Assignment 4

Group 28

Danylo Vasylyshn, 1815709

Sam van der Velden, 1017871

Stan Meijerink, 1222737

Problem formulation

The goal of the problem is to classify an image into known classes in a partially labeled dataset with anomalies. While the goal consists of two parts, classifying images as well as identifying anomalies, the task could be described as a classification task with anomaly detection.

The data is split into 4 groups, one without labels or anomalies, one with labels but no anomalies, and two with both labels and anomalies. The last two groups are representative data sets and thus will be used for testing purposes.

As a one-hot verdict is made for all classes, the most effective way to measure the performance of this model is the ratio of the true positive rate over the false positive rate. Or more simply, the accuracy of all positive measures, when testing against the anomalous data. This is reflected in the ROCcurves shown in the evaluation section of this report.

Model formulation

For this model transfer learning is used between two new models. In the discussion, the element of transfer learning will be discussed in detail. The first model reduces images to 2d vector space with the use of a variational auto-encoder. As this task does not take labels into account this model is trained with the unlabeled data, as it is the largest.

The second model uses linear layers and softmax to classify images. Part of the first model is used in the second model which is trained to decode and classify images using the limited available labeled data.

The loss function for the first model is the Kullback-Leibler divergence together with the reconstruction loss measured by the mean squared error. The KL-loss is given by the formula:

$$KLLoss(mu,\ logSigma)\ =\ 0.5\ \cdot\ sum(mu^{2}\ +\ e^{2*logSigma}\ -\ 2\ \cdot\ logSigma\ -\ 1)$$

In this mu is the output of the second last linear layer of the encoder model, $logSigma$ is the output of the last layer of the encoder model.
The MSE loss is given by the following formula:

$$ReconstructionLoss(\widehat{x},\ x)\ =\ MSE(\widehat{x},\ x,\ reduction\ =\ sum)$$

In this formula $\widehat{x}$ are the predicted pixel values of the image while x are the actual pixel values of the image. With the reduction mode of “sum”, the mean squared error is then calculated.
This makes the total loss of the first model the following formula:

$$FirstModelLoss\ =\ KLLoss\ +\ ReconstructionLoss$$

The loss function of the second model is the cross-entropy loss as described below:

$$CrossEntropyLoss(pred,\ label) = \{l_1, \dots, l_N\}^{\top}$$

$$l_n = \ -w_{y_n} \cdot log \frac{exp(label_{n,\ pred_n})}{\sum_{c=1}^{c} exp(label_{n,c})} \cdot 1$$

In this formula $pred$ are the predictions of which label it should be, $label$ is the correct label for each data point, w_y are the weights of the predictions and c are the number of classes.

Implementation and training

The first model (see Table 1) consists of an encoding and a decoding phase and is trained using the unlabeled data without anomalies. It goes through 4 linear layers. After the first two linear layers, a rectified linear unit function is applied. The result from this layer is passed on to two other linear layers, generating the outputs henceforward named mu and log_sigma. Mu, sigma, and epsilon are used for reparameterization according to the following formula ($z = mu + sigma \cdot epsilon$), where sigma is the exponential of log_sigma, and epsilon is a tensor of length sigma with random samples from a normal distribution. The decoding phase consists of 3 linear layers and aims to reconstruct the image from the representation in latent space. The initial layer embeds 30 input dimensions into 400, the second layer embeds 400 into 400 and the last layer embeds 400 into 1024 dimensions which are identical to the starting dimensionality. After the first 2 layers, a rectified linear unit function is applied. On the output of the last layer, the sigmoid function is applied. The loss function used to backpropagate over this is the loss when reconstructing the image from the embedding by the encoder into latent space. This loss is defined by the mean square error of the pixel values.

Encoder layers		**Decoder layers**
Linear(1024, 400)		Linear(30, 400)
ReLu()		ReLu()
Linear(400, 400)		Linear(400, 400)
ReLu()		ReLu()
mu = Linear(400, 30)	sigma = Linear(400, 30)	Linear(400, 1024)
z = mu + sigma * epsilon		Sigmoid()

Table 1: layers of the encoder and decoder of the first model.

The second model (see table 2) uses the embedding of the first model along with 2 linear layers. The first layer embeds 30 to 200 dimensions after which a rectified linear unit function is applied, then there is a layer that embeds from 200 to 5 dimensions. The results are then run through a softmax function to end up with a probability distribution for the 5 classes. These classes are used for classification where the maximum likelihood is used to classify. The cross-entropy (as described in the model formulation) is then used to calculate the loss for backpropagation.

Classifier model
Linear(30, 200)
ReLu()
Linear(200, 5)
Softmax()

Table 2: layers of the classifier model

Experimentation and evaluation

Experimentation with the first model yielded little improvements. The dimension of the latent space was changed from 10 to 30. Which decreased the loss from 22.5 to 22.2. The aforementioned linear layers in the encoding phase were previously convolutional layers with kernel size 5 and padding 0, stride 1 for the first layer, and 2 for the other 2 layers. This did not yield an increase in performance. Thus with running time in mind the decision was made to convert these to linear layers. Secondly, the hidden dimension from the classifier was changed to combat underfitting at low dimensionality.

The model was evaluated again shortly in the graph below (see figure 1). A sample was taken of the labeled dataset in which the model is able to correctly classify 89/100 samples.

