https://www.overleaf.com/project/617f27219290e9610de799e6

Kinship Software System

Justin Andal, Ameya Morajkar, Shikhir Goel

September 1, 2022

Abstract

For our project, we look to take an image of a group of people and determine the individuals related to each other. The first phase is to run our image through a object detector to segregate the subjects within the image into their own sub-image. Next we take a pair of the sub-images and input them into a Siamese Network. This would output whether the sub-images are related or not depending on how similar their feature vectors are. After comparing each sub-image with each other, we output a CSV of which sub-images are related or not.

1 Introduction

When looking at a group of people, sometimes we can determine who is related to who just based on looking at them. One of the most obvious ways to find out if people are related is by matching up facial features. For example, if two people have the same nose, eye structure, jawline, etc. we become more confident to say that these people are related or not. But how can we get a computer to see what we see? In this project, we look to explore this question by creating a Kinship Software System that would take an image that contains a group of people and determine who is related to who.

Currently online, all that is out there is people being able to take individual image of a person and have a CNN classify if they are related to another individual image of another person. This is good, but it only works on images with just one person within it. We look to expand that ability to take in images with groups of people and see who we can classify as related. We could just take an individual image of each person in the group photo and input them into the regular Kinship Classifier, but that can be more work. Having an image already containing a group of people, you can already make the next assumption on determining whether it is a family photo or not. Our system will comprise of first detecting each person in the photo, pair them with each person and determine whether a pair is related or not.

2 Procedure

2.1 Detection

For our project, we decided to use Microsoft's object detector, megadetector. It's a high quality object detector, created more specifically for camera trap images to detect wildlife. The reason we chose this type of detector was because they built their detector to handle animals, people and vehicles. The reason behind that was to know when there is noise within a camera trap data set, such that they can reduce their data set when images had only people or vehicles within them. But for our purposes, we look to focus on people within an image and ignore any other type of object. A more general object detector like yolo might detect too many objects within our photos so we wanted to use a more specific detector to get what is really important to us for our system. In Figure 1, you can see that megadetector was able to detect each person within the image and label that it has detected a person with a high confidence.

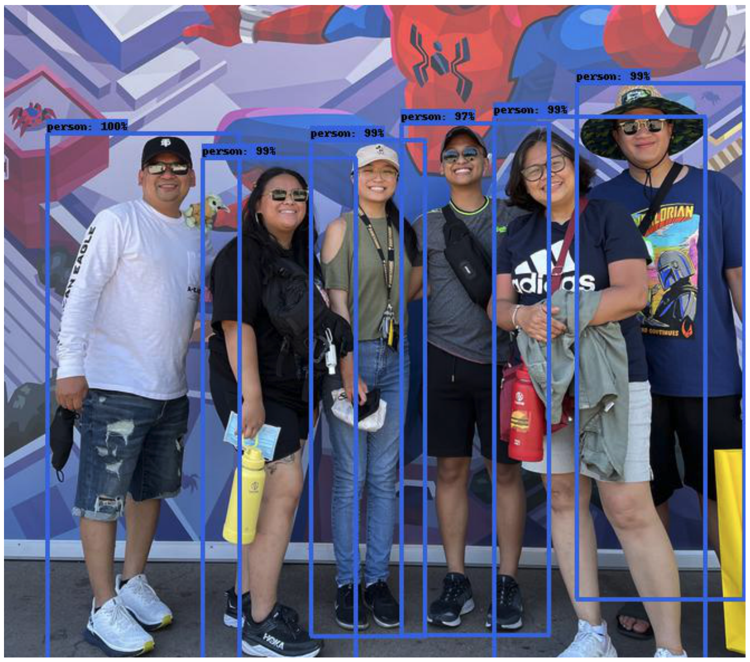
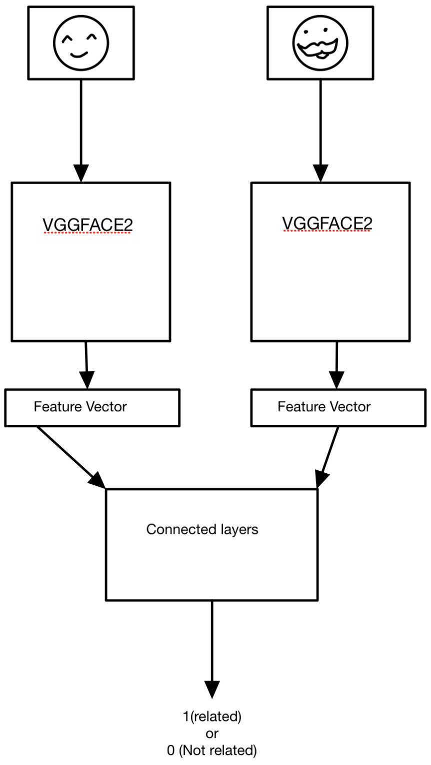


