پیاده سازی مقاله تشخیص سرطان پوست خوش‌خیم از بدخیم با استفاده از شبکه های کانولوشنی ترکیبی

مقدمه

در این گزارش تلاش بر این خواهد بود که شبکه‌های کانولوشنی ترکیب شده پیشنهادی در مقاله را پیاده سازی کرده و آن را با سایر مدل‌های معروف کانولوشنی مقایسه کنیم. برای این کار از دیتاست مربوطه موجود در سایت Kaggle استفاده می‌کنیم که شامل چندین عکس از سرطان های پوست خوش‌خیم وبدخیم به صورت پوشه بندی شده می‌شود.

ابزار های لازم

این پیاده سازی در قالب فایل نوت بوک پایتون(.ipynb) و در محیط Google Colab انجام شده است. در این پیاده سازی از کتابخانه TensorFlow برای ساخت شبکه‌ها، OpenCV، Scikit Image و NumPy برای کار با تصاویر، Matplotlib برای رسم نمودارها و Scikit Learn برای محاسبه معیارها استفاده شده است.

```
import cv2
import os
from random import choice
from google.colab import drive
import numpy as np
import matplotlib.pyplot as plt
from skimage.exposure import equalize_hist, equalize_adapthist, histogram
import patoolib
from sklearn.manifold import TSNE
from tensorflow.keras.applications import EfficientNetB0, DenseNet121, Xception, vgg16, resnet50
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from tensorflow.keras import Sequential, Model
from tensorflow.keras.layers import Dense, Input, GlobalAveragePooling2D, BatchNormalization, Dropout, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical, plot_model
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score, confusion_matrix,\
 classification_report, plot_confusion_matrix, roc_curve, precision_score, recall_score
```

توابع کمکی و مورد نیاز

در ابتدای کار توابع عمومی مورد نیاز خود را تعریف می‌کنیم:

1. تابع EarlyStopping – این تابع در شبکه‌های عصبی کمک میکند شبکه Overfit نشود، بدین صورت که با گرفتن تعداد تکرار متوالی برای بررسی و معیاری خاص آن را بررسی می‌کند که اگر آن معیار بعد از تعداد تکرار متوالی مشخص شده بهبود نیابد آموزش را متوقف می‌کند.

```
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
```

2. بهینه ساز Adam – این تابع در واقع یک بهینه ساز است که اساس کار آموزش شبکه عصبی است.

```
optimizer = Adam(learning_rate=0.0001)
```

3. تابع ShowImage – این تابع به کمک کتابخانه Matplotlib آدرس عکس را گرفته و آن را نمایش می‌دهد.

```
def ShowImage(*, img, title):
    plt.title(title)
    plt.imshow(img)
    plt.show()
```

4. تابع plot_hist – این تابع history آموزش یک مدل را گرفته و نمودارهای روند دقت و loss آن را نمایش می‌دهد.

```
def plot_hist(hist):
    plt.plot(hist.history["accuracy"])
    plt.plot(hist.history["val_accuracy"])
    plt.title("model accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()

    plt.plot(hist.history["loss"])
    plt.plot(hist.history["val_loss"])
    plt.title("model loss")
    plt.ylabel("loss")
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()
```

5. تابع ReadImage – این تابع آدرس عکس را گرفته با کمک کتابخانه OpenCV آن را باز می‌کند و عکس را برمی‌گرداند.

```
def ReadImage(*, Path):
    src = cv2.imread(Path)
    return src
```

6. تابع ResizeImage – این تابع یک عکس و ابعاد مورد نیاز را گرفته و عکس را به ابعاد داده شده در می‌آورد.

```
def ResizeImage(*, img, width, height):
    resized = cv2.resize(img, (width, height), interpolation = cv2.INTER_AREA)
    return resized
```

7. تابع NormalizeImage – این تابع یک عکس را گرفته و آن را به بازه (0,1) می‌برد.

```
def NormalizeImage(*, img)
    norm_image = cv2.normalize(img, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
    return norm_image
```

8. تابع GaussianBlurImage – این تابع عکس را گرفته و تاری گوسی را روی آن اعمال می‌کند.

```
def GaussianBlurImage(*, img, kernel):
    GBlur = cv2.GaussianBlur(img, kernel, cv2.BORDER_DEFAULT)
    return GBlur
```

