DISNET: DISNEY CARTOON RECOGNITION AND CLASSIFICATION NETWORK

Daniel Chan
Student# 1007110127
danielsebastian.chan@mail.utoronto.ca

Tatiana Leon Prado
Student# 1006852835
tatiana.leonprado@mail.utoronto.ca

Nishith Gandhi
Student# 1004742200
nish.gandhi@mail.utoronto.ca

Muaaz Nisar
Student# 1007936450
muaaz.nisar@mail.utoronto.ca

ABSTRACT

This document is the final report for the APS360 project: DisNet. The report will highlight our motivations behind our project. It will also include the current state of the art image classification models and explain the architecture used. A deep analysis on results and challenges will conclude the report.
—-Total Pages: 9

1 INTRODUCTION

Have you ever created something but someone else has taken credit for it? Or seen the image of a famous character on YouTube, but the creator hasn't given the proper acknowledgement to the company who owns the character?

Copyright infringement has been one of the most serious problems that have hampered the development of the entertainment industry. Unfortunately, due to the limitations of existing image classification and search services, these infringements have not been properly identified, and the number of infringements has increased over the years (1). To tackle this issue, we propose a photo classification AI with a scope on Disney cartoon classics.

Our project topic stemmed from a collective group interest in cartoons and a desire to find a way to address copyright related to the misuse of these cartoons in the media. The reason we chose Disney cartoons is that not only do these hold fond memories of our childhood, but they have more authentic data available. Because of this, many of these characters have become mainstream, and therefore, animation companies need a way to protect their cartoon characters from copyright violations.

A machine learning algorithm that is useful for classifying images is Convolutional neural networks, and we will use pre-trained weights as tools to achieve our goal.

2 ILLUSTRATION, BACKGROUND & RELATED WORK

To give a better sense of what has already been done in this area, we will present five related works in the field. First, we have **Content Id**, YouTube's automated content identification system. Many copyright owners use this to identify and manage their copyright-protected content on YouTube easily. Videos uploaded to YouTube are scanned against a database of audio and visual content submitted to YouTube by copyright owners. Then, if Content ID finds a match, it applies a "Content ID claim" to the matching video, which can result in blocking the video or tracking the video's viewership statistics. (2)

There are several well-known datasets **MNIST** (3) is one of the most widely-used datasets in the simple image recognition field using machine learning (ML). It is a handwritten digits database

consisting of 60000 training samples and 10000 test samples. In addition, there are small datasets like **Caltech256** (4): a challenging set of 256 object categories containing 30607 images, **PASCAL** (5): a developing image database since 2015 that contains image information for Classification. Nevertheless, all the datasets mentioned above are about collections of real-world-style images. Therefore, it is pioneering to construct a cartoon-style image database. In fact, there is a "high demand for high quality, representative, large-scale datasets for cartoons."(6)

Studies by Beihang University explain how they approached this by building their own model: **ICartoon** . In the development of ICartoon, the dataset was created following these steps: hierarchical data collection, data filtering process, and Q/A manual annotation. According to the report, it consisted in first doing a face detection branch where they "manually label 60,000 images with 109,810 cartoon face bounding boxes... 0.80 was used as the train set and 0.20 was used as the test set" (6). They enhanced the detection of the model by mixing the dataset with real human faces. ICartoon "developed a Q/A system to annotate the identity information of cartoon faces manually."(6)

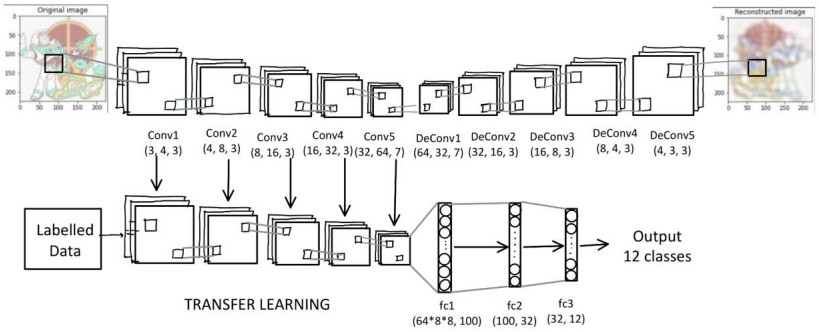

Figure 1.0: Illustration of the architecture of a Convolutional AutoEncoder that utilizes transfer learning. The model identifies and learns features in an image. This features are decoded in order to recreate the image and see how much the model has learnt.