Figure 1: Example output image from MegaDetector, labeling each detection as a person

2.2 Kinship Classifier

In order to determine kinship between two people, we will be focusing on the similarity of facial features between them. How we can do that is by creating a Siamese Network that passes each image into a CNN and compares their outputted feature vectors together. If they come out similar, then the network will predict that they are related.



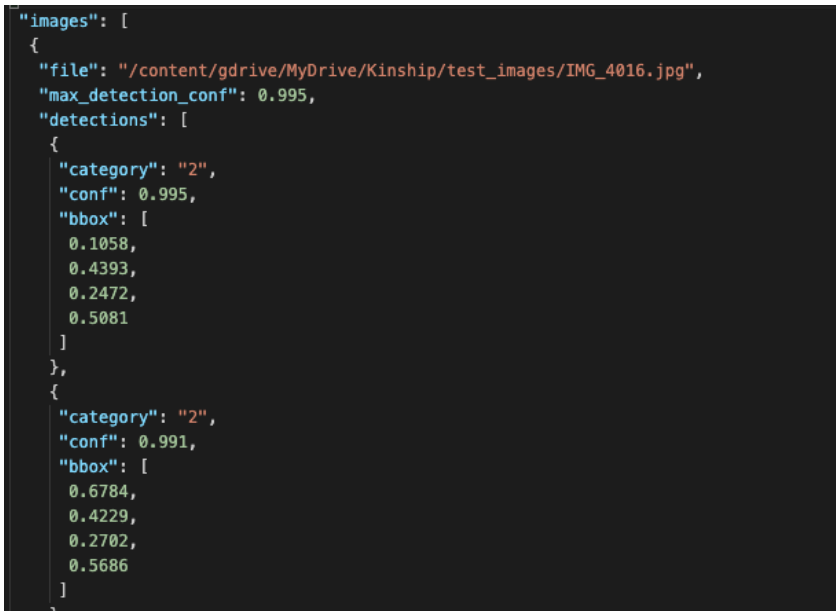
To speed up the training process, we use Transfer Learning to be able to extract features within our images. This is possible because pretrained models have already been trained to focus on extracting important features within an image. Luckily there is a pretrained model called VGGFace2. It has been trained on 3.3 million face images, making it useful in extracting important facial features.

2.2.1 Data set Definition

The data set we chose to use was a shared Kaggle dataset called **Families in the Wild: A Kinship Recognition Benchmark**. Because it has been used for competitions, we can trust that it's a clean dataset that works well in most cases. For our training data, its a folder that contains multiple subfolders, each representing a family. Inside each family subfolder then contains another set of subfolders, each containing multiple different images of each person that belongs in that family. The images are all squared images, focused on just the face of the person.

2.3 Kinship Software System

Putting it all together, our system will first take in a set of images that contain groups of people. Next we pass each image through megadetector, outputting a JSON file containing each detection for each photo. For each detection, there are coordinates where each detection is at within the image, shown in Figure 2. We then use those coordinates to crop out each person into their own individual image. Each image is then placed into a folder that will contain each person detected within that specific image.

