SPRING 2019

CS464 Final Report

Introduction to Machine Learning

Selin Özdaş 21400537
Büşra Arabacı 21401688
Cem Demirpek 21301643
Doğan Can Eren 21400329
Elif Engin

Introduction

The drastic increase in the number of artworks that are digitized created a data pool. This pool required people to find ways to retrieve and archive the data. Artists use different concepts to describe paintings such as texture, colors, patterns and balance. The main problem caused by this situation is that some features may give high variations. Mona is a categorization tool of fine art paintings in terms of their specifications such as genre, style etc. by processing visual features. To demonstrate Mona, we are going to use a publicly available dataset from Kaggle which is Painter by Numbers prepared by Kiri Nichol.

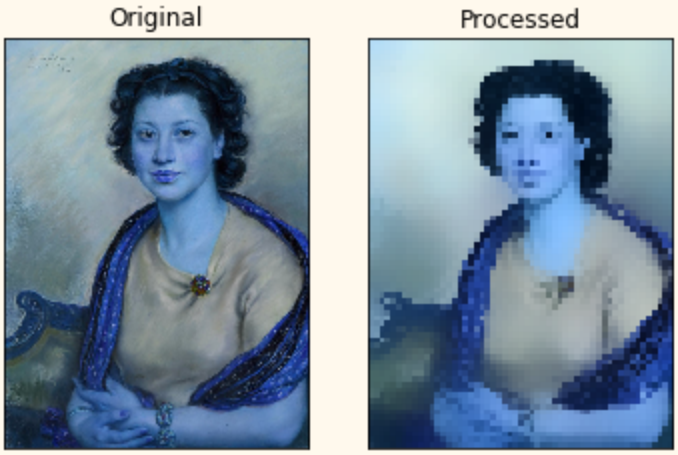
Background Information

“According to which metrics does an art historian categorizes the artwork?“ is one of the major questions arose after the institutions of art gain power[2]. While there were subjective opinions about how art historians do that, emergence of new technologies enhanced our understanding of artworks. Art historians turned image processing techniques to good account to get a better understanding of artworks. It is found out that we can encode information of fine art paintings by applying unsupervised and supervised learning methodologies[3]. Additionally, discovery of convolutional neural network(CNN) modelling was a breakthrough in image classifications tasks. It turned out that CNN modelling is beneficial due to the decrease in parameters and the reduction in the amount of training data to save time. It was like seeing the greater picture by using a smaller frame.

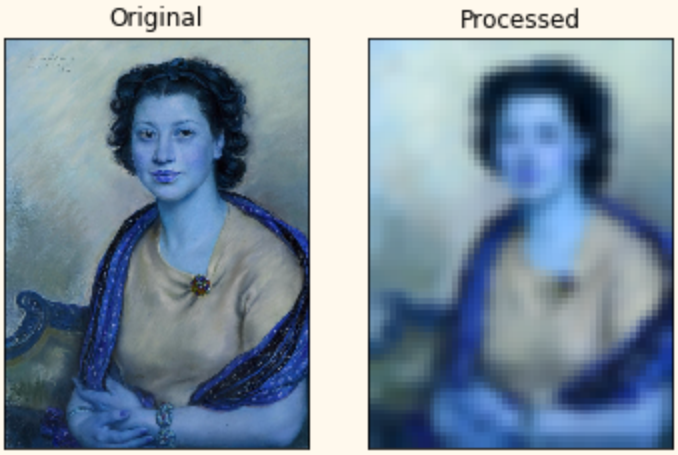
Work Done

Preprocessing images

In order to feed the images to machine learning and deep learning models, some preprocessing needed to be done because images in the dataset varies in size and color balance. Image sizes were changed and all images were converted to same size. Also, rgb color values are normalized to the range between 0 and 1 because using other values causes neural networks to converge to 1 for all outputs since the input values are high which affect the classification. Moreover, gaussian blur filter was applied to images to remove noise. Bilateral image filtering was tried and the errors could not resolved, however it looks promising as it produces a clean image for training:



Compared to Gaussian filtering:



