Actor Identification with Deep Learning

Alexander Gabourie (gabourie@stanford.edu), Connor McClellan (cjmcc@stanford.edu)
Github: https://github.com/cjmcclellan/CS230DeepActor.git

1. Motivation/Introduction:

Prolific actors are generally easy to recognize in movies/tv shows, but it is often the case in which an actor feels familiar, but you just cannot put a name to the face. This circumstance leaves the viewer with the option to either guess or pull out their phone, laptop, etc. and look up who the actor is. While guessing can be fun, this process certainly detracts from the viewing experience. In this project, we aim to quickly satisfy the curious by automating the actor lookup process by using deep learning and tracking algorithms. The ultimate goal of this project is to extend those learnings to real-time actor identification with the click of a button. However, for this course, the goal is to address a modest part of this project.

2. Project Description/Objectives:

To effectively identify actors in movies/tv shows in real-time, a series of processing steps must be executed: face detection, face identification, and object tracking. Face detection uses an algorithm which scans images, from coarse-to-fine, and looks for facial features. This creates bounding boxes which captures actor's faces. An example of this can be seen in the green boxes of Fig. 1. With face segments of the current frame, we can encode each face using an Inception-ResNet Convolutional Neural Network (CNN) which has been pre-trained on a face database. These encodings can then be passed through face identification algorithms which are trained on either sets of actor's faces or character specific images taken from movies or tv shows. Since an actor and their character information is easily linked, a positive actor identification allows us to label both sets of information on-screen as seen by the text in Fig. 1. Lastly, since the process of face detection and identification is expensive, object tracking algorithms can be used to follow a character until a camera change. Object tracking is a crucial component if this process is to run in real time.

Fig. 1: Target output of completed project. The green boxes contain faces, the name above each box is the character, and below is the actor.

To implement the process described above would be a large undertaking and so we aim to focus on the face identification part of the project. The face detection models have already been described by R. Aljundi *et al* in [1] and implemented by D. Sandberg in [2]. We use their algorithms for that part of the project. Since object tracking is not a primary objective of the course project, we did not address this part of the project and it will not be included in this report.

3. Dataset/Tooling:

Much of the following work is based on Schroff *et al*'s face recognizer work in [3] implemented in TensorFlow by D. Sandberg in [2]. The package, called FaceNet, has trained an Inception-ResNet network (V1) in [4] using the CASIA-WebFace [5] dataset for facial embedding extraction and a Multi-task CNN [1] using the WIDER FACE [6] dataset for face detection. We use trained weights for both networks in our project. Additionally, we use PyTorch [7] and Scikit-learn [8] to create identification algorithms based off the embeddings from the Inception-ResNet network.

For the actor identification process, we were required to build our own dataset. We sourced our images from Google Images as it returned the most relevant images based on a query and there are programming interfaces to expedite the process. Original queries were of the form '<Character> <Movie> <Year> <Actor>'. Since actors are generally dressed in costume when in character, we queried character names, along with movie titles for context. We also included the year as some movies have been remade

under the same name. Unfortunately, despite many query combinations, we ran into problems. For example, if the movie title is the name of a character, the results are poor (i.e. character: Mugatu, movie: Zoolander). Ultimately, our datasets still came from Google Images, with a mix of images of actors both in and out of character, but a manual refinement of the returned results was required. Manual data filtering was slow and resulted in smaller than desired datasets, likely hurting the performance of our deep learning models.

Another consequence of building the database, in part, manually was it limited the number of characters we aimed to identify. All of our results are based on the movie 'The Ridiculous 6' which has dozens of people in the cast. We decided to focus on the classification of only seven characters. This meant that our test set, which is comprised of screenshots from the movie, was selected from scenes with containing only the characters we trained on. The movie-specific classifiers (to be discussed) cannot classify everyone in that movie.

4. Approach/Results:

The following subsections will cover the face detection and face embedding neural networks, how they work, and how we used them. Following that, we will discuss the three different ways we classified and identified seven characters from 'The Ridiculous 6': 1. Support Vector Machines, 2. Softmax classification trained with the cross-entropy loss, and 3. Siamese network trained with triplet loss.