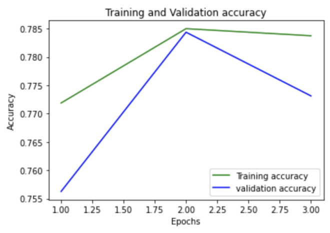


```
"images": [
 {
  "file": "/content/gdrive/MyDrive/Kinship/test_images/IMG_4016.jpg",
  "max_detection_conf": 0.995,
  "detections": [
   {
    "category": "2",
    "conf": 0.995,
    "bbox": [
     0.1058,
     0.4393,
     0.2472,
     0.5081
    ]
   },
   {
    "category": "2",
    "conf": 0.991,
    "bbox": [
     0.6784,
     0.4229,
     0.2702,
     0.5686
    ]
```

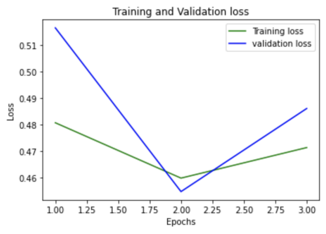
Figure 2: Example output JSON file containing the file and what detections were made in the image and what category the detection is (2 == person)

Next we need to create a CSV file to contain all pairing permutations of each person within a group photo. For example, if we have a photo containing 3 people (A,B and C), then there will be 6 different permutations between the 3 people. So we go through each group image folder, and pair each cropped image with each other, and place those pairings in a CSV sheet to be used when making our kinship predictions.

Now that we have each person paired up, we then take our trained kinship classifier, and pass each pairing into it and obtain a prediction. These predictions are then labeled next to it's corresponding pair, indicating whether they are related or not.



(a) Baseline Accuracy



(b) Baseline Loss

Figure 3: History graphs of how training performance of our baseline kinship classifier

3 Results and Analysis

For this system, we need to measure performance of our Kinship Classifier and the Kinship Software System separately. This is because even though our accuracy for a classifier itself gets better, it could make the system it self worse or not improve at all. First we will discuss baseline performance of each part and then talk about how we made it better in our ablation studies.

3.1 Baseline Kinship Classifier Performance

For our baseline model, we trained it for 10 epochs, with a batch size of 32, drop out ratio of .01, and steps per epoch of 100. This resulted in a validation accuracy of 78% and validation loss of 0.46.

In Figure 3 we can see the training performance for training and validation accuracy. The loss and accuracy trends also look reasonable to say that it is training in the right direction. Unfortunately our system crashed when training for 10 epochs and stopped at 7 epochs. Luckily we saved the best model weights and was able to plug it back into the model and train for 3 more epochs, as shown in Figure 3.

Our data did not include clear labels for the testing set of images so we based our performance on how well the validation set was. But having a validation accuracy at 78% was a pretty decent baseline for our model. Having the VGG pretrained model helped a lot to get this solid baseline accuracy.

3.2 Baseline Kinship Software System Performance

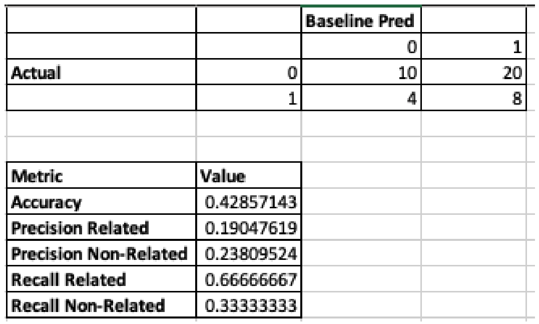
To test our system, we used two images, one containing 4 people and another image with 6 people. Both images contain people related to someone within the image. After we plugged in our baseline model into our system, we get the following confusion matrix and metrics shown in Figure 4. Interestingly, our accuracy was a very low (42%) given that our model finished with a validation accuracy of 78%.

Our thinking on why it is so low is because the images that we pass in to our model to predict kinship, includes more than just the face of the person. An individual photo includes the full body and even sometimes parts of another individual if they are standing really close to each other. This is a problem because our classifier is trained for face relationships, so by introducing more features than just the face, the classifier will potentially take in unnecessary features and use that to determine relationship. For example, maybe two people are wearing the same shirt, it could use that to predict as related even though they don't have similar facial features.