9. تابع DrawHistogram – این تابع عکس را گرفته و تبدیل به بردار کرده و نمودار هیستوگرام(میله‌ای) آن را رسم می‌کند.

```
def DrawHistogram(*, img, range, title):
    plt.hist(img.flat, bins=100, range=range)
    plt.title(title)
    plt.show()
```

10. تابع CLAHE – این تابع عملیات CLAHE(Contrast Limited Adaptive Histogram Equalization) را روی عکس اعمال می‌کند.

```
def CLAHE(*, img, cliplimit, kernel):
    cl_img = equalize_adapthist(img[:,:,0], clip_limit=cliplimit, kernel_size=kernel)
    return np.dstack((cl_img, img[:, :, 1], img[:, :, 2]))
```

لود کردن دیتاست و پیش پردازش

در ابتدای کار دیتاست را آنزیپ می‌کنیم

```
patoolib.extract_archive('/content/drive/MyDrive/Skin Cancer-Malignant-Benign.zip')
```

سپس مشابه پوشه‌بندی دیتاست یک پوشه‌بندی ایجاد می‌کنیم تا دیتای پیش پزدازش شده را درون آن نگهداری کنیم.

```
ScaledDataRoot = '/content/ScaledData/'

ScaledDataTrain = '/content/ScaledData/train'
ScaledDataTest = '/content/ScaledData/test'

ScaledDataTrainBenign = '/content/ScaledData/train/benign'
ScaledDataTrainMalignant = '/content/ScaledData/train/malignant'

ScaledDataTestBenign = '/content/ScaledData/test/benign'
ScaledDataTestMalignant = '/content/ScaledData/test/malignant'

if os.path.isdir(ScaledDataRoot)== False:
    os.mkdir(ScaledDataRoot)

    os.mkdir(ScaledDataTrain)
    os.mkdir(ScaledDataTest)

    os.mkdir(ScaledDataTrainBenign)
    os.mkdir(ScaledDataTrainMalignant)

    os.mkdir(ScaledDataTestBenign)
    os.mkdir(ScaledDataTestMalignant)
```

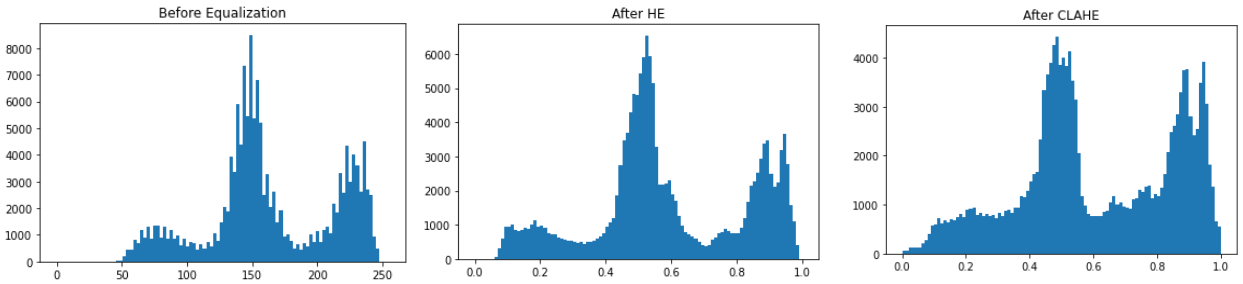
در این مرحله تابع پیش پردازش دیتاست را پیاده سازی می‌کنیم. طبق گفته مقاله ابتدا تمامی عکس‌ها باید به ابعاد 224×224 بروند سپس نرمالایز شده و بعد از آن تاری گوسی رو آن‌ها اعمال شود. در نهایت نیز عملیات CLAHE روی آن‌ها اعمال می‌شود.

```python
def PreprocessImages(*, DataPath, ScaledPath, ImagesWidth, ImagesHeight):
  print(f'Preprocess on Folder {DataPath} ... :', end=' ')
  for image in os.listdir(DataPath):
      try:
        src = ReadImage(Path=os.path.join(DataPath, image))
        resized = ResizeImage(img=src, width=ImagesWidth, height=ImagesHeight)
        norm_image = NormalizeImage(img=resized)
        GBlur = GaussianBlurImage(img=norm_image, kernel=(5,5))
        cl_img = CLAHE(img=GBlur, cliplimit=2.0, kernel=(8,8))
        plt.imsave(f'{ScaledPath}/{image}', cl_img)
      except Exception as e:
        print(e)
        print(f'There is a Problem in Processing image {os.path.join(DataPath, image)}')
  print('Done!')
```

```
Preprocess on Folder /content/Skin Cancer-Malignant-Benign/train/benign ... : Done!
Preprocess on Folder /content/Skin Cancer-Malignant-Benign/train/malignant ... : Done!
Preprocess on Folder /content/Skin Cancer-Malignant-Benign/test/benign ... : Done!
Preprocess on Folder /content/Skin Cancer-Malignant-Benign/test/malignant ... : Done!
```

