IMAGE CLASSIFICATION OF THE ILLUSTRATION OF EVERYDAY OBJECTS

Jeffery Coulter, Aditya Joseph James, Manasi Swaminathan

Luddy School of Informatics, Computing, and Engineering
Indiana University
Bloomington, Indiana

1. INTRODUCTION

Image classification is the branch of machine learning and deep learning that performs the task of classifying images into distinct classes. The challenge is to accurately predict the class the input image belongs to. Some of the challenges include noise due to the background of the subject, image quality, different objects photographed with similar background, to name a few.

Image classification has a wide range of applications from categorizing pictures, medical image processing and is a subset of computer vision. Performing image classification with machine learning and deep learning algorithms needs a large dataset for the model to work accurately. Machine learning and Deep learning techniques have become very popular in performing this task.

Neural networks are designed to identify patterns in images just like the human brain. The main advantage of neural networks is learning features directly from the training data using mapping functions. Several models have been introduced to perform classification and the objective of this project is to analyze them.

Our study looks to carry out image classification on less commonly used data such as clipart, painting and quickdraw images to name a few. We found a collection of images which has been broadly categorized based on the type of image, namely painting, clipart, quickdraw,infograph, real photograph, sketch [1]. In each of these categories we have 345 image classes (e.g. airplane, clock, ball, bird, etc.). We use this datasets to answer the research questions that are elaborated below.

Our project will aim to carry out the following objectives. 1) We aim to select three broad classes among ourselves and then carry out image classification across 345 classes subject to enough samples being available. 2) We also would like to understand if models differentiate between color and black and white images, for this we plan to implement a clustering algorithm. 3) Given that our datasets would be vastly different it would also be interesting to carry out a comparison study to see which technique was able to solve the problems better.

2. RELATED WORK

Image classification has been around for a few decades and has largely focused on real world images such as photographs of people, places, animals etc. Even the most commonly available datasets such as MNIST [2] are based on handwritten digits. There are very few, if any, widely available dataset on other forms of images such as clipart, paintings, freehand drawing, info-graphics etc [3].

However, Yanqing Zhou et al. have tried to solve the problem of cartoon classification which is very close to the problem that we are trying to solve [3]. In their paper they faced challenges of not having enough cartoon style data for their analysis and turned to several other approaches to generate more data for their analysis. They were able to implement a DNN based solution to classify cartoon images.

Another study that worked on the problem of classifying art was one carried out by Young-Min Kim in which the author used a CNN to classify comic art [4]. In the article the author took 8 volumes of the comic 109 manga, this is because each volume was illustrated by a different artist hence the illustration would be different in each of them. We thought that a CNN model would also be sufficient to classify clipart given that a manga comic has a lot more features as compared to a clipart image.

3. CHALLENGES

This project will be relatively challenging since only one of us has a background in image processing. However, most of the differences between image analysis and other machine learning tasks is related to data pre-processing, for example, image scaling and grayscale conversion. Given that there are 345 classes a random guess would predict with an accuracy of only 0.3%. This will make the classification even more challenging.

We also find that there is uneven distribution of the images across different classes which would result in class imbalances. In some cases we also see that there are multiple objects in the same image such as hand and pencil being held in the hand. Since both these are also separate classes it we would have to deal with a lot of miss-classification cases.

4. METHODS

For the first research question of image classification, we implemented three architectures for clipart dataset classification based on previous work by Busfield and Yagmur [5, 6]. We also wanted to compare the performance of the state of the art algorithms such as GoogLeNet to carry out classification for both broad categories (clipart, painting, real, etc.) and also for classification of classes within a category [7]. This would help us understand and compare a manually tuned architecture to a state of the art architecture such as GoogLeNet.

Additionally, since each class of images will include varying quality (drawings, photographs, clipart, etc.), we may find it beneficial to try eroding the images before feeding them into the CNN. Such preprocessing can be used for edge-detection and will thus make differences in quality (photograph vs clipart) less significant and the network will be able to focus primarily on geometric objects [8].