Another potential mishap could be due to having to reshape our image down because our model only takes 224x224 images. Most of our data is too big so we have to shrink them down to fit that shape which can cause us to loose some information.

In Figure 4, we can see that the model also contains way too many false positives, resulting in a precision of predicting related to be really low at 19%. This could be due to too many similar features between two people that don't include their facial features. For example, most people have two arms and legs so that could be contributing to thinking too many people to be related within an image.

In our ablation studies, we look to improve our model given the issues presented in our baseline.



		Baseline Pred	
		0	1
Actual	0	10	20
	1	4	8

Metric	**Value**
Accuracy	0.42857143
Precision Related	0.19047619
Precision Non-Related	0.23809524
Recall Related	0.66666667
Recall Non-Related	0.33333333

Figure 4: Confusion Matrix and Metrics of our system using our baseline model

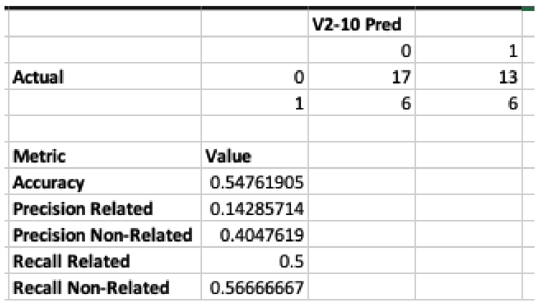
3.3 Ablation Studies

Given the baseline, it doesn't seem like we are doing too well in performance. The biggest issue is the inability to focus on just the face within our cropped images. One obvious solution is to crop just the face of each person. But given limited time to complete this project, we weren't able to execute this solution. Instead we look to try to generalize our classifier better through hyperparameter tuning in hopes that it could focus on just picking facial features given the whole figure of our subjects.

3.3.1 Hyperparameter Tuning

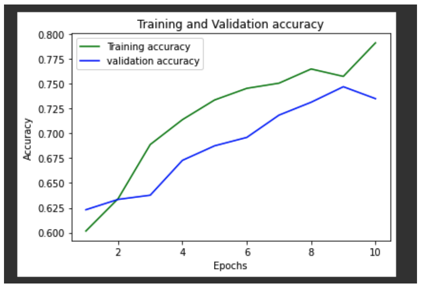
We want to best generalize our model such that it doesn't classify based on one or two specific things and become a classifier too specific to one thing. In the real world, data isn't clean so we want it to do better in real world scenarios. In order to do so, the parameters we will be looking to tune are batch size, epochs, and dropout ratio. To recap, our baseline had a batchsize=32, epochs=10, and dropout ratio=0.01.

First, our batch size might have been too big, causing it to wait too long before updating it's weights. So we halved it to a batchsize=16 because we want to feed in less images at each step in an epoch, such that we can update our weights more often which should help it generalize better. This will also be more efficient because we wont be overloading our GPU with big batches to train at each step in an epoch. The results came out 10% better with an accuracy of 54%, shown in Figure 5. It looks like the model learned to get less false positives but got more False negatives than the baseline. Interestingly the classifier ended with a validation accuracy of 74% so potentially it generalized better to handle our full body images. This is possibly due to the model having more time to update it's weights to generalize the face better. So although we have a lower validation accuracy, we actually have better weights to work with our type of images. We can also see in Figure 6, that the model hasn't overfitted yet so it's possible that there is still more training to be done.