پس از انجام پیش پردازش و ذخیره دیتاست حاصل، بصورت رندوم یکی از عکس ها را انتخاب کرده و نمودار هیستوگرام مرحله به مرحله آنرا در طول پیش پردازش نمایش می‌دهیم.

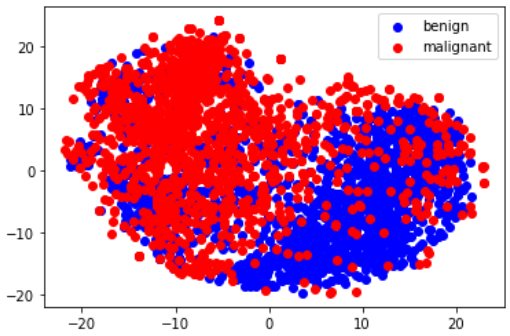
```python
image = choice(os.listdir(folder_benign_train))
src = ReadImage(Path=os.path.join(folder_benign_train, image))
DrawHistogram(range=(0,255), img=src, title='Before Equalization')
resized = ResizeImage(img=src, width=224, height=224)
norm_image = NormalizeImage(img=resized)
GBlur = GaussianBlurImage(img=norm_image, kernel=(5,5))
DrawHistogram(range=(0,1), img=GBlur, title='After HE')
cl_img = CLAHE(img=GBlur, cliplimit=2.0, kernel=(8,8))
DrawHistogram(range=(0,1), img=cl_img, title='After CLAHE')
```



همچنین نمودار t-SNE دیتاست را رسم می‌کنیم.(هر نقطه یک عکس است که به به فضای دو بعدی نگاشت شده است) در واقع این نمودار نشان می‌دهد که نمونه‌ها چقدر درهم آمیخته شده و جدا سازی آن‌ها سخت است.

```python
Data = []
label = []

for image in os.listdir(folder_benign_train):
    src = ReadImage(Path=os.path.join(folder_benign_train, image))
    Data.append(src.flatten())
    label.append(0)

for image in os.listdir(folder_malignant_train):
    src = ReadImage(Path=os.path.join(folder_malignant_train, image))
    Data.append(src.flatten())
    label.append(1)

X = np.array(Data)
y = np.array(label)
```

```
X_embedded = TSNE(n_components=2, learning_rate='auto', init='random', perplexity=90.0,).fit_transform(X)

plt.scatter([X_embedded[i][0] for i in range(len(X_embedded)) if label[i]==0], [X_embedded[i][1] for i in range(len(X_embedded)) if label[i]==0],
            color='blue', label='benign')
plt.scatter([X_embedded[i][0] for i in range(len(X_embedded)) if label[i]==1], [X_embedded[i][1] for i in range(len(X_embedded)) if label[i]==1],
            color='red', label='malignant')

plt.legend()
plt.show()
```



حال در این مرحله دیتاست پیش پردازش شده را لود می‌کنیم. همزمان با لود کردن، طبق گفته مقاله باید عملیات دیتا افزونی(Data Augmentation) نیز انجام شود. به منظور این کار از flipping استفاده میشود همچنین ویژگی fill_mode باید روی nearest تنظیم شود.

```
train_gen = ImageDataGenerator(
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')

test_gen = ImageDataGenerator()
```

```
train_ds = train_gen.flow_from_directory('/content/ScaledData/train',target_size=(224, 224),
                                         batch_size=16, shuffle=True)
test_ds = test_gen.flow_from_directory('/content/ScaledData/test',target_size=(224, 224),
                                       batch_size=16, shuffle=True)
```

