The Decade Classification of RGB Images using Convolutional Neural Networks and k-Means Clustering

Nicholas Abad (32105207), Pavels Dembo (32144420)
Saarim Aatri (33661839), Mya Zhang (32268500) and Ang Li (32265175)

SCC403: Data Mining

Lancaster University
January 21, 2018

Contributions

Within the project the jobs were carried out by the people below:

1. **Nicholas Abad:**
   (a) Create a Python script in order to locally save files into correct folders
   (b) Create a Python script in order to transform each of our images into an array that can be used within convolutional neural networks using the Keras library
   (c) Implement a convolutional neural network using our preprocessed data
   (d) Gather data regarding the accuracy of human classification per decade
   (e) Analyze, interpret, and compare the results of our model to human accuracy
2. **Pavels Dembo:**
   (a) Create a Python script that resizes or crops locally saved images in order to transform all images into a uniform size
   (b) Pre-process the data to eliminate missing and inaccurate features
   (c) Create a Python script in order to transform each of our images into an array that can be used within convolutional neural networks using the Keras library
   (d) Implement a convolutional neural network using our preprocessed data
   (e) Gather data regarding the accuracy of human classification per decade
   (f) Analyze, interpret, and compare the results of our model to human accuracy
3. **Saarim Aatri:**
   (a) Validation of original dataset to ensure equal number of images in each year
   (b) Create a script that takes the original dataset and randomly extracts image URLs to generate three subsets that are then used for our training, testing, and validation sets
4. **Mya Zhang:**
   (a) Pre-process the data in order to keep the consistency with CNN method
   (b) Create a Python script that performs k-Means clustering
   (c) Create a Python script that reduces dimensions of our data by testing on different parameters
   (d) Analyze, interpret, and compare the results of our k-Means Algorithm to human accuracy and CNN method
5. **Ang Li:**
   (a) Pre-process the data in order to keep the consistency with CNN method
   (b) Create a Python script that performs k-Means clustering
   (c) Create a Python script that tests the results of k-Means clustering
   (d) Analyze, interpret, and compare the results of our k-Means Algorithm to human accuracy and CNN method

Abstract

Due to the fact that classifying images by their date is an extremely prevalent issue in today's day and age, we decided to follow previous works and attempt to solve this problem through the use of a convolutional neural network and through the use of the k-Means clustering algorithm. With an aim to classify images per decade, we found that improving the amount of images that our algorithms are trained on slightly improves accuracy rates, which we determined to be the best performance metric to use throughout our project. With data regarding the accuracy rates of untrained humans being obtained in previous studies as well as through our own data collection method, we found that the accuracy rates that we subsequently obtained in both of our models performed higher than both of the accuracy rates of untrained humans. Our project goes into detail about previous studies, both of the methods that were implemented, and goes into detail about our conclusions and future works.

1 Introduction and Objectives (Nicholas)

With a photograph's age being of the utmost importance in subjects such as photography, history, geography, and archeology, estimating the date in which a photo has been taken has long been a problem that many people have been unable to solve. By relying primarily on domain expertise as well as on the contents within an image, date estimation has typically been a manual process in which experts in these fields use contextual clues, such as the make and model of cars or fashion trends within the current picture itself, to make an educated guess of the date in which a particular photograph has been taken. Despite being an inefficient and unreliable method for dealing with such a common problem, this has unfortunately been the norm when it comes to classifying images and it was only until recently that research has begun to try to come up with a better solution, particularly in a machine learning context.

Following the original study done by Palermo et al. in 2014 [1], several researchers have been attempting to classify images by creating features of their own and testing the performance of these features using machine learning techniques such as neural networks and support vector machines. In an attempt to follow their lead, our group decided to address this problem in our own way, though taking inspiration from three previous research groups before us. With an aim to classify images per decade between the years of 1930 through 1970, our ultimate goal was to use a previously created 1,021,215 image dataset and implement a convolutional neural network in order to train and test our model. Using this neural network then allowed us to train our model using varying amounts of images per year and two different image resizing methods (cropping and resizing) in an attempt to discover how the size of our training set and how the image resizing method affects our final accuracy. We then gathered our own data regarding untrained humans' image classification per decade and compared this to our aforementioned 6 accuracy measures. In addition to the convolutional neural network, we also set out to use k-Means in order to cluster our data and subsequently gathered the accuracy measures using $k = 5$ and $k = 10$. In general, it will be shown that more clusters provide higher accuracy to color images.

As our final result, we used accuracy as our main performance metric in both experiments and produced a testing accuracy using six differing datasets for the neural network and twelve differing datasets for k-Means, both of which can be found within *Table 1* and *Table 2* respectively. In regards to the neural network, despite increasing the amount of training images for cropped data, the accuracy did not quite change, which is unexpected. However, for the resized images, increasing the amount of training observations did increase the accuracy and it will be shown that the highest accuracy comes when using the 20,000 image resized dataset. The accuracy was obtained to be 0.3282. In regards to the k-Means algorithm, we varied the amount of clusters to be either 5 or 10 and then calculated the accuracy for both colored and black and white images. In this case, it will be shown that the highest accuracy comes when using a 10,000 image dataset and using black and white with 5 clusters. This accuracy was found to be 0.3854.