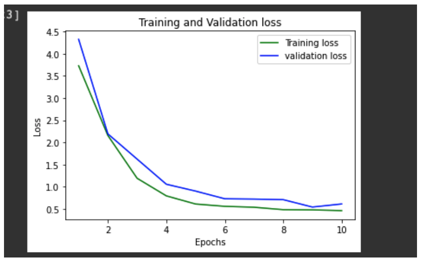


		V2-10 Pred	
		0	1
Actual	0	17	13
	1	6	6
Metric	**Value**		
Accuracy	0.54761905		
Precision Related	0.14285714		
Precision Non-Related	0.4047619		
Recall Related	0.5		
Recall Non-Related	0.56666667		

Figure 5: Confusion Matrix and Metrics of our system using our V2 model with Batch Size =16



(a) Accuracy with Batch Size = 16



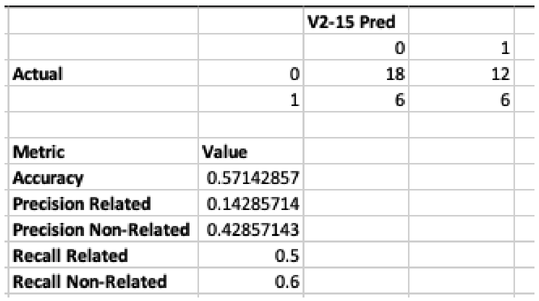
(b) Loss with Batch Size = 16

Figure 6: History graphs of how training performance of our classifier after changing to a batch size equal to 16

Knowing that, we increased our epochs to 15 and kept our batch size to 16. This caused our validation accuracy to go up to 75% and our system's accuracy to go up slightly to 57%, which is not a huge jump, as shown in Figure 7. We can also see that it improved on false positives but remained the same for false negatives. It's validation accuracy also didn't change much, as shown in Figure 8, so we decided to increase the epochs again to 18 because 20 caused our resources to crash due to low RAM. Interestingly, our performance went way down to 45% as shown in Figure 9. This could be due to it over training and giving us the same result as our baseline because it is now no longer generalizing. So this shows that our sweet spot is 15 epochs with a batch size of 16. Although the metrics aren't very good, this is our best result given these parameters.

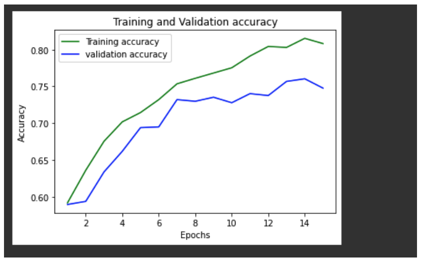
Next, we will try to improve upon this development and tune our drop out ratio. Drop out is when we randomly drop certain connection within our CNN. Our baseline rate was set to 0.01, meaning our model would set 1% of our inputs to zero. This would mean that we are less likely to drop a connection, therefore training with most of the inputs. So we look to increase the ratio to see if dropping more inputs would regularize the learning.

After increasing the dropout rate to 0.2, our accuracy went back down to 45%, as shown in Figure 10. This could be due to the model dropping too many connection. This tells us that we need to keep most of the weights in order for our classifier to pay attention to the face when given a full body image. Therefore,

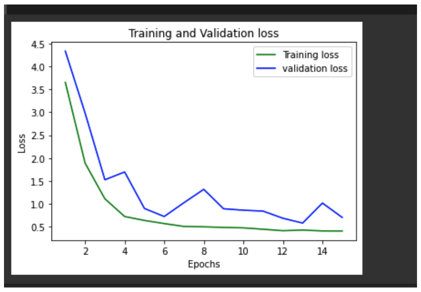


		V2-15 Pred	
		0	1
Actual	0	18	12
	1	6	6
Metric	**Value**		
Accuracy	0.57142857		
Precision Related	0.14285714		
Precision Non-Related	0.42857143		
Recall Related	0.5		
Recall Non-Related	0.6		

Figure 7: Confusion Matrix and Metrics of our system using our V2 model with Batch Size =16 and 15 epochs