طراحی و آموزش مدل‌ها

برای طراحی مدل پیشنهادی از سه مدل معروف EfficientNetB0، DenseNet121 و Xception استفاده می‌کنیم. این سه مدل در کنار هم تشکیل بردار ویژگی برای هر عکس را داده و یک شبکه Fully Connected در نهایت از روی بردار ویژگی کار تشخیص را انجام می‌دهد. هر سه مدل بصورت آماده در کتابخانه TensorFlow وجود دارند. آن‌ها را لود کرده و از وزن‌های شبکه آموزش داده شده روی دیتاست معروف ImageNet استفاده می‌کنیم و سپس آن‌ها را آپدیت می‌کنیم.(این همان مفهوم یادگیری انتقالی است)

- **Efficient Net B0**

ابتدا این شبکه را از کتابخانه TensorFlow لود کرده سپس لایه ها ابتدایی و انتهایی آن را حذف کرده و لایه ابتدایی را با توجه به ابعاد عکس های دیتاست، 224×224 در نظر می‌گیریم. همچنین در لایه آخر با توجه به دو کلاسه بودن مسئله و گفته مقاله از دو نورون همراه با تابع فعال‌ساز Sigmoid استفاده شده است.

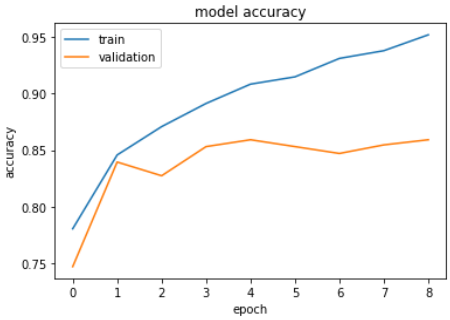
```
inputs = Input(shape=(224, 224, 3))
ModelEfficientNetB0 = EfficientNetB0(include_top=False, input_tensor=inputs, weights="imagenet")

x = GlobalAveragePooling2D(name="avg_pool")(ModelEfficientNetB0.output)
x = BatchNormalization()(x)

outputs = Dense(2, activation="sigmoid", name="pred")(x)
ModelEfficientNetB0.trainable = True
```

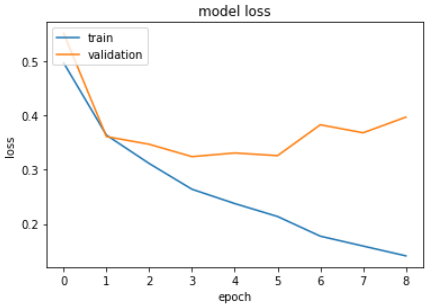
سپس مدل را با تابع هزینه binary cross entropy و بهینه ساز Adam کامپایل کرده و سپس با 30 تکرار و سایز دسته 16 (batch Size = 16) آموزش می‌دهیم. همچنین با کمک تابع EarlyStopping مراقب هستیم مدل Overfit نشود.

```
ModelEfficientNetB0 = Model(inputs, outputs, name="EfficientNet-B0")
ModelEfficientNetB0.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=["accuracy"])

hist = ModelEfficientNetB0.fit(train_ds, epochs=30, validation_data=test_ds, verbose=2, batch_size=16, callbacks=[early_stopping])
plot_hist(hist)
```

```
Epoch 1/30
165/165 - 47s - loss: 0.4974 - accuracy: 0.7804 - val_loss: 0.5520 - val_accuracy: 0.7470 - 47s/epoch - 286ms/step
Epoch 2/30
165/165 - 27s - loss: 0.3640 - accuracy: 0.8457 - val_loss: 0.3612 - val_accuracy: 0.8394 - 27s/epoch - 163ms/step
Epoch 3/30
165/165 - 27s - loss: 0.3115 - accuracy: 0.8707 - val_loss: 0.3470 - val_accuracy: 0.8273 - 27s/epoch - 165ms/step
Epoch 4/30
165/165 - 28s - loss: 0.2640 - accuracy: 0.8912 - val_loss: 0.3242 - val_accuracy: 0.8530 - 28s/epoch - 167ms/step
Epoch 5/30
165/165 - 28s - loss: 0.2375 - accuracy: 0.9082 - val_loss: 0.3310 - val_accuracy: 0.8591 - 28s/epoch - 168ms/step
Epoch 6/30
165/165 - 28s - loss: 0.2137 - accuracy: 0.9147 - val_loss: 0.3260 - val_accuracy: 0.8530 - 28s/epoch - 167ms/step
Epoch 7/30
165/165 - 28s - loss: 0.1773 - accuracy: 0.9310 - val_loss: 0.3831 - val_accuracy: 0.8470 - 28s/epoch - 168ms/step
Epoch 8/30
165/165 - 28s - loss: 0.1590 - accuracy: 0.9378 - val_loss: 0.3683 - val_accuracy: 0.8545 - 28s/epoch - 171ms/step
Epoch 9/30
165/165 - 28s - loss: 0.1409 - accuracy: 0.9518 - val_loss: 0.3971 - val_accuracy: 0.8591 - 28s/epoch - 168ms/step
Epoch 9: early stopping
```

