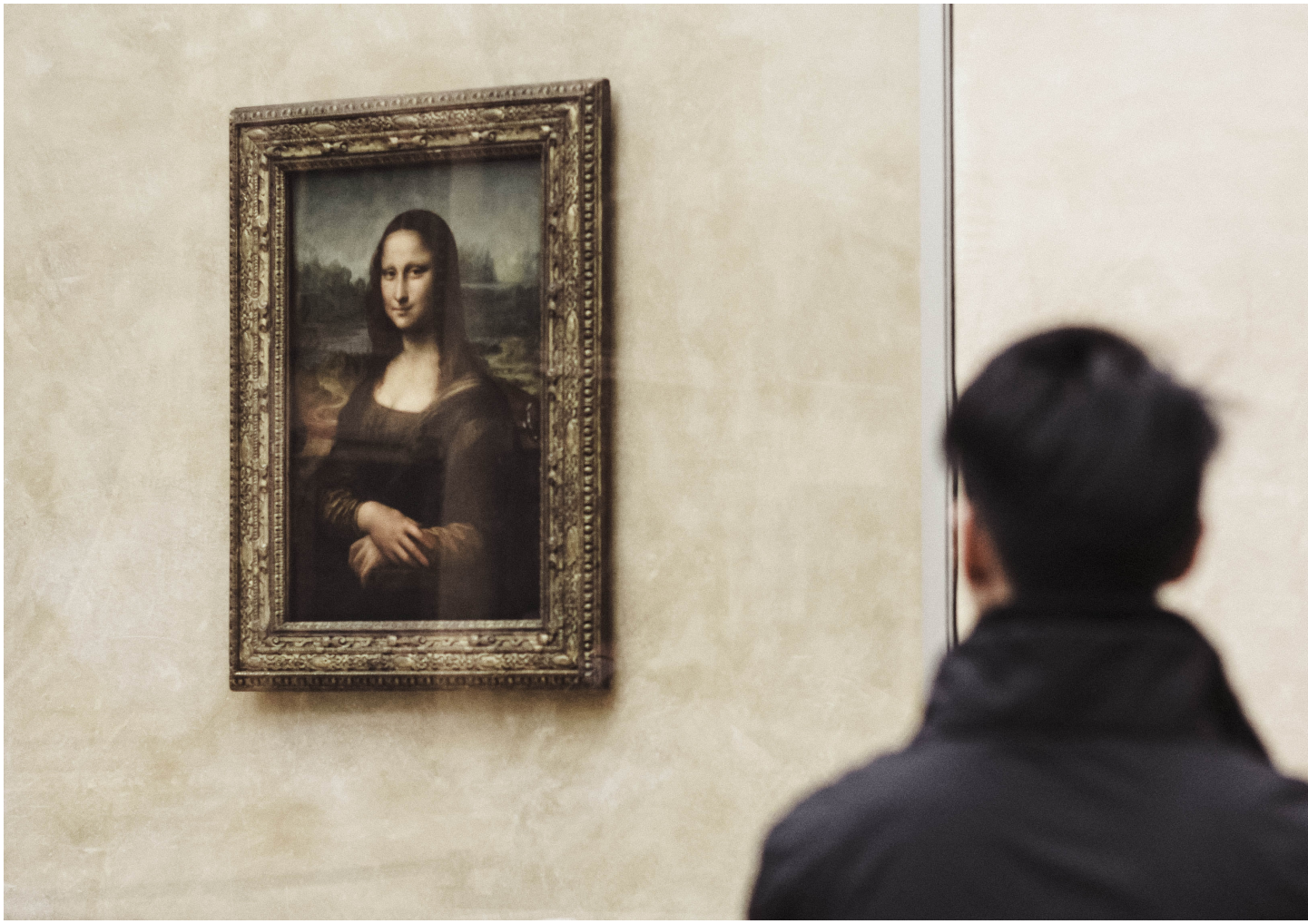
Matching The Art Work To The Artist.

