Age, Gender, and Ethnicity Detection using CNNs

Team: Ahmed Ahmed (519, Ty), Praneeth Alla (519, Yang), Tirtha Kharel (519, Aditya)
Project Mentor TA: Ty

Abstract

In this paper, we examine the predictive capabilities of using lightweight deep learning models for gender/age/ethnicity classification trained on images of faces in the UTKFace dataset. With an emphasis on speed, age/gender/ethnicity classification are used extensively in the field of computer vision and ensuring an efficient yet accurate prediction is vital in advancement of computer vision. Therefore, this paper aims to show that CNNs are a fast and accurate option for these classification problems. We propose 3 separate CNN architectures to classify each of the attributes (age, gender, and ethnicity) with precision, recall, and f1 being the evaluation metrics. Our models achieve about 91% precision, 94% recall, and 93% f1 for gender prediction, 69% precision, 68% recall, and 68% f1 for age prediction, and 85% precision, 85% recall, and 85% f1 for ethnicity prediction. We found that despite the limited data and model complexity, we were still able to obtain relatively high levels of accuracy with fast performance.
Github Link to Code: https://github.com/doublea1186/CIS519-Project

Introduction

Computer Vision has been an immensely complicated yet useful topic in deep learning since the 1960s. Some of the most widely studied tasks pertaining to computer vision are age, gender, and ethnicity prediction. Successfully creating an accurate model for this task would lend itself towards a wide variety of applications in the real world. Specifically, these classification problems have many applications in healthcare, self-driving cars, security and marketing. Further, the models could be generalized to predict other similar variables, such as height/weight/fitness etc, which would further benefit society in applications such as healthcare. However, current work done on this problem is complicated, convoluted, and takes too long for a real world analysis. As such, speed and compactness are challenges we would like to address in this paper.

Our proposition, while applying concepts learned in CIS519 and attempting to achieve the best performance possible, is twofold. Firstly we will attempt to maximize performance by trying out different variations of CNNs and transfer learning. Secondly, we will attempt to minimize the time necessary to predict on a dataset by creating simple architectures, which still have excellent predictive power. As students of an introductory ML course, we do not expect to beat state of the art research, but we do expect to achieve relatively high evaluation metrics, while creating a model that could be immediately applied in a real world setting without computational/speed limitations.

Related Prior Work

CNNs for Face Detection: Kalinovskii et. al. [1] explored using CNNs in the place of more complex algorithms that require more computational power in order to solve facial recognition problems. The high computational complexity of existing algorithms limit their ability to be used to process high quality images and videos. Kalinovskii et. al. were able to create a model using a cascade of compact CNNs in order to efficiently solve facial recognition problems with an accuracy comparable to more complex algorithms. However, the speed of this algorithm far surpassed existing algorithms, suggesting that CNNs are a good choice for our models. We will attempt to show that these significant speed improvements hold for age, gender, and ethnicity prediction as well.

CNNs for Age and Gender Classification: Levi et. al. [2] provide even more evidence that CNNs are one of the best options for facial recognition within the domain that we are interested in (age and gender prediction). Like us, they treated age prediction as a classification problem rather than a regression problem. They used a CNN with three convolutional layers and two fully-connected layers. Through their research, Levi et. al. found that the usage of deep convolutional neural networks resulted in significant improvements in model accuracy. Additionally, Antipov et. al [4], who used CNNs for gender prediction, found that CNNs need much less training data to achieve high levels of accuracy. We believe that these results can be extended to ethnicity prediction as well.

Formal Problem Setup (T, E, P)

T: An image i is an RGB image of size w x h x c, where w is the width of the image in pixels, h is the height of the image in pixels, and c is the number of input channels. In this case, there are three input channels corresponding to the RGB values, which range from 0 to 255. For each image i, there are 3 variables we want to predict: age, gender, ethnicity. We will learn a predictor f(i) to approximate the age, gender, and ethnicity of the image. Age, gender, and ethnicity prediction are all classification problems.
E: We will use the UTKFace dataset of about 10,000 training images with the correct labels. The labels of each face image are embedded in the file name, where age is between 0-116, gender is 0/1 denoting male/female, and race is an integer from 0-4 denoting White, Black, Asian, Indian, and others. Age, Gender, and Ethnicity are all Classification problems. We will further use image augmentation, utilizing rotations, dilations, cropping, flipping, etc to increase the size of our dataset. This will allow us to fit a more complex model without overfitting and making the model more robust to transformations. We will use about 60% of the data for training, 20% for validation (for hyperparameter tuning), and 20% for training (for our final performance metrics).
P: We will use the negative loglikelihood of the softmax loss function (same as cross-entropy loss) over the images. From this loss function, we will be using the precision, recall, and f1 scores of each model as the standard metric of performance.