مدل در تکرار 9ام متوقف شد و دقت 95 درصدی روی داده های آموزشی و دقت تقریبا 86 درصدی روی داده‌هایی که ندیده بود داشت.





- **Dense Net 121**

ابتدا این شبکه را از کتابخانه TensorFlow لود کرده سپس لایه ها ابتدایی و انتهایی آنرا حذف کرده و لایه ابتدایی را با توجه به ابعاد عکس های دیتاست، 224×224 در نظر می‌گیریم. همچنین در لایه آخر با توجه به دو کلاسه بودن مسئله و گفته مقاله از دو نورون همراه با تابع فعال‌ساز Sigmoid استفاده شده است.

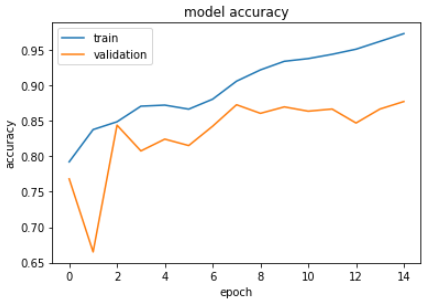
```
ModelDenseNet121 = DenseNet121(include_top=False, input_shape=(224,224,3), weights="imagenet")
x = Flatten()(ModelDenseNet121.output)

outputDenseNet121 = Dense(2, activation='sigmoid')(x)

ModelDenseNet121.trainable = True
```