Abhijit Krishna Menon
menon.ab@husky.neu.edu

April 12, 2020



Contents

1 Introduction

Artwork is one of the most abstract ways of expressing oneself. As someone who is not a huge fan of art or very appreciative of it, I have always wondered how people distinguish between the styles of various artists and painters. Given that image processing takes a lot of computation power, I felt that this project would be best suited to show the true change, higher and parallel computation brings about while training a Machine learning model.

2 Objective

The objectives of this project include:

- To understand more about the analysis of paintings, I decided to implement a Machine Learning model to predict what painting belongs to which artist.
- The further scope for this, is to try paintings by random artist and generate a score on how much their artwork is related to work of certain other famous artists that are a part of the data set that I am working with.

3 Methodology

During the process of this project I worked on trying to implement different neural network architectures and hyper parameters. I eventually settled for one, that was both computationally efficient as well as giving me a decent result.

Computation efficiency was important to me primarily because of the lack of computation results laid out to me. The discovery cluster kept timing out and my local computer was short of RAM to perform certain memory intensive functions.

My final model architecture was as below:

1. Resnet50 Baseline Model (Figure 1)
2. Flatten Layer
3. Dense Layer to 512 Edges.
4. Dense Layer to 16 Edges.
5. Output 1 with prediction of 11 classes probabilities.
6. Flatten Layer
7. Dense Layer to 512 Edges.
8. Dense Layer to 16 Edges.
9. Final output with prediction of 11 classes probabilities.

Now, to explain my architecture. The output from the baseline Resnet50 structure is fed to my following layers. One will notice 2 similar architectures back to back. The first architecture is used find the optimum Learning rate for training the model and I have initiated a reduction on learning rate every time there is a plateau in training. I run this chunk of the architecture for 10 epochs and find one optimum learning rate. I then use that learning rate and feed it into the second chunk, which I then run for approximately 50 epochs to then get final classification The number of epochs for each layer is something I kept switching around to see which gives me the best results while still remaining computationally efficient.

To run this model and to show comparisons between the following:

1. Discovery Cluster (Tesla K80 GPU)
2. Google Co-lab Pro (Tesla 40C)

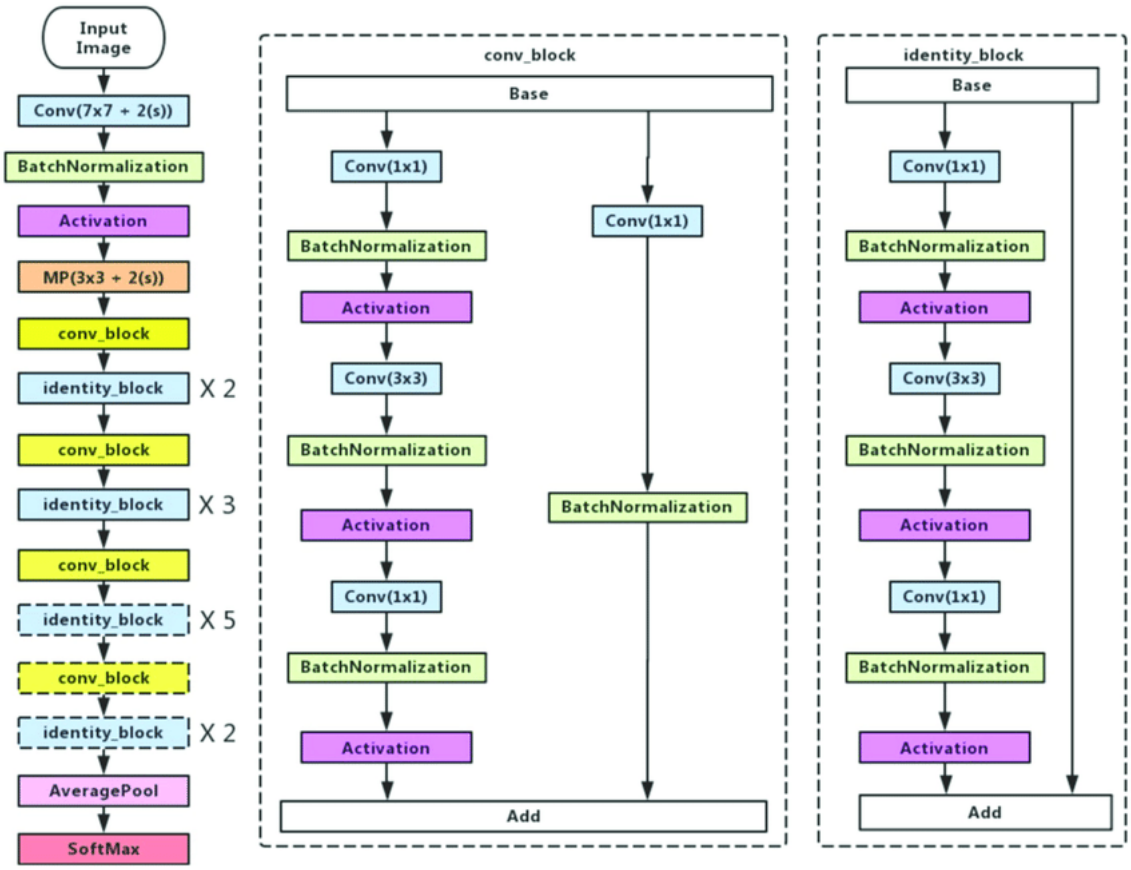


Figure 1: Resnet50 Architecture

3. Google Co-Lab Pro CPU
4. Mac Book Pro CPU(2.2 GHz 6-Core Intel Core i7)

I trained this model using the Keras framework running on a Tensorflow backend.

4 Dataset

- The source of the data is Best Art Work of All Time - Dataset from Kaggle.
- The size of the dataset is 2GB.
- This data set contains three files.
  - artists.csv: data set of information for each artist.
  - images.zip: collection of images (full size), divided in folders and sequentially numbered.
  - resized.zip: same collection but images have been resized and extracted from folder structure.

To begin pre-processing of data and building input data structure, I parsed through the images folder and got all the images of a particular artist from the artist's folder. Post getting labelled data, I used the artists.csv file to build a weights for each class on the basis of how many paintings of each artist is present in the given folders. The class weights are important because we do not want artists with more paintings to be getting predicted more often. We thus have normalise the weights for each class.
Post this, I pass the images directory through an ImageDataGenerator for image data preprocessing and normalisation. This included re-scaling the images. Zooming in by 0.7 to improve pixel density. I then built a training image generator and a validation image generator using the flow-from-directory

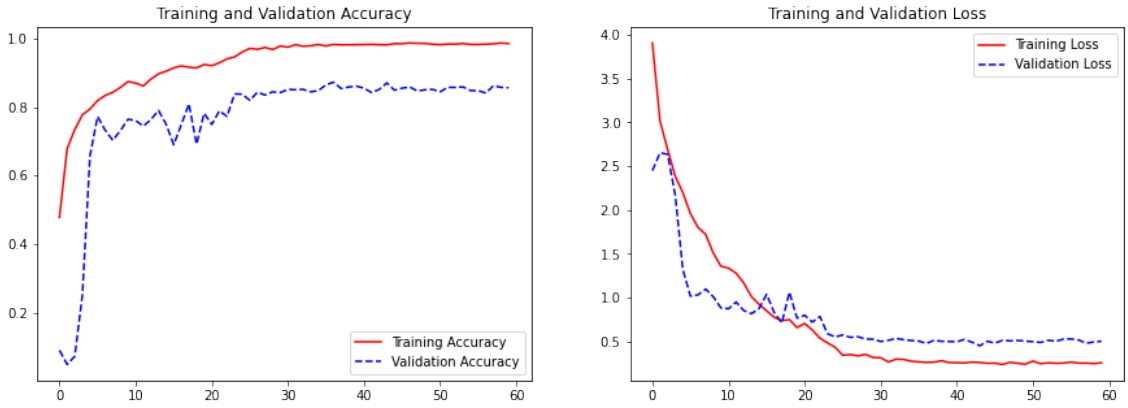


Figure 2: Training and Validation Cycle

function as part of the Tensorflow library.
I now had 3444 images for training split into 11 classes. I also had 855 images for the validation split split into 11 classes. For the training we have a batch size of 215 and for validation we have a batch size of 53.

