable Pool + Batch Normalisation	0.99267	0. 94650

4 Final Model Architecture

Given that the classification models did not work well for the task of anomaly detection, autoencoders trained using normal data outshine significantly. As such, the **Shallow Autoencoder with L1 Regularisation** that learns sparse representation of images and **Convolution Autoencoder with Max Pooling** that learns spatial features were used for the final 2 predictions.