St. Pölten University of Applied Sciences

/ medien & digitale technologien

A Machine-Learning Based Approach for 2D Character Animation

<Subtitle>

Bachelor Thesis

Bachelor Course on Creative Computing
at St. Pölten University of Applied Sciences

Submitted by:
Georg Becker
01228308

Advisor: <Pre-Title> <FirstName> <LastName>, <Pos-Title>

<Place>, <DD>.<MM>.<YYYY>

Declaration

I assure that

- I have written this work independently, have not used other sources and aids than those indicated and have not made use of any other unauthorized assistance.

- I have not yet submitted this topic to an assessor in Austria or abroad for assessment or in any form as an examination paper.

- this work corresponds to the work assessed by the assessor.

Date: ______________________________ Signature: ______________________________

Abstract

Introduction: Warum behandeln wir das Thema

Purpose: Welches Problem soll gelöst werden

Method: Wie wurde die Problemlösung gemacht

Product: Was war das Ergebnis

Conclusion: Was sind die Folgerungen / Schlussfolgerungen aus den gewonnen Erkenntnissen

keine Referenzen und Zitate

Contents

1 Introduction **1**

2 Method **2**

3 Results / Ergebnisse **15**

3.1 First Section 15

3.1.1 First Subsection 15

4 Discussion / Diskussion **16**

5 Conclusion / Fazit **17**

Bibliography **18**

List of Figures **19**

List of Tables **20**

List of Listings **21**

Appendices **22**

A Appendix 22

B Appendix 23

1 Introduction

2 Method

Literature review

I reviewed previous work, focusing on two areas. I explored already available methods for creating animations from sketches by performing skeleton classification and reviewed previous work dealing with the classification of sketched objects.

Related work

Eitz et al. (2012) collected a dataset of 20,000 sketches and divided them into 250 categories of 80 images each. Humans recognized on average 73.1% of these sketches correctly. This dataset is used in my work to train and validate the classifier to choose which animation is the most appropriate to show.

Huang et al. (2022) proposes a pipeline to create rigged and animated characters from a single image. Their solution aims for a holistic approach, requiring no user intervention, to assist non-professional users in creating animated characters. The proposed pipeline performs contour extraction with salient object detection and extrudes a 3D mesh from geometry generated by applying constrained Delaunay to the contours. Afterwards, a skeleton is estimated using a mean curve method and an animation is transferred onto the skeleton. In our work, we want to follow a similar philosophy of no user interaction and hope to improve the believability of the animated results by not only classifying the skeleton type but also the subject class of the input sketch.

In Smith et al. (2023b) a system for automatically animating drawings of human figures is introduced. The system comprises of a figure detection step using Mask R-CNN He et al. (2018) to extract the figures' bounding box and utilises a pipeline of quantitisation, morphological closing, dilating, flood filling and retention of the largest remaining polygon for extracting the subject's mask. This gave more effective results than using Mask R-CNN for their usecase. We will investigate both apporaches for our work to decide which one we want to use for extracting masks for sketeches of quadrupeds. The introduced pipeline then uses a custom pose estimation model based on a ResNet-50 backbone to predict the current pose. The result of the pose estimation and masking process is used to generate a rigged model by using the predicted joint positions to determine the skeleton and assigning each triangle of the mesh to the bone, nearest to the triangle's centroid. For animating the pipeline uses motion captured data of human performers to retarget 3D motion to the 2D result.

Training classification models

We used a subset of the dataset provided by Eitz et al. (2012) and Sangkloy et al. (2016) to train our classification models. Only the classes "cat" and "dog" were taken as training data for our models. To train and evaluate our models we used the scikit-learn library introduced by Pedregosa et al. (2011).

kNN classifier

We trained a kNN classifier with the pixel values of the input images. Before using the values to train the model, we resized the images to a size of 64 times 64 pixels, and flattened the array to get a feature vector with 12288 entries, ranging from 0 to 255 in value. To find the best-performing k, we performed a grid search with cross-validation on 3 folds leading to k = 5 as the model with the highest accuracy at 61.8798%.

SVM classifier

We trained an SVM classifier with a total of 1544 labeled images of sketches of cats and dogs. Before training the model we resized the images to a size of 64 times 64 pixels, and flattened the array to get a feature vector with 4480 entries. The images were imported as grayscale images. The SVM classifier performed with an accuracy of 53.7578%.