5 Results and Analysis

My major factor in deciding to go with a particular architecture was the processing power I had to back running the particular structure. So while I initially built a more dense network, I realise that most of the processing I had was running out of memory.
This meant I started from a basic network and worked upwards to get to the best possible network that I could without runnning into memory issues.
Post the selection of the architecture I ran the model and generated predictions and generated and overall F1 score of 86% as my accuracy score.

The training and validation cycle of my model can be seen in the figure below.

The performance of my model across artists can be seen from the heatmap(Figure 3).

The prediction results for each artist can be seen from the table Figure 4 generated. We see the highest F1 score for the artist Albrecht Durer, followed by Marc Chagall and Edgar Degas.

Post the generation of results I tried 2 things.

1. Try prediction of the artist of random paintings from within my dataset. This can be seen in Figure 5
2. Picked a random painting from the internet of one of these painters and predicted which painter the painting is a closest match to. This can be seen in

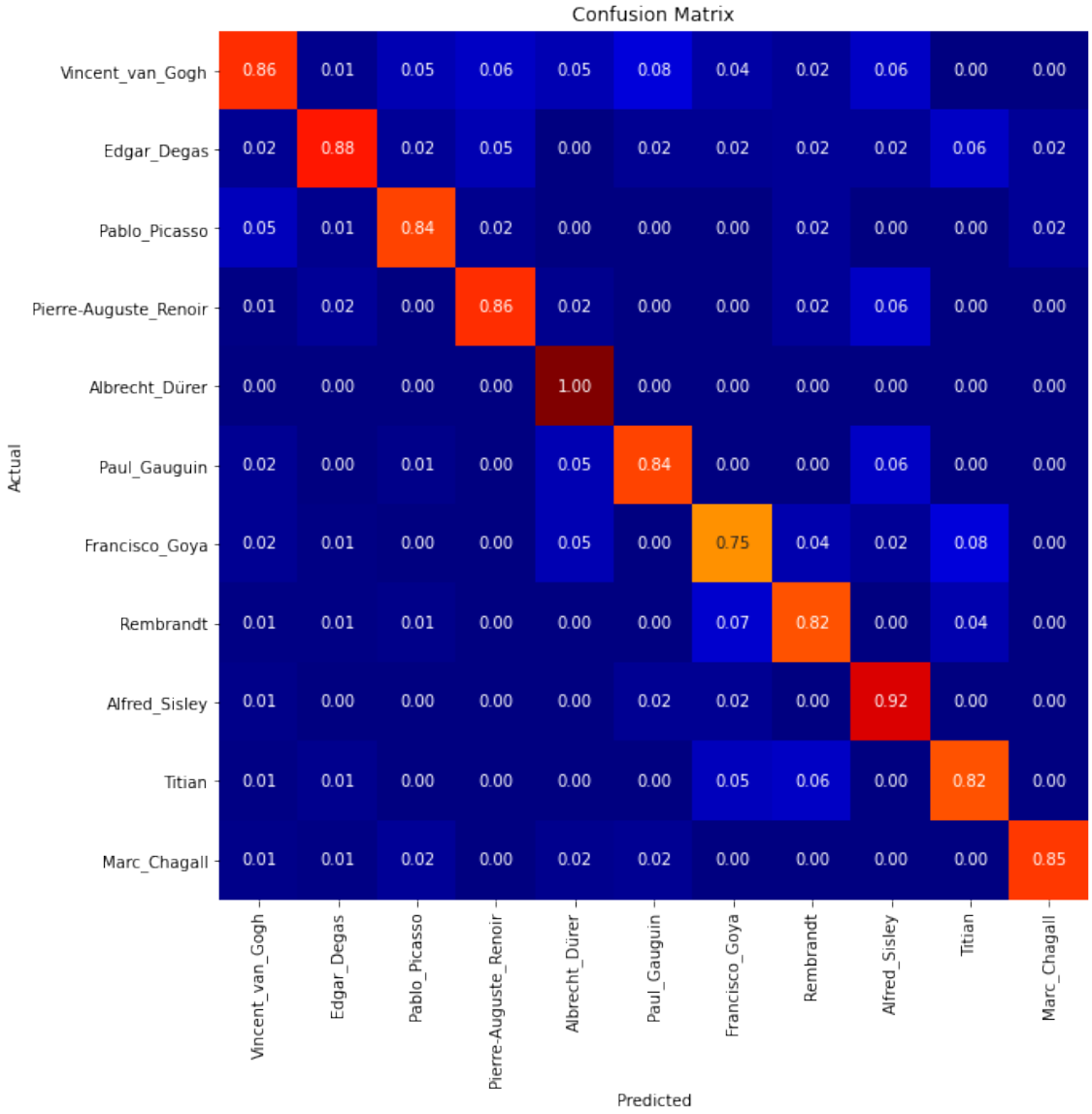


Figure 3: Output Heatmap

6 Hardware Performance Comparisons

In this section, I will go through the performance across the 4 different hardware specs that I used to train the model on. The hardware are as follows:

1. **Discovery Cluster (Tesla K80 GPU)**
   The Tesla K80 was a professional graphics card by NVIDIA, launched in November 2014. Built on the 28 nm process, and based on the GK210 graphics processor, in its GK210-885-A1 variant, the card supports DirectX 12. The GK210 graphics processor is a large chip with a die area of 561 mm and 7,100 million transistors. Tesla K80 combines two graphics processors to increase performance. It features 2496 shading units, 208 texture mapping units, and 48 ROPs, per GPU. NVIDIA has paired 24 GB GDDR5 memory with the Tesla K80, which are connected using a 384-bit memory interface per GPU (each GPU manages 12,288 MB). The GPU is operating at a frequency of 562 MHz, which can be boosted up to 824 MHz, memory is running at 1253 MHz.