3 Data Processing

No dataset for cartoon characters, expansive enough, was available publicly. To avoid starting from scratch, we picked a small Kaggle dataset as a head start, and expanded on it. We picked 12 characters total and downloaded 200 different pictures from the internet for each one.

A thorough cleaning process was carried out to standardise the data with a software called Canva. First, discarding low resolution examples. Secondly, centering the image to focus on the character's features (mainly the face and body). Thirdly, resizing the image to a 224 x 224 shape (Figure 2.0). Finally, transforming the image to a JPG. At the end, our first dataset had 2052 pictures total, and we decided to train our first Convolutional model with this.

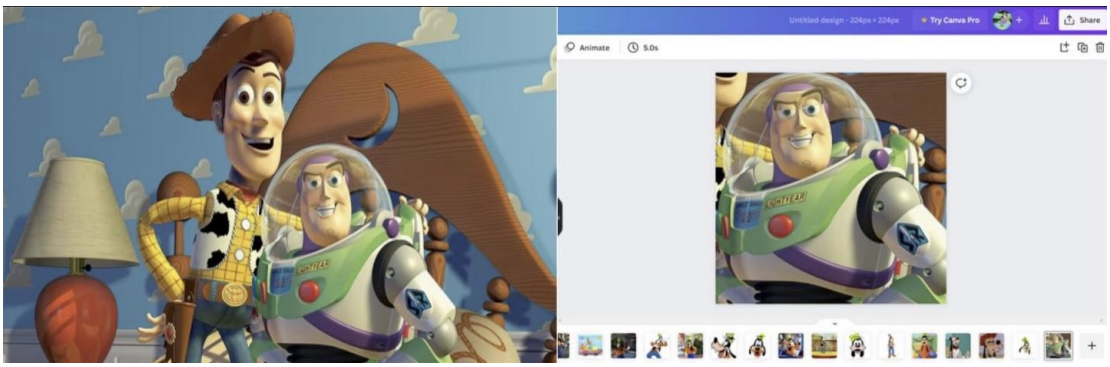

Figure 2.0: Original image compared to end result (7)

However, 2052 pictures was too little for the model to properly learn features and get high enough accuracy for unseen data. Several issues were encountered in the creation of a dataset. First and foremost, the lack of said dataset. But in the actual creation of the set, we found that downloading, resizing, and transforming the data in batches is particularly tedious and time consuming, especially when working with thousands of images and lack of software that does it automatically. To avoid an exhaustive process to find and clean new pictures, we took the route of data augmentation.

We began by applying 14 different filters, to create 14 new copies of every image, therefore expanding our dataset without the need to find new images and edit the size again. To mass edit the images we used PhotoScape X, a photo editing software that enables you to batch edit, fix and enhance photos. Here's an example on making a blurred copy:

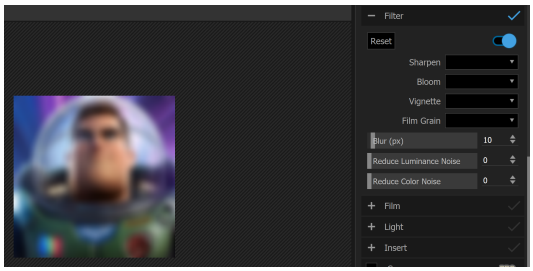

Figure 3.0: Blurred Image of Buzz (8)

At the end of the data augmentation phase, the database had almost 30k pictures total. We tried using this dataset on google colab, but it was so massive that we ran out of RAM when loading the data. Consequently, we reduced the dataset to 12k and trained the Convolutional model again and our second model which was an AutoEncoder. 70% of the pictures were used for training, 15% for validation, and 15% for testing.