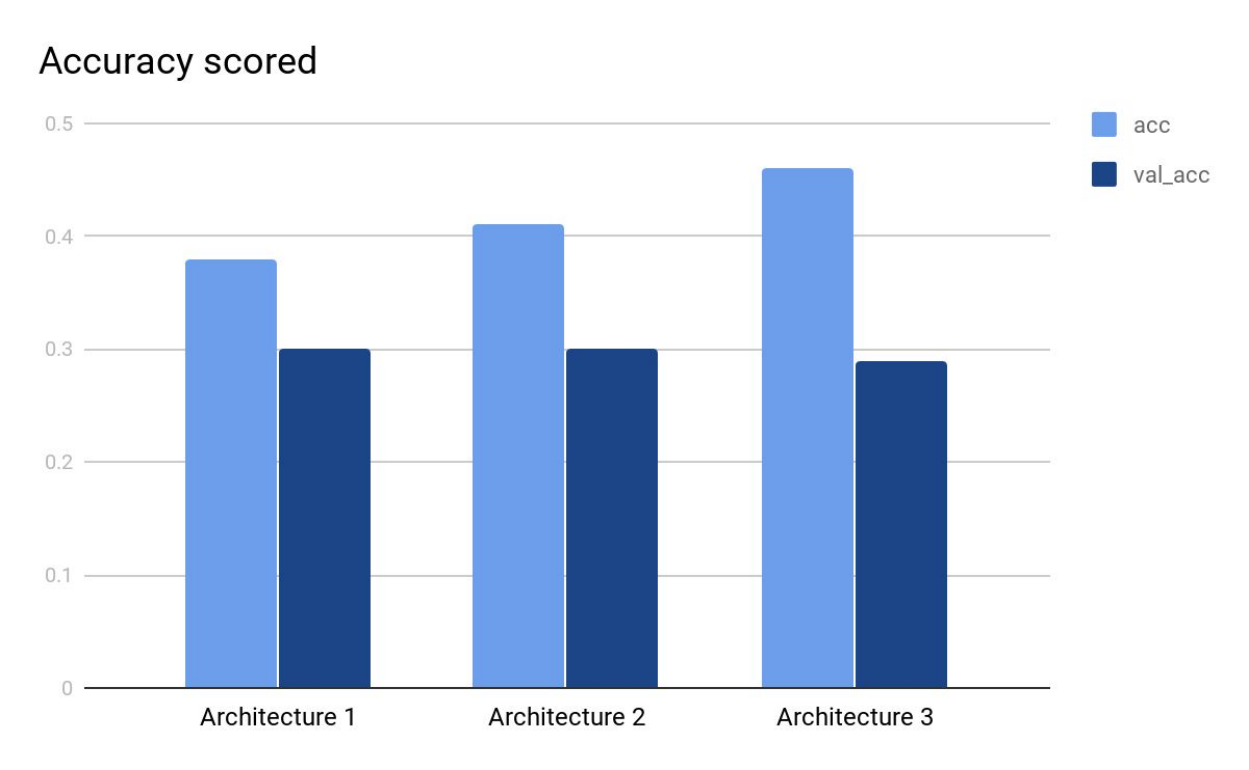
Opencv library is used for image resizing and noise reduction. Additionally, some data cleaning was done in the dataset because dataset includes some corrupted images and images with mode different than rgb.

Image classification

Our focus for this iteration is to classify images according to their genre and style. Artificial neural networks and convolutional neural networks were trained for this purpose. Since they are both multiclass classification, we only focused on genre and in the next iteration, we will apply the best model for style classification.

We had 8241 samples in the dataset after data cleaning. For the genre and style classification, since we have 127 styles and 40 genres and the task becomes

multiclass classification. Since it is multiclass classification, we encoded the classes by using one hot encoding technique. Therefore, for genre classification, we had one hot vectors of length 40 and for style we had one hot vectors of 127 for labels.
We split the dataset into train and test set. We used ⅛ of the data as test set.
We first trained artificial neural network models for creating baseline model. We tried different architectures by changing hidden layer number and number of neurons in the layers. Also, we experimented with different optimizers. For these neural networks, stochastic gradient descent algorithm is used with batch size 1.
Results with artificial neural networks:



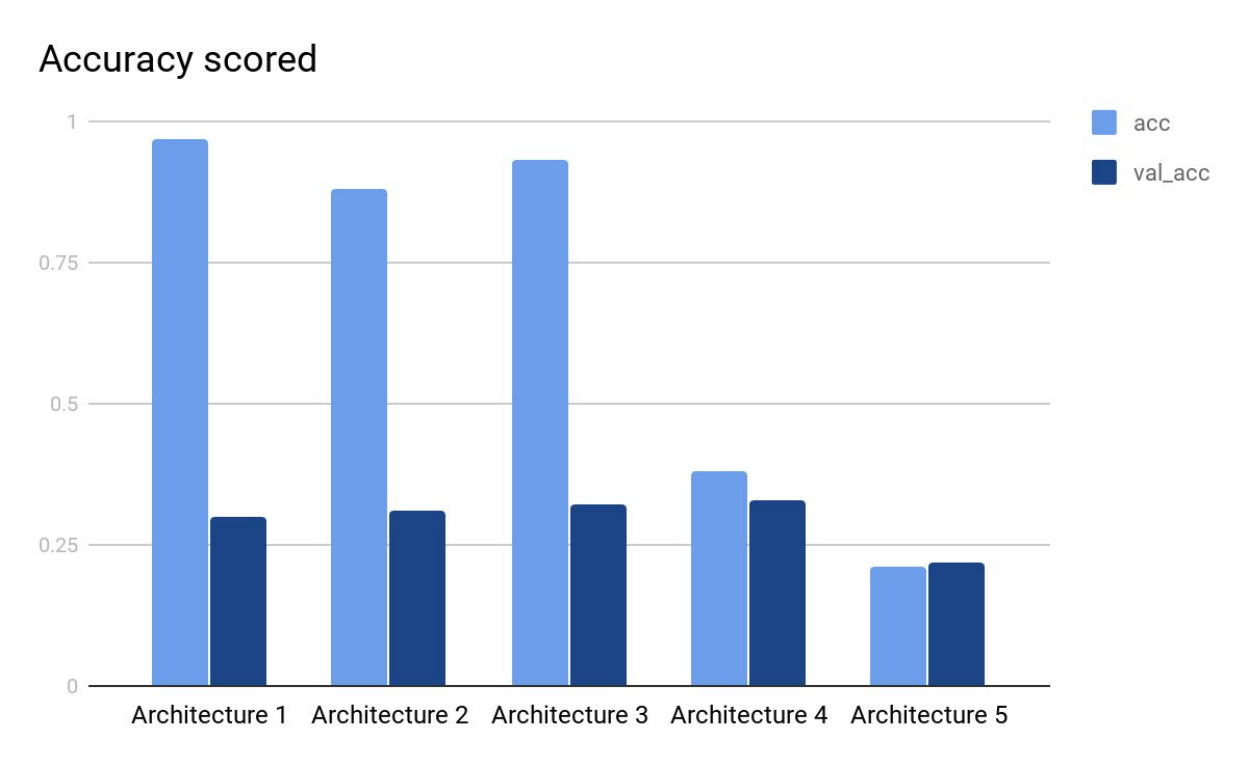
Architecture: 100*100*3 -> 256 -> 64 -> 40 Results: acc:0.38, val_acc: 0.30
Architecture: 100*100*3 -> 1024 -> 256 -> 64 -> 40 Results: acc: 0.41 , val_acc: 0.30
Architecture: 100*100*3 -> 3000 -> 40 Results: acc: 0.46 , val_acc: 0.29

According to artificial neural network results, network with one hidden layer and large hidden layer size performs best on training set. But their results on test set are similar. As we can see these architectures are not powerful enough to capture images features. They give low accuracies because they are simple neural

network architectures. Therefore, we tried Convolutional Neural Network architectures for image classification task.

CNN Models

With Convolutional Neural Networks, we experimented with different number of convolutional layers, different number of hidden layers. Also, we experimented with adding max pooling option but we realised that it does not help in our case. Pooling helps on increasing the receptive field of a filter, which is defined as the region in the input space that a particular Convolutional Neural Network's feature is looking at[1]. It would help if we were looking for a specific region in the input like eyes, but as we are not; it reduces our model's accuracy.