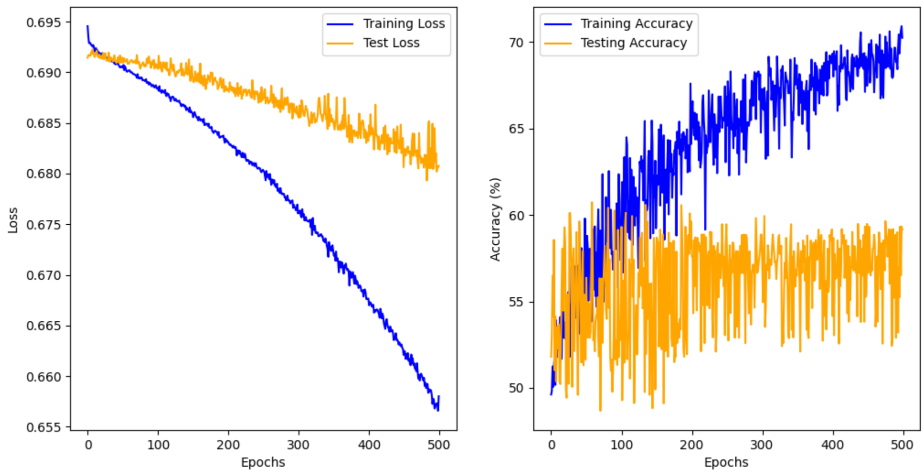
Neural network classifier

We created and trained a Neural Network binary classifier using pytorch Paszke et al. (2019). This is the network's setup:

```
class MyNN(nn.Module):
def __init__(self):
    super().__init__()
    self.linear_relu_stack = nn.Sequential(
        nn.Linear(in_features=3*128*128,out_features=16*16),
        nn.ReLU(),
```

```
            nn.Linear(in_features=16*16,out_features=16*16),
            nn.ReLU(),
            nn.Linear(in_features=16*16,out_features=1)
        )
def forward(self, x, **kwargs):
        x = x.view(x.size(0), -1)
        logits = self.linear_relu_stack(x)
        return logits
```

The network was trained on 80% of a collection of in total 1544 labeled images, depicting sketches of cats and dogs. The images where resized to a size of 128 times 128 pixels before being fed into the network. Training the network on 80% of the dataset for 500 epochs with a learning rate of 0.0001, led to the network performing with a 59.15% accuracy on the unseen test data.



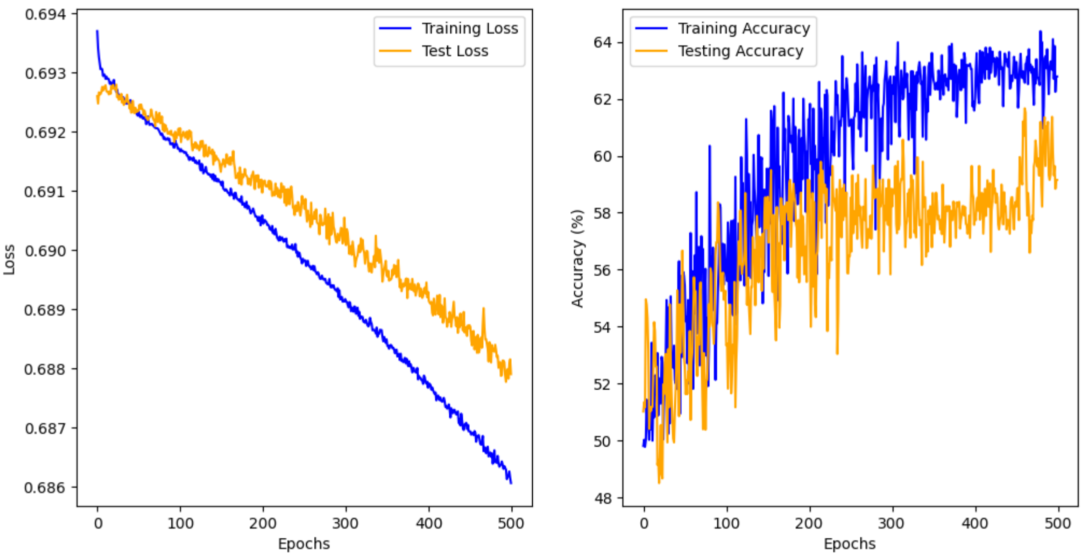
CNN classifier

We trained a Convolutional Neural Network classifier. This is the network's setup:

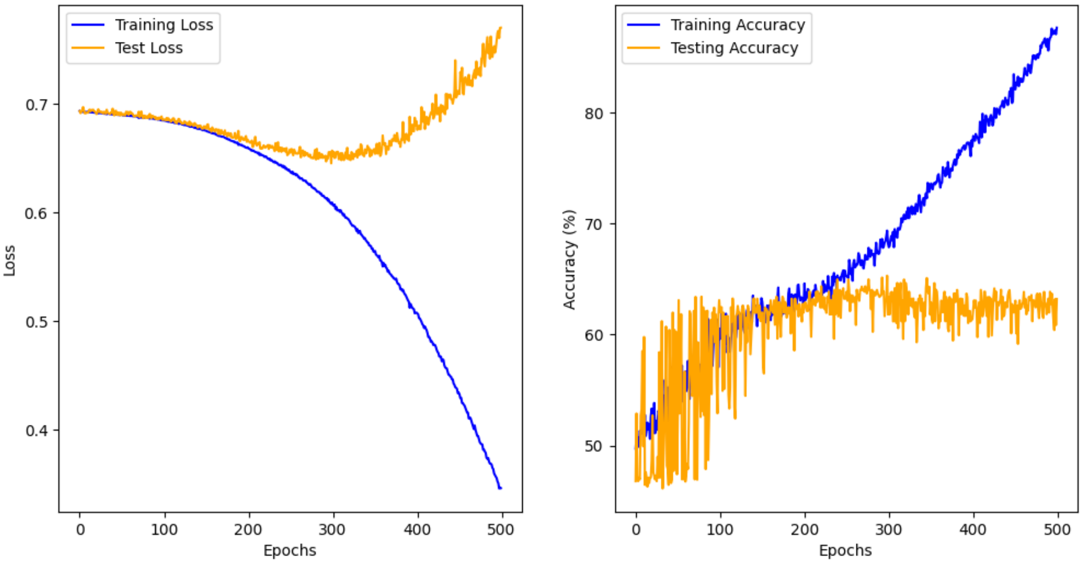
```
class MyCNN(nn.Module):
def __init__(self):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
        kernel_size=3, stride=1,
        padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
        kernel_size=3,
        stride=1,
        padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

```

```
    self.fc1 = nn.Linear(64 * 32 * 32, 512)
    self.fc2 = nn.Linear(512, 1)

def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-1, 64 * 32 * 32)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

Training the network analogous to the previously mentioned Neural Network classifier for 500 epochs with a learning rate of 0.0001, yielded an accuracy of 59.15% on unseen test data.



Repeating the setup with a learning rate of 0.001 led to an accuracy of 63.2% on unseen test data.



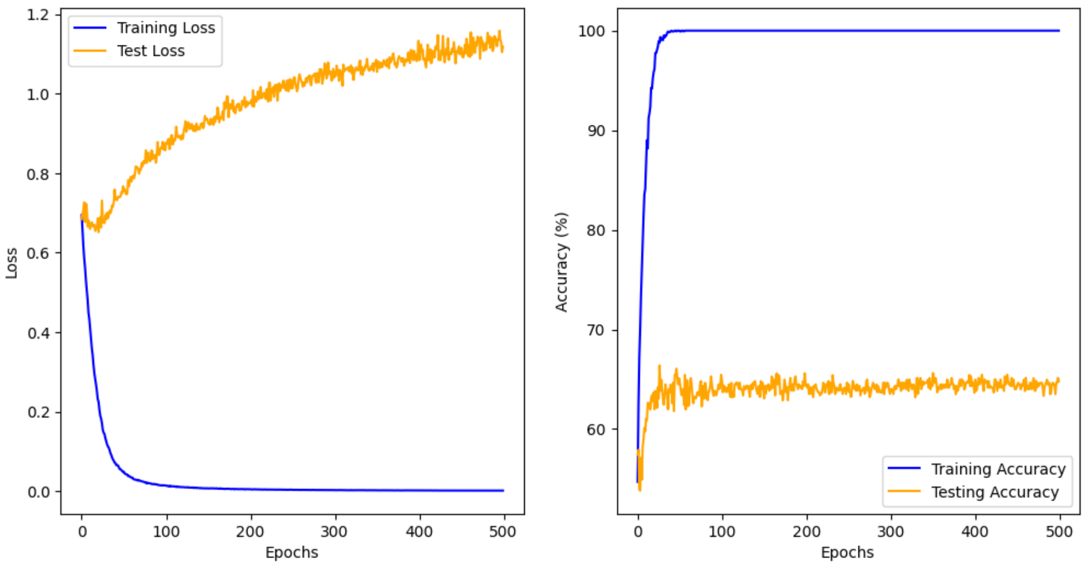
Enhanced CNN

We used a slightly more complex CNN architecture adding batch normalization after each convolutional layer. Also we added an additional Convolution. Additionally we use a dropout layer before the final fully connected layer.

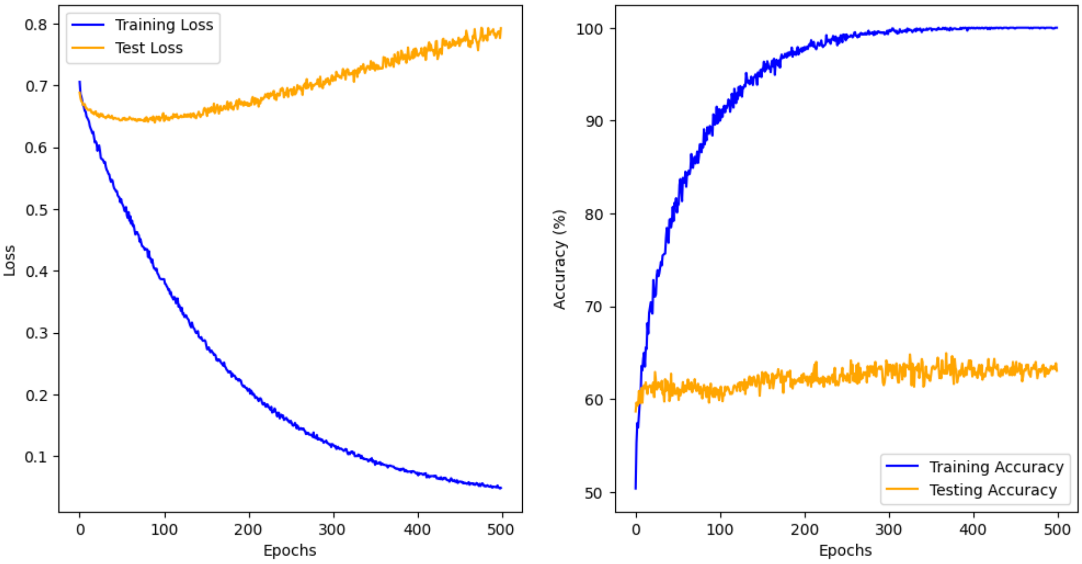
```
class EnhancedCNN(nn.Module):
    def __init__(self):
        super(EnhancedCNN, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
         kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=32,
         kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
        kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=64,
        kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(64 * 32 * 32, 512)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 1)

```