Here is our final database: Final_Dataset.

4 Baseline Model

To improve the understanding and performance of our project, we will compare it with a simpler model. Given ANNs ability to take in a lot of inputs, and process them to infer hidden as well as complex, non-linear relationships, ANNs are playing a significant role in image and character recognition. (9) At the end of Lab 2 we used an ANN model which classified an image into one of two classes: "cat" or "dog." Our baseline model took inspiration from this, but instead of classifying them into two classes we will classify into 12.

For our baseline model, the number of inputs is 12 as well as the number of outputs. After splitting the data into training, testing and validation, in order to visualise how this model was working we found both the training loss and training curve for validation accuracy vs the number of iterations:

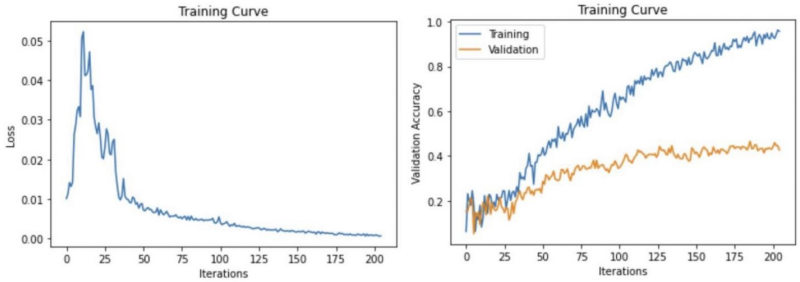

Figure 5.0: Baseline Model Training Curve

The above image shows the best result of training the ANN network. Our model has a training accuracy of 0.75 and a validation accuracy of 0.42. We notice that the model's performance improves when increasing the batch size and epochs to the optimal. Finding the right batch size can be challenging and requires a lot of trials. In general, the batch size can be increased if the dataset increases, to have a lower training time. Overall, even though using ANNs for classifying an image into 2 classes, for our model it doesn't generate desirable results.

Discussion: ANNs are used for datasets that are smaller. In general, CNNs tend to be more powerful and a more accurate way of solving classification problems. In general, CNN tends to be a more powerful and accurate way of solving classification problems. ANNs are still dominant for problems where datasets are limited, and image inputs are not necessary.

Here is our Baseline Model: Baseline Model.

5 ARCHITECTURE

Model #1

The architecture of the Convolutional Network incorporated the use of transfer learning from AlexNet to divide the model into two components (Figure 4.0).

First AlexNet.features is the neural network component that is used to compute convolutional features. The team then builds a trainable classifier that takes these features as input. The neural network component of AlexNet has 5 convolutional layers with 5x5 kernels and 3 max pooling layers. The first 2 convolutional layers are each followed with a max pooling layer. The other 3 convolutional layers are attached together, and a max pooling layer is attached after the last convolutional layer.

The classifier the team built contains 3 fully connected layers. The output layer has 12 neurons , as the total number of classes. The first hidden layer has 100 neurons. The second hidden layer has 32. The activation function of ReLU is used to ensure non-linearity. After tuning the hyperparameters, the team found that the optimal training parameters were a batch size of 250, learning rate of 0.01 and 21 epochs. The team achieved an accuracy of 0.8469 on the validation dataset and 0.9994 on the training dataset.

The low validation accuracy made us think that transfer learning from AlexNet was insufficient for our model and that we would need to look at other routes. That led us to create an AutoEncoder and using VGG and InceptionNet as transfer learning alternatives. None of which gave better accuracy. After finalizing and expanding the dataset to 12k images, we reran this first model, and to our surprise, it achieved an accuracy of 99% in both validation and training.

Here is the Convolutional Model: AlexNet.