Additionally, by creating our own image classification test and gathering the results, we found that of the people that we polled, their average accuracy rate was about .203846, which is slightly lower than the 0.26 accuracy rate of untrained humans obtained in previous studies. However, with our top-performing neural network accuracy rate being 0.3282 and our top-performing k-means accuracy rate being 0.3854, our models perform better than both of these metrics.

2 Related Works Regarding Image Classification

2.1 Primary Motivations (Nicholas and Pavels)

One of our first motivations for this project came from Palermo et al.'s paper, "Dating Historical Images" [1], in which they attempted to classify images per decade by firstly computing seven unique features for each of their images and consequently evaluated the performance of these features using Support Vector Machines. By using Amazon's Mechanical Turk to gather data from untrained humans, they then compared the results of their model and showed that even their worst-performing feature showed a better overall accuracy than the responses that they received from those aforementioned human subjects, who had a success rate of 26.0% of identifying the correct decade that each of these images came from.

Similarly and by using the work of Palermo as inspiration, "Color Features For Dating Historical Color Images" [3] by Fernando et al., also aimed to extract certain characteristics from images in order to classify images by decade. Through the use of their two own unique color descriptors, color derivatives and color angles, they then used both linear and non-linear Support Vector Machines to evaluate the performance of their features and obtained a performance accuracy of 85.5%.

Further building upon the original study of Palermo et al., Eric Müller, Matthias Springstein, and Ralph Ewerth conducted a new study in an attempt to classify images based on their date, but did so by first using Flickr's API in order to create a dataset consisting of $1,021,215$ images that were taken between the years 1930 and 1999 [2]. With this dataset, they then trained $1,020,095$ of their images using GoogLeNet, a 22-layer deep network, and consequently used the remaining images within either their testing or validation set. By using absolute mean error as their main performance metric, it was shown that GoogLeNet came closer to predicting the year that a photograph was taken by roughly 3.6 years when being compared to untrained humans.

Using these three bodies of work as our primary sources of inspiration, we thus decided on a project that combines elements of each of these works into one. With a similar aim of attempting to classify images based on the decade that the image was taken, we set out to use the abundant $1,020,095$ image dataset that Müller created in order to train and test a neural network model as well as use k-Means in order to experiment on clustering algorithms. In addition to wanting to maximize our testing accuracy, we also decided to incorporate Palermo's idea of comparing this accuracy to the accuracy of untrained humans, which was found to be 26%.

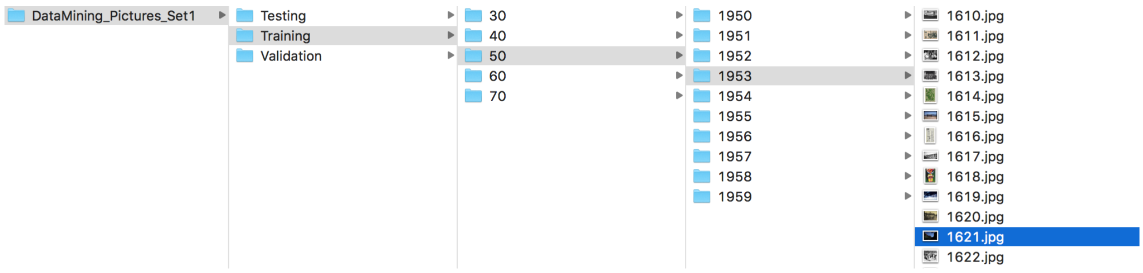
3 Methods and Solutions

3.1 Data Preprocessing

3.1.1 Creating Random Subsets of the Dataset (Saarim)

Through the use of Microsoft Excel's built in functions and a VBA script, we found it necessary to first create a program that generates a random data set without replacement from the $1,020,095$ image dataset that we will be working with. Functions such as *match* (to find row number in a given range for a specific query), *index* (to go to a row number provided and get the data present there), *rand* (for generating random numbers), *rank* (for ranking random numbers to get unique random row numbers to pick random data from) and *min* (to get minimum value in a range) have been used to achieve the desired results. The algorithm not only generates a random data set based of a given sample size, but it also makes sure that any previously generated sample data present in the original sheet, is not repeated in the new uniquely generated data set.

Using this script, we then took three random samples without replacement of our original dataset and generated a small, medium, and large dataset in order to compare how varying the set size affects our model. The small dataset consisted of a total of $5,000$ images in which the training set specifically had 100 images per year, resulting in $1,000$ images per decade. Similarly, the medium dataset consisted of a total of $10,000$ images in which the training set had 200 images per year or $2,000$ images per decade. The large dataset then consisted of $20,000$ images with the training set having 400 images per year totaling 4000 images per decade. Each of these sets maintained the same ratio of 70-15-15, in which the training set consisted of 70% of the dataset while the testing and validation set each consisted of 15% of the data.