4.1. Data Exploration

The exploration of the clipart data showed that we might have an issue of samples across the various classes. This leads us to believe that we might have some issues during the modeling process. We also believe that given that a random guess for predicting a class would be 0.3% for an image a lower accuracy might still be acceptable. Below is the box plot for the clipart data. We see that the median number of samples across the 345 classes is close to 100.

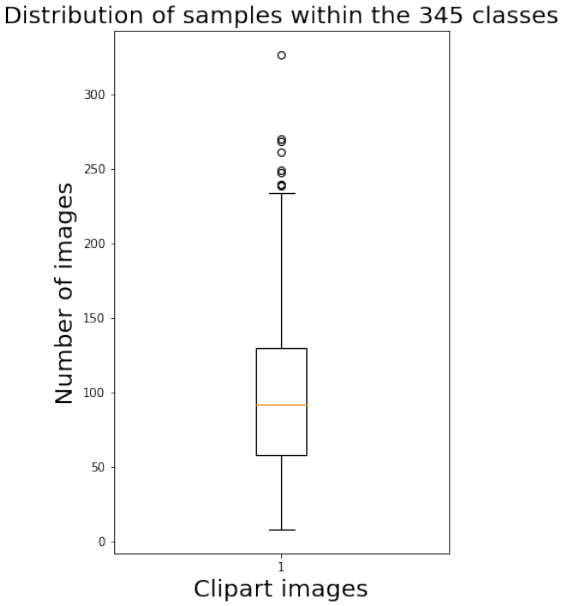


Fig. 1: Example of image distribution between classes for the clipart category.

We referred to an article which tries to solve a real world classification problem i.e. actual image data which is stored on the local drive [5]. The analysis by Busfield aims to carry out a binary classification of a machine part being defective and non defective. Thus, his approach did not have the issue of low sample size. Given a binary classification problem the architecture was also much more simple as compared to a more complex problem such as a multi-class classification problem.

The approach followed by their solution required the training and test data to be in separate folders. The data we got had the training and test data in a single folder, but a text file was provided that tagged which data was supposed to be used for training and which data was supposed to be used for test. We had to design a script that would create the directory structure so as to implement their approach.Below is a graphical representation of the transformation.

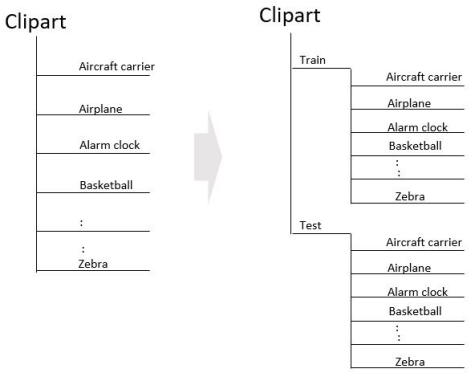


Fig. 2: Example of folder structure.

4.2. Pre-processing

As discussed above in Section 4, something we might like to try is to use an edge-detection algorithm to reduce some of the differences between the images. Currently, we've implemented a Sobel filter.

Fig. 3: Application of sobel filter to different images of asparagus.

In the image above, there are three pictures of asparagus, a drawing, a real image, and a black and white sketch along with the corresponding filtered versions. By applying the filter and reducing the images to basic geometric objects, we hoped to allow the our models to more easily recognize the objects across categorical groups. However, this proved to be computationally inefficient. After converting 10 images to their edges and extrapolating the time to determine how long it would take to convert a whole training set, we found that it would take a couple of days.

4.3. Image Orientation

From our initial observation we noticed that the images do not have the same orientation. This would cause some of the images to be miss-classified. In the example below there are three images. We observe two of the images the orientation seems to be horizontally mirrored while the third image has two swords and the orientation is also reversed.

Fig. 4: Varying image orientations in the clipart category.

To fix this we would need to ensure that the rotated/flipped images are also considered. We would need to explore if tensorflow can handle this scenario.

4.4. Class Imbalance

As noted earlier we see that not all classes have the same number of images. This would result in some classes not having enough samples to learn from. To handle this we might need to apply some sort of undersampling or oversampling or augmentation techniques [9].

5. CLASSIFICATION ACROSS CLASSES

We began with looking at only the clipart class. We made this selection because clipart seemed to have a fair balance of detail and simplicity compared to the other categories. For example, we felt that real images and infographs might contain too much detail for some of our simpler architectures, while sketches and quickdrawings might have too little.