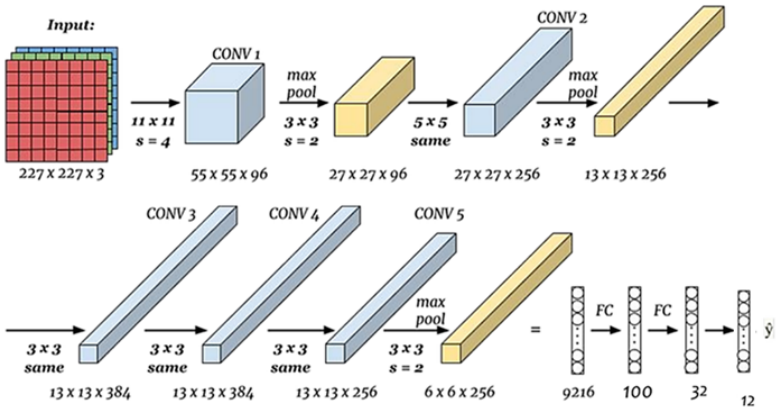

Figure 4.0: Diagram of Model

Model #2

The standard architecture of the AutoEncoder consisted of two components; the encoder and the decoder. The encoder consisted of 3 convolutional layers, 2 of which had 3x3 kernels while one with 7x7 kernels, creating a low dimensional embedding space. The decoder being a mirrored image of the encoder used transposed convolution reconstructs the input image. All layers had a stride of 2 and padding of 1.

Using this standard architecture the team performed a hyperparameter search, including a search for the optimal number of layers. Therefore, the final model consisted of 5 convolutional layers for the encoder part and 5 transposed convolutional layers for the decoder part. The last layer of encoder and first layer of decoder had kernels of 7x7 while all other layers had 3x3. Only the first layer of the encoder increased the number of channels from 3 to 4 while all other layers double the amount of channels, and vice versa for the decoder. Strides and padding were kept constant. A summary of the hyperparameter search is provided here. , while an illustration of the final model is shown in figure 1. The subsequent classifier that was to receive 8x8x64 tensors from the encoder had three fully connected layers with ReLU activation, finally converging to 12 classes. The output was then passed through torch.sigmoid() function to obtain probabilities.

Training the model for 30 epochs gave a satisfactory validation loss of 0.0495, however, upon plotting the reconstruction images it was seen that the network had hardly learnt any visual features. Academic literature showed that training an AutoEncoder for a high number of epochs is ideal. Therefore, using accelerated GPUs, the network was trained for 1000 epochs, yielding much better reconstructions, yet still insufficient. Consequently, transferring the weights from the encoder to a convolution block and adding a classifier only yielded a validation accuracy of 46%.

Here is the AutoEncoder Model: AutoEncoder.

6 Primary Model

6.1 Quantitative Results

Originally, we used an AlexNet pre-trained model set which had 2 convolutional layers with 2 max pooling layers and 3 attached convolutional layers with 1 max pool. After training with about 2,000 images, we reached 84.7% validation accuracy.

Since we were not satisfied with the result, we decided to perform transfer learning using an AutoEncoder. For the AutoEncoder to learn valuable features and representations by reconstruction of original images, it needed a much larger dataset than 2,000 images, as mentioned earlier in data processing, since we could not find expansive databases for the cartoon characters. Instead, we augmented the images ourselves such as image rotations, blurring, and colour manipulation like grey scaling and brightness changes. At the end of image augmentation, we had over 30,000 images.

