**EMOTION RECOGNITION USING FACIAL EXPRESSIONS**

**(AI PROJECT)**

**GROUP 11:**

MOHSIN RAZA

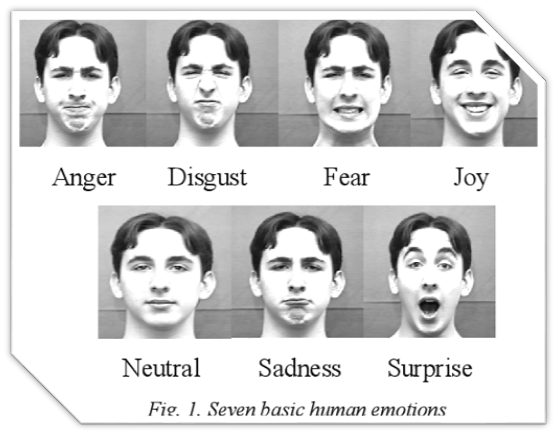
IJTIHAD HUSSAIN

SHOAIB AKHTAR

ABDUR RAFFAY

**INTRODUCTION**

* Emotion recognition is the process of identifying human emotion. People vary widely in their accuracy at recognizing the emotions of others. By using Facial Emotion Recognition, businesses can process images, and videos in real-time for monitoring video feeds or automating video analytics, thus saving costs and making life better for their users.
* A technology which Uses biometric markers to detect emotions in human faces.
* This technology is a sentiment analysis tool and is able to automatically detect the expressions.



**CONT.**

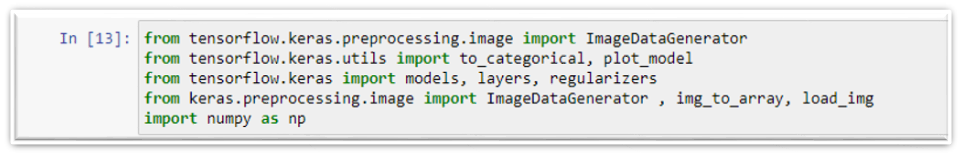
* So basically, there's already work done on emotion recognition and we can easily find it online but we have added some more by ourselves.
* AI can detect emotions by learning what each facial expression means and applying that knowledge to the new information presented to it. Emotional artificial intelligence, or emotion AI, is a technology that is capable of reading, imitating, interpreting, and responding to human facial expressions and emotions.
* Companies have also been taking advantage of emotion recognition to drive business outcomes. For the upcoming release of Toy Story 5, Disney plans to use facial recognition to judge the emotional responses of the audience. Apple even released a new feature on the iPhone X called Animoji, where you can get a computer simulated emoji to mimic your facial expressions. It’s not so far off to assume they’ll use those capabilities in other applications soon.

**APPLICATIONS & IMPORTANTCE**

* Market research
* Video game testing
* It is the most ideal way to assess the effectiveness of any business content
* It can provide valuable information about the sentiment of a target audience towards a marketing message, product or brand

**LIBRARIES**

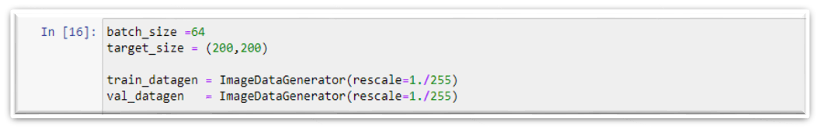
* **Numpy**
* **Keras**
* **Tensorflow**



