```
pip install tensorflow
```

```
   ------------------------------- -------- 4.4/5.5 MB 1.9 MB/s eta 0:00:01
   -------------------------------- ------- 4.5/5.5 MB 1.9 MB/s eta 0:00:01
   --------------------------------- ------ 4.6/5.5 MB 1.9 MB/s eta 0:00:01
   ---------------------------------- ----- 4.7/5.5 MB 1.9 MB/s eta 0:00:01
   ---------------------------------- ----- 4.8/5.5 MB 1.9 MB/s eta 0:00:01
   ----------------------------------- ---- 4.9/5.5 MB 1.9 MB/s eta 0:00:01
   ------------------------------------ --- 5.0/5.5 MB 1.9 MB/s eta 0:00:01
   ------------------------------------ --- 5.1/5.5 MB 1.9 MB/s eta 0:00:01
   ------------------------------------- -- 5.2/5.5 MB 1.9 MB/s eta 0:00:01
   -------------------------------------- - 5.3/5.5 MB 1.9 MB/s eta 0:00:01
   ---------------------------------------  5.4/5.5 MB 1.9 MB/s eta 0:00:01
   ---------------------------------------  5.5/5.5 MB 1.9 MB/s eta 0:00:01
   ---------------------------------------- 5.5/5.5 MB 1.9 MB/s eta 0:00:00
Downloading tensorflow_estimator-2.15.0-py2.py3-none-any.whl (441 kB)
   ---------------------------------------- 0.0/442.0 kB ? eta -:--:--
   ---------- ----------------------------- 112.6/442.0 kB 3.3 MB/s eta 0:00:01
   -------------------- ------------------- 225.3/442.0 kB 2.8 MB/s eta 0:00:01
   ----------------------------- ---------- 327.7/442.0 kB 2.3 MB/s eta 0:00:01
   ---------------------------------------- 442.0/442.0 kB 2.3 MB/s eta 0:00:00
Downloading tensorflow_io_gcs_filesystem-0.31.0-cp311-cp311-win_amd64.whl (1.5 MB)
   ---------------------------------------- 0.0/1.5 MB ? eta -:--:--
   -- ------------------------------------- 0.1/1.5 MB 5.1 MB/s eta 0:00:01
   ----- ---------------------------------- 0.2/1.5 MB 2.9 MB/s eta 0:00:01
   ------- -------------------------------- 0.3/1.5 MB 2.9 MB/s eta 0:00:01
   ----------- ---------------------------- 0.4/1.5 MB 2.7 MB/s eta 0:00:01
   --------------- ------------------------ 0.6/1.5 MB 2.7 MB/s eta 0:00:01
   ----------------- ---------------------- 0.7/1.5 MB 2.7 MB/s eta 0:00:01
   -------------------- ------------------- 0.8/1.5 MB 2.6 MB/s eta 0:00:01
   ---------------------- ----------------- 0.9/1.5 MB 2.6 MB/s eta 0:00:01
   ---------------------- ----------------- 0.9/1.5 MB 2.6 MB/s eta 0:00:01
   --------------------------- ------------ 1.1/1.5 MB 2.4 MB/s eta 0:00:01
   ----------------------------- ---------- 1.1/1.5 MB 2.3 MB/s eta 0:00:01
   ------------------------------ --------- 1.2/1.5 MB 2.2 MB/s eta 0:00:01
   -------------------------------- ------- 1.3/1.5 MB 2.1 MB/s eta 0:00:01
   ---------------------------------- ----- 1.3/1.5 MB 2.1 MB/s eta 0:00:01
   ------------------------------------- -- 1.4/1.5 MB 2.0 MB/s eta 0:00:01
   ---------------------------------------  1.5/1.5 MB 2.0 MB/s eta 0:00:01
   ---------------------------------------- 1.5/1.5 MB 1.9 MB/s eta 0:00:00
Using cached termcolor-2.4.0-py3-none-any.whl (7.7 kB)
Downloading google_auth-2.28.1-py2.py3-none-any.whl (186 kB)
   ---------------------------------------- 0.0/186.9 kB ? eta -:--:--
   -------- ------------------------------- 41.0/186.9 kB 1.9 MB/s eta 0:00:01
   ----------------------- ---------------- 112.6/186.9 kB 1.7 MB/s eta 0:00:01
   ---------------------------------------- 186.9/186.9 kB 1.4 MB/s eta 0:00:00
Downloading google_auth_oauthlib-1.2.0-py2.py3-none-any.whl (24 kB)
Using cached tensorboard_data_server-0.7.2-py3-none-any.whl (2.4 kB)
Downloading requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Downloading rsa-4.9-py3-none-any.whl (34 kB)
Downloading oauthlib-3.2.2-py3-none-any.whl (151 kB)
   ---------------------------------------- 0.0/151.7 kB ? eta -:--:--
   ---------- ----------------------------- 41.0/151.7 kB 2.0 MB/s eta 0:00:01
   -------------------------------- ------- 122.9/151.7 kB 1.4 MB/s eta 0:00:01
   ---------------------------------------- 151.7/151.7 kB 1.3 MB/s eta 0:00:00
Installing collected packages: libclang, flatbuffers, termcolor, tensorflow-io-gcs-filesystem, tensorflow-estimator, tensorboar
Successfully installed absl-py-2.1.0 astunparse-1.6.3 flatbuffers-23.5.26 gast-0.5.4 google-auth-2.28.1 google-auth-oauthlib-1.
Note: you may need to restart the kernel to use updated packages.
```