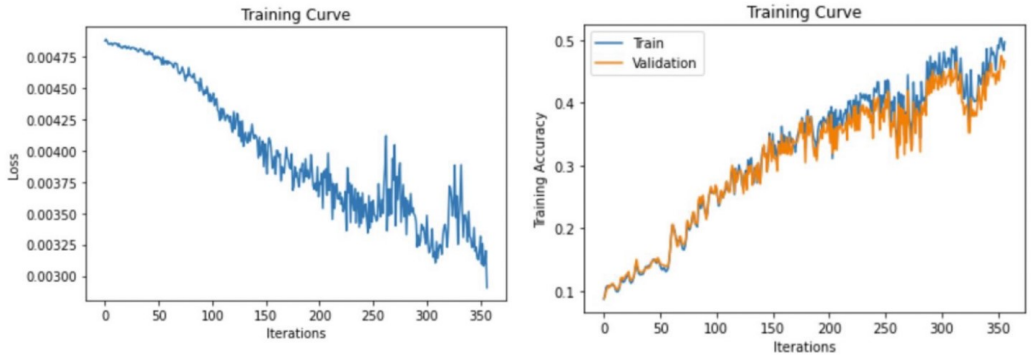

Figure 6.0: training curve of classifier with Autoencoder pretraining

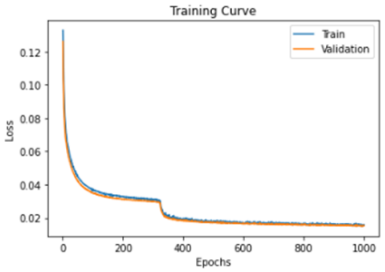

Figure 6.1: Training curve of Autoencoder

Unfortunately, due to limited RAM and computational resources, we had to reduce the size of the dataset down to 12,000 images over the 12 characters. When we tried to train the AutoEncoder, we realised that 31 epochs using the 12,000 images was not enough to train the AutoEncoder. Therefore, we trained the AutoEncoder for 1000 epochs, achieving a validation loss of 0.00974, yet due to poor reconstructions, the AutoEncoder pretrained weights were able to achieve a validation accuracy of only 46%.

Therefore, we tried a few different pre-trained networks such as ResNet and VGG which did not give good results similar to the AutoEncoder. Finally, we decided to fall back to AlexNet and train it with the 12,000 images compared to the 2,000 images from before, and do a more expansive hyperparameter search. This gave incredible results; Our final validation accuracy came out 97.2%. Tests with new and previously unseen images also gave similar promising results.

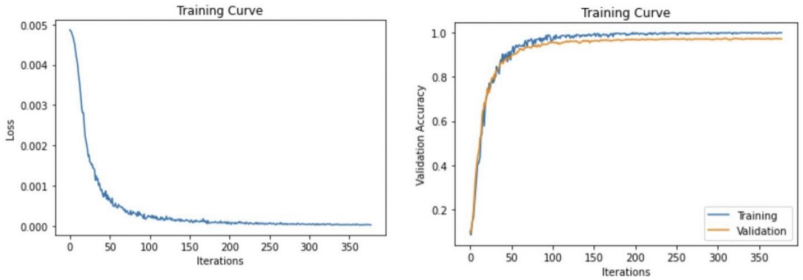

Figure 6.2: Training curve of AlexNet with 12,000 image dataset

6.2 Qualitative Results

After trying several architectures, we decided to go with AlexNet pre-training for our model. The model takes 224x224 images as input and maps them to 12 possible output classes (Disney characters). The ground truth label is also displayed for reference. We used the probability 50% as the cutoff for making a discrete prediction. Figure 7.1 shows a sample output of our model.

```
[18]
        print(characters)
        print(prob)
        print(f'output: {characters[pred[0]]}, label: {characters[labels[0]]}\n')
        if cntr == 10:
          break

['Buzz', 'Donald', 'Goofy', 'Mickey', 'Minnie', 'Pluto', 'Pumba', 'Simba', 'Tigger', 'Timon', 'Wall-e', 'Woody']
tensor([[0.5009, 0.7310, 0.5002, 0.5316, 0.5001, 0.5025, 0.5000, 0.5028, 0.5000,
         0.5000, 0.5000, 0.5000]], device='cuda:0', grad_fn=<SigmoidBackward0>)
output: Donald, label: Donald

['Buzz', 'Donald', 'Goofy', 'Mickey', 'Minnie', 'Pluto', 'Pumba', 'Simba', 'Tigger', 'Timon', 'Wall-e', 'Woody']
tensor([[0.7306, 0.5355, 0.5055, 0.5000, 0.5001, 0.5115, 0.5000, 0.5049, 0.5000,
         0.5099, 0.5587, 0.5233]], device='cuda:0', grad_fn=<SigmoidBackward0>)
output: Buzz, label: Buzz

['Buzz', 'Donald', 'Goofy', 'Mickey', 'Minnie', 'Pluto', 'Pumba', 'Simba', 'Tigger', 'Timon', 'Wall-e', 'Woody']
tensor([[0.5001, 0.5005, 0.5380, 0.5000, 0.5001, 0.5002, 0.7307, 0.5081, 0.5015,
         0.5246, 0.5000, 0.5004]], device='cuda:0', grad_fn=<SigmoidBackward0>)
output: Pumba, label: Pumba
```