```
    def forward(self, x):
        x = self.pool1(
            F.relu(
                self.bn2(
                    self.conv2(F.relu(self.bn1(self.conv1(x))))
                )
            )
        )

        x = self.pool2(
            F.relu(
                self.bn4(
                    self.conv4(F.relu(self.bn3(self.conv3(x))))
                )
            )
        )

        x = x.view(-1, 64 * 32 * 32)

        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.fc2(x)
        return x
```

Training this model with a learning rate of 0.001 for 500 epochs resulted in an accuracy of 64.79% on previously unseen test data. It yielded an accuracy of 100% on the training data. We therefore suspect it to be overfitted.



Training the model with a learning rate of 0.0001 for 500 epochs resulted in an accuracy of 64.07% on previously unseen test data.

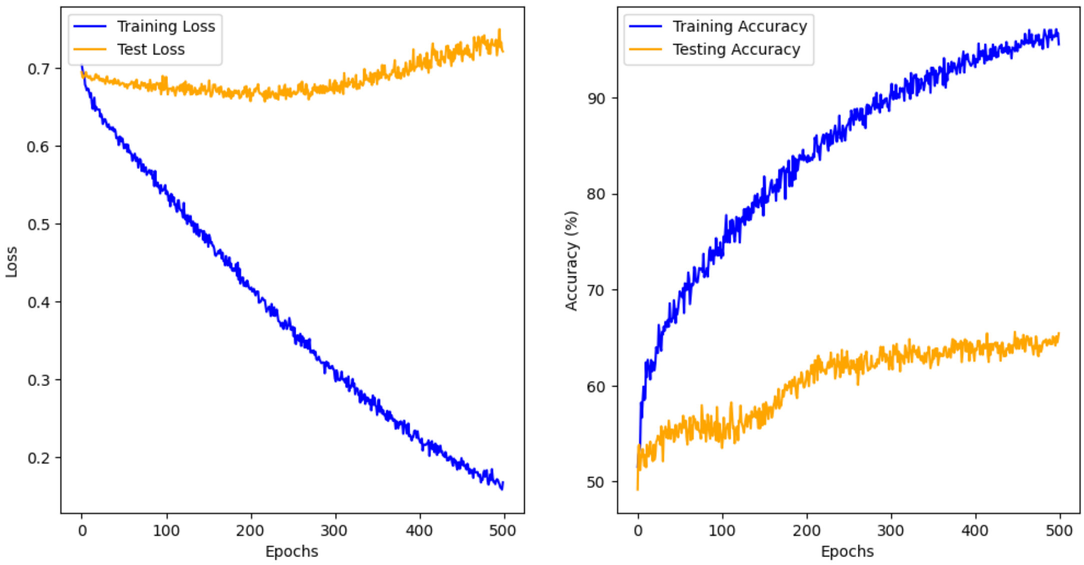


We introduced 2 additional dropout layers to the network architecture. The network is described like this:

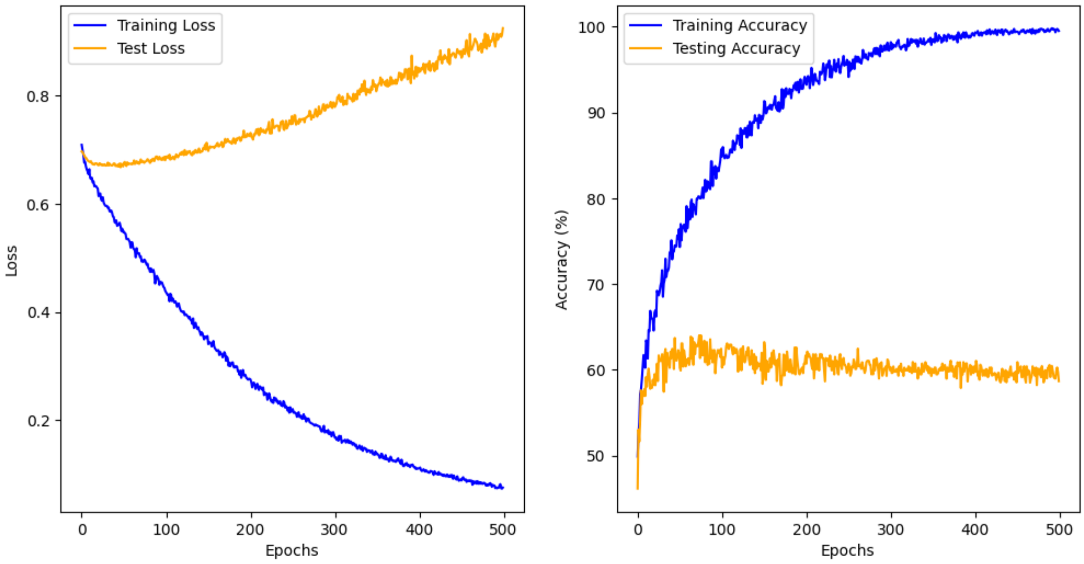
```
class EnhancedCNNMoreDropout(nn.Module):
def __init__(self):
    super(EnhancedCNNMoreDropout, self).__init__()