Within the clipart category, we saw that there was an issue regarding the number of samples in each class. We had seen that there was a median of only 100 samples per class. This resulted in us not being able to have enough data to train the model. To handle this we took a call to select only those classes that had more than 200 samples. On applying this filtering criteria we shortlisted 17 classes from the original 345 classes.

For these selected classes, we pre-processed them by converting them to black and white and resizing them to 64x64. We used the same preprocessing and data-preparation as carried out by Busfield [5] as it had a ready made skeleton of data extraction and loading. Once we did this we proceeded to try out various architectures for training the model. To train the model we used the Carbonate environment available to us from IU. Our model uses CNN implementations in both Tensorflow and PyTorch. We decided to try this as we had issues with performance when executing the model using PySpark.

5.1. Architecture and results

We implemented four architectures in total, beginning with a relatively simple deep CNN and eventually working up to GoogLeNet [7]. As one would expect, the more trivial models performed worse, however, even GoogLeNet had some difficulties with certain classes, such as infograph images and sketches. We suspect this is likely due to the abundance of information in infographs and the lack of information in sketches.

5.1.1. Architecture 1

For our evaluation, we tried a few architectures which are based on the architectures by Busfield and Yagmur [5, 6]. We added modifications to these architectures for the various executions. We chose to augment the images horizontally as that was most common variation that we had seen in the training data.

We found that the architecture above was able to achieve an accuracy of 37.1%, this was much better than the initial model result of 4.3%. We believed that this could be improved further and tried a couple of more architectures.

5.1.2. Architecture 2

To further improve the model we decided to have kernel regularization in each convolutional layer. Apart from this we added batch normalization, max pooling and a dropout to the convolutional layers before connecting to a fully connected layer.

We found that this method performed marginally worse than the previous architecture even though the loss curves seem to suggest otherwise. This was because the initial loss was very high and dropped to a much lower value. The scale of the y axis gives a false idea that the losses are very low. The accuracy obtained using this technique was 35.8%.

```
---------------------- Model Summary ------------------------
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 64, 64, 1)         0

 random_flip_1 (RandomFlip)  (None, 64, 64, 1)         0

 conv2d_8 (Conv2D)           (None, 60, 60, 6)         156

 max_pooling2d_5 (MaxPooling  (None, 30, 30, 6)        0
 2D)

 dropout_8 (Dropout)         (None, 30, 30, 6)         0

 conv2d_9 (Conv2D)           (None, 26, 26, 16)        2416

 max_pooling2d_6 (MaxPooling  (None, 13, 13, 16)       0
 2D)

 dropout_9 (Dropout)         (None, 13, 13, 16)        0

 flatten_2 (Flatten)         (None, 2704)              0

 dense_6 (Dense)             (None, 256)               692480

 dropout_10 (Dropout)        (None, 256)               0

 dense_7 (Dense)             (None, 128)               32896

 dropout_11 (Dropout)        (None, 128)               0

 dense_8 (Dense)             (None, 17)                2193

=================================================================
Total params: 730,141
Trainable params: 730,141
Non-trainable params: 0
```

Fig. 5: Description of architecture 1.

Fig. 8: Accuracy and loss results for architecture 2.

5.1.3. Architecture 3

In this method we kept 16 convolutions in all the six layers but different kernel sizes of 2, 3 and 5 which were applied in gaps of two convolutional layers. We also had l2 kernel regularizers for each of six layers.

We found that this architecture performed the best across the three that we tested. it gave a training accuracy of 45%, validation accuracy of 36% and a testing accuracy of 42.4%. Given that a random guess would be 5.9% we feel that this is still an acceptable model for prediction given the low training data.