   Being a dual-slot card, the NVIDIA Tesla K80 draws power from 1x 8-pin power connector, with power draw rated at 300 W maximum. This device has no display connectivity, as it is not designed to have monitors connected to it. Tesla K80 is connected to the rest of the system using

Classification Report:

	precision	recall	f1-score	support
Vincent_van_Gogh	0.86	0.86	0.86	174
Edgar_Degas	0.91	0.88	0.90	139
Pablo_Picasso	0.88	0.84	0.86	86
Pierre-Auguste_Renoir	0.88	0.86	0.87	65
Albrecht_Dürer	0.86	1.00	0.92	65
Paul_Gauguin	0.87	0.84	0.85	62
Francisco_Goya	0.80	0.75	0.77	57
Rembrandt	0.82	0.82	0.82	51
Alfred_Sisley	0.81	0.92	0.86	51
Titian	0.82	0.82	0.82	51
Marc_Chagall	0.95	0.85	0.90	47
accuracy			0.86	848
macro avg	0.86	0.86	0.86	848
weighted avg	0.86	0.86	0.86	848

Figure 4: Output Table

Figure 5: Internal Dataset Predictions

a PCI-Express 3.0 x16 interface. The card measures 267 mm in length, and features a dual-slot cooling solution.

2. **Google Co-lab Pro (Tesla 40C GPU)** The Tesla K40c was an enthusiast-class professional graphics card by NVIDIA, launched in October 2013. Built on the 28 nm process, and based on the GK180 graphics processor, in its GK180-890-A1 variant, the card supports DirectX 12. The GK180 graphics processor is a large chip with a die area of 561 mm and 7,080 million transistors. It features 2880 shading units, 240 texture mapping units, and 48 ROPs, . NVIDIA has paired 12 GB GDDR5 memory with the Tesla K40c, which are connected using a 384-bit memory interface. The GPU is operating at a frequency of 745 MHz, which can be boosted up to 876 MHz, memory is running at 1502 MHz.

   Being a dual-slot card, the NVIDIA Tesla K40c draws power from 1x 6-pin + 1x 8-pin power connector, with power draw rated at 245 W maximum. This device has no display connectivity, as it is not designed to have monitors connected to it. Tesla K40c is connected to the rest of the system using a PCI-Express 3.0 x16 interface. The card measures 267 mm in length, and features a dual-slot cooling solution. Its price at launch was 7699 US Dollars.