Methods

Baseline approaches we compare against:

For gender classification, we used GModel1 as the baseline model since it was the simplest and first CNN model created. For age classification, we used BaselineModel2, which was the simplest CNN model, and the pre-trained googlenet architecture as the baseline models. The last layer of the googlenet model was replaced with a 6 output fully connected layer, and the weights for all other layers were frozen. For ethnicity classification, we used a pre-trained resnet50 model as the baseline model (using transfer learning). The last layer of the model was replaced with a 5 output fully connected layer, and the weights for all other layers were frozen.

Implementation Details:

We used Pytorch/Apache Spark to train our deep learning models using GPUs. Images in the dataset were labeled “[age]_[gender]_[ethnicity]_[datetime].jpg”, so we used the image names in order to obtain the labels. Because some of the files were improperly named, we iterated through the dataset to remove such files. We further used image augmentation consisting of horizontal/vertical flips, random rotations, and random perspectives in order to increase the size of our dataset to about 19,558 images. We also added normalization to the images for better training, and we resized the images from (200, 200) to (64, 64) in order to drastically decrease runtime. During training, we use a learning rate scheduler to decrease the learning rate every few epochs, which helps achieve better results. We separated the data into a train/validation set for training and hyper parameter tuning with gridsearch and a test set for final evaluation metrics. We used the Adam Optimizer and binary cross entropy for all of our models.

GModel1 was a CNN with 2 blocks of 2 convolution relu, convolution, relu, batchnorm, maxpool2d(2, 2), and 25% percent dropout followed by a single fully connected layer with 2 outputs. It was trained with early stopping with a start learning rate of .05, regulation of 5e-4, and a learning rate decay of .1 every 5 epochs over a total of 15 epochs. GModel2 was a CNN with 4 blocks each consisting of convolution, relu, batchnorm, maxpool2d(2, 2), and 25% dropout followed by a single fully connected layer with 2 outputs. It was trained with early stopping with a start learning rate of .01, regulation of 5e-4, and a learning rate decay of .1 every 10 epochs over a total of 40 epochs. GModel3 is a transfer learning model with the resnet50 architecture with the last fully connected layer removed and replaced with a 2 output fully connected layer. The gradients are then frozen for all layers except the last and then trained on the images. It was trained with early stopping with a start learning rate of .01, regulation of 5e-4, and a learning rate decay of .1 every 10 epochs over a total of 40 epochs. Finally, GModel4 is another transfer learning model with the googlenet architecture used as a feature extractor in which the last fully connected layer (with 1000 outputs) is taken as inputs to a random forest (Using Apache Spark for efficient computation). It used a random forest, with a min sample split of 8, 100 estimators, max depth of 20, and used gini impurity. The detailed architectures for GModel1 and GModel2 are provided in the supplementary materials

For the age prediction model we first started by conducting a test to see how well we could predict ages that were not binned. We found that our scores were very low when we did not bin the ages, therefore we decided to organize the ages into 6 different classifications. Starting with the transfer learning model (Googlenet), we used an initial learning rate of 0.005, which was decreased by a factor of 0.1 every 10 epochs for 30 epochs. BaselineModel2 was a custom CNN architecture with 3 convolutional layers and one fully connected linear layer. Batch normalization was applied to all of the layers, and a 25% dropout was applied. We used an initial learning rate of .003, which was decreased by a factor of 0.1 every 5 epochs for a total of 15 epochs. We also used a regularization parameter that was set to 5e-4. Our final model for age classification, AgeClassifier2, used 4 convolutional layers and 2 fully connected layers. Batch normalization was applied to every convolutional layer. The reLu activation function was used for every layer except for the final fully connected layer. We applied a dropout of 25% on every other convolutional layer and a dropout of 50% on the second to last fully connected layer. The regularization parameter we used was 6e-4 and we used an initial learning rate of 0.0015, which was decreased by a factor of 0.1 every 10 epochs for a total of 35 epochs. The detailed architectures for both models are shown in the supplementary materials.