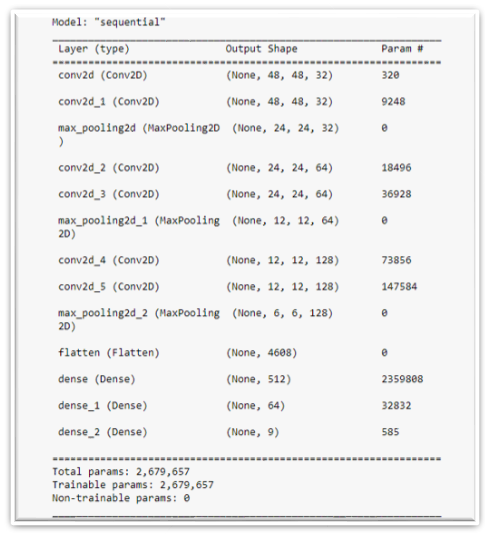
**SAMPLE DATA**



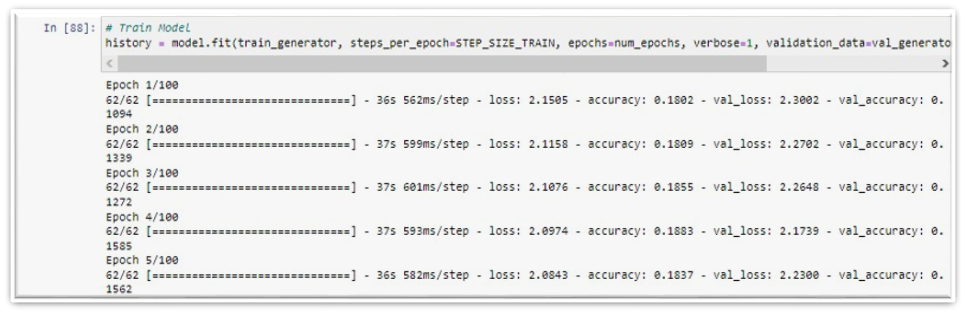
**SETTING HYPERPARAMETERS**



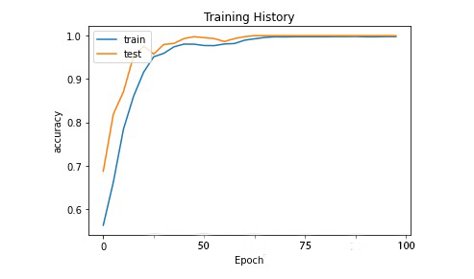
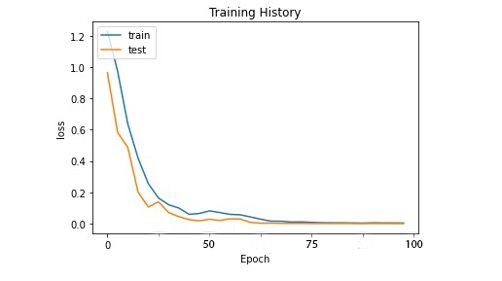
**MODAL SUMMARY:**



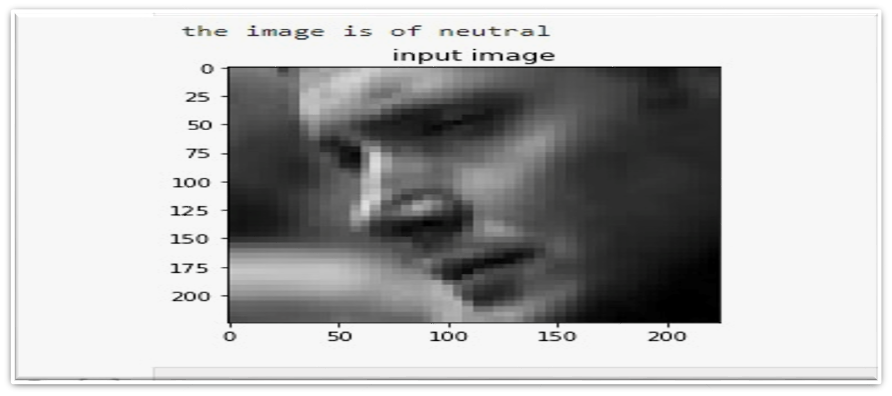
**MODAL TRAINING:**



**LOSS & ACCURACY GRAPHS:**



**TESTING:**



**CODE:**

{

"cells": [

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {

"scrolled": true

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"['test', 'train']\n"

]

}

],

"source": [

"import os\n",

"print(os.listdir('fer2013'))"

]