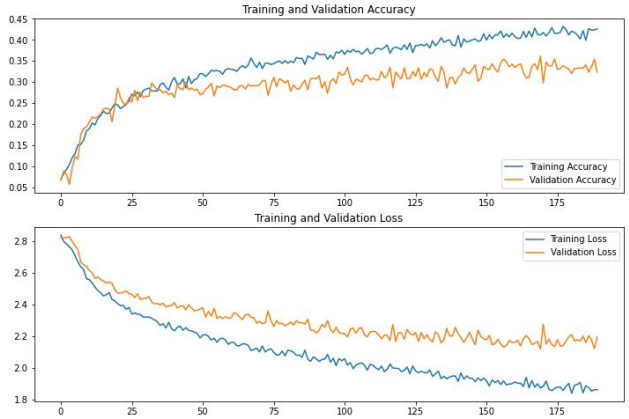


Fig. 6: Accuracy and loss results for architecture 1.

Fig. 9: Accuracy and loss results for architecture 3.

5.1.4. GoogLeNet

We next tried the GoogLeNet architecture, a 122 layer CNN which uses ”Inception” layers which act to widen the network by applying multiple convolutional layers at once. For the most part, we kept the architecture the same as described in the paper [7]. However, our implementation included batch-normalization at the end of each convolutional layer, which hadn’t been developed at the time of the original GoogLeNet’s creation.

Not only was the accuracy better than our previous architectures, we found that the epochs required to converge were significantly reduced from 150-160 epochs down to 20-50 epochs.

```
---------- Model Summary ----------
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 64, 64, 1)         0

 random_flip_1 (RandomFlip)  (None, 64, 64, 1)         0

 conv2d_10 (Conv2D)          (None, 62, 62, 16)        160

 batch_normalization_6 (Batc  (None, 62, 62, 16)       64
 hNormalization)

 conv2d_11 (Conv2D)          (None, 60, 60, 16)        2320

 batch_normalization_7 (Batc  (None, 60, 60, 16)       64
 hNormalization)

 max_pooling2d_7 (MaxPooling  (None, 30, 30, 16)       0
 2D)

 dropout_12 (Dropout)        (None, 30, 30, 16)        0

 conv2d_12 (Conv2D)          (None, 28, 28, 32)        4640

 batch_normalization_8 (Batc  (None, 28, 28, 32)       128
 hNormalization)

 conv2d_13 (Conv2D)          (None, 26, 26, 32)        9248

 batch_normalization_9 (Batc  (None, 26, 26, 32)       128
 hNormalization)

 max_pooling2d_8 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 dropout_13 (Dropout)        (None, 13, 13, 32)        0

 conv2d_14 (Conv2D)          (None, 11, 11, 64)        18496

 batch_normalization_10 (Bat  (None, 11, 11, 64)       256
 chNormalization)

 conv2d_15 (Conv2D)          (None, 9, 9, 64)          36928

 batch_normalization_11 (Bat  (None, 9, 9, 64)         256
 chNormalization)

 max_pooling2d_9 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_14 (Dropout)        (None, 4, 4, 64)          0

 flatten_3 (Flatten)         (None, 1024)              0

 dense_9 (Dense)             (None, 256)               262400

 dropout_15 (Dropout)        (None, 256)               0

 dense_10 (Dense)            (None, 128)               32896

 dense_11 (Dense)            (None, 17)                2193

=================================================================
Total params: 370,177
Trainable params: 369,729
Non-trainable params: 448
```

Fig. 7: Description of architecture 2.