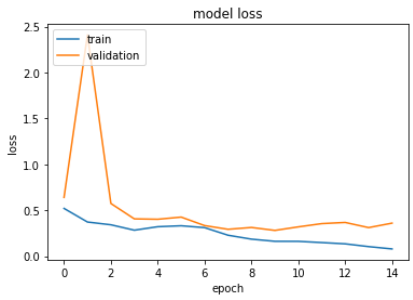
سپس مدل را با تابع هزینه binary cross entropy و بهینه ساز Adam کامپایل کرده و سپس با 30 تکرار و سایز دسته 16 (batch Size = 16) آموزش می‌دهیم. همچنین با کمک تابع EarlyStopping مراقب هستیم مدل Overfit نشود.

```
ModelDenseNet121 = Model(inputs=ModelDenseNet121.input, outputs=outputDenseNet121, name="DenseNet-121")
ModelDenseNet121.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=["accuracy"])

hist = ModelDenseNet121.fit(train_ds, epochs=30, validation_data=test_ds, verbose=2, batch_size=16, callbacks=[early_stopping])
plot_hist(hist)
```

```
Epoch 1/30
165/165 - 50s - loss: 0.5192 - accuracy: 0.7922 - val_loss: 0.6394 - val_accuracy: 0.7682 - 50s/epoch - 301ms/step
Epoch 2/30
165/165 - 31s - loss: 0.3711 - accuracy: 0.8377 - val_loss: 2.4091 - val_accuracy: 0.6652 - 31s/epoch - 191ms/step
Epoch 3/30
165/165 - 32s - loss: 0.3410 - accuracy: 0.8487 - val_loss: 0.5729 - val_accuracy: 0.8439 - 32s/epoch - 194ms/step
Epoch 4/30
165/165 - 32s - loss: 0.2820 - accuracy: 0.8707 - val_loss: 0.4053 - val_accuracy: 0.8076 - 32s/epoch - 195ms/step
Epoch 5/30
165/165 - 32s - loss: 0.3200 - accuracy: 0.8722 - val_loss: 0.4011 - val_accuracy: 0.8242 - 32s/epoch - 193ms/step
Epoch 6/30
165/165 - 32s - loss: 0.3303 - accuracy: 0.8665 - val_loss: 0.4246 - val_accuracy: 0.8152 - 32s/epoch - 195ms/step
Epoch 7/30
165/165 - 32s - loss: 0.3099 - accuracy: 0.8805 - val_loss: 0.3323 - val_accuracy: 0.8424 - 32s/epoch - 195ms/step
Epoch 8/30
165/165 - 32s - loss: 0.2272 - accuracy: 0.9060 - val_loss: 0.2918 - val_accuracy: 0.8727 - 32s/epoch - 196ms/step
Epoch 9/30
165/165 - 32s - loss: 0.1845 - accuracy: 0.9219 - val_loss: 0.3117 - val_accuracy: 0.8606 - 32s/epoch - 195ms/step
Epoch 10/30
165/165 - 32s - loss: 0.1610 - accuracy: 0.9340 - val_loss: 0.2796 - val_accuracy: 0.8697 - 32s/epoch - 194ms/step
Epoch 11/30
165/165 - 32s - loss: 0.1605 - accuracy: 0.9378 - val_loss: 0.3182 - val_accuracy: 0.8636 - 32s/epoch - 195ms/step
Epoch 12/30
165/165 - 32s - loss: 0.1472 - accuracy: 0.9439 - val_loss: 0.3538 - val_accuracy: 0.8667 - 32s/epoch - 194ms/step
Epoch 13/30
165/165 - 32s - loss: 0.1334 - accuracy: 0.9511 - val_loss: 0.3664 - val_accuracy: 0.8470 - 32s/epoch - 195ms/step
Epoch 14/30
165/165 - 32s - loss: 0.1030 - accuracy: 0.9621 - val_loss: 0.3102 - val_accuracy: 0.8667 - 32s/epoch - 195ms/step
Epoch 15/30
165/165 - 32s - loss: 0.0779 - accuracy: 0.9731 - val_loss: 0.3590 - val_accuracy: 0.8773 - 32s/epoch - 196ms/step
Epoch 15: early stopping
```

مدل در تکرار 15ام متوقف شد و دقت 97 درصدی روی داده های آموزشی و دقت تقریبا 88 درصدی روی داده‌هایی که ندیده بود داشت.





- **Xception**

ابتدا این شبکه را از کتابخانه TensorFlow لود کرده سپس لایه ها ابتدایی و انتهایی آنرا حذف کرده و لایه ابتدایی را با توجه به ابعاد عکس های دیتاست، 224×224 در نظر می‌گیریم. همچنین در لایه آخر با توجه به دو کلاسه بودن مسئله و گفته مقاله از دو نورون همراه با تابع فعال‌ساز Sigmoid استفاده شده است.

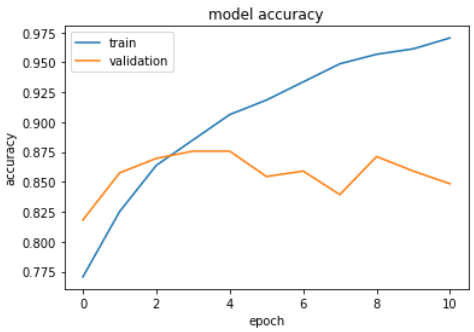
```
ModelXception = Xception(include_top=False, input_shape=(224,224,3), weights="imagenet")
x = Flatten()(ModelXception.output)
outputXception = Dense(2, activation='sigmoid')(x)

ModelXception.trainable = True
```