},

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {},

"outputs": [],

"source": [

"train\_path = 'fer2013/train'\n",

"val\_path = 'fer2013/test'"

]

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {},

"outputs": [],

"source": [

"import matplotlib.pyplot as plt\n",

"def plot\_images(img\_dir, top=10):\n",

" all\_img\_dirs = os.listdir(img\_dir)\n",

" img\_files = [os.path.join(img\_dir, file) for file in all\_img\_dirs][:5]\n",

" \n",

" plt.figure(figsize=(10, 10))\n",

" \n",

" for idx, img\_path in enumerate(img\_files):\n",

" plt.subplot(5, 5, idx+1)\n",

" \n",

" img = plt.imread(img\_path)\n",

" plt.tight\_layout() \n",

" plt.imshow(img, cmap='gray') "

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {

"scrolled": true

},

"outputs": [

{

"data":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/angry')"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/disgust')"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {

"scrolled": true

},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/fear')"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/happy')"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/neutral')"

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"metadata": {

"scrolled": true

},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/sad')"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {

"scrolled": true

},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/surprise')"

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {

"scrolled": true

},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/confused')"

]

},

{

"cell\_type": "code",

"execution\_count": 15,

"metadata": {},

"outputs": [

{

"data": {

"image/png":

"text/plain": [

"<Figure size 720x720 with 5 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"plot\_images(train\_path+'/flustered')"

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {

"scrolled": false

},

"outputs": [],

"source": [

"from tensorflow.keras.preprocessing.image import ImageDataGenerator \n",

"from tensorflow.keras.utils import to\_categorical, plot\_model\n",

"from tensorflow.keras import models, layers, regularizers"

]

},

{

"cell\_type": "code",

"execution\_count": 17,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"['angry', 'confused', 'disgust', 'fear', 'flustered', 'happy', 'neutral', 'sad', 'surprise']\n"

]

}

],

"source": [

"emotion\_labels = sorted(os.listdir(train\_path))\n",

"print(emotion\_labels)"

]

},

{

"cell\_type": "code",

"execution\_count": 18,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Found 3998 images belonging to 9 classes.\n",

"Found 450 images belonging to 9 classes.\n"

]

}

],

"source": [

"batch\_size =64\n",

"target\_size = (48,48)\n",

"\n",

"train\_datagen = ImageDataGenerator(rescale=1./255)\n",

"val\_datagen = ImageDataGenerator(rescale=1./255)\n",

"\n",

"train\_generator = train\_datagen.flow\_from\_directory(\n",

" train\_path,\n",

" target\_size=target\_size,\n",

" batch\_size=batch\_size,\n",

" color\_mode=\"grayscale\",\n",

" class\_mode='categorical',\n",

" shuffle=True)\n",

"\n",

"val\_generator = val\_datagen.flow\_from\_directory(\n",

" val\_path,\n",

" target\_size=target\_size,\n",

" batch\_size=batch\_size,\n",

" color\_mode=\"grayscale\",\n",

" class\_mode='categorical')"

]

},

{

"cell\_type": "code",

"execution\_count": 19,

"metadata": {},

"outputs": [],

"source": [

"input\_shape = (48,48,1) # img\_rows, img\_colums, color\_channels\n",

"num\_classes = 9"

]

},

{

"cell\_type": "code",

"execution\_count": 20,

"metadata": {

"scrolled": true

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Model: \"sequential\"\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