4.1. Pre-designed Networks:

A. Face Detection – Multi-task CNN:

The face detection portion of the project is handled by a multi-task cascaded CNN (MTCNN) design [1] that incorporates three CNN stages trained on the WIDER FACE database [6]. The MTCNN design is built upon a coarse-to-fine detection approach. Before passing an image to the first CNN stage, it is resized many times to create an 'image pyramid' as seen in Fig. 2. The first CNN stage is the Proposal CNN (P-Net) which quickly scans the image for possible candidates. This is followed by a Refine CNN (R-Net) to reject false positives and calibrate the bounding box. Lastly, an Output CNN (O-Net) further prunes the possible candidates and identifies the eyes, nose, and corners of the mouth on the face. This result can be seen under the 'Finalize Boxes' label in Fig. 2. This cascading CNN structure decreases the face detection time over other CNN architectures making it ideal for real-time the real-time actor detection application.

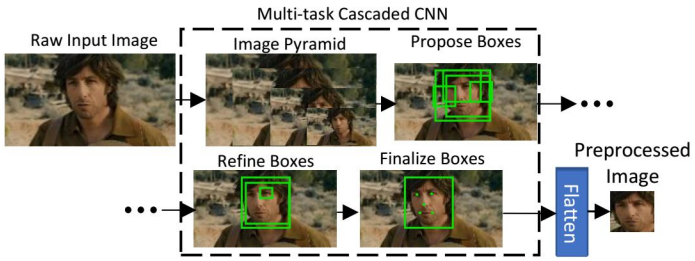

Fig. 2: Datapath diagram of the MTCNN showing image resizing, bounding boxes proposals (P-Net), refinement (R-Net), and finalization (O-Net).

Using the code from FaceNet [2], we set up scripts to run the MTCNN. Overall, the MTCNN works well on most photos containing faces. Various poses and skin pigmentation are easily recognized as seen in Fig. 3 (a)-(c). There are some circumstances where the model struggles, although rightfully so. We found partially obscured faces (Fig. 3 (d)) to cause trouble at times. Low-resolution faces, as in Fig. 3 (e) are also difficult to recognize, which will be a problem if characters on-screen are far away from the camera. Lastly, the MTCNN sometimes finds false positives in pictures of shrubbery or mountains (not shown), although not often.

B. Face Embedding:

With the faces found with the MTCNN, we extract the face embeddings with an Inception-ResNet CNN implemented in FaceNet [2]. Before passing a face image to the network, we flatten the image to a standard size (160×160×3). The Inception-ResNet is pre-trained, using a softmax loss function, on 494,414 images with 10,575 unique identities and images of size 453×453×3 provided in the CASIA-WebFace database [5]. The Inception-ResNet architecture itself is quite complex and will not be shown here, however, you can find diagrams of FaceNet's implementation by referring to the V1 architecture in reference [4]. With this pre-trained network, we pass flattened face images through the network and extract the output from the embedding tensor. This face embedding is a 128-dimesional representation of the face data. For the course project, we are not concerned with real-time processing so we precompute all embeddings for images in our training, validation, and test sets. By doing this, we can save time on training and evaluation as we only need to compute/backpropagate through a small classification network.

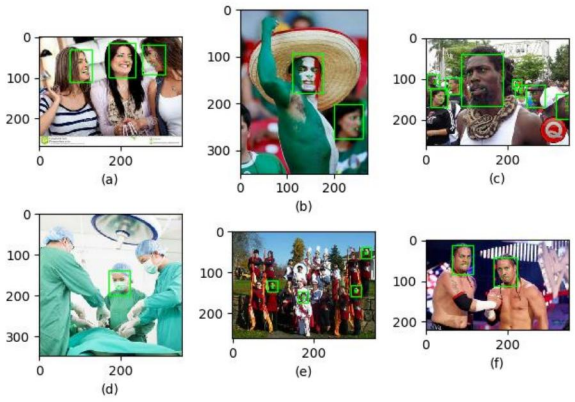

Fig. 3: (a)-(f) shows outputs from face detection model. Bounding boxes and points labeling the eyes, nose, and corners of the mouth are displayed.

