```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Dropout, Reshape, Concatenate, Leaky
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
```

```
image_dimensions = {'height':256, 'width':256, 'channels':3}
```

```
class Classifier:
    def __init__():
        self.model = 0

    def predict(self, x):
        return self.model.predict(x)

    def fit(self, x, y):
        return self.model.train_on_batch(x, y)

    def get_accuracy(self, x, y):
        return self.model.test_on_batch(x, y)

    def load(self, path):
        self.model.load_weights(path)
```

```
class Meso4(Classifier):
    def __init__(self, learning_rate = 0.001):
        self.model = self.init_model()
        optimizer = Adam(lr = learning_rate)
        self.model.compile(optimizer = optimizer,
                           loss = 'mean_squared_error',
                           metrics = ['accuracy'])

    def init_model(self):
        x = Input(shape = (image_dimensions['height'],
                           image_dimensions['width'],
                           image_dimensions['channels']))

        x1 = Conv2D(8, (3, 3), padding='same', activation = 'relu')(x)
        x1 = BatchNormalization()(x1)
        x1 = MaxPooling2D(pool_size=(2, 2), padding='same')(x1)

        x2 = Conv2D(8, (5, 5), padding='same', activation = 'relu')(x1)
        x2 = BatchNormalization()(x2)
        x2 = MaxPooling2D(pool_size=(2, 2), padding='same')(x2)

        x3 = Conv2D(16, (5, 5), padding='same', activation = 'relu')(x2)
        x3 = BatchNormalization()(x3)
        x3 = MaxPooling2D(pool_size=(2, 2), padding='same')(x3)

        x4 = Conv2D(16, (5, 5), padding='same', activation = 'relu')(x3)
        x4 = BatchNormalization()(x4)
        x4 = MaxPooling2D(pool_size=(4, 4), padding='same')(x4)

        y = Flatten()(x4)
        y = Dropout(0.5)(y)
        y = Dense(16)(y)
        y = LeakyReLU(alpha=0.1)(y)
        y = Dropout(0.5)(y)
        y = Dense(1, activation = 'sigmoid')(y)

        return Model(inputs = x, outputs = y)
```

```
meso.load('/content/Meso4_DF ')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
dataGenerator = ImageDataGenerator(rescale=1./255)

generator = dataGenerator.flow_from_directory(
    '/content/drive/MyDrive/DATA/data',
    target_size=(256, 256),
    batch_size=1,
    class_mode='binary')
```

```
Found 3218 images belonging to 2 classes.
```

```
X, y = generator.next()

print(f"Predicted likelihood: {meso.predict(X)[0][0]:.4f}")
print(f"Actual label: {int(y[0])}")
print(f"\nCorrect prediction: {round(meso.predict(X)[0][0])==y[0]}")

plt.imshow(np.squeeze(X));
```

```
1/1 [==============================] - 0s 317ms/step
Predicted likelihood: 0.9096
Actual label: 1
1/1 [==============================] - 0s 56ms/step

Correct prediction: True
```

```
correct_real = []
correct_real_pred = []

correct_deepfake = []
correct_deepfake_pred = []

misclassified_real = []
misclassified_real_pred = []

misclassified_deepfake = []
misclassified_deepfake_pred = []
```

```
for i in range(len(generator.labels)):

    X, y = generator.next()
    pred = meso.predict(X)[0][0]

    if round(pred)==y[0] and y[0]==1:
        correct_real.append(X)
        correct_real_pred.append(pred)
    elif round(pred)==y[0] and y[0]==0:
        correct_deepfake.append(X)
        correct_deepfake_pred.append(pred)
    elif y[0]==1:
        misclassified_real.append(X)
        misclassified_real_pred.append(pred)
    else:
        misclassified_deepfake.append(X)
        misclassified_deepfake_pred.append(pred)

    if i % 1000 == 0:
        print(i, ' predictions completed.')

    if i == len(generator.labels)-1:
        print("All", len(generator.labels), "predictions completed")
```

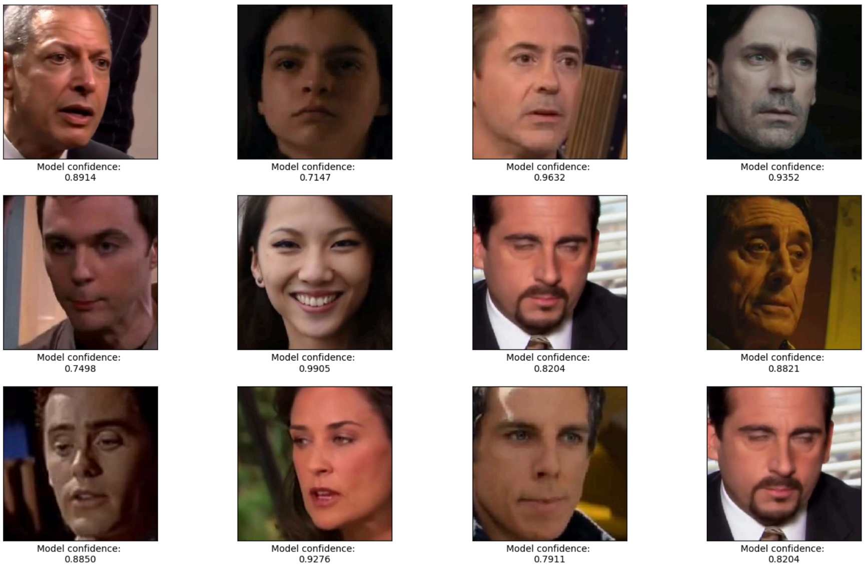
```
1/1 [==============================] - 0s 40ms/step
0  predictions completed.
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 44ms/step
```

```
def plotter(images, preds):
    fig, axes = plt.subplots(3, 4, figsize=(16, 9))
    subset = np.random.choice(len(images), 12, replace=False)

    for i, ax in enumerate(axes.flatten()):
        idx = subset[i]
        ax.imshow(np.squeeze(images[idx]))
        ax.set_xlabel(f"Model confidence: \n{preds[idx]:.4f}")
        ax.set_xticks([])
        ax.set_yticks([])

    plt.tight_layout()
    plt.show()
```

```
plotter(correct_real, correct_real_pred)
```



```
def plotter(images,preds):
    if len(images) <= 1:
        print("Not enough images to plot.")
        return

    fig = plt.figure(figsize=(16,9))
    num_images = min(12, len(images))
    subset = np.random.randint(0, len(images), num_images)
    for i,j in enumerate(subset):
        fig.add_subplot(3,4,i+1)
        plt.imshow(np.squeeze(images[j]))
        plt.xlabel(f"Model confidence: \n{preds[j]:.4f}")
        plt.tight_layout()
        ax = plt.gca()
        ax.axes.xaxis.set_ticks([])
        ax.axes.yaxis.set_ticks([])
    plt.show();
    return
```

```
plotter(misclassified_real, misclassified_real_pred)
```