" Layer (type) Output Shape Param # \n",

"=================================================================\n",

" conv2d (Conv2D) (None, 48, 48, 32) 320 \n",

" \n",

" conv2d\_1 (Conv2D) (None, 48, 48, 32) 9248 \n",

" \n",

" max\_pooling2d (MaxPooling2D (None, 24, 24, 32) 0 \n",

" ) \n",

" \n",

" conv2d\_2 (Conv2D) (None, 24, 24, 64) 18496 \n",

" \n",

" conv2d\_3 (Conv2D) (None, 24, 24, 64) 36928 \n",

" \n",

" max\_pooling2d\_1 (MaxPooling (None, 12, 12, 64) 0 \n",

" 2D) \n",

" \n",

" conv2d\_4 (Conv2D) (None, 12, 12, 128) 73856 \n",

" \n",

" conv2d\_5 (Conv2D) (None, 12, 12, 128) 147584 \n",

" \n",

" max\_pooling2d\_2 (MaxPooling (None, 6, 6, 128) 0 \n",

" 2D) \n",

" \n",

" flatten (Flatten) (None, 4608) 0 \n",

" \n",

" dense (Dense) (None, 512) 2359808 \n",

" \n",

" dense\_1 (Dense) (None, 64) 32832 \n",

" \n",

" dense\_2 (Dense) (None, 9) 585 \n",

" \n",

"=================================================================\n",

"Total params: 2,679,657\n",

"Trainable params: 2,679,657\n",

"Non-trainable params: 0\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

]

}

],

"source": [

"# Build Model\n",

"model = models.Sequential()\n",

"\n",

"model.add(layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='same', input\_shape=input\_shape)) #, data\_format='channels\_last', kernel\_regularizer=regularizers.l2(0.01)))\n",

"model.add(layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='same'))\n",

"model.add(layers.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))\n",

"\n",

"model.add(layers.Conv2D(64, kernel\_size=(3, 3), activation='relu', padding='same'))\n",

"model.add(layers.Conv2D(64, kernel\_size=(3, 3), activation='relu', padding='same'))\n",

"model.add(layers.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))\n",

"\n",

"model.add(layers.Conv2D(128, kernel\_size=(3, 3), activation='relu', padding='same'))\n",

"model.add(layers.Conv2D(128, kernel\_size=(3, 3), activation='relu', padding='same'))\n",

"model.add(layers.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))\n",

"\n",

"model.add(layers.Flatten())\n",

"\n",

"model.add(layers.Dense(512, activation='relu'))\n",

"model.add(layers.Dense(64, activation='relu'))\n",

"\n",

"model.add(layers.Dense(num\_classes, activation='softmax'))\n",

"\n",

"model.summary()"

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"metadata": {},

"outputs": [],

"source": [

"# Compile Model\n",

"model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])"

]

},

{

"cell\_type": "code",

"execution\_count": 149,

"metadata": {},

"outputs": [],

"source": [

"num\_epochs = 100\n",

"STEP\_SIZE\_TRAIN = train\_generator.n//train\_generator.batch\_size\n",

"STEP\_SIZE\_VAL = val\_generator.n//val\_generator.batch\_size"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"scrolled": false

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epoch 1/50\n",

"399/399 [==============================] - 73s 181ms/step - loss: 2.1347 - accuracy: 0.1732 - val\_loss: 2.3056 - val\_accuracy: 0.1111\n",

"Epoch 2/50\n",

"399/399 [==============================] - 72s 180ms/step - loss: 2.1255 - accuracy: 0.1795 - val\_loss: 2.2688 - val\_accuracy: 0.1111\n",

"Epoch 3/50\n",

"399/399 [==============================] - 72s 182ms/step - loss: 2.1251 - accuracy: 0.1795 - val\_loss: 2.2900 - val\_accuracy: 0.1111\n",

"Epoch 4/50\n",

"399/399 [==============================] - 73s 183ms/step - loss: 2.1242 - accuracy: 0.1795 - val\_loss: 2.2594 - val\_accuracy: 0.1111\n",

"Epoch 5/50\n",

"399/399 [==============================] - 74s 185ms/step - loss: 2.1228 - accuracy: 0.1800 - val\_loss: 2.2923 - val\_accuracy: 0.1111\n",

"Epoch 6/50\n",

"295/399 [=====================>........] - ETA: 18s - loss: 2.1268 - accuracy: 0.1760- ETA: 2"

]

}

],

"source": [

"# Train Model\n",

"history = model.fit(train\_generator, steps\_per\_epoch=STEP\_SIZE\_TRAIN, epochs=num\_epochs, verbose=1, validation\_data=val\_generator, validation\_steps=STEP\_SIZE\_VAL)"