4.2 Movie Specific Models:

Movie specific models refers to classifiers that must be trained on a movie-by-movie basis. Here we choose two models to evaluate: a support vector classifier (baseline/reference model), and a neural network with a softmax output layer. These models can be trained to label an arbitrary number of classes, but their effectiveness decreases, and training difficulty increases with the number of classes. Because of this, if a classifier only has to worry about the actors in a single movie, then the number of classes can stay small, and the model accuracies will improve. This also allows for each classifier to train based on the costume an actor might be wearing (i.e. wig, mustache, prosthetics, etc.) which might confuse more general classifiers. However, this requires a training set and a set of trained parameters for every title. The following two methods have been trained using a set of 551 examples with ~50-100 examples for each of the seven characters we try to classify in 'The Ridiculous 6.' These images are taken from Google and preprocessed using methods described above. The test set consists of 84 examples taken from screenshots of the movie and preprocessed.

A. Support Vector Classification (SVC):

While not a neural network, we wanted to include a classic machine learning algorithm as a benchmark for classification. For this we use the Scikit-learn [8] implementation with a linear kernel. Multiclass support is handled by a one-vs-one scheme and the implementation is more than quadratic with the number of samples meaning our relatively small dataset and number of classes work well for this algorithm. Default model parameters were used in training and the resulting model tested with an accuracy of 95.2% (80/84).

B. Softmax:

Softmax is a natural extension of logistic regression to multiple classes and is well suited for actor classification. Since our softmax layer outputs seven units, one for each character to classify, we must transform the 128-dimension face embedding to a 7-dimensional vector. The network that gave the best validation results is shown in Fig. 4.

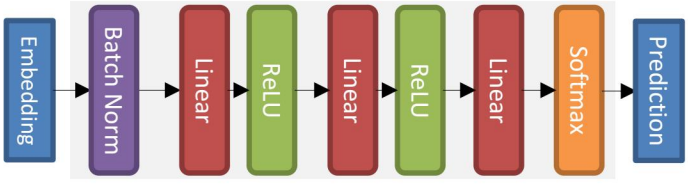

Fig. 4: Architecture of neural network with softmax output layer and two hidden layers.

In PyTorch, though evaluations yield the same classifications as a traditional softmax layer, we use the LogSoftmax layer which pairs nicely with the negative log likelihood loss. The loss function is as follows:

$$Loss(z, c) = \frac{-\log(e^{z[c]})}{\sum_j e^{z[j]}}$$

where z is the softmax output vector and c is the class indes. The equation above is simply the negated log of the softmax function. This is what PyTorch defines as the cross-entropy loss. We found that two hidden layers gave the best results, although deeper networks did not severely degrade performance. The linear layers, from left to right, have 256, 128, and 7 hidden units and weights were initialized with Xavier initialization using a normal distribution [9]. We also found that batch norm was only effective when applied to the input embeddings. For batch norm to work, we trained on minbatches each with 32 random training examples (no examples repeat per epoch and no minibatches match between epochs). The implementation of batch norm and minibatches drastically reduced training and validation loss. We also used a decaying learning rate where $\alpha_0 = 1.0\times10^{-4}$ and $\alpha := 0.99\alpha$ every 10 epochs.

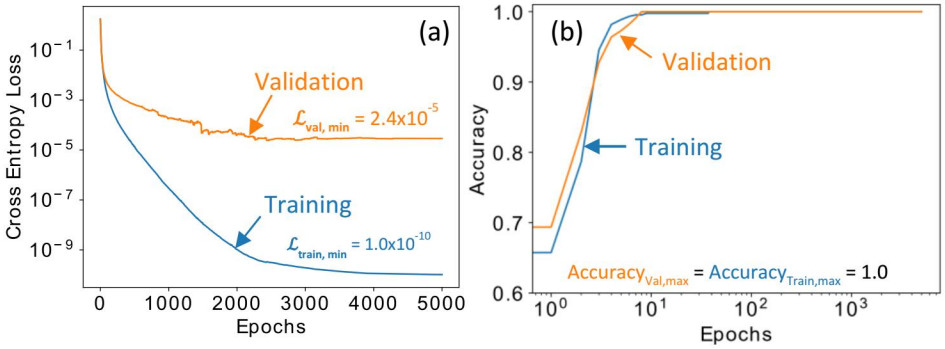

Fig. 5: (a) Cross entropy loss for the validation and training set of softmax network. (b) Prediction accuracy for the same network.