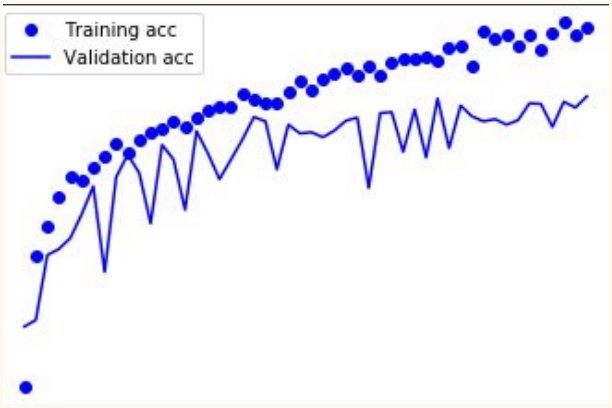
Architecture: kernelsize : 9, conv32 -> conv32-> conv64-> conv128 -> 512 -> 128 -> 40
Results: acc:0.97, val_acc: 0.30
Architecture: kernelsize : 3, conv32 -> conv64-> conv64-> conv96 -> conv32->128->40
Results: acc:0.88, val_acc: 0.31
Architecture: kernelsize : 3, conv32 -> conv64-> conv64-> conv32 ->128->40
Results: acc:0.93, val_acc: 0.32
Architecture: kernelsize : 3, conv32 -> conv64-> conv64-> conv128-> 256->128->40
Results: acc: 0.38, val_acc: 0.33

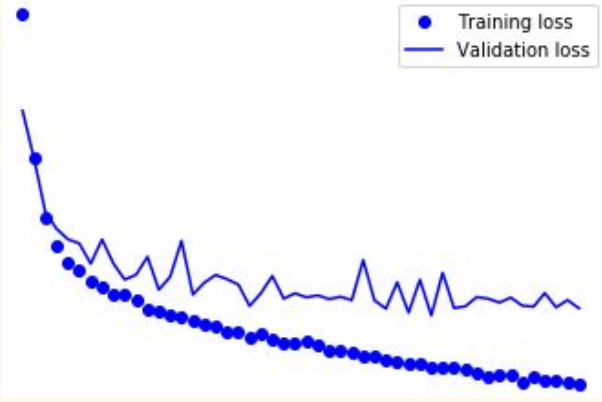
Architecture: kernelsize : 5, conv32 -> conv64-> conv64-> conv96 -> conv32->128->40
Results: acc:0.21, val_acc: 0.22
Architecture: kernelsize : 5, conv96 -> conv128-> conv128-> conv96 -> conv64->512->40 Results: acc:0.92, val_acc: 0.29
For the convolutional neural networks, we obtained good results for the training set. However, it gives bad accuracy for test set. This is a sign for overfitting. There are several ways to solve overfitting problem and one of them is adding regularization to the model. We experimented on the best result we get by increasing dropout rates by .25, and we get the following results:
Architecture: kernelsize : 3, conv32 -> conv64-> conv128-> 256->128->40
Results: acc: 0.33, val_acc: 0.35

Final CNN Model

Model 1

This model includes 3 convolutional layer followed by 2 fully-connected layer. After each layer (except the last) there is 0.5 dropout rate. Between the convolutional layers there are three max-pooling layers with 2 x 2 window size. The preferred optimizer is Adam.
Results = Epoch 50 loss: 1.9588 - acc: 0.4075 - val_loss: 2.2906 - val_acc: 0.3610