DataMining_Pictures_Set1	Testing	30	1950	1610.jpg
	Training	40	1951	1611.jpg
	Validation	50	1952	1612.jpg
		60	1953	1613.jpg
		70	1954	1614.jpg
			1955	1615.jpg
			1956	1616.jpg
			1957	1617.jpg
			1958	1618.jpg
			1959	1619.jpg
				1620.jpg
				1621.jpg
				1622.jpg

Figure 1: *Example of the folder schema used for saving pictures using the three separate .csv files. The highlighted image,* 1621*.jpg, came from the first set, is being used within the training set, belongs to the 1950's decade, and was originally taken specifically in the year 1953.*

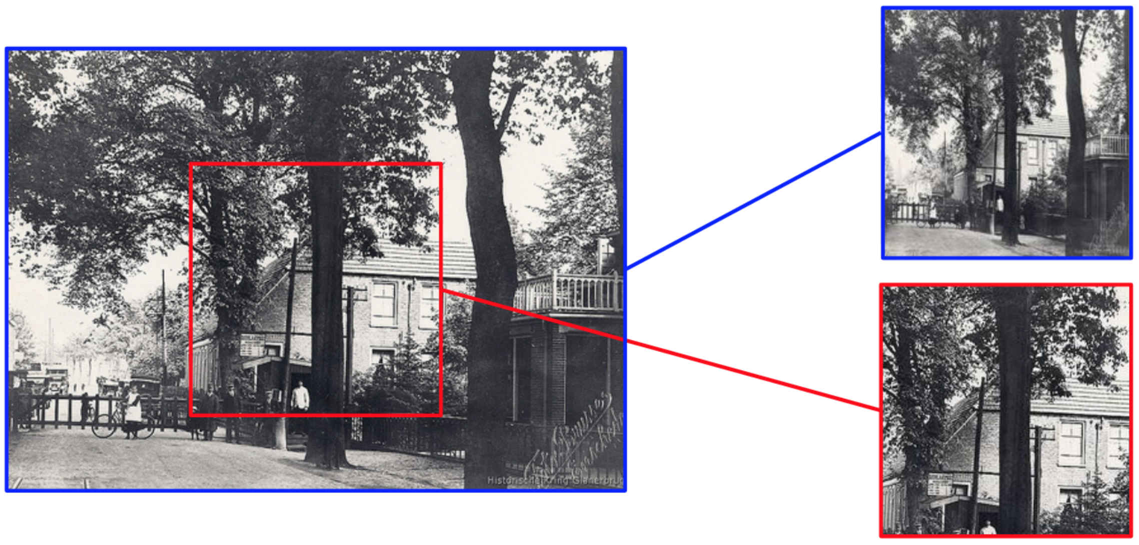


Figure 2: *Example of the two methods that we used to downsize images. The original image could be seen on the left while the **red** boxes represent how we cropped our images and the **blue** boxes represent how we resized our images.*

3.1.2 Locally Saving Files In Correct Folders (Nicholas and Pavels)

With these three separate .csv files, we then created a Python script in order to read in each URL within each file and save these images locally in a specific format that would be easiest for our group to work with. This script would first take into account the first .csv file, which was labeled *Set 1*, and create a new folder, *DataMining_Picture_Set1*. Within this *DataMining_Picture_Set1* folder, three new sub-folders were then created corresponding to the training, testing, and validation sets that we will be using. For all three of the folders labeled *Training*, *Testing*, and *Validation*, new sub-folders were then created within each in order to indicate the decade that each image belonged to. Additionally and within each of the 5 unique decade folders, 10 more sub-folders were created to correspond to each of the years within that decade. Images for this set were saved accordingly and this process was repeated again for both *Set 2* as well as with *Set 3*. An example of the folder schema can be found in *Figure 1*.

3.1.3 Resizing and Cropping Images (Nicholas and Pavels)

Due to images within our three datasets each having different individual sizes, we found it necessary to transform every picture to a uniform size of 256 by 256 through the usage of both cropping and resizing in order to test whether the chosen method has an effect on our final results. In order to do so, we thus created a Python script that would first take in the folders seen in Figure 1 and subsequently create two new folders following the same schema. One of the new folders contained resized images of each of the original photographs while the other folder contained cropped images of each of the original photographs. When dealing with where to crop our photos from and

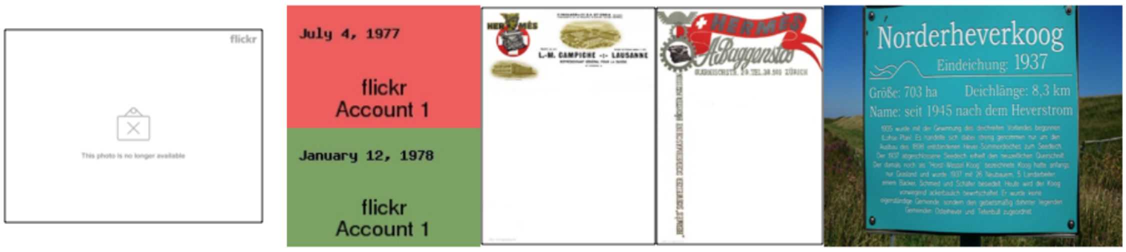


Figure 3: *Examples of missing and incomplete pictures within our dataset. Inaccurate images (right) such as these ones had to be manually omitted from out datasets.*

because our images in this case needed to be cropped down to the aforementioned 256 by 256 size, we determined that it would be best to crop from the direct center of the original picture. Examples of both of these methods have been shown above in *Figure 2*.