The results of our training can be seen in Fig. 5. In Fig. 5 (a), we can see the training loss continue to decrease as the validation loss levels out suggesting that we may be overfitting. After retraining with many different λ values for L2 regularization, we found that L2 regularization greatly increased both the training and validation loss. Since the validation loss stops improving around 2500 epochs, we decided to use the model trained to the 2500th epoch and used early stopping as a form of regularization.

With all hyperparameters tuned and all weights learned, we tested the softmax network on the test set. The resulting accuracy was 92.9% (78/84). While this is worse than the SVC, we might expect more training data to improve this classifier more than the SVC because it is a neural network. The incorrect predictions for both classifiers can be seen in Fig. 6. We can see that most errors are on either dark images or low-resolution photos. For those cases, the classifiers may not be at fault as the embeddings from the Inception-ResNet may be poor. The softmax classifier also incorrectly classifies Will Forte (bottom red squares), which might suggest that more training data is needed for his character.

Fig. 6: Incorrect predictions using softmax classifier (red), SVC (blue), or both (purple)

4.3 General Identification Models

A. Triplet Loss:

Movie specific models, discussed above, require a separate dataset and trained model for every movie. However, a general face ID model, which works for all movies is also of interest. Here we implemented a face classifier training on the triplet loss, which follows Schroff *et al*'s approach in his Facenet paper [3]. By using triplet loss, our model can identify faces by comparing an unknown face with reference faces to determine likeness and then label the input with the ID of the reference most similar. As a result, we only need one reference face for each actor in the movie to compare with unknown faces, making this model easier to apply to our actor ID application. However, triplet loss requires a training database of triplets (anchor, positive, and negative example faces) which are more difficult to construct than the databases used

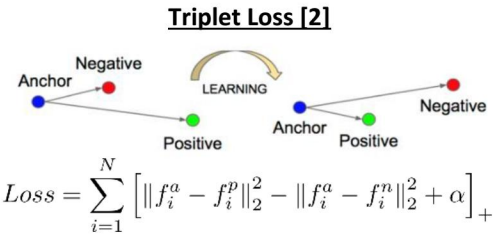

Fig. 7: Triplet loss from [ref]. Training is improved over a Siamese loss by pushing the anchor embedding away from the negative

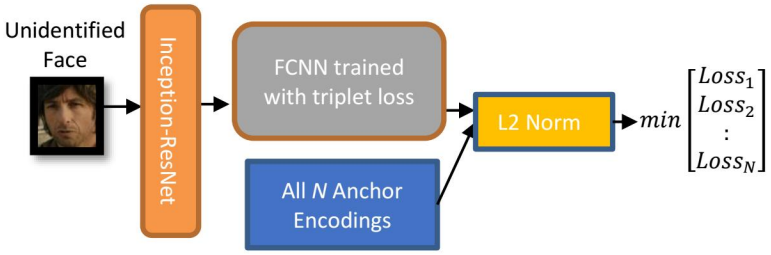

Fig. 8: Schematic showing face detection using a model trained on triplet loss. Processing is sped up by pre-compiling the encodings of all anchor reference faces

for the softmax classifier. By programmatically searching Google Images, we were able to compile 250 different actors faces for triplet loss training.

Fig. 7 displays the triplet loss concept for face recognition from [3] as an extension of a Siamese network by adding a negative face example to push the model to map similar faces and differentiate different faces. Selecting the anchor, positive, and negative face triplet training examples is important, as the negative example should be like the anchor for effective training. We used a gender ID script to predict gender based on actor name [10] to pair negatives and anchors with the same gender to improve training.