Model Summary =

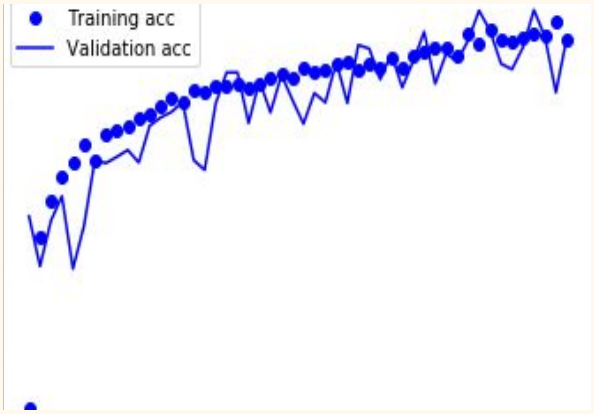
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 48, 48, 32)        896
_________________________________________________________________
batch_normalization_7 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
```

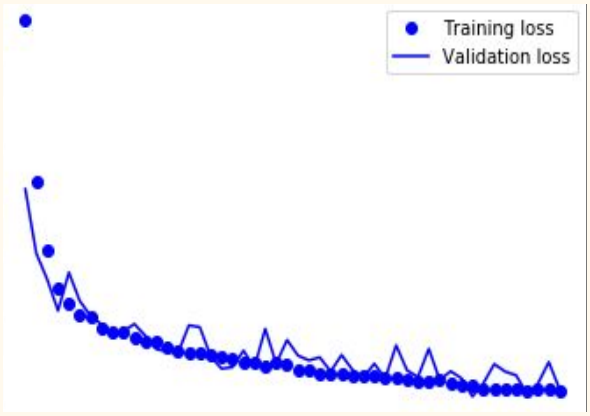
```
_________________________________________________________________
dropout_7 (Dropout)          (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 11, 11, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 9, 9, 64)          18496
_________________________________________________________________
batch_normalization_8 (Batch (None, 9, 9, 64)          256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                65600
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 40)                2600
=================================================================
Total params: 97,224
Trainable params: 97,032
Non-trainable params: 192
_________________________________________________________________
```

Model 2

As we overfit to the data, to decrease overfitting we changed the max pooling window size to be 3 x 3 for the first two max-pooling layers and decreased the overfitting problem and the loss without considerable amount of decrease in accuracy.

Results = Epoch 47 loss: 2.2842 - acc: 0.3396 - val_loss: 2.2750 - val_acc: 0.3540





Model Summary =

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 48, 48, 32)        896
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 16, 16, 32)      0
_________________________________________________________________
batch_normalization_23 (Batc (None, 16, 16, 32)      128
_________________________________________________________________
dropout_29 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 4, 4, 32)        0
_________________________________________________________________
batch_normalization_24 (Batc (None, 4, 4, 32)        128
_________________________________________________________________
dropout_30 (Dropout)         (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 2, 2, 64)          18496
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 1, 1, 64)        0
_________________________________________________________________
batch_normalization_25 (Batc (None, 1, 1, 64)        256
```

```
_________________________________________________________________
dropout_31 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_14 (Dense)             (None, 64)                4160
_________________________________________________________________
batch_normalization_26 (Batc (None, 64)                256
_________________________________________________________________
dropout_32 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_15 (Dense)             (None, 40)                2600
=================================================================
Total params: 36,168
Trainable params: 35,784
Non-trainable params: 384
_________________________________________________________________
```

Model 3

We added L1 regularization to the data to make it better. Even though it did not help us to increase our accuracy, now our validation accuracy becomes more than the training accuracy

Results = Epoch 48 loss: 2.2435 - acc: 0.3299 - val_loss: 2.2247 - val_acc: 0.3505

Model Summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_150 (Conv2D)          (None, 48, 48, 32)        896
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization_150 (Bat (None, 16, 16, 32)        128
_________________________________________________________________
dropout_152 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_151 (Conv2D)          (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 4, 4, 32)          0
_________________________________________________________________
```

```
batch_normalization_151 (Bat (None, 4, 4, 32)          128
_________________________________________________________________
dropout_153 (Dropout)        (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 2, 2, 64)          18496
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
batch_normalization_152 (Bat (None, 1, 1, 64)          256
_________________________________________________________________
dropout_154 (Dropout)        (None, 1, 1, 64)          0
_________________________________________________________________
flatten_28 (Flatten)         (None, 64)                0
_________________________________________________________________
dense_55 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_153 (Bat (None, 32)                128
_________________________________________________________________
dropout_155 (Dropout)        (None, 32)                0
_________________________________________________________________
dense_56 (Dense)             (None, 40)                1320
=================================================================
Total params: 32,680
Trainable params: 32,360
Non-trainable params: 320
_________________________________________________________________
```

Results & Conclusion

Since we are doing multiclass classification and our class size is large, we could not get high accuracies. Although convolutional neural networks generally perform good in image classification tasks, they are not enough for large classes. Also, our dataset size is not enough. We think that our models cannot see enough examples for classification from this dataset. Our models would work better with more data and better regularization techniques. Most articles say that going deeper than going wider gives best results. Even though we have experimented with too many models, we could not see this effect on our accuracy.

References

[1] D. H. T. Hien and D. H. T. Hien, "A guide to receptive field arithmetic for Convolutional Neural Networks," *Medium*, 05-Apr-2017. [Online]. Available: https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807. [Accessed: 05-Apr-2019].

[2] Stam, Deirdre Corcoran. "HOW ART HISTORIANS LOOK FOR INFORMATION." Art Documentation: Journal of the Art Libraries Society of North America, vol. 3, no. 4, 1984, pp. 117–119. JSTOR, www.jstor.org/stable/27947353.

[3]T. E. Lombardi. "The classification of style in fine-art painting". ETD Collection for Pace University. Paper AAI3189084., 2005.

[4]K. Sorokina, "Image Classification with Convolutional Neural Networks", Medium, 2019. [Online]. Available: https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8.