3.1.4 Cleaning the Dataset (Pavels)

As previously mentioned, we now have a grand total of six datasets that we will be working with:

(i) 5,000 cropped images	(ii) 10,000 cropped images	(iii) 20,000 cropped images
(iv) 5,000 resized images	(v) 10,000 resized images	(vi) 20,000 resized images

As previously mentioned in *Section 3.1.1*, all datasets were split into proportions of 70-15-15, where 70% of all images were used for training, and the remaining 30% of images were split evenly between the validation and testing sets. With these datasets in hand, the next step involved preprocessing which was done in two parts. Firstly, all missing images were removed. This was accomplished by comparing the hashes of each image against a known hash that represented a missing image. The second part of cleaning the data was more labor intensive, as it required a manual inspection through every folder to detect and remove inaccurate images. Such images were categorized as containing over certain percentage of identical color or containing misleading information. In order to see examples of missing and incomplete pictures, please refer to *Figure 3*.

3.1.5 Creating Arrays Consisting of RGB Values (Nicholas and Pavels)

As our final step of our data preprocessing phase, it was necessary to create a Python script in order to transform each of our images into an array in order to feed it into our convolutional neural network and k-Means clustering algorithm. In doing so, we used the NumPy library as well the Python Imaging Library (PIL) to first load in each image and iterate through the image pixel by pixel. At each of those $256 \times 256 = 65536$ pixels, we would generate a 3-tuple of the red, green and blue (RGB) values and append each one of these 3-tuples into a single NumPy array. By repeating this process for the remaining images, we would then have a multitude of image arrays, each of which represent an individual image's RGB values at each pixel. With each of these image arrays, we then initialized an empty array and appended those individual image arrays into that new array. We found this to be necessary in order to put this in the correct format for both of our algorithms.

3.2 Building a Neural Network (Nicholas and Pavels)

Convolutional Neural Networks can be regarded as a subset of feed forward Neural Networks which are commonly applied for classification of visual data. The backbone of every CNN is its architecture which tends to vary depending on the type of the problem that is being solved. There is no right or wrong design approach, however most successful approaches have been extensively analysed in various literatures. As such, the developmental and perfection of a CNN is an incremental process where heuristics are tuned until most optimal results are found.

Due to the uniqueness of the problem as well as the little amount of information available about the subject of classifying historical images, a large amount of time was devoted to exploring effectiveness of various architectures. The exploration started by analysing existing architectures

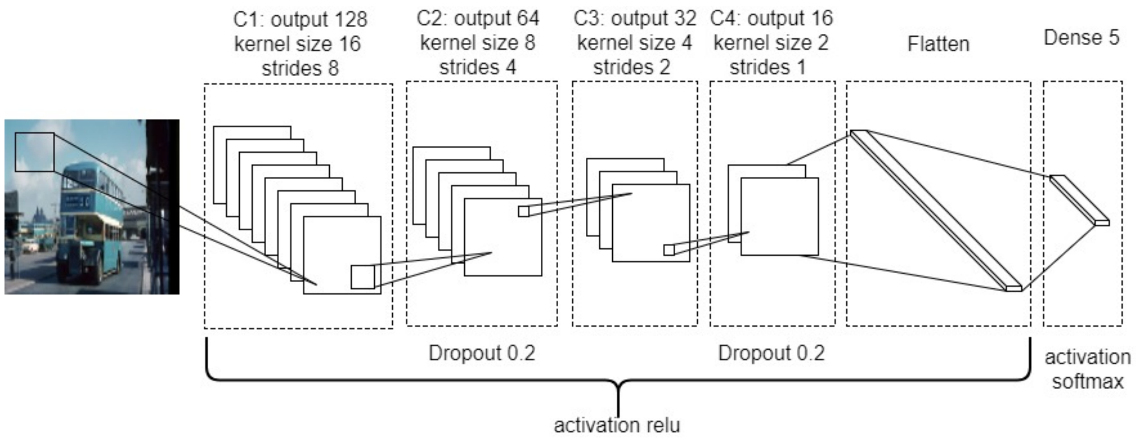


Figure 4: *The architecture of the convolutional neural network that was used.*

which are employed for MNIST and CIFAR datasets [4, 5]. Both architectures have their fair share of similarities such that they both employ some amount of convolutional and pooling layers and various activation functions. Key distinction that was made at the start is that the two architectures mentioned above focus on classification of an object displayed on the image, rather than the image itself. Based on this justification, pooling was removed from our architecture since its primary aim is to detect different positions of a given feature. Further experimentation revealed that by using large kernel filters, the CNN would be forced to look at the larger section of the image, instead of locating isolated features. Again, this is crucial to our task since we are interested at the image as the whole and not at a specific feature.