3. **Google Co-Lab Pro CPU** Hardware Specs:
   - 1xsingle core hyper threaded Xeon Processors @2.3Ghz i.e(1 core, 2 threads)

Figure 6: Predicted image from a url.
Predicted artist = Titian
Prediction probability = 72.94976711273193

- n1-highmem-2 instance
- 2vCPU @ 2.2GHz
- 13GB RAM
- 33GB Free Space
- maximum 24 hours

4. **Mac Book Pro CPU(2.2 GHz 6-Core Intel Core i7)** Hardware Specs:
   - 12.2 GHz 6-Core Intel Core i7(12 threads)
   - 16 GB 2400 MHz DDR4
   - 256 GB Free Space

Our final results from the 4 hardware specs were as follows:

Device	Time Taken for Training(Seconds)
Discovery Cluster(Tesla K80)	7711.00
Google CoLab Pro(Tesla 40C)	17547.00
Google CoLab Pro(CPU)	85652.00
Mac Book Pro(CPU)	66628.00

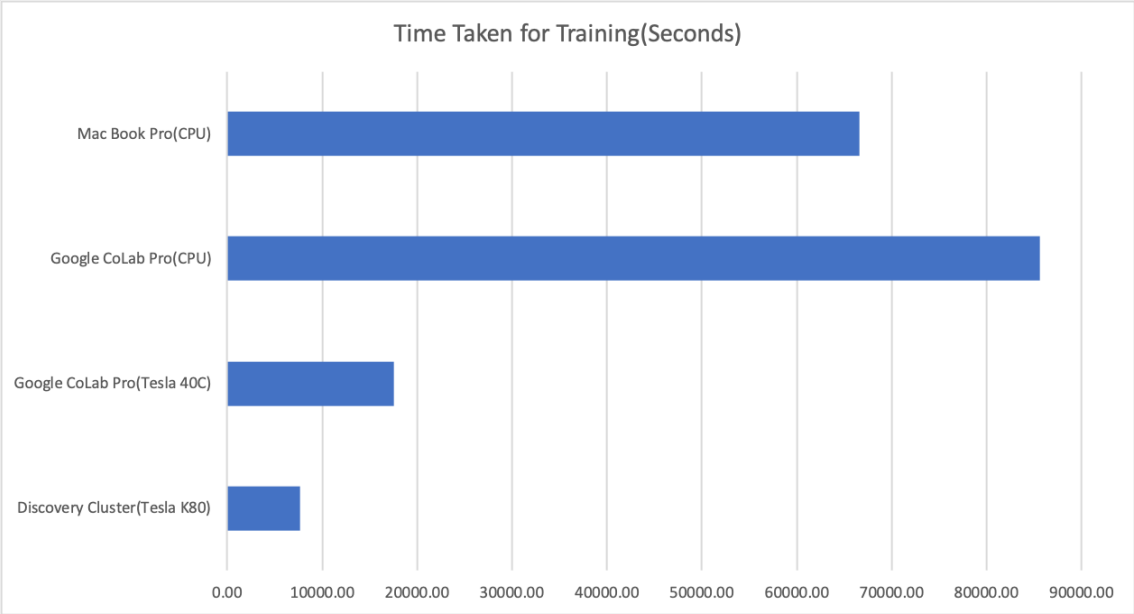


Figure 7: Hardware Performance Results

7 Conclusion

From the experiment performed to test the performance of these various hardware specifications on the training of the Machine Learning model in question, we can conclude that the Tesla 40C GPU provided by the Google CoLab Pro performs the most efficiently amongst all the 4 candidates.

We also conclude that we have got good results from the ML model that we have trained(86% F1 score). This however, could have been improved if the network was made more dense. If access to higher processing power was a given then this experiment could be tried in the future.

8 Future Scope for Improvement

The possible directions for future improvement of this project include:

1. Build a more dense model given better procesing power.
2. Train on a larger dataset consisting of more artists and their painting.
3. Build a web app and deploy the project.