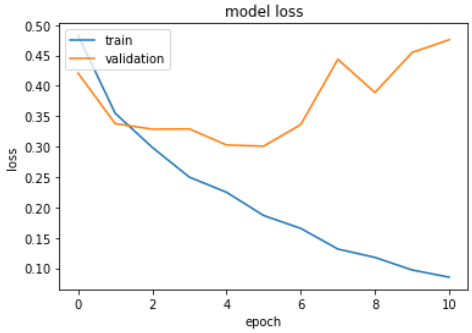
سپس مدل را با تابع هزینه binary cross entropy و بهینه ساز Adam کامپایل کرده و سپس با 30 تکرار و سایز دسته 16 (batch Size = 16) آموزش می‌دهیم. همچنین با کمک تابع EarlyStopping مراقب هستیم مدل Overfit نشود.

```
ModelXception = Model(inputs=ModelXception.input, outputs=outputXception, name="Xception")
ModelXception.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=["accuracy"])

hist = ModelXception.fit(train_ds, epochs=30, validation_data=test_ds, verbose=2, batch_size=16, callbacks=[early_stopping])
plot_hist(hist)
```

```
Epoch 1/30
165/165 - 60s - loss: 0.4829 - accuracy: 0.7706 - val_loss: 0.4208 - val_accuracy: 0.8182 - 60s/epoch - 366ms/step
Epoch 2/30
165/165 - 52s - loss: 0.3549 - accuracy: 0.8252 - val_loss: 0.3377 - val_accuracy: 0.8576 - 52s/epoch - 316ms/step
Epoch 3/30
165/165 - 52s - loss: 0.2987 - accuracy: 0.8639 - val_loss: 0.3286 - val_accuracy: 0.8697 - 52s/epoch - 313ms/step
Epoch 4/30
165/165 - 52s - loss: 0.2497 - accuracy: 0.8851 - val_loss: 0.3291 - val_accuracy: 0.8758 - 52s/epoch - 314ms/step
Epoch 5/30
165/165 - 52s - loss: 0.2249 - accuracy: 0.9063 - val_loss: 0.3027 - val_accuracy: 0.8758 - 52s/epoch - 314ms/step
Epoch 6/30
165/165 - 52s - loss: 0.1867 - accuracy: 0.9185 - val_loss: 0.3007 - val_accuracy: 0.8545 - 52s/epoch - 313ms/step
Epoch 7/30
165/165 - 52s - loss: 0.1655 - accuracy: 0.9336 - val_loss: 0.3361 - val_accuracy: 0.8591 - 52s/epoch - 314ms/step
Epoch 8/30
165/165 - 52s - loss: 0.1315 - accuracy: 0.9488 - val_loss: 0.4437 - val_accuracy: 0.8394 - 52s/epoch - 313ms/step
Epoch 9/30
165/165 - 52s - loss: 0.1177 - accuracy: 0.9568 - val_loss: 0.3890 - val_accuracy: 0.8712 - 52s/epoch - 314ms/step
Epoch 10/30
165/165 - 52s - loss: 0.0971 - accuracy: 0.9613 - val_loss: 0.4548 - val_accuracy: 0.8591 - 52s/epoch - 313ms/step
Epoch 11/30
165/165 - 52s - loss: 0.0852 - accuracy: 0.9704 - val_loss: 0.4758 - val_accuracy: 0.8485 - 52s/epoch - 314ms/step
Epoch 11: early stopping
```

مدل در تکرار 11ام متوقف شد و دقت 97 درصدی روی داده های آموزشی و دقت تقریبا 85 درصدی روی داده‌هایی که ندیده بود داشت.





ساخت دیتاست با استفاده از مدل‌ها

در این مرحله مدل‌ها روی دیتاست پیش پردازش شده پیش‌بینی کرده و به ازای هر مدل یک زوج مرتب بدست می‌آید. با کنار هم قرار دادن زوج های مرتب حاصل، یک بردار 6 تایی به ازای هر عکس بدست می‌آید که آن ها را ذخیره می‌کنیم. دیتاست حاصل یک ماتریس از نوع numpy array است.

```
yPredEfficientNetB0 = ModelEfficientNetB0.predict(train_ds_Unshuffle)
yPredDenseNet121 = ModelDenseNet121.predict(train_ds_Unshuffle)
yPredXception = ModelXception.predict(train_ds_Unshuffle)

FeaturedDataSet = np.concatenate([yPredEfficientNetB0, yPredDenseNet121, yPredXception], axis=1)
FeaturedDataSet.shape
```

```
(2637, 6)
```

ساخت و آموزش Meta Learner

در این بخش یک شبکه ساده Fully Connected طراحی می‌شود که روی دیتاست تولید شده از پیش‌بینی مدل‌ها آموزش می‌بیند.

```
MetaLearner = Sequential()
MetaLearner.add(Dense(8, input_dim=6, activation='relu'))
MetaLearner.add(Dense(4, activation='relu'))
MetaLearner.add(Dense(1, activation='sigmoid'))
```