After conduction further experimentations the architecture shown in the *Figure 4* was chosen. The final architecture employs 4 convolution layers in a proceeding manner. The output of each layer is a halved, which in turn reduces the dimensionality of the implemented CNN. Each layer implements a fairly large kernel and proportionally to that a large striding window. As mentioned previously, this was implemented to force the CNN to look at the larger sections of the image, instead of focusing on local features. Since the chosen dataset employs wide range of images for each class in terms of different motives and colours, it was crucial to ensure that the model would not overfit on the training data. Various tests were conducted with and without drop-out layers. The results revealed that with the drop out layers implemented, the CNN tends to have marginally higher accuracies for both validation and test sets. Presumably, this occurs because drop-out layers prevent the model to develop intricate connections within the hidden layers, which in turn eliminates the problem of overfitting.

The loss function measures compatibility between a prediction and the ground truth label. As such it is crucial for any architecture to minimise the loss, as it acts as one of the primary indicators of poor convergence of the model. Unlike the previous processes of selecting parameters, work published by Muller et al. implements stochastic gradient descent optimisation function with learning rate of 0.001, learning rate decay 1e-6 and momentum of 0.9. After various testing, we found these parameters to work well for us.

Lastly, proposed architecture also employs various activation functions both after each convolutional layer and the output layer. Activations layers that were used for the our CNN can be inferred from the *Figure 4*.

3.3 K-Means Clustering

3.3.1 Different Approach on K-Means Clustering (Mya and Ang)

Using the method detailed in *Section 3.1.1*, three datasets were used that contained 5,000, 10,000, and 20,000 images respectively. As the k-Means algorithm focuses on clustering existing data into different groups by similarities, there is no need for validation set to minimize overfitting, however training sets and testing sets are still required for implementation and subsequent analysis of the results. After discarding the validation sets, our three datasets were subsequently reduced to contain 4,200, 8,500, and 17,000 images with 82% of the respective data belonging to the training set and the remaining 18% belonging to the testing set. Due to the nature of the k-

Means algorithm when dealing with color segmentation, the sizes of images are not required to be the same. Therefore, data resizing and cropping are not applicable here as we have kept all downloaded images in their original format. After manual inspection of the datasets detailed in *Section 3.1.4*, data cleaning code was implemented in python by manually inputting the label of inaccurate images, which contains high percentage of certain color or containing inaccurate information. In order to match the way in which data is inputted into the k-Means clustering algorithm, we used the NumPy library as well the Python Imaging Library (PIL) to first load in each image and iterate through the image pixel by pixel. Due to images being of different sizes, we summed up each image's hue, saturation, and value measurements for each pixel. After performing this operation on all the images, we obtained a tuple with the value of (Hue, Saturation, Value).

3.3.2 Methodology Applied in the k-Means Algorithm (Ang and Mya)

asd While containing different red, green, and blue values, each image can be considered as an RGB array. For a dataset containing 5000 images, the dataset needed to be formatted into a matrix of the size $5000 \times 3 = 15000$. Unlike the CNN, the k-Means algorithm aims to separate data observations into different clusters where each data point belongs to the cluster with the nearest mean, or nearest distance on a graph. Hence, a 5000×3 RGB matrix is not feasible to implement K-Means due to our constrains, as the aim here is to calculate the closest distance among each data point and find a best cluster. RGB measurements have another disadvantage on black and white images, where the color black is represented by the three-tuple (0,0,0), and the color white is represented as (255,255,255)[7]. These extreme measurements would cause the resulting clusters to be heavily skewed to 0 or 255 when demonstrating black and white images. Although RGB is straightforward for image identification, HSV separates color information (Chroma) and image intensity or brightness level (Luma), which is more useful for image segmentation. As HSV components signify Hue, Saturation and Value (gray intensity) of a pixel, they are not correlated to each other in terms of color as each component has its own role in defining the property of that pixel. Hue provides information regarding color (wavelength in other terms), saturation always shows how much percentage of white is mixed with that color, and value is the magnitude of that color (or Intensity)[6]. As all components of HSV space vary, they do not follow the same scale for representation of the values. The characteristics of hue and saturation can be only applied to color images because it is meaningless for black and white images. Value, on the other hand, is suitable for both black & white and color images. Hence, instead of normalisation, our approach is to apply all three values of H, S, V for color images, and only use V when implementing black and white images. To better represent the picture, *Equation 1* (below) is used to calculate the average H, S, V values of each pixel to quantitatively represent one image in an array of (H,S,V).

$$\mu_{H,S,V} = \frac{1}{n}\sum_{i=1}^{n} H_i S_i V_i \tag{1}$$

The k-Means clustering algorithm is a method of vector quantisation, originally from signal processing, that is popular for cluster analysis in data mining. In our case, the most important elements of the algorithm distance d, is defined below

$$d = \sqrt{(H' - H)^2 + (S' - S)^2 + (V' - V)^2} \tag{2}$$

in which H represents hue, S represents saturation, V represents value. This distance d is the basic Euclidean distance between two points in three dimensions.