    self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1)
    self.bn1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1)
    self.bn2 = nn.BatchNorm2d(32)
    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
    self.dropout1 = nn.Dropout(0.3)

    self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1)
    self.bn3 = nn.BatchNorm2d(64)
    self.conv4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1)
    self.bn4 = nn.BatchNorm2d(64)
    self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
    self.dropout2 = nn.Dropout(0.4)

    self.fc1 = nn.Linear(64 * 32 * 32, 512)
    self.dropout3 = nn.Dropout(0.3)
    self.fc2 = nn.Linear(512, 1)

def forward(self, x):
    x = self.pool1(F.relu(self.bn2(self.conv2(F.relu(self.bn1(self.conv1(x)))))))
```

```
        x = self.dropout1(x)

        x = self.pool2(F.relu(self.bn4(self.conv4(F.relu(self.bn3(self.conv3(x)))))))
        x = self.dropout2(x)

        x = x.view(-1, 64 * 32 * 32)

        x = F.relu(self.fc1(x))
        x = self.dropout3(x)
        x = self.fc2(x)
        return x
```

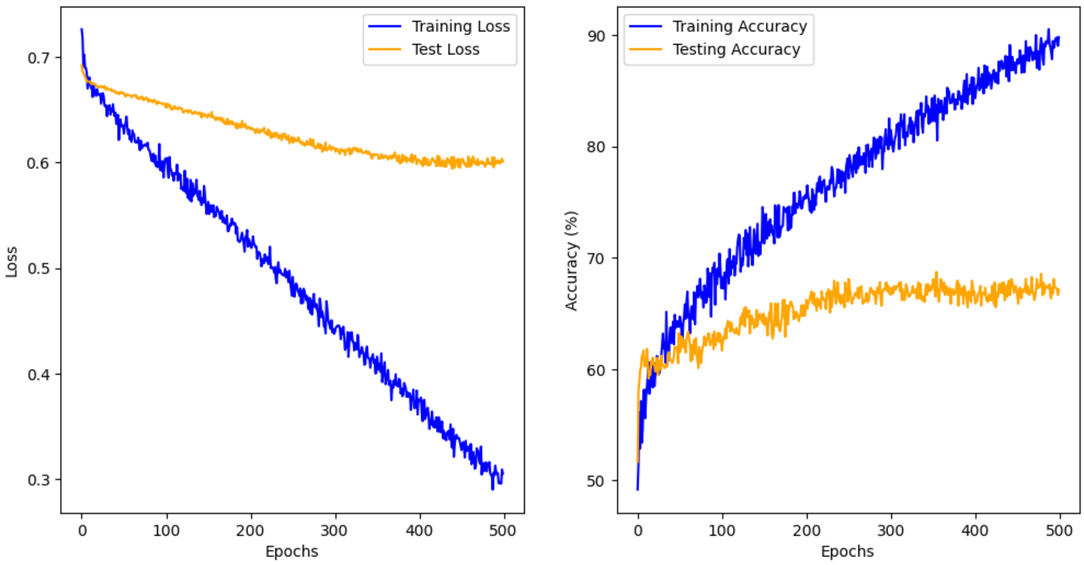
Introducing a dropout with a value of 0.3 after each convolution block resulted in a model that achieved 65.43% on previously unseen test data after being trained for 500 epochs with a learning rate of 0.0001.



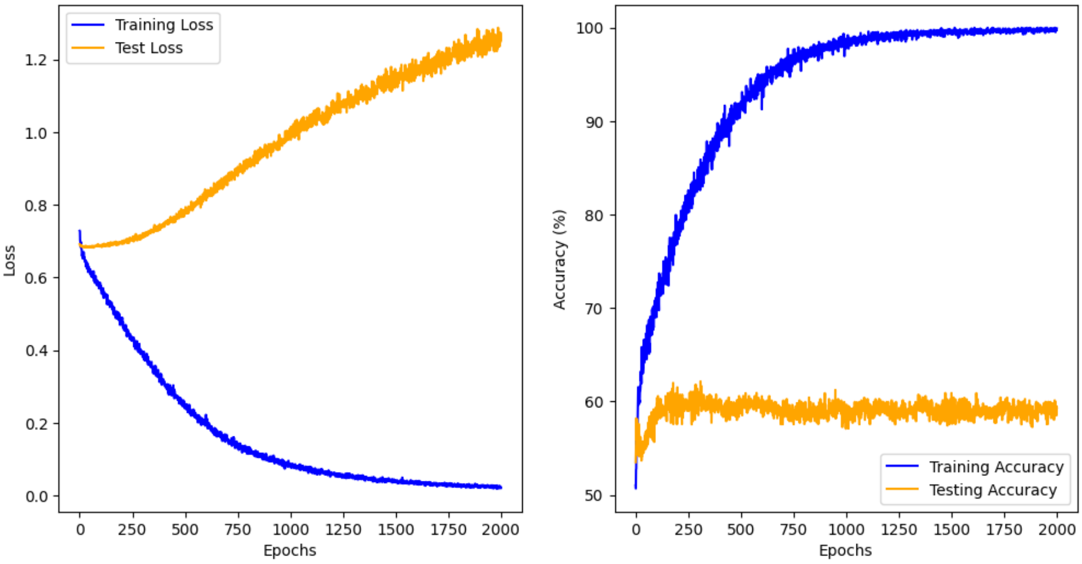
Setting the dropout to a value of 0.1 resulted in a model that achieved an accuracy of 58.68% on previously unseen test data after training for 500 epochs with a learning rate of 0.0001.



Setting the dropout to a value of 0.5 resulted in a model that achieved an accuracy of 67.14% on previously unseen test data after training for 500 epochs with a learning rate of 0.0001. The graph led us to believe the model might benefit from training for a larger amount of epochs.



Training with 2000 epochs did not yield an improvement, during our exeriment the model achieved an accuracy of 59.33% on previously unseen test data after training.

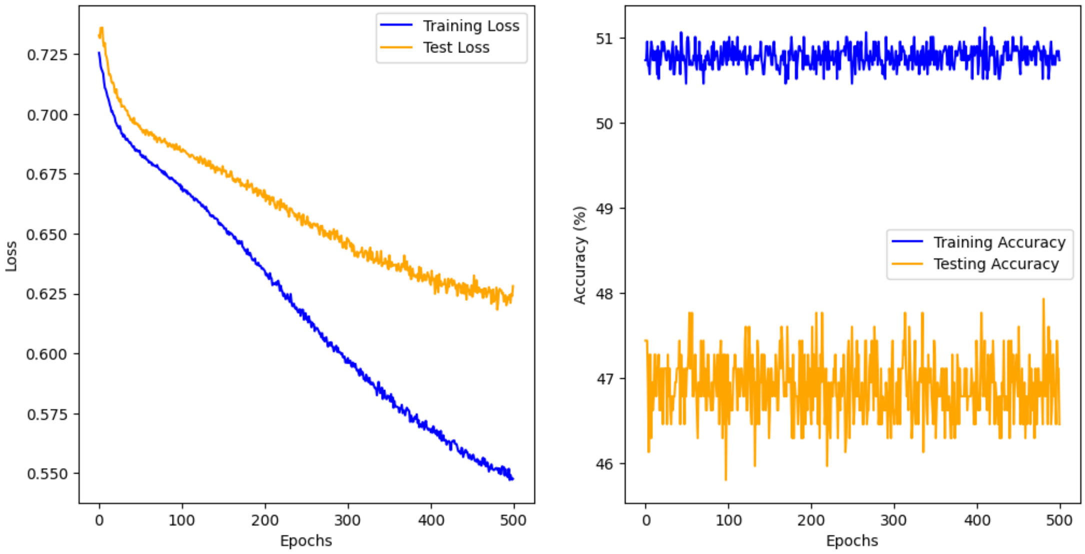


ResNet50 classifier

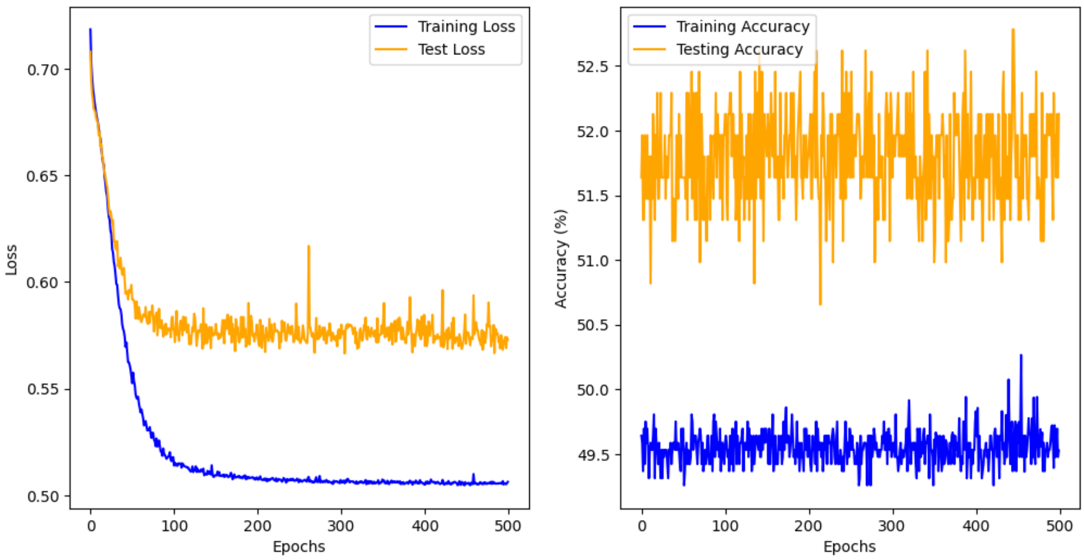
We trained a ResNet50 classifier using this pytorch code:

```
class MyResNet50(nn.Module):
def __init__(self):
    super(MyResNet50, self).__init__()
    self.resnet = models.resnet50(pretrained=True)
    num_features = self.resnet.fc.in_features
    self.resnet.fc = nn.Linear(num_features, 1)

def forward(self, x):
    x = self.resnet(x)
    return torch.sigmoid(x)
```

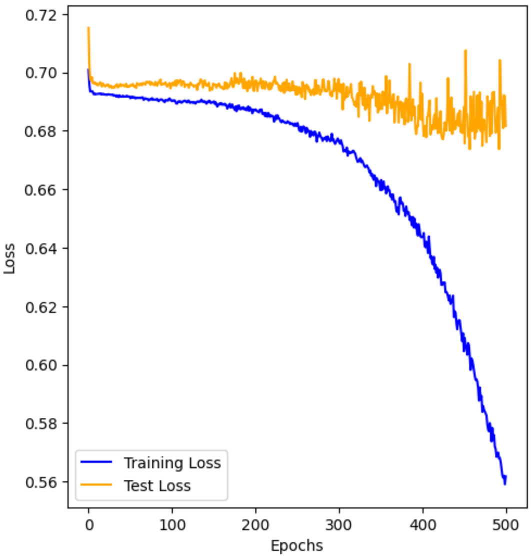
Training the network for 500 epochs with a learning rate of 0.0001 resulted in an accuracy of 46.46% on previously unseen test data.

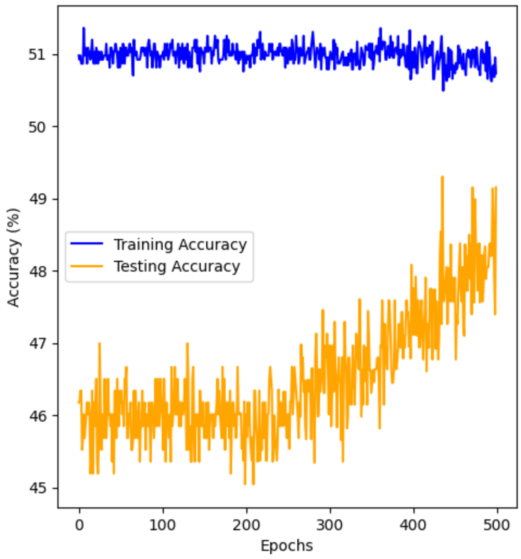


Training for 500 epochs with a learning rate of 0.001 resulted in an accuracy of 52.13% on previously unseen test data.

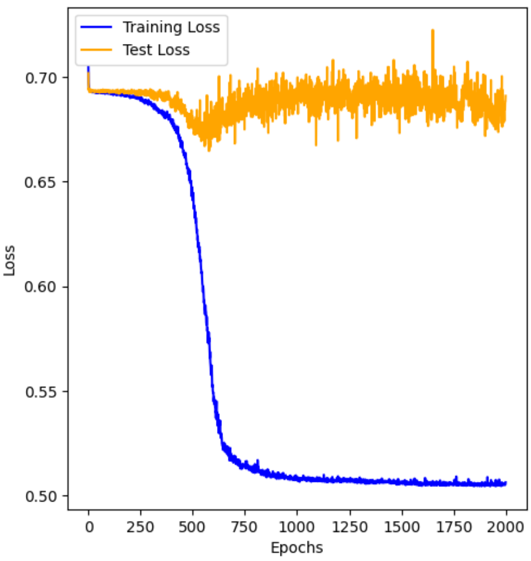


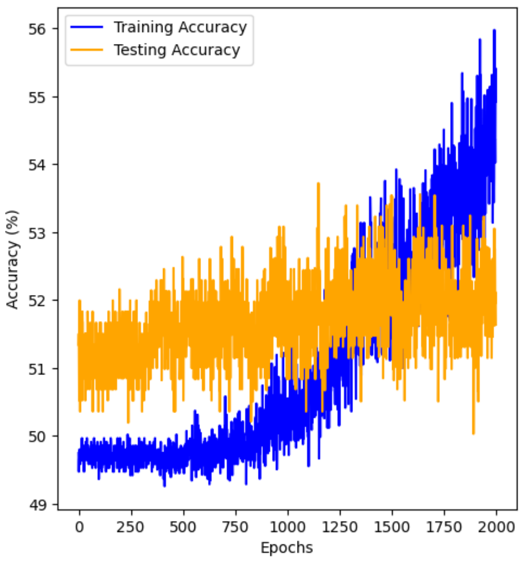
Training the model from scratch by setting pretrained=False, with a learning rate of 0.001 for 500 epochs resulted in an accuracy of 49.15% on previously unseen test data





Training the model from scratch, with a learning rate of 0.001 for 2000 epochs resulted in an accuracy of 51.64% on previously unseen test data





Implementing the pipeline

For this work, we reimplemented the pipeline proposed by Korpitsch (2023), and adapted the code where needed.

Sketchdetection

To repeat the steps introduced by Korpitsch (2023) we collected the dataset provided by Sarvadevabhatla et al. (2017). In Smith et al. (2023a), a Mask-R-CNN, as described by He et al. (2018), is used to detect the bounding boxes of figures drawn by children.

3 Results / Ergebnisse

Presenting found literature in a useful way

3.1 First Section

Ich bin Text, Text, Text[1]

3.1.1 First Subsection

[1] http://mfg.fhstp.ac.at

4 Discussion / Diskussion

Comparison of presented technologies/methods/projects

Kritische Diskussion / Vergleich der Ansätze

Welche Methoden werden zumeist genutzt, warum?

Überblick / Zusammenfassung der gefundenen Literatur in einer sinnvollen Kategorisierung / Charakterisierung