Using the pretrained model from [2] for computing face embeddings, we then trained a fully-connected NN similar to Fig. 4, but no softmax layer, using triplet loss. Once trained, the triplet-trained model can be evaluated on accuracy by taking unknown faces from the train or validation set and comparing the computed encoding with all known actors faces using L2 norm and taking the min loss as the predicted actor (Fig. 8). The loss and accuracy results per epoch are plotted in Fig. 9, showing an increase in validation and training accuracy over simply using the embeddings from the pretrained Inception-ResNet. We found that tuning the margin hyperparameter (α, Fig. 7) had a large effect on the performance of our model with a margin of 12 being optimal.

We then tested our model on the same data as the SVC and softmax models, achieving 88% accuracy, less than the SVC and softmax. We attribute the low accuracy to a lack of training data, as creating the triplet examples proved time consuming. However, we only used one reference image for each of the 7 actors in the test set, requiring less actor face images per movie than the SVC and softmax models. Therefore, with a larger training set, the triplet loss trained model could perform as well as the other models while requiring less data collection per movie.

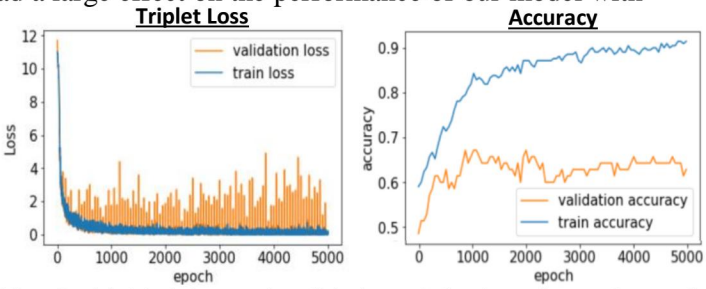

Fig. 9: (a) Training and validation triplet loss for each epoch (average over all minibatches). (b) Training and validation accuracy for each epoch. Eventually, the validation accuracy stops improving but training accuracy continues to improve, suggesting more training data is needed to improve model performance.

5. Conclusion:

In this project, we successfully implemented a face detection scheme for identifying actors in movies/tv shows. We used pre-trained models for face segmentation and face embedding, then applied transfer learning for developing models to identify actors. We found that SVC and softmax models trained for each movie were fairly accurate (95%-93%), while a general ID model trained using triplet loss was less accurate (88%) but required less data from each movie. Future work will focus on creating better data collection for better training results, connecting all networks for a complete video to actor ID solution, and implementing an object tracking algorithm for real-time actor identification.

6. Contributions:

A. Gabourie developed the database builder, SVC and softmax models. C. McClellan developed the triplet loss training. Both set up the MTCNN and Inception-ResNet models and worked on the reports/poster.

7. References:

[1] K. Zhang, Z. Zhang, Z. Li, Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks", *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499-1503, Oct 2016.

[2] D. Sandberg, 'FaceNet,' (2017), *GitHub*, https://github.com/davidsandberg/facenet

[3] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering." *Proceedings of the IEEE conference on computer vision and pattern recognition*. (2015)

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi, "Inception-V4 inception-ResNet and the impact of residual connections on learning", *Proc. AAAI*, pp. 1-3, 2017.

[5] Yi, Dong, *et al.* "Learning face representation from scratch." *arXiv preprint arXiv:1411.7923* (2014).

[6] S. Yang, P. Luo, C. C. Loy, and X. Tang, "WIDER FACE: A Face detection benchmark." arXiv:1511.0652 (2015).

[7] A. Paszke, *et al*, 'PyTorch,' (2017), *GitHub*, https://github.com/pytorch/pytorch

[8] F. Pedregosa, *et al,* "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[9] X. Glorot, Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", *Proc. AISTATS*, pp. 249-256, 2010.

[10] I. S. Pérez, 'Gender Guesser,' (2018), *GitHub*, https://github.com/lead-ratings/gender-guesser