In reference to *Figure 5*, "Initiation" is the section of the program that initiates all the variables, as well as reads in all the images. Each image will be calculated pixel by pixel to get the average value of Hue, Saturation and Value of the image. This data is then put into a 3-dimensional list of HSV values. In the next step, "Set n Centers Randomly", n could be either 5 or 10 in our case. We equally split the whole dataset into (number of centers) sets and pick the 1st image of those subsets as center points to make it a "random" center. In the step "Get n New Clusters", the program looks at the whole dataset again and assign all the images to the center which has the minimum distance d. By doing this, new clusters are generated. In the step "Calculate N New Centers", the program looks at each cluster, calculates the average value of (H, S, V) of all the images in the cluster, and assign this point as the new center point of this cluster. In the step "Same Centers?", the program compares old centers with new centers. If they are not the same, the program will go back to step

"Get n New Clusters" with new center points. It will continue iterating until the new center points equal with the old ones. Then the program steps into "Get Results of Clusters". In this step, the program outputs all the clusters with all their components, including how many images are in each decade, and their respective percentage. This information is then used in the analysis part. Finally, in the step "Label Clusters", the program obtains the percentages of how many images are in each decade. According to the rank, the program assigns each clusters to each decade. As the result, each decade will be assigned with same amount of clusters.

4 Results

4.1 Neural Network Results (Nicholas and Pavels)

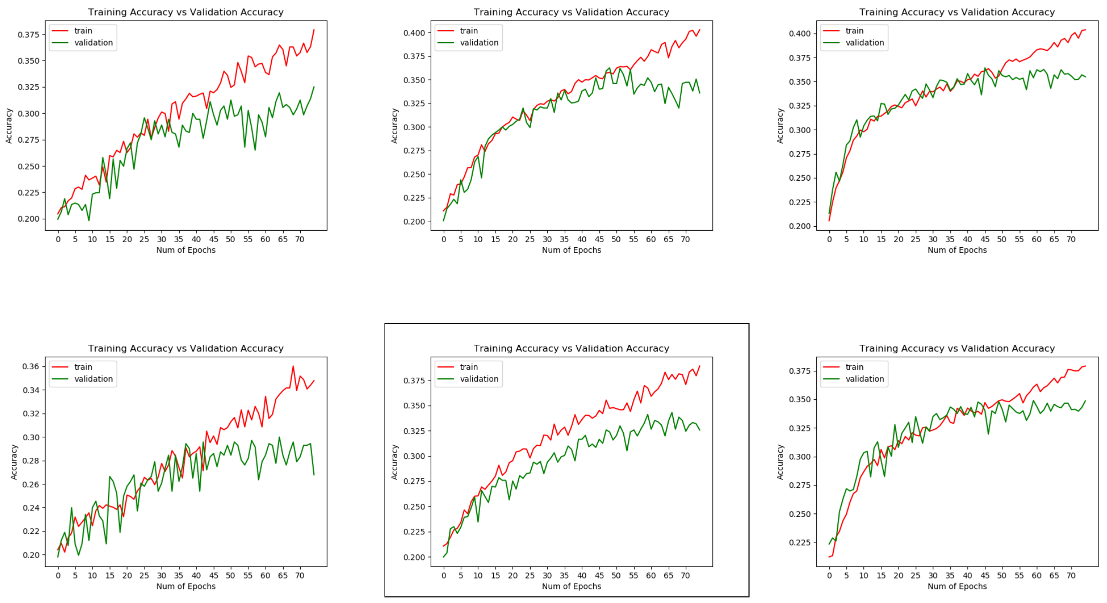


Figure 5: *Top: Training and validation accuracies across the three **RESIZED** image datasets. Bottom: Training and validation accuracies across three **CROPPED** image datasets. The images within the left column are the datasets containing 5,000 total images, the images within the middle column are the datasets containing 10,000 total images, and the images within the right column are the datasets containing 20,000 total images. The x-axis is the number of epochs while the y-axis is the accuracy.*