5 Conclusion / Fazit

Was kann man daraus lernen?

Was fehlt?

Ideen für zukünftige Forschung

Bibliography

Eitz, M., Hays, J., & Alexa, M. (2012). How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, *31*(4), 44:1–44:10.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask r-cnn.

Huang, Z., Han, R., Huang, J., Yin, H., Qin, Z., & Wang, Z. (2022). Automatically generate rigged character from single image. *ACM Multimedia Asia*. https://doi.org/10.1145/3469877.3490565

Korpitsch, T. (2023). *Semantic-aware animation of hand-drawn characters* [Master's thesis, Research Unit of Computer Graphics, Institute of Visual Computing and Human-Centered Technology, Faculty of Informatics, TU Wien]. https://www.cg.tuwien.ac.at/research/publications/2023/korpitsch-2023-sao/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Sangkloy, P., Burnell, N., Ham, C., & Hays, J. (2016). The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Trans. Graph.*, *35*(4). https://doi.org/10.1145/2897824.2925954

Sarvadevabhatla, R. K., Dwivedi, I., Biswas, A., Manocha, S., & Babu, R. V. (2017). Sketchparse : Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks.

Smith, H. J., Zheng, Q., Li, Y., Jain, S., & Hodgins, J. K. (2023a). A method for animating children's drawings of the human figure.

Smith, H. J., Zheng, Q., Li, Y., Jain, S., & Hodgins, J. K. (2023b). A method for animating children's drawings of the human figure. *ACM Trans. Graph.*, *42*(3). https://doi.org/10.1145/3592788

List of Figures

List of Tables

Code Listings

Appendices

A Appendix

LoHrem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.LoHrem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. LoHrem

B Appendix

LoHrem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.LoHrem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. LoHrem