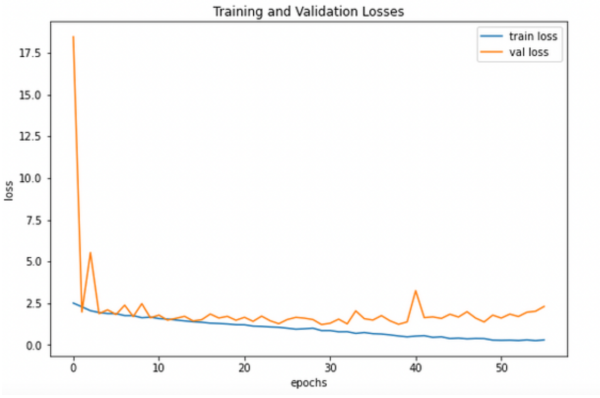


Fig. 11

Fig. 12: Training and validation loss for clipart classification using GoogLeNet.

```
---------- Model Summary ----------
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 64, 64, 1)         0

 random_flip_1 (RandomFlip)  (None, 64, 64, 1)         0

 conv2d_16 (Conv2D)          (None, 63, 63, 16)        80

 batch_normalization_12 (Bat  (None, 63, 63, 16)       64
 chNormalization)

 conv2d_17 (Conv2D)          (None, 62, 62, 16)        1040

 batch_normalization_13 (Bat  (None, 62, 62, 16)       64
 chNormalization)

 max_pooling2d_10 (MaxPoolin  (None, 31, 31, 16)       0
 g2D)

 dropout_16 (Dropout)        (None, 31, 31, 16)        0

 conv2d_18 (Conv2D)          (None, 29, 29, 16)        2320

 batch_normalization_14 (Bat  (None, 29, 29, 16)       64
 chNormalization)

 conv2d_19 (Conv2D)          (None, 27, 27, 16)        2320

 batch_normalization_15 (Bat  (None, 27, 27, 16)       64
 chNormalization)

 max_pooling2d_11 (MaxPoolin  (None, 13, 13, 16)       0
 g2D)

 dropout_17 (Dropout)        (None, 13, 13, 16)        0

 conv2d_20 (Conv2D)          (None, 9, 9, 16)          6416

 batch_normalization_16 (Bat  (None, 9, 9, 16)         64
 chNormalization)

 conv2d_21 (Conv2D)          (None, 5, 5, 16)          6416

 batch_normalization_17 (Bat  (None, 5, 5, 16)         64
 chNormalization)

 max_pooling2d_12 (MaxPoolin  (None, 2, 2, 16)         0
 g2D)

 dropout_18 (Dropout)        (None, 2, 2, 16)          0

 flatten_4 (Flatten)         (None, 64)                0

 dense_12 (Dense)            (None, 256)               16640

 dropout_19 (Dropout)        (None, 256)               0

 dense_13 (Dense)            (None, 128)               32896

 dense_14 (Dense)            (None, 17)                2193

=================================================================
Total params: 70,705
Trainable params: 70,513
Non-trainable params: 192
```

Fig. 10: Description of architecture 3.

After applying the model to clipart category, we moved on to the categories of reals, infographs, sketches, and quick drawings. We found the best results occurred for real images and, somewhat surprisingly, quick drawings, which reached accuracies on average of 68% and 76% respectively.

GoogLeNet was unable to produce successful results for infographs, likely for reasons described above. Similarly, classification of sketches only achieved mediocre results. On average, infograph classification attained an accuracy of 44.4% and sketch classification achieved an accuracy of 55.6%. Additionally, the sketch classification struggled with over-fitting despite implementing an early-stop mechanism with the validation set.

a)

b)

Fig. 13: Training and validation loss for a) real image classification and b) quick draw classification using GoogLeNet.

5.1.5. Transfer Learning ResNet-50

Finally we implemented ResNet-50 transfer learning on the data using the learned weights from imagenette [10]. We wanted to test the performance of this architecture to the ones that we had used. We referred to an implementation of transfer learning done by Isbarov and Potdar to carry out the transfer learning [11, 12]. We found that it performed much better than Architecture 1-3 with the testing accuracy of 78.53%. The loss and accuracy curves below show that the accuracy is far superior to what we had obtained.

5.2. Intermediate discussion

Based on our three custom architectures we decided that architecture 3 was the best for classifying the clipart data. We felt that the limitation of the data across classes resulted in the model not being able to learn effectively. We believe that given more data for each class we could have got a much higher accuracy.

Having said that we do think an accuracy of 42.4% in the test data is far better than a random guess of 5.9%. Hence, we

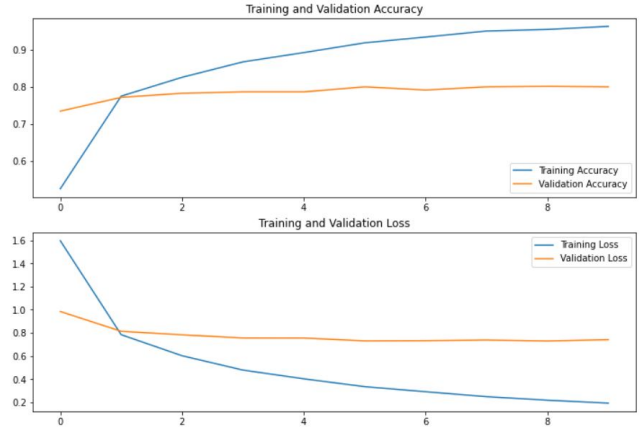


Fig. 14: Accuracy and loss results for ResNet-50

are happy with the results obtained from our analysis.