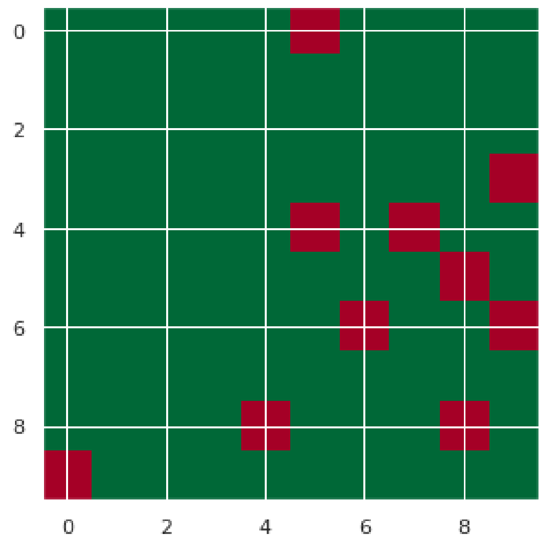

Figure 1: Evaluation of the classification model

Additionally the model was tested against anomaly data. For this, the anomaly-likelihood was calculated by calculating the Elbo elementwise. This was done by first calculating the aforementioned KL-loss and reconstruction loss, which was then combined with a constant term which is denoted by the formula: $ConstantTerm = xDim \cdot 0.5 \cdot log(\pi)$.
Where x_dim is the dimensionality of the input. Sigma is not included in this formula as it is assumed to be constant at 1/sqrt(2)

The image below represents a likelihood distribution where it can be seen that the likelihoods for anomalies are decently separated from the normal cases. This can be further seen in the roc curve below (see figure 2).

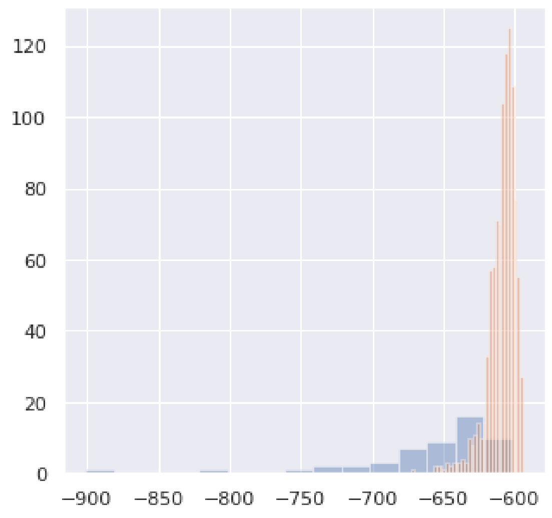

Figure 2: Distribution of the anomaly likelihood (blue) against the normal likelihood (yellow)

In the curve below (see figure 3) we can see the ROCcurves denoting the ratio of the false positive rate and the true positive rate. A false positive represents a normal case identified as an anomaly while a true positive case represents a correctly identified anomaly. As can be seen, the reliability of the model is quite decent. The image contains two models for 2 sample images from the two representative test sets.

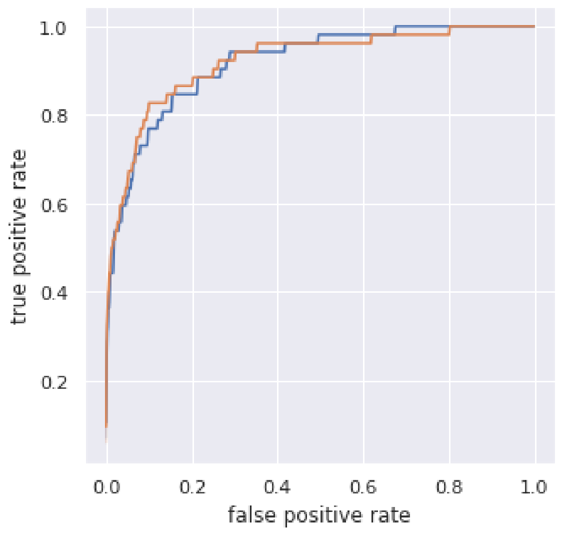

Figure 3: ROCcurve of the true positive rate against the false-positive rate

Conclusion

Despite having few samples containing labeled data, by applying transfer learning from a model trained to embed images in latent space, we can more effectively train a classifying neural network with limited data. The ROCcurve shows that the model performs significantly better than a random allocation.

Discussion

As far as potential improvements go, there as some suggestions that can be made:

- The model is trained on images of objects taken from the same angle. The generalization, therefore, does not extend to the object but rather the silhouette of the object at a similar angle.
- The model is trained on low-quality images. Higher quality images would not only allow for better generalization. It would also allow for potential subcategorization as for instance a secondary model can be trained with transfer learning to classify different kinds of shoes.

Concerning transfer learning; In this case, the use of transfer learning is suitable as the training set for the initial variational auto-encoder network and the training set for the secondary classification network are both subsets of the same data, thus they both share the same features. The exception to this are the anomalies in the tertiary and quaternary dataset, but these are labeled as such and considered their own class. Had the data for the primary network contained different features or simply less or more of them compared to the second network, transfer learning would not have been suitable as the learned embeddings do not generalize properly to the dataset the model is transferred to.