For the baseline ethnicity classification model (transfer learning with ResNet50), we used an initial learning rate of 0.003, which was decreased by a factor of 0.1 every 10 epochs over a total of 30 epochs. We also used a regularization parameter of 0.001. EthnicityModel2 was a custom CNN architecture with four convolutional layers and 3 fully-connected linear layers. Batch normalization was applied to all of the convolutional layers, and 25% dropout was applied to the second and fourth convolutional layers. The reLu activation function was used for all layers except for the last fully-connected layer. We used a regularization parameter of 5e-4, and an initial learning rate of 0.005, which was decreased by a factor of 0.1 every 5 epochs over a total of 15 epochs. Finally, EthnicityModel3 was also a custom CNN architecture with four convolutional layers and 3 fully-connected layers. Batch normalization and max-pooling were applied to all of the convolutional layers, and 25% dropout was applied to the second and fourth convolutional layers. The reLu activation function was used for all layers except for the last fully-connected layer. We used a regularization parameter of 5e-4, and an initial learning rate of 0.001, which was decreased by a factor of 0.1 every 5 epochs over a total of 15 epochs. The detailed architectures for EthnicityModel2 and EthnicityModel3 are shown in the supplementary materials.

Experimental Results

Questions:
We aim to answer the following questions: (1) Which model(s) predict age/gender/ethnicity the best? (2) Do our models predict quickly? (3) How well do our models generalize? (4) What works for this problem and what doesn't? (5) Does the limited size of our dataset inhibit performance?

We compare our models for each feature on the test data and see which one predicts better, given the time it takes to predict, as highlighted below, which solves [1] and [2]. Despite the limited size of the dataset, we were still able to achieve very high accuracy (91%), which answers [5]. Our evaluation metrics for train/test were almost identical at the end of training for each model and therefore we can conclude that our models generalize well which solves [3]. We can further conclude that Transfer learning with Random Forest is not a good technique to classify gender which solves [4].

	Precision	Recall	F1	Test Prediction Time (s)
GModel1	.872	.884	.878	4.96
GModel2	.893	.905	.899	5.34
GModel3	.912	.943	.927	5.79
GModel4	.791	.783	.787	2.51

We therefore have GModels 1-3 which predict increasingly more accurately but take an increasing amount of time to predict, which shows the ability of CNNs to predict gender with high metrics (slightly lower than state of the art). We also showcase GModel4, which predicts worse than the other 3 models, but takes a fraction of the time. Therefore, it may be applicable in circumstances where speed is much more valuable than accuracy.

The results of our models for age prediction are shown below.

	Precision	Recall	F1	TestPrediction Time (s)
Googlenet (Transfer learning)	.252	.280	.241	2.93
BaselineModel2 (initial CNN model)	.642	.654	.640	2.71
AgeModel2	.690	.682	.684	2.55

We found that for the problem of age prediction, the transfer learning model did not perform well. We did however improve our precision and recall drastically with a custom CNN, which answers [1]. Furthermore, the models predicted on the test dataset fairly quickly, and adding to the complexity didn't hinder the time taken [2]. We did find, however, that because of the limited size dataset and added complexity, the model did overfit a bit more than the other models, which answers [4]. However, we were still able to achieve a relatively high accuracy (70%), which answers [5].

The results of our models for ethnicity prediction are summarized below.