9 Appendix

9.1 Important Code Chunks

```
batch_size = 16
train_input_shape = (224, 224, 3)
n_classes = artists_top.shape[0]

train_datagen = ImageDataGenerator(validation_split=0.2,
                                   rescale=1./255.,
                                   zoom_range=0.7,
                                  )

train_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="training",
                                                    shuffle=True,
                                                    classes=artists_top_name.tolist()
                                                   )

valid_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="validation",
                                                    shuffle=True,
                                                    classes=artists_top_name.tolist()
                                                   )

STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print("Total number of batches =", STEP_SIZE_TRAIN, "and", STEP_SIZE_VALID)
```

Listing 1: Image Preprocessing

```
start_train_time = datetime.now()

# Load pre-trained model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=
    train_input_shape)

for layer in base_model.layers:
    layer.trainable = True


# Add layers at the end
X = base_model.output
X = Flatten()(X)

X = Dense(512, kernel_initializer='he_uniform')(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

X = Dense(16, kernel_initializer='he_uniform')(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

output = Dense(n_classes, activation='softmax')(X)

model = Model(inputs=base_model.input, outputs=output)


optimizer = Adam(lr=0.0001)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

n_epoch = 10

```

```
early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1,
                           mode='auto') #restore_best_weights=True

reduce_lr = (monitor='val_loss', factor=0.1, patience=5,
                              verbose=1, mode='auto')

checkpoint_path = '/content/drive/My Drive/Artist to Artwork/checkpoints/cp.ckpt'
# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_best_only=True,
    period=1)


model.save_weights(checkpoint_path.format(epoch=0))

print("Training Model 1")
# Train the model - all layers
history1 = model.fit_generator(generator=train_generator, steps_per_epoch=
    STEP_SIZE_TRAIN,
                               validation_data=valid_generator, validation_steps=
    STEP_SIZE_VALID,
                               epochs=n_epoch,
                               shuffle=True,
                               verbose=1,
                               callbacks=[reduce_lr,cp_callback],
                               # use_multiprocessing=True,
                               # workers=32,
                               class_weight=class_weights
                              )



for layer in model.layers:
    layer.trainable = True

optimizer = Adam(lr=0.0001)

model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

print("Training Model 2")
n_epoch = 50
history2 = model.fit_generator(generator=train_generator, steps_per_epoch=
    STEP_SIZE_TRAIN,
                               validation_data=valid_generator, validation_steps=
    STEP_SIZE_VALID,
                               epochs=n_epoch,
                               shuffle=True,
                               verbose=1,
                               callbacks=[reduce_lr, early_stop,cp_callback],
                               # use_multiprocessing=True,
                               # workers=32,
                               class_weight=class_weights
                              )


#model.save_weights('model_checkpoint2')

train_end_time = datetime.now()

total_time = train_end_time - start_train_time

if tf.test.gpu_device_name():
    print(f"Time taken to train model with GPU --> {total_time}")
else:
    print(f"Time taken to train model with CPU --> {total_time}")
```

Listing 2: Modelling

```
from sklearn.metrics import *
import seaborn as sns

tick_labels = artists_top_name.tolist()

def showClassficationReport_Generator(model, valid_generator, STEP_SIZE_VALID):
    # Loop on each generator batch and predict
    y_pred, y_true = [], []
    for i in range(STEP_SIZE_VALID):
        (X,y) = next(valid_generator)
        y_pred.append(model.predict(X))
        y_true.append(y)

    # Create a flat list for y_true and y_pred
    y_pred = [subresult for result in y_pred for subresult in result]
    y_true = [subresult for result in y_true for subresult in result]

    # Update Truth vector based on argmax
    y_true = np.argmax(y_true, axis=1)
    y_true = np.asarray(y_true).ravel()

    # Update Prediction vector based on argmax
    y_pred = np.argmax(y_pred, axis=1)
    y_pred = np.asarray(y_pred).ravel()

    # Confusion Matrix
    fig, ax = plt.subplots(figsize=(10,10))
    conf_matrix = confusion_matrix(y_true, y_pred, labels=np.arange(n_classes))
    conf_matrix = conf_matrix/np.sum(conf_matrix, axis=1)
    sns.heatmap(conf_matrix, annot=True, fmt=".2f", square=True, cbar=False,
                cmap=plt.cm.jet, xticklabels=tick_labels, yticklabels=tick_labels,
                ax=ax)
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_title('Confusion Matrix')
    plt.show()

    print('Classification Report:')
    print(classification_report(y_true, y_pred, labels=np.arange(n_classes),
    target_names=artists_top_name.tolist()))

showClassficationReport_Generator(model, valid_generator, STEP_SIZE_VALID)
```