We also find that CNNs are very powerful in predicting the classes of images which are man made such as clipart. This is because each clipart would bear the finger print of the illustrator and hence would expect to be much different even within the same class. However, in our analysis we find that the model was still able to classify these samples well, even though the accuracy does not reach the performance of the state of the art approaches such as GoogLeNet.

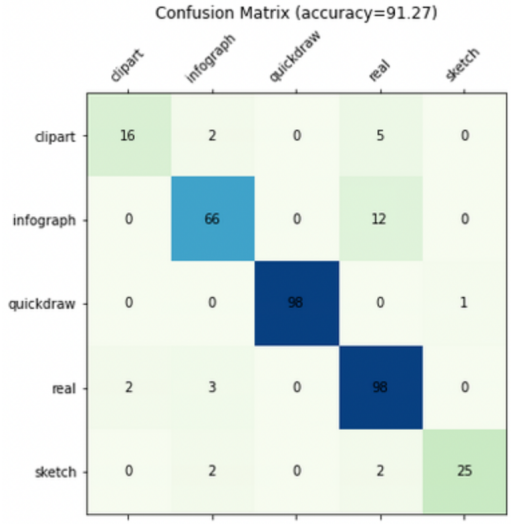
6. CLASSIFICATION ACROSS CATEGORIES

In addition to classifying images within a given category, we also wanted to classify images by their category, that is, to determine whether an image is real, a sketch, an infograph, etc. For this, we excluded the painting category since there were issues unzipping the download. The model then attempted to classify across five categories - real images, infographs, clipart, sketches, and quick drawings.

We found that this performed well with an average accuracy of 89.0%. This could be due to the lower number of possible categories compared to our earlier studies. Furthermore, the differences between these categories is arguably greater than the differences between images within a category, that is, a drawing of a bird and a drawing of a cat share more similarities than an arbitrary real image and a drawing. This is likely also contributed to the higher accuracy scores for the tests run for this problem.

The confusion matrix in the figure above shows that even with a relatively small data set of about 1500 images, the model can perform quite well. As expected, the most trouble came from distinguishing real images from infographs, but for the most the classifications are correct.

a)



b)

Fig. 15: The a) training and validation losses and b) confusion matrix for a single instance of running the GoogLeNet model on categorical classification.

7. CLUSTERING

In this research question of differentiating colored and black and white images we combine random images from the painting dataset, quickart dataset, and the black and white pictures from the clipart dataset to make one dataset set of 1839 images. The goal is to separate these images as black and white and color and to check how effective is the algorithm in doing so using the actual count of the black and white, and colored images.

The clustering technique works well for this as it puts each point in a group or cluster. The algorithm we use here is the kmeans clustering for color separation. Kmeans clustering is an unsupervised machine learning algorithm that identifies k centroids and assigns each of the data point to a cluster based on the distance to the centroid. Here each centroid represents the color palette of the image.

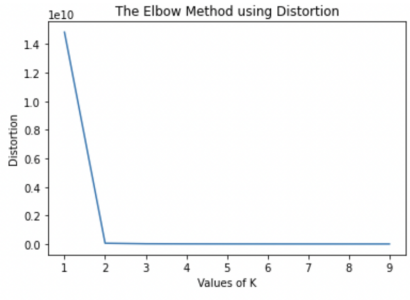


Fig. 16: The elbow curve for black and white image, optimal k value is 2

7.1. Image Processing

Given an image of size $M \times N$ there are $M \times N$ pixels each having three components red, green and blue known as RGB. These pixels are considered as the data points for clustering. To process the images we use the opencv library. The opencv library helps in reading each image and convert it into an image array. When the image is read it is in the BGR format (Blue, Green, Red) but in order to perform analysis the image is converted to RGB.

