STAT 542 — Statistical Learning — Spring 2024

University of Illinois Urbana-Champaign

Midterm Project Report

Face Recognition Using CNN & ResNet50

By Group 6: Rosin Gu, Gustavo Nino Chaparro, Mingzhan Yang, Shannon Ooi, Yixuan Deng, Yuchen Wang, Yaxin Yang

Professor Jingbo Liu

March 29, 2024

Section 1: Machine Learning Task and Goal

Throughout their lives, individuals may be found in various locations, adorned with diverse accessories such as glasses, spanning different ages, and exhibiting varying hair colors. Because of this, facial recognition technology has become widely used. Facial recognition technology is pivotal for secure authentication and access control across various devices like smartphones and computers. It extends to granting secure entry into buildings and aiding law enforcement in identifying suspects or missing persons. This technology enhances public safety by facilitating criminal investigations and tracking down individuals, making it imperative to verify identities by comparing current images with past records.

We utilize an extensive dataset containing images of 31 renowned Hollywood celebrities from Kaggle. These images capture various scenarios, showcasing each celebrity in various attire, accessories, and environmental contexts, under diverse lighting conditions. Our dataset comprises an average of 82 meticulously curated photos per celebrity, resulting in a substantial collection of 2,562 images. Additionally, to ensure uniformity in image processing, we resize all images to a standard size of 224x224 pixels. To improve computational efficiency and promote the exploration of stochastic solutions, we organize the dataset into 32 randomly selected batches.

To facilitate recognition, our objective is to identify key facial features. This project aims to use machine learning algorithms to extract facial features to correctly identify individuals. The two machine learning algorithms used in this project are convolutional neural network (CNN) and Residual Neural Network (ResNet). Comparison statistics (accuracy, type I error, etc.) will be used to understand the performance of both algorithms. Our goal is to understand which algorithm performs better in our setting.

Note that a GitHub Link will be included in the appendix as reference of the code for cleaning, CNN and ResNet.

Section 2: Methodology

We aim to use both CNN and ResNet architectures to classify our facial data. CNN is a class of deep neural networks consisting of convolutional, pooling, and fully connected layers. Convolutional layers are the fundamental building blocks in CNN. In a single convolutional layer, a kernel or filter defined on some small region of the given input is applied to the input through a sliding window and generates a feature map after that. This allows CNN to extract spatio-temporal features from the input. Pooling layers are usually inserted between consecutive convolutional layers, which reduce the number of learnable parameters while keeping significant features. At the end of a CNN, a fully connected layer integrates all the extracted features to generate an output.

Adding more layers could significantly increase the accuracy but at the same time encountering the vanishing gradients problem, whereby gradients used to update networks become extremely small or reduced to zero as they backpropagate from the output layers to the earlier layers. This would result in slow convergence, low training stability and result in difficulties in capturing long-term dependencies. Hence, Residual Network (ResNet) was used to tackle the problem by introducing “skip connections” also known as the residual block, where a bypassing mechanism creates extra links from a single layer to a future layer skipping several intermediate layers. So instead of just learning the original function, the learned mapping becomes the sum of the original function and an identity mapping. This results in a more direct flow of gradients in the backpropagation process.

Section 3: Results & Comparison

We developed 2 different CNN architectures, our first model incorporates four 2-dimensional convolution layers with 32, 64, 64, 96, and 32 neurons for each layer respectively. All layers also use the rectifier (ReLU) activation function and have the padding method of “valid”. We also incorporated batch normalization between these layers to standardize the data before moving to the next layer. Pooling layers using the max-pooling method were added after each convolution layer. A dropout layer with a rate of 0.2 was added after the final pooling layer. We then have our first fully connected (FC) layer using ReLU activation and a unit size of 128, followed by another FC layer with softmax activation that returns the network output to match the number of image classes (31) that we have. Our second model is nearly identical to the first, except we removed the layer with 96 neurons before the last convolution layer. This increased the number of total parameters from 247,071 to 669,503. Between the 2 architectures, the same learning rate is used while the number of epochs is tuned.

As for the performance of these two models, the training accuracy for the first model with epoch 20 is around 74% while the training accuracy for the second model with epoch 20 is 89%. The model is then applied to the validation set. However, the performance is rather poor, a max accuracy of 41% is achieved at epoch 40 for the first model and around 35% accuracy is achieved for the second model at epoch 24.

We also developed 2 different ResNet architectures based on the existing ResNet50 and ResNet101 from Tensorflow by flattening these base models and connecting to a fully connected layer of 1024 nodes and then leading to the final classification procedure. It turns out that for training accuracy, they can achieve around 92% and 99% respectively. When

considering computational efficiency and memory requirements, the ResNet50-based model is preferred.

By comparing the performance of CNN and ResNet, for the training process, ResNet is better. However, due to runtime issues, only CNN is tested for a validation set. The low accuracy for that suggests the possibility of overfitting.

Section 4: Difficulties/Further Discussion

First, the RESNET model uses a longer time than CNN. As we choose different epochs and layers to test, the RESNET model can run for as much as 3 hours as the number of layers and epochs gets larger. After our experiments, increasing the number of layers and epochs is able to increase the accuracy of the model. But at the same time, it will lead to overfitting of the model. So choosing the right number is crucial, and finally, we chose an epoch of 20. The second difficulty is that we need to choose between RESNET 50, RESNET 101, and RESNET 152 for the base model. We tried RESNET 34 as well as RESNET 100 and found that the first model led to underfitting and the second to overfitting. Finally, we chose RESNET 50. For further discussion, we can try different models to save time.

For our CNN models, we originally anticipated the second architecture would have a huge increase in runtime and chose to not run the second model to epoch 50. Overall the second model seems to perform better as it reached a higher accuracy score much earlier than the first. However, both models seem to suffer from overfitting issues as the validation accuracy never reached higher than 41%. Additionally, both accuracy scores seem to plateau after reaching 90%/40% for training and validation respectively. Also, we notice that the run time of the code is long, it takes ~2 hours to finish running 20 epochs for both models. We

could include parallel programming to accelerate the process and use GPU to make the model run faster.

Appendix

Model	CNN-1	CNN-2
Architecture	Conv32 Conv64 Conv64 Conv96 Conv32 FC128	Conv32 Conv64 Conv64 Conv32 FC128
# total parameters	247,071	669,503
# trainable parameters	246,495	669,119
Avg. runtime/epoch (seconds)	275	290
Max training accuracy	93.11%	92.96%
Max validation accuracy	40.64%	35.86%
Accuracy at epoch 20 (Train/Validation)	74.37%/27.69%	89.32%/30.68%

Table 1. CNN Model Architecture and Summary

Model	ResNet50	ResNet101
Architecture	ResNet50 FC1024	ResNet101 FC1024
# total parameters	25717663	44788127
# trainable parameters	2129951	2129951
Time (sec)/epoch	~540	~890
Accuracy	92%	99%

Table 2. ResNet Model Architecture and Summary in 20 epoch

Link to GitHub: [GitHub link to Project Code]

Link to Dataset: [https://www.kaggle.com/datasets/vasukipatel/face-recognition-dataset]

```
classes = list(data_set_tensor.class_indices.keys())
base_model  = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling='avg')
model_resnet = Sequential([
    base_model,
    Flatten(),
    Dense(1024, activation = "relu"),
    Dropout(0.2),
    Dense(len(classes), activation = "softmax")
])
```

Code 1. ResNet Model tuning

```
model_resnet.compile(
    loss= "categorical_crossentropy",
    optimizer = Adam(),
    metrics = ["accuracy"]
)
model_resnet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 31)                31775

=================================================================
Total params: 25717663 (98.11 MB)
Trainable params: 2129951 (8.13 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

Code 2. ResNet Model Parameters

```
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    subset='training',
    shuffle = True)
val_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    subset='validation',
    shuffle = True)

classes = list(train_generator.class_indices.keys())
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Conv2D(96, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(classes),activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Code 3: CNN Tuning Parameters

Contribution:

Gustavo Nino Chaparro	Data cleaning, slides and report
Yixuan Deng	Build Resnet Model and related part of report
Mingzhan Yang	CNN Model building, CNN Model turning, slides and report
Rosin Gu	Slides and report
Shannon Ooi	ResNet methodology, Github, slides and report
Yuchen Wang	CNN model building, slides and report

Yaxin Yang	Model tuning, slides and report