Through the use of our convolutional neural network that was described in detail throughout *Section 3.2*, we sought out to train each of our six datasets using 75 epochs, and then finally test our model using the testing set. When evaluating our model on our testing set, the main performance metric used was accuracy, which is similar to previous studies [1][2], and can be defined as the simple fraction of the number of correct predictions over the total number of images tested. After the model was tested, we thus received six final accuracy metrics which can be found in *Table 1*. Unfortunately, when producing these accuracy metrics, we found there not to be that significant of a difference between using the cropped images and resized images. When comparing the cropped 5000 image dataset to the cropped 20000 image dataset, the resulting difference was only a negligent 0.0001% difference. However, when comparing the same two values of that of the resized image datasets, the difference in the two accuracies was 0.027%, in which using the 5000 image dataset resulted in an accuracy rate of 0.3012 while the 20000 image dataset resulted in an accuracy rate of 0.3282. Because of the relatively small differences in both of the datasets, we first came to the conclusion that specifically in our dataset and by using our current convolutional neural network architecture, choosing between these two methods of shrinking an image does not quite matter as neither produces a significant benefit over the other. In fact, of these two methods, the highest accuracy of the cropped datasets was 0.3206% while the highest accuracy of the resized datasets was 0.3282%.

Table 1: Testing Accuracy of all 6 datasets when put into our CNN

	5,000 Image Dataset	10,000 Image Dataset	20,000 Image Dataset
Cropped	**0.3205**	**0.3203**	0.3206
Resized	0.3012	0.3070	**0.3282**

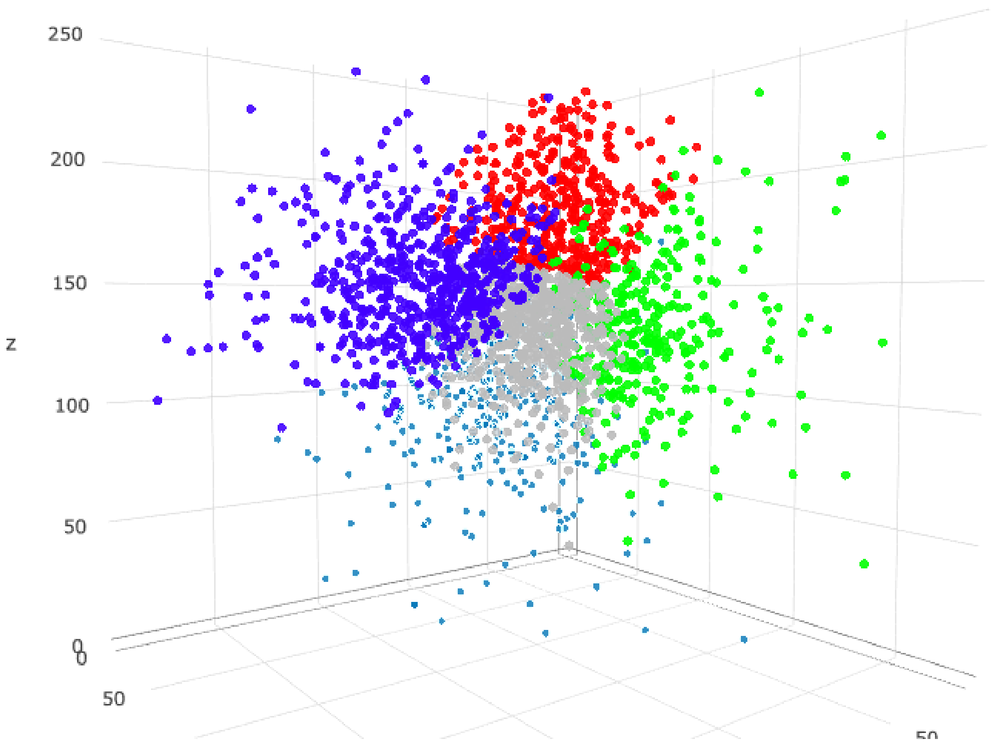


Figure 6: *The distribution of clusters on 5000 images. Many data points are gathered near 0 because of black and white images.*

On the other hand, by increasing the amount of images that our model is trained on, our testing accuracy improved on both the cropped and the resized datasets, which was to be expected. By training a model on more images per decade, our convolutional neural network is better able to learn more about what is important within an image when trying to classify images by decade. However, this comes at a cost as the potential of over-fitting may come into play. Through the analysis of of the graphs located in *Figure 6*, we found there not to be that significant of a case of over-fitting as the validation lines (green lines) in these graphs are relatively close to the training lines (red lines). This implies that over-fitting isn't heavily prevalent when training our model on these datasets.

4.2 K-Means Algorithm Results (Ang and Mya)

When analysing the k-Means algorithm, we used accuracy as our performance metric in order to compare these metrics to what was obtained when using the CNN. We utilised the testing sets with labels to verify the accuracy of produced clusters. To reduce bias and investigate how the varying k affects results, we tested our algorithm by calculating more clusters using the same training set. The results are shown in Table 2. In general, more clusters provide higher accuracy to color images, except for the 5000 image dataset where the accuracy for 5 clusters and 10 clusters is marginally different.