]

},

{

"cell\_type": "code",

"execution\_count": 115,

"metadata": {},

"outputs": [],

"source": [

"# Save Model\n",

"models.save\_model(model, 'fer2013\_cnn.h5')"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {

"scrolled": true

},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'STEP\_SIZE\_VAL' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m~\\AppData\\Local\\Temp/ipykernel\_10592/1269167435.py\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m 1\u001b[0m \u001b[1;31m# Evaluate Model\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 2\u001b[1;33m \u001b[0mscore\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mmodel\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mevaluate\_generator\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mval\_generator\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0msteps\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mSTEP\_SIZE\_VAL\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 3\u001b[0m \u001b[0mprint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m'Test loss: '\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mscore\u001b[0m\u001b[1;33m[\u001b[0m\u001b[1;36m0\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 4\u001b[0m \u001b[0mprint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m'Test accuracy: '\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mscore\u001b[0m\u001b[1;33m[\u001b[0m\u001b[1;36m1\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'STEP\_SIZE\_VAL' is not defined"

]

}

],

"source": [

"# Evaluate Model\n",

"score = model.evaluate\_generator(val\_generator, steps=STEP\_SIZE\_VAL) \n",

"print('Test loss: ', score[0])\n",

"print('Test accuracy: ', score[1])"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# Show Train History\n",

"keys=history.history.keys()\n",

"print(keys)\n",

"\n",

"def show\_train\_history(hisData,train,test): \n",

" plt.plot(hisData.history[train])\n",

" plt.plot(hisData.history[test])\n",

" plt.title('Training History')\n",

" plt.ylabel(train)\n",

" plt.xlabel('Epoch')\n",

" plt.legend(['train', 'test'], loc='upper left')\n",

" plt.show()\n",

"\n",

"show\_train\_history(history, 'loss', 'val\_loss')\n",

"show\_train\_history(history, 'accuracy', 'val\_accuracy')"

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [

{

"ename": "NameError",

"evalue": "name 'load\_img' is not defined",

"output\_type": "error",

"traceback": [

"\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",

"\u001b[1;31mNameError\u001b[0m Traceback (most recent call last)",

"\u001b[1;32m~\\AppData\\Local\\Temp/ipykernel\_10592/2117660961.py\u001b[0m in \u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m 2\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 3\u001b[0m \u001b[0mpath\u001b[0m \u001b[1;33m=\u001b[0m \u001b[1;34m\"C:/Users/dell/Desktop/Emotion\_Recognition\_using\_FE/fer2013/test/angry/Training\_3908.jpg\"\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 4\u001b[1;33m \u001b[0mimg\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mload\_img\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mpath\u001b[0m\u001b[1;33m,\u001b[0m \u001b[0mtarget\_size\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;36m224\u001b[0m\u001b[1;33m,\u001b[0m\u001b[1;36m224\u001b[0m\u001b[1;33m)\u001b[0m \u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\u001b[0;32m 5\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m 6\u001b[0m \u001b[0mi\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mimg\_to\_array\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mimg\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m/\u001b[0m\u001b[1;36m255\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",

"\u001b[1;31mNameError\u001b[0m: name 'load\_img' is not defined"

]

}

],

"source": [

"# path for the image to see if it predics correct class\n",

"\n",

"path = \"C:/Users/dell/Desktop/Emotion\_Recognition\_using\_FE/fer2013/test/angry/Training\_3908.jpg\"\n",

"img = load\_img(path, target\_size=(224,224) )\n",

"\n",

"i = img\_to\_array(img)/255\n",

"input\_arr = np.array([i])\n",

"input\_arr.shape\n",

"\n",

"pred = np.argmax(model.predict(input\_arr))\n",

"\n",

"print(f\" the image is of {op[pred]}\")\n",

"\n",

"# to display the image \n",

"plt.imshow(input\_arr[0])\n",

"plt.title(\"input image\")\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3 (ipykernel)",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.9.6"

}

},

"nbformat": 4,

"nbformat\_minor": 4

}