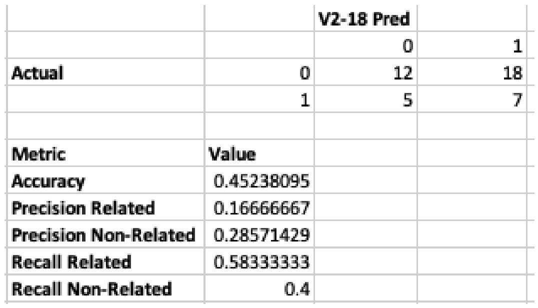


(a) Accuracy with Batch Size = 16 and Epochs = 15



(b) Loss with Batch Size = 16 and Epochs = 15

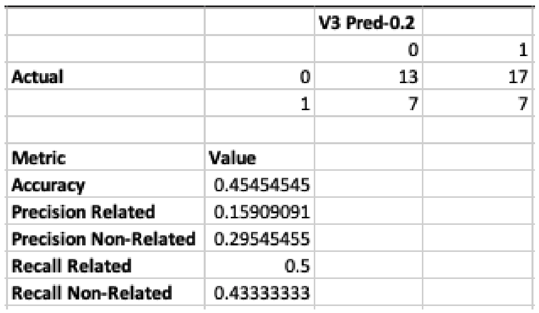
Figure 8: History graphs of how training performance of our classifier after changing to a batch size equal to 16 and epochs equal to 15



		V2-18 Pred	
		0	1
Actual	0	12	18
	1	5	7
Metric	Value		
Accuracy	0.45238095		
Precision Related	0.16666667		
Precision Non-Related	0.28571429		
Recall Related	0.58333333		
Recall Non-Related	0.4		

Figure 9: Confusion Matrix and Metrics of our system using our V2 model with Batch Size =16 and epochs at 18

in this case it is best to keep our dropout rate low given the circumstances of having to classify the full body and not just the individuals face.



		V3 Pred-0.2	
		0	1
Actual	0	13	17
	1	7	7
Metric	Value		
Accuracy	0.45454545		
Precision Related	0.15909091		
Precision Non-Related	0.29545455		
Recall Related	0.5		
Recall Non-Related	0.43333333		

Figure 10: Confusion Matrix and Metrics of our system using our V2 model with Batch Size =16, epochs = 18, dropout=0.2

4 Conclusion

Completing a full blown Deep Learning software system in 10 weeks is tough but not impossible. One of the most amazing thing about today's day in age, is that most of the tools we need are out there, and it's just our job to take those tools and put them together in a useful and meaningful way.

Although our system doesn't have the best performance, I believe we created a solid minimal viable product. Everything is designed in a modular way such that it is easy to remove and add parts that you'd need to make the system better. For example, one of the things I wanted to add was a gender and age classifier so that we can get better deduce the type of relationships within an image. So if our system classifies two people as related, we can then pass it through an age and gender classifier and see if one person is older female (call them A) and the other person is a younger male (call them B), then we can potentially say A is B's mother, and B is A's son. Obviously it's not that easy because what if A was a grandmother or an auntie, etc. But what's important is that we have created a foundation to create more possibilities of getting more meaningful information out of an image containing groups of people.

Our biggest hurdle was resources not being compatible or fast enough to get us our results back. For example, we wanted to apply data augmentation, but because of slow resources and limited time, we weren't able to pre-process our data to be augmented and add variety to the model's learning. But we have something that is workable and able to be improved on.

5 Project Submission Files

In the project folder you should expect to contain the following files:

- **Kinship-software-system.ipynb** : The final project AI software system
- **Final-VGGFace.ipynb** : The notebook to train the Kinship Classifier
- **Final-data-analysis.csv** : CSV sheet containing all the data analysis from the hyperparameter tuning
- **software-test-images** : Folder containing two images that can be used as input to the software system
- **Kinship** : Folder containing an input folder with the train and test images for the classifier

References

- Mansar Youness. Deep Neural Networks for Kinship prediction using face photos. May 16, 2019: https://towardsdatascience.com/deep-neural-networks-for-kinship-prediction-using-face-photos-f2ad9ab53834
- Github repo Kinship Classifier Script. September 3, 2019:

  https://github.com/bhargavasatyamani/kaggle-competitions/blob/master/Recognizing
- MegaDetector Github Repo:

  https://github.com/microsoft/CameraTraps/blob/master/megadetector.md
- Jason Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. December 3, 2018:

  https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/