Figure 7.0: Snapshot of sample output of the model

From extensive testing, we see that our model identifies the characters with an accuracy of 97.2%. The model performs well mainly through extensive hyperparameter search, and due to the sheer size and variety of the dataset. Finally, one area where our model lacks a bit is distinguishing between "Mickey Mouse" and "Minnie Mouse". This may be because the two characters only have subtle differences in their edges and textures. Additionally, some image augmentations, such as blurring and colour distortion could have further obscured their textures and edges. Before we finalised using AlexNet, we also used an architecture with an AutoEncoder. Unfortunately, we did not have enough RAM and computational power to execute it well. We verified this by testing the encoder and decoder parts by themselves. A sample result can be seen in Figure 7.1. Even after 1000 epochs, the decoder could not fully re-create the original images, and only reached a final accuracy of 46%.

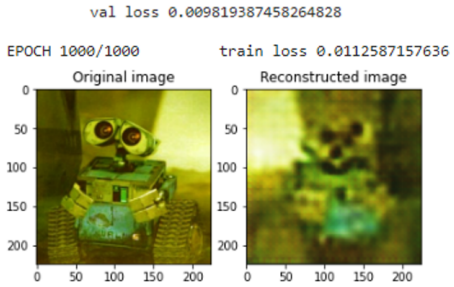

Figure 7.1: AutoEncoder reconstructing an image of Wall-e

7 Evaluate model on new data

To verify if the results were a good representation of learning and not overfitting, we had set aside a portion of our database that the model had never seen before and used that for the tests.

From the finalised 12k image dataset we separated 15% of the dataset for testing. This totalled to 1,800 images of unseen data for a testing phase. For our model, it was simple, if it identified all images correctly, it meant that we had successfully tackled the classification problem that we were setting.

We used the AlexNet Convolutional model and trained it with the 12k images, achieving high validation and training accuracy. Once the model was fully trained, it achieved an accuracy of 88.3% on the unseen testing images. This can be seen in this document: AlexNet.

To have visual evidence, we also ran some of this unseen data through the AutoEncoder to see how the decoded image compared to the original one. Here is a test example of Mickey:

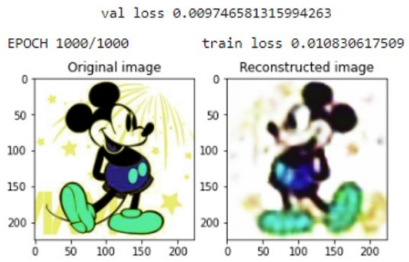

Figure 8.0: A reconstructed image of Mickey

While the decoded image is distinguishable, it is evident that even at 1000 epochs, it isn't close to being perfect. While it did provide more training examples and help improve the models for recognition and classification. Some of the filters we applied such as blurriness or saturation, might have led the model to learn wrong features. Some of the pictures from the data augmentation phase are beyond recognizable from the original character and this led the AutoEncoder output to have a distorted image. To improve on this, we should use said filters with reduced intensity when augmenting the data.

8 Ethical Considerations

There are a few ethical considerations that we need to account for with our model. In a case where Disney uses our model to identify copyright infractions, our model would replace many employees who would have been originally responsible for enforcing copyrights. Most of these employees would lose their jobs in that case.

Secondly, despite having an accuracy of 97.2%, since our model goes over thousands of images over the internet, it would give several false positives in the process. These false positives could lead to counter lawsuits unless verified by humans, as it would incorrectly cause someone to be copyright struck.

Finally, since our model can only process 224x224 size images, it would be able to accept a very limited number of pictures from the internet. However, this can easily be fixed in future versions by adding a section at the beginning of the model that transforms the input to acceptable parameters.