	Precision	Recall	F1	Test Prediction Time (s)
ResNet50 (Transfer Learning)	0.601	0.639	0.610	6.19
EthnicityModel2 (Custom CNN)	0.732	0.749	0.736	6.14
EthnicityModel3 (Custom CNN)	0.852	0.854	0.853	6.19

Similar to age classification, we found that the baseline transfer learning model did not perform that well for ethnicity prediction. We obtained much higher accuracy when we built custom CNN in order to solve the problem, which answers [1] above. As shown below, we were able to achieve about 85% accuracy with CNNs. Furthermore, the models were able to predict on the test dataset relatively quickly, and we found that increasing the complexity of the model only contributes to a slight increase in prediction time (see EthnicityModel2 vs. EthnicityModel3 in the table below). As a result, we can conclude that CNNs are able to solve the ethnicity classification problem quickly and efficiently, which solves [2]. We also found that for the custom CNNs, the metrics on the training dataset were very similar to the metrics on the testing dataset, indicating that our models generalize well, which answers [4]. Despite the small size of the dataset, we were still able to achieve a relatively high accuracy (85%), which answers [5]. This is consistent with existing literature that states that CNNs need much less training data to achieve high accuracy.

Conclusions and Future Work

We conclude by asserting that while our models may not have the same state of the art performance as other published models, we were able to get significant results with limited data and model size. We believe that if we were to train our models with many more different images, we will be able to achieve similar results as published papers but with faster performance. Overall, it is clear to see how powerful CNNs are in regards to image classification and we look forward to experimenting with this problem and similar problems in the future.

Ethical Considerations and Broader Impacts:

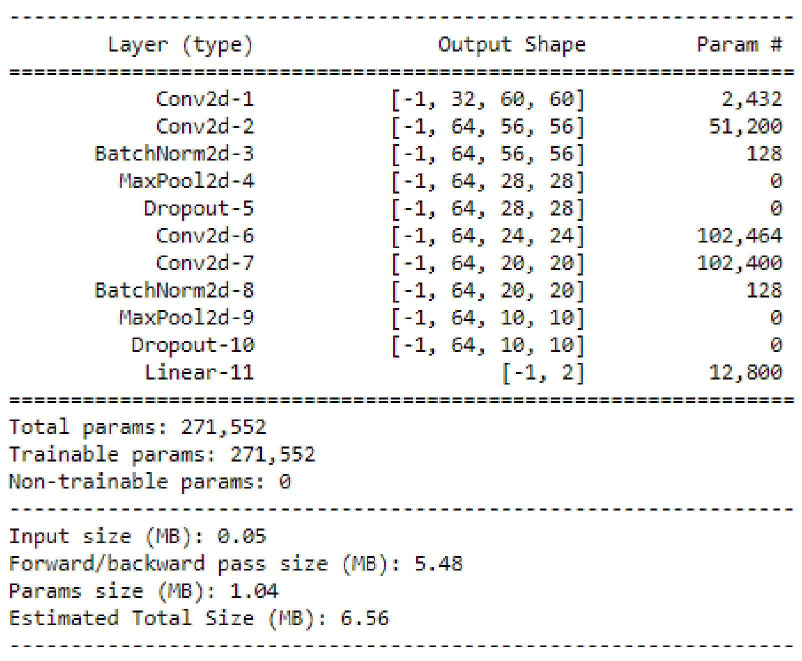
The data was collected with the permission of the people in the images, and none of the data has any private information. We use multiple evaluations metrics to make sure we are not biased against either gender/age/ethnicity. In terms of real world application, the models created would need to be retrained on a representative sample of the population indicative on the area of application, rather than the data we have used for this project. This is because Improperly detecting someone’s age, gender, or ethnicity can have significant negative implications, including the denial of opportunities for a certain individual.

Prior Work / References:

1. Kalinovskii I.A., Spitsyn V.G., “Compact Convolutional Neural Network Cascade for Face Detection”
2. Gil Levi and Tal Hassner, “Age and Gender Classification using Convolutional Neural Networks”. CVPR (2015)
3. Yicheng An, Jiafu Wu, Chang Yue, “CNNs for Face Detection and Recognition”
4. Antipov, Grigory, Sid-Ahmed Berrani, and Jean-Luc Dugelay. "Minimalistic CNN-based ensemble model for gender prediction from face images." Pattern recognition letters 70 (2016)
5. https://github.com/rachit-shah/Apparent-Age-Prediction-using-CNN