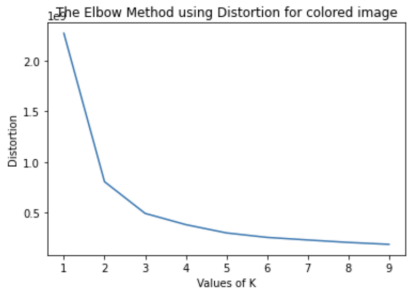


Fig. 17: The elbow curve for colored image, optimal k value is 3.

7.2. Choosing k using elbow method

The elbow method is a technique to determine the best value of k. It is performed by running the k means algorithm with various k values and calculating the distortions. The distortions for each k value is plotted to visualize the elbow. The elbow is the region that determines optimal k value. The intuition is that in a black and white image the dominant colors are black and white so the k value set to differentiate black and white and color images should be 2. This can be proved by performing the elbow method on a black and white image. The elbow method is also performed on a colored image to determine the ideal k value to extract the dominant colors in a colored image.

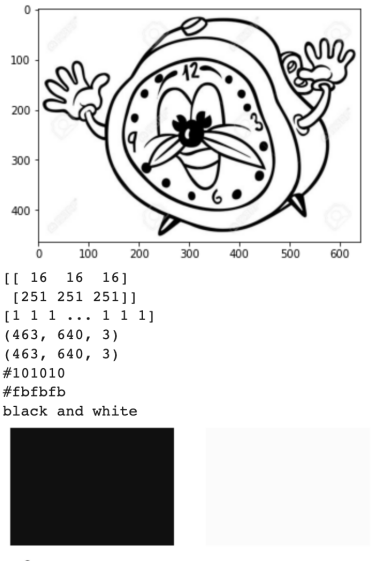


Fig. 18: Result of clustered black and white image (k=2)

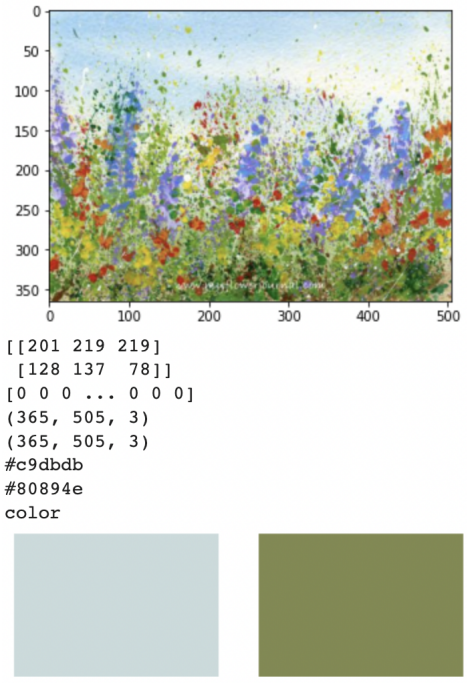


Fig. 19: Result of clustered colored image (k=2)

7.3. Implementation and Results

The k value is fixed to 2 for getting the two most dominant colors in the image based on [13, 14]. The centroids are returned after performing Kmeans on every image. The centroid represents the RGB value of the dominant colors. For black and white images the dominant colors are shades or variations of black and white. Some pictures may have more of white than black or vice versa. The RGB value is converted to hex value. To determine and separate the black and white and colored images the RGB value of the two dominant colors is converted to CIE color space and then the distance between the color space of black and white with the two dominant colors is calculated respectively. A threshold is set with respect to the distance to determine whether the colors are considered in the shades of black and white.

To analyze the dominant colors in a colored image the k value is fixed to 3 based on the elbow method performed on a colored image. A palette of dominant colors is returned.