Listing 3: Result generation

9.2 Training on Different Hardware

10 References

- Resnet and it's variants Article
- Original author of base code that I modified.
- Comprehensive Guide to Convolutional Neural Networks

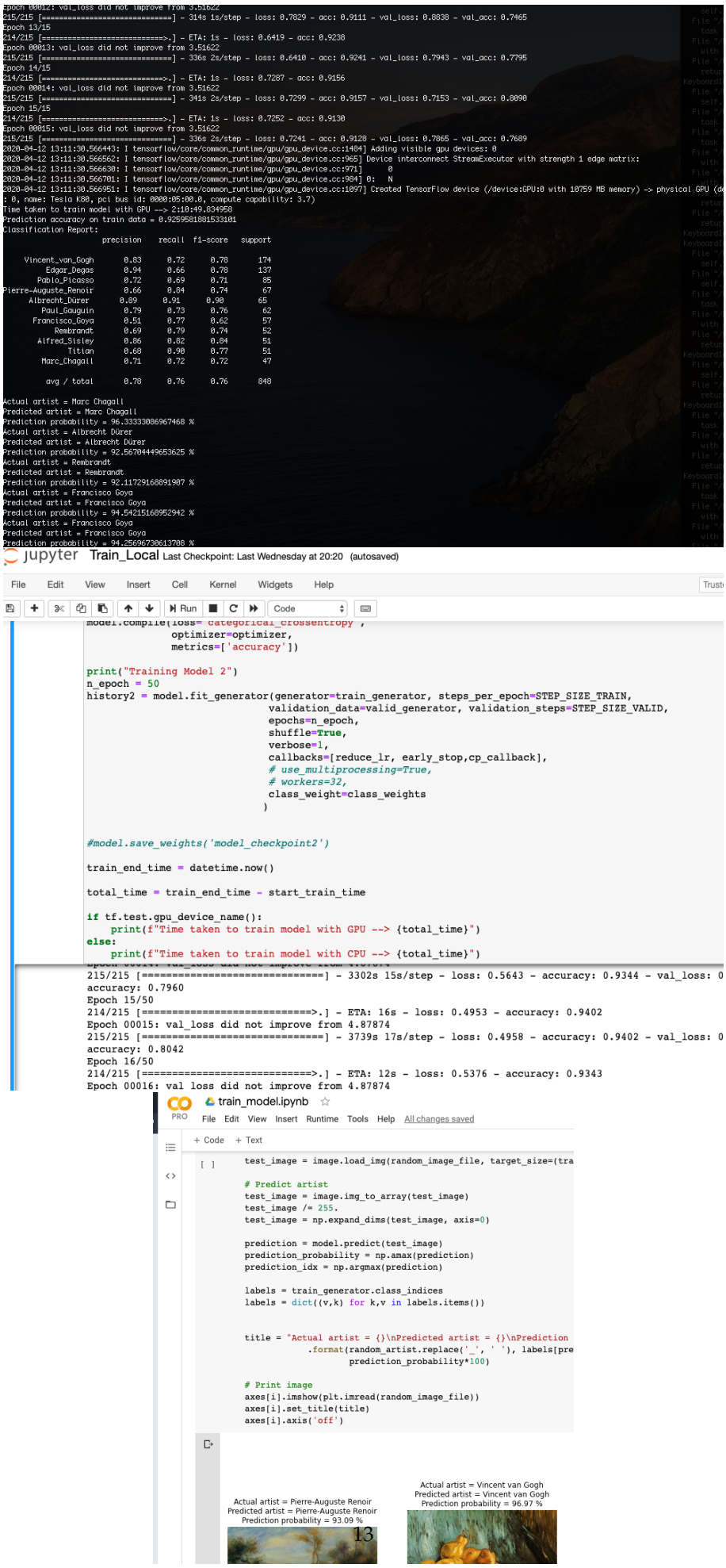


Figure 8: Running model on different hardware backends.