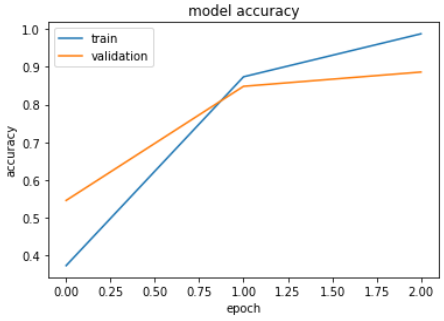
این مدل را نیز همانند مدل های قبلی با تابع هزینه binary cross entropy کامپایل کرده و با 3 تکرار و سایز دسته 16 آموزش می‌دهیم.

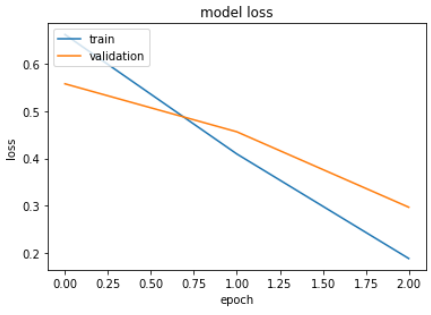
```
MetaLearner.compile(optimizer='adam', loss="binary_crossentropy", metrics=["accuracy"])
```

```
hist = MetaLearner.fit(FeaturedDataSet, train_ds_Unshuffle.labels, epochs=3,
                       validation_data=(FeaturedDataSetTest, test_ds_Unshuffle.labels), verbose=1, batch_size=16)
plot_hist(hist)
```

```
Epoch 1/3
165/165 [==============================] - 1s 4ms/step - loss: 0.6629 - accuracy: 0.3724 - val_loss: 0.5584 - val_accuracy: 0.5455
Epoch 2/3
165/165 [==============================] - 1s 3ms/step - loss: 0.4095 - accuracy: 0.8737 - val_loss: 0.4565 - val_accuracy: 0.8485
Epoch 3/3
165/165 [==============================] - 1s 4ms/step - loss: 0.1878 - accuracy: 0.9879 - val_loss: 0.2966 - val_accuracy: 0.8864
```

در اینجا نیز دقت تقریبا 99 درصدی روی داده ها آموزشی و دقت 88 درصدی روی دیتاست تست داریم.

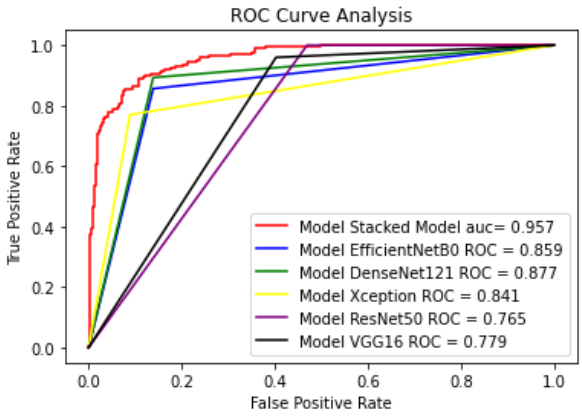




ارزیابی و مقایسه

در این بخش مدل پیشنهادی را با مدل‌های Efficient Net B0، Dense Net 121، Xception، Res Net 50 و VGG16 مقایسه می‌کنیم.(تمامی مدل‌ها به روش بیان شده پیاده سازی شده‌اند)

Model Name	Accuracy	F1 Score	AUC Score	Precision	Recall	Specificity
Stacked Model	0.89	0.87	0.88	0.88	0.86	0.91
Efficient Net B0	0.86	0.85	0.86	0.84	0.86	0.86
Dense Net 121	0.88	0.87	0.88	0.84	0.89	0.86
Xception	0.85	0.82	0.84	0.88	0.77	0.91
Res Net50	0.74	0.78	0.77	0.64	1.00	0.53
VGG16	0.76	0.79	0.78	0.67	0.96	0.6



همانطور که مشاهده میشود مدل پیشنهادی بجز Recall در بقیه معیار ها از بقیه مدل ها بالاتر است.