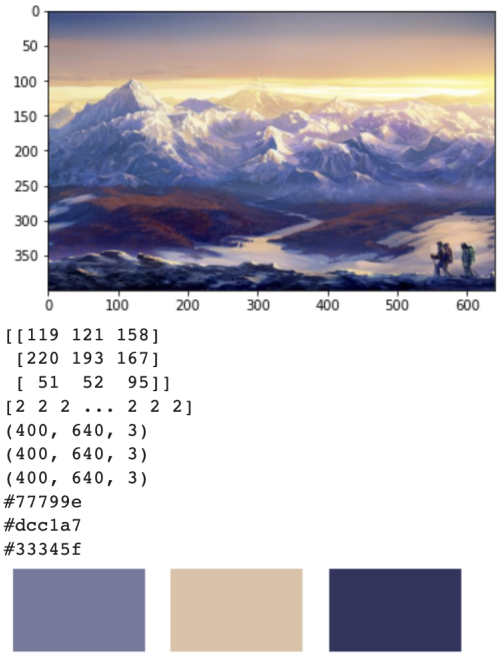


Fig. 20: Result of clustered colored image for 3 dominant colors (k=3)

```
No of black and white images in the data: 786
No of black and white images clustered: 802
No of colored images in the data: 1053
No of colored images clustered: 1037
```

Fig. 21: Count of black and white and color images, comparing the actual count of images with the count after clustering

The results in Fig. 21 show that there could be a few mismatches depending on the threshold chosen. This is a limitation that can be worked on.

8. DISCUSSION AND FUTURE WORK

We believe that we have been successful in answering the research problems that we set out to solve. With the clipart classification we were able to take data that has high variability

due to the artist's interpretation of the class and obtained an accuracy better than a random guess. Even though we were not able to achieve the state of the art performance for this classification we were happy that we were able to make significant improvement from the original approach of close to 4.3% accuracy .

For the classification of non photographic images we think that a more complex architecture needs to be used. One of the main reasons is because the illustration is based on the creativity of the illustrator. This results in a lot of non standard images which would make it very difficult for the model to learn patterns. We did explore transfer learning by implementing ResNet-50 with learned weights from Imagenette. But, even with this we did not get state of the art accuracy of over 95% in the test data. One of the future paths that we see is to further increase the complexity of the architectures but this will involve a lot more training time. We have used the carbonate environment for our project and faced memory issues even on carbonate so this training resources might be a constraint for building more complex models.

Kmeans clustering works efficiently in clustering the images based on the colors using the pixels and returning the centroids that represent the palette. The method of calculating similar colors based on the color space and the threshold set to determine whether the image is black and white is very subjective. An alternative method in the future can help separate the images effectively.

9. ACKNOWLEDGEMENTS

We would like to thank the UITS team for providing us with the necessary resources to run our programs on Carbonate. We would also like to thank Prof. Williamson for the guidance and feedback offered to us during the course of the project.

10. REFERENCES

[1] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang, "Moment matching for multi-source domain adaptation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1406–1415.

[2] Yann LeCun and Corinna Cortes, "MNIST handwritten digit database," 2010.

[3] Yanqing Zhou, Yongxu Jin, Anqi Luo, Szeyu Chan, Xiangyun Xiao, and Xubo Yang, "Toonnet: a cartoon image dataset and a dnn-based semantic classification system," in *Proceedings of the 16th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry, VRCAI 2018, Hachioji, Japan, December 02-03, 2018*, Koji Mikami, Zhigeng Pan, Matt Adcock, Daniel Thalmann, Xubo Yang, Tomoki Itamiya, and Enhua Wu, Eds. 2018, pp. 30:1–30:8, ACM.

[4] Young-Min Kim, "Feature visualization in comic artist classification using deep neural networks.," *J. Big Data*, vol. 6, pp. 56, 2019.

[5] Tim Busfield, "Image classification with tensorflow," .

[6] Aktas Yagmur, "Fine-tuning a cnn model for image classification," .

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[8] Syed Mohammad Abid Hasan and Kwanghee Ko, "Depth edge detection by image-based smoothing and morphological operations," *Journal of Computational Design and Engineering*, vol. 3, no. 3, pp. 191–197, 2016.

[9] Ray, "Computer vision: How to tackle the problem of class imbalance in image datasets?," .

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," 2015, cite arxiv:1512.03385Comment: Tech report.

[11] Jafar Isbarov, "Transfer learning with tensorflow," .

[12] Keyur Potdar, "Keras split train test set when using imagedatagenerator," .

[13] Karan Bhanot, "color identification in images," .

[14] Rohan Sethi, "Color extraction," 2020.