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Activation, Dropout, BatchNormalization
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
from skimage import transform
```

```
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy
```

```
photo_size=224
```

```
def prepare_dataset(data_dir):
    datagen = ImageDataGenerator(
        rescale= 1/255,
        rotation_range=40,
        width_shift_range=.2,
        height_shift_range=.2,
        shear_range=.1,
        horizontal_flip=True,
        fill_mode='nearest',
        zoom_range=.2,
```

```
    )
    generator = datagen.flow_from_directory(
        data_dir,
        target_size=(photo_size,photo_size),
        class_mode='binary',
        batch_size=128,
        classes=['non_autistic','autistic']
    )
    return generator


train_data=prepare_dataset('E:/Apps/VS Code/Projects Data Set/ASD Prediction/train')
validation_data = prepare_dataset('E:/Apps/VS Code/Projects Data Set/ASD Prediction/valid')
test_data=prepare_dataset('E:/Apps/VS Code/Projects Data Set/ASD Prediction/test')
```

```
Found 2536 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Found 300 images belonging to 2 classes.
```

```
validation_data.class_indices
```

```
{'non_autistic': 0, 'autistic': 1}
```

VGG16

```
def create_vgg_model():
    from keras.applications.vgg16 import VGG16
    from keras.models import Model
    from keras.layers import Dense
    from keras.layers import Flatten
    # load model without classifier layers
    model = VGG16(include_top=False, input_shape=(photo_size, photo_size, 3))
    for layer_idx in range(len(model.layers)):
        if layer_idx not in [1,2,3,15,16,17,18]:
            model.layers[layer_idx].trainable = False
    # add new classifier layers
    flat1 = Flatten()(model.layers[-1].output)
    class1 = Dense(256, activation='relu')(flat1)
    output = Dense(95, activation='softmax')(class1)
    # define new model
    model = Model(inputs=model.inputs, outputs=output)
    return model


vgg_model=create_vgg_model()
vgg_model.summary()
```

```
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_fun

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_po

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
```

```
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 95)                24415

=================================================================
Total params: 21161887 (80.73 MB)
Trainable params: 13565343 (51.75 MB)
```

```
vgg_model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
vgghist=vgg_model.fit(
    train_data,
    epochs=100,
    validation_data=validation_data
)
vgg_model.save("vgg_model50.h5")
```

```
Epoch 1/100
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValu

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eage

20/20 [==============================] - 1222s 60s/step - loss: 1.7027 - accuracy: 0.4685 - val_loss: 0.7166 - val_accuracy: 0.
Epoch 2/100
20/20 [==============================] - 1240s 62s/step - loss: 0.6836 - accuracy: 0.5761 - val_loss: 0.7125 - val_accuracy: 0.
Epoch 3/100
20/20 [==============================] - 991s 49s/step - loss: 0.6698 - accuracy: 0.5856 - val_loss: 0.6717 - val_accuracy: 0.5
Epoch 4/100
20/20 [==============================] - 855s 42s/step - loss: 0.6450 - accuracy: 0.6179 - val_loss: 0.6290 - val_accuracy: 0.6
Epoch 5/100
20/20 [==============================] - 762s 38s/step - loss: 0.6373 - accuracy: 0.6317 - val_loss: 0.6331 - val_accuracy: 0.6
Epoch 6/100
20/20 [==============================] - 797s 39s/step - loss: 0.6341 - accuracy: 0.6234 - val_loss: 0.5837 - val_accuracy: 0.6
Epoch 7/100
20/20 [==============================] - 737s 37s/step - loss: 0.6117 - accuracy: 0.6538 - val_loss: 0.6103 - val_accuracy: 0.6
Epoch 8/100
20/20 [==============================] - 727s 36s/step - loss: 0.5677 - accuracy: 0.7121 - val_loss: 0.6033 - val_accuracy: 0.6
Epoch 9/100
20/20 [==============================] - 768s 38s/step - loss: 0.6230 - accuracy: 0.6696 - val_loss: 0.5910 - val_accuracy: 0.6
Epoch 10/100
20/20 [==============================] - 788s 40s/step - loss: 0.5453 - accuracy: 0.7228 - val_loss: 0.6068 - val_accuracy: 0.7
Epoch 11/100
20/20 [==============================] - 815s 41s/step - loss: 0.5472 - accuracy: 0.7263 - val_loss: 0.5899 - val_accuracy: 0.6
Epoch 12/100
20/20 [==============================] - 864s 43s/step - loss: 0.5229 - accuracy: 0.7386 - val_loss: 0.5528 - val_accuracy: 0.7
Epoch 13/100
20/20 [==============================] - 799s 40s/step - loss: 0.5295 - accuracy: 0.7397 - val_loss: 0.5650 - val_accuracy: 0.6
Epoch 14/100
20/20 [==============================] - 769s 38s/step - loss: 0.5228 - accuracy: 0.7366 - val_loss: 0.5422 - val_accuracy: 0.7
Epoch 15/100
20/20 [==============================] - 845s 43s/step - loss: 0.5198 - accuracy: 0.7362 - val_