Table 2: Testing Accuracy for k-Means Clustering Algorithm

	5,000 Image Dataset	10,000 Image Dataset	20,000 Image Dataset
5 Clusters - Color	**0.3204**	0.2523	**0.2449**
5 Clusters - Black and White	0.2489	**0.3854**	0.0215
10 Clusters - Color	0.3097	**0.3131**	0.3099
10 Clusters - Black and White	**0.2667**	0.0412	**0.3805**

5 Conclusion

5.1 Limitations and Future Work (Nicholas and Pavels)

Throughout our project, we believed that one of our primary limiting factors was our ability to train, validate, and test our convolutional neural network in the sense that we did not quite have the appropriate computational power to do exactly what we wanted. With only our personal laptops in hand, we were unable to train out dataset on larger subsets of the data simply because it would take an unreasonable time to do so. Additionally, within the dataset itself, another limitation came in *Section 3.1.4* when we found it necessary to manually inspect every folder to detect and remove images that we categorized as being inaccurate. Due to inaccurate images not having a particular format, style, or size, we could not create an automated program to check whether an image should be omitted from the dataset. Rather, we needed to manually inspect images ourselves, in which subjectivity rather than objectivity is required, which could therefore bring forth differing results if multiple people were to perform the same task. In terms of the convolutional neural network and with a multitude of changeable hyperparameters within it, potential future work includes implementing several different convolutional neural network architectures in which these parameters, such as the activation function used or using something other than stochastic gradient descent for example, differed in an attempt to produce the greatest testing accuracy for the dataset. With these different models in hand, k-fold cross validation could then be applied in order to aggregate the results and produce a mean estimate for our model's accuracy. Additional future work could also include the increase of the amount of images that the aforementioned networks were trained on to give a better estimate.

In an attempt to shy away from the use of neural networks, it is also worth mentioning that due to many previous studies using support vector machines for the image classification problem that we have presented, future work could also include mimicking these ideas and/or slightly modifying what has been done in the past. Finally, given the nature of the problem, a simple feed-forward neural network could also be tried as well.

In regards to implementing the k-Means algorithm, the biggest limitation we faced was finding the most suitable method for eliminating images' year labels. One of the primary applications of the k-Means algorithm is clustering or unsupervised classification which only applies to the data without labels. However, the dataset came with labels, and it was hard to eliminate all the labels when analysing training results, since the results needed to be mapped back to labels. Furthermore, the nature of the dataset also limited the accuracy of our results (10% of the downloaded images were missing data). Our approach was to ignore missing images due to time constrains, which reduced the number of samples for both training and testing sets and increased bias as the result. This problem could be avoided in further research by improving our extracting methodologies when obtaining images from the web. Finally, although the original dataset contains one million images, hardware and time constrained us from increasing the size of our sets bigger than 20,000 images. With larger datasets, we expect that results would be less biased according to Central Limit Theory.

5.2 Final Remarks (Nicholas and Pavels)

With an overall aim of estimating the decade in which a photograph was taken, it has been shown that when implementing the convolutional neural network detailed in Section 3.2 on both cropped and resized images, we obtained test accuracies that ranged between 30 and 32%, the highest being that of training our neural network on a 20,000 resized image dataset in which a testing accuracy of 32.82% was obtained. Although our testing accuracy was not as high as previous studies, we were still able to develop a new conclusion being that the method in which we decided to reduce the size of our image, whether that is through image resizing or cropping, does not affect our final outcome. However, despite obtaining relatively low accuracy rates, our convolutional neural network, produced a higher accuracy rate when being compared to the data that we collected as well as Palermo's gathered data on untrained humans, which were found to be 20.3846% and 26% respectively.

References

[1] Frank Palermo, James Hays, and Alexei A. Efros *Dating Historical Color Images* 2014 <http://graphics.cs.cmu.edu/projects/historicalColor/HistoricalColor-ECCV2012.pdf>

[2] Eric Müller, Matthias Springstein, and Ralph Ewerth *"When Was This Picture Taken?" – Image Date Estimation in the Wild* 2017 <https://link.springer.com/content/pdf/10.1007%2F978-3-319-56608-5_57.pdf>

[3] Basura Fernando, Damien Muselet, Rahat Khan, and Tinne Tuytelaars *"Color Features For Dating Historical Color Images"* 2014 <https://lirias.kuleuven.be/bitstream/123456789/479431/2/3833_postprint.pdf>

[4] *"CNN implementation for MNIST dataset"*<https://github.com/kerasteam/keras/blob/master/examples/mnist_cnn.py>

[5] *"CNN implementation for CIFAR10 dataset"*<https://github.com/kerasteam/keras/blob/master/examples/cifar10_cnn.py>

[6] Lidiya Georgieva, Tatyana Dimitrova, and Nicola Angelov *"RGB and HSV colour models in colour identification of digital traumas images"* 2005 <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.519.3568&rep=rep1&type=pdf>

[7] Rakib Hassan, Romana Rahman Ema, and Tajul Islam *"Color Image Segmentation using Automated K-Means Clustering with RGB and HSV Color Spaces"* 2017 <https://pdfs.semanticscholar.org/5cfb/01d9f4a8e29756f761bb6dc4cece4920df5e.pdf>