9 Difficulties and Challenges

Throughout the project the team encountered several challenges from data collection, augmentation, applying transfer learning, to performing hyperparameter search which required knowledge beyond what was achieved in the labs.

The first challenge the team faced was not having a publicly available dataset, for which the team had to handpick 2000 images and then following internet guides, carry out data augmentation. However, the extent of data augmentation and some specific filters proved counterproductive for the network.

Following this, the convolutional Autoencoder was implemented. This required a more extensive understanding of PyTorch to perform the transfer learning. Most internet resources guided how to implement the AutoEncoder on common datasets like MNIST however many variations were made to make the model function on our dataset. Furthermore, transfer learning was also more popularly discussed with torchvision models like AlexNet or VGG, or using libraries like Keras which was not the scope of our project. Therefore, we had to learn more extensive techniques to transfer the AutoEncoder weights to the classifier.

10 Discussion and Concluding Thoughts

In a nutshell, the quantitative results we achieved from our exploration, specifically the training curves, are evidence that the coding, model architecture, and hyperparameter search were successful. However, limitations in computational resources, time constraints, and method of data augmentation limited the AutoEncoder from being our final model. In the end, AlexNet proved to be an amazing model for image classification and even though our cartoon images were highly distinct from real-life images, AlexNet pretrained weights were able to achieve astonishing accuracy, using the advantage of an abundance of training data. We are also able to conclude that transfer learning, in general, was a good approach for our model since we already had limited computing and pre-trained weights require less time to fine-tune.

To improve on our existing model, more specifically to achieve similar results with the AutoEncoder as AlexNet, data augmentation could be performed with filters that have less intensity. That way the new images would be different from original ones, but not to the extent that they get indistinguishable amongst several classes. Moreover, the model can be experimented with fewer number of classes to see if the model better generalizes a smaller number of characters. Characters with similar shape and structure (Mickey and Minnie) can also be eliminated to make the classification task simpler. Finally, in a future investigation, we would like to reduce the batch size and increase the number of checkpoints we do throughout the testing phase as one of the problems we struggled with was very long run times. In conclusion, the image classification network we propose is a promising solution to companies looking to protect their character's use in the media.

REFERENCES

[1] Disney Trademark Infringement: UpCounsel 2022," UpCounsel. [Online]. Available: https://www.upcounsel.com/disney-trademark-infringement.

[2] How Content ID works - YouTube Help," Google. [Online]. Available: https://support.google.com/youtube/answer/2797370?hl=en: :text=Some copyright owners use Content,to YouTube by copyright owners.

[3] THE MNIST DATABASE," MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[4] Griffin, Gregory and Holub, Alex and Perona, Pietro (2007) Caltech-256 Object Category Dataset. California Institute of Technology . https://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001]

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman,"The Pascal Visual Object Classes (VOC) Challenge," International Journal of Computer Vision, vol. 88, no. 2. Springer Science and Business Media LLC, pp. 303–338, Sep. 09, 2009. doi:10.1007/s11263-009-0275-4.

[6] Y. Zheng, Y. Zhao, M. Ren, H. Yan, X. Lu, J. Liu, and J. Li, "Cartoon Face Recognition: A Benchmark Dataset," Proceedings of the 28th ACM International Conference on Multimedia, pp. 2264–2272, Oct. 2020.[Online]. Available: ACM Digital Library, https://dl.acm.org/doi/abs/10.1145/3394171.3413726 [Accessed: 05-Jun-2022].

[7] Toy story 4 finally has a release date," The Independent, 10-Apr-2018. [Online]. Available: https://www.independent.co.uk/arts-entertainment/films/news/toy-story-4-release-date-pixar-disney-cast-plot-tom-hanks-latest-a8297326.html. [Accessed: 12-Jul-2022].

[8] Lightyear," Disney Movies, 17-Jun-2022. [Online]. Available: https://movies.disney.com/lightyear.

[9] J. Mahanta, "Introduction to Neural Networks, Advantages and Applications," Medium, 12-Jul-2017.[Online]. Available: https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207.