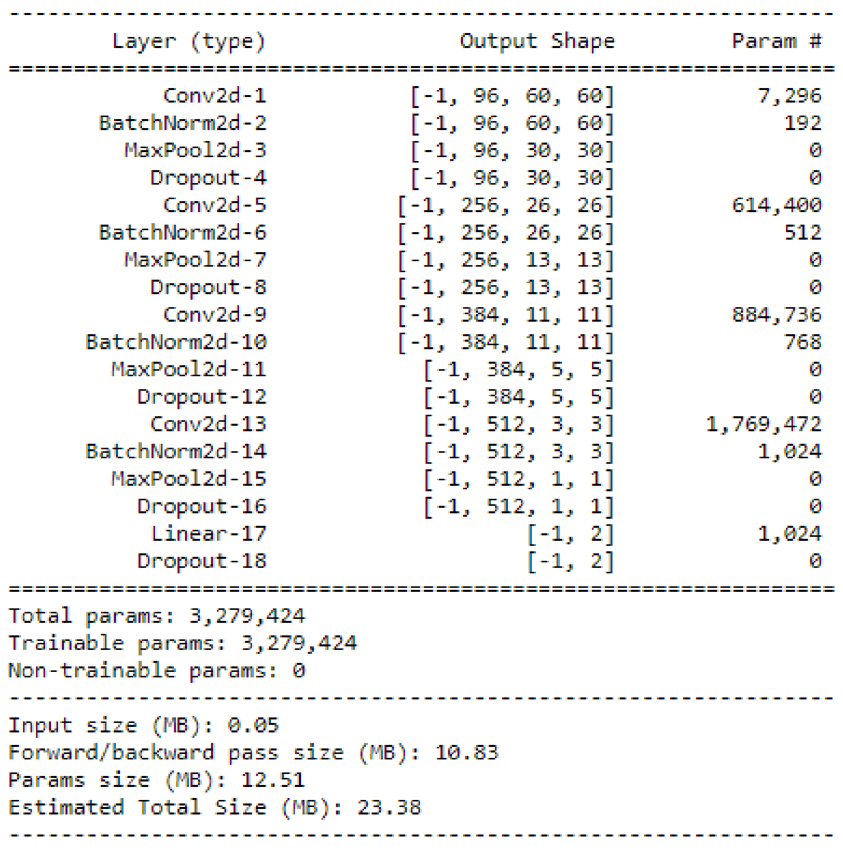
Supplementary material:

GModel1 Architecture:



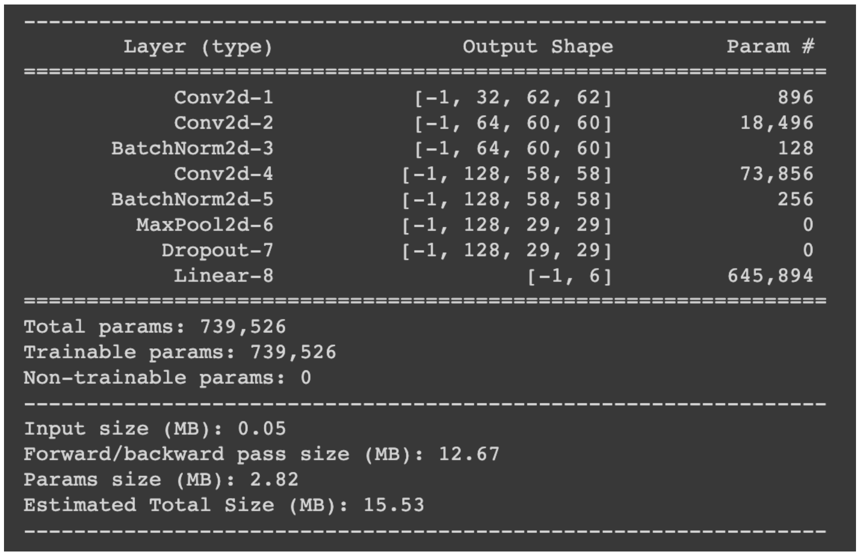
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 60, 60]           2,432
            Conv2d-2           [-1, 64, 56, 56]          51,200
       BatchNorm2d-3           [-1, 64, 56, 56]             128
         MaxPool2d-4           [-1, 64, 28, 28]               0
           Dropout-5           [-1, 64, 28, 28]               0
            Conv2d-6           [-1, 64, 24, 24]         102,464
            Conv2d-7           [-1, 64, 20, 20]         102,400
       BatchNorm2d-8           [-1, 64, 20, 20]             128
         MaxPool2d-9           [-1, 64, 10, 10]               0
          Dropout-10           [-1, 64, 10, 10]               0
           Linear-11                    [-1, 2]          12,800
================================================================
Total params: 271,552
Trainable params: 271,552
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 5.48
Params size (MB): 1.04
Estimated Total Size (MB): 6.56
----------------------------------------------------------------
```

GModel2 Architecture:



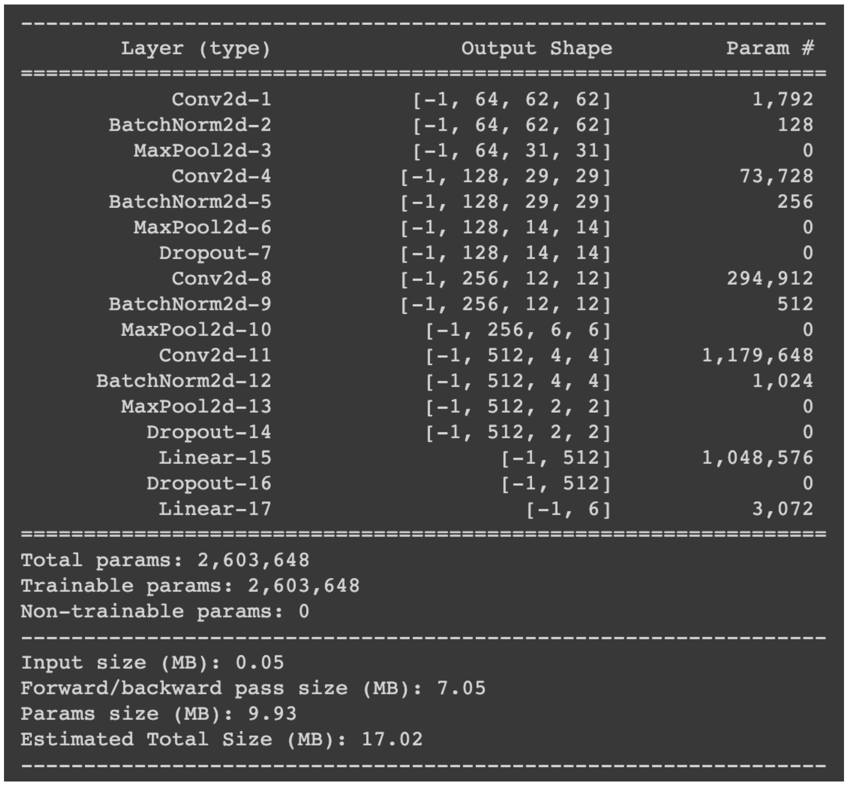
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 60, 60]           7,296
       BatchNorm2d-2           [-1, 96, 60, 60]             192
         MaxPool2d-3           [-1, 96, 30, 30]               0
           Dropout-4           [-1, 96, 30, 30]               0
            Conv2d-5          [-1, 256, 26, 26]         614,400
       BatchNorm2d-6          [-1, 256, 26, 26]             512
         MaxPool2d-7          [-1, 256, 13, 13]               0
           Dropout-8          [-1, 256, 13, 13]               0
            Conv2d-9          [-1, 384, 11, 11]         884,736
      BatchNorm2d-10          [-1, 384, 11, 11]             768
        MaxPool2d-11            [-1, 384, 5, 5]               0
          Dropout-12            [-1, 384, 5, 5]               0
           Conv2d-13            [-1, 512, 3, 3]       1,769,472
      BatchNorm2d-14            [-1, 512, 3, 3]           1,024
        MaxPool2d-15            [-1, 512, 1, 1]               0
          Dropout-16            [-1, 512, 1, 1]               0
           Linear-17                    [-1, 2]           1,024
          Dropout-18                    [-1, 2]               0
================================================================
Total params: 3,279,424
Trainable params: 3,279,424
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 10.83
Params size (MB): 12.51
Estimated Total Size (MB): 23.38
----------------------------------------------------------------
```

BaselineModel2 (Age)



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 62, 62]             896
            Conv2d-2           [-1, 64, 60, 60]          18,496
       BatchNorm2d-3           [-1, 64, 60, 60]             128
            Conv2d-4          [-1, 128, 58, 58]          73,856
       BatchNorm2d-5          [-1, 128, 58, 58]             256
         MaxPool2d-6          [-1, 128, 29, 29]               0
           Dropout-7          [-1, 128, 29, 29]               0
            Linear-8                    [-1, 6]         645,894
================================================================
Total params: 739,526
Trainable params: 739,526
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 12.67
Params size (MB): 2.82
Estimated Total Size (MB): 15.53
----------------------------------------------------------------
```

AgeClassifier2



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 62, 62]           1,792
       BatchNorm2d-2           [-1, 64, 62, 62]             128
         MaxPool2d-3           [-1, 64, 31, 31]               0
            Conv2d-4          [-1, 128, 29, 29]          73,728
       BatchNorm2d-5          [-1, 128, 29, 29]             256
         MaxPool2d-6          [-1, 128, 14, 14]               0
           Dropout-7          [-1, 128, 14, 14]               0
            Conv2d-8          [-1, 256, 12, 12]         294,912
       BatchNorm2d-9          [-1, 256, 12, 12]             512
        MaxPool2d-10            [-1, 256, 6, 6]               0
           Conv2d-11            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-12            [-1, 512, 4, 4]           1,024
        MaxPool2d-13            [-1, 512, 2, 2]               0
          Dropout-14            [-1, 512, 2, 2]               0
           Linear-15                  [-1, 512]       1,048,576
          Dropout-16                  [-1, 512]               0
           Linear-17                    [-1, 6]           3,072
================================================================
Total params: 2,603,648
Trainable params: 2,603,648
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 7.05
Params size (MB): 9.93
Estimated Total Size (MB): 17.02
----------------------------------------------------------------
```

EthnicityModel2 Architecture

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 61, 61]             784
       BatchNorm2d-2           [-1, 16, 61, 61]              32
            Conv2d-3           [-1, 32, 29, 29]           8,224
       BatchNorm2d-4           [-1, 32, 29, 29]              64
           Dropout-5           [-1, 32, 29, 29]               0
            Conv2d-6           [-1, 64, 26, 26]          32,832
       BatchNorm2d-7           [-1, 64, 26, 26]             128
            Conv2d-8           [-1, 64, 12, 12]          65,600
       BatchNorm2d-9           [-1, 64, 12, 12]             128
          Dropout-10           [-1, 64, 12, 12]               0
           Linear-11                  [-1, 256]       2,359,552
           Linear-12                  [-1, 128]          32,896
           Linear-13                    [-1, 5]             645
================================================================
Total params: 2,500,885
Trainable params: 2,500,885
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 2.40
Params size (MB): 9.54
Estimated Total Size (MB): 11.99
----------------------------------------------------------------
```

EthnicityModel3 Architecture

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 61, 61]           3,136
       BatchNorm2d-2           [-1, 64, 61, 61]             128
         MaxPool2d-3           [-1, 64, 30, 30]               0
            Conv2d-4          [-1, 128, 27, 27]         131,200
       BatchNorm2d-5          [-1, 128, 27, 27]             256
         MaxPool2d-6          [-1, 128, 13, 13]               0
           Dropout-7          [-1, 128, 13, 13]               0
            Conv2d-8          [-1, 512, 10, 10]       1,049,088
       BatchNorm2d-9          [-1, 512, 10, 10]           1,024
        MaxPool2d-10            [-1, 512, 5, 5]               0
           Conv2d-11           [-1, 1024, 2, 2]       8,389,632
      BatchNorm2d-12           [-1, 1024, 2, 2]           2,048
        MaxPool2d-13           [-1, 1024, 1, 1]               0
          Dropout-14           [-1, 1024, 1, 1]               0
           Linear-15                  [-1, 256]         262,400
           Linear-16                  [-1, 128]          32,896
           Linear-17                    [-1, 5]             645
================================================================
Total params: 9,872,453
Trainable params: 9,872,453
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 6.79
Params size (MB): 37.66
Estimated Total Size (MB): 44.49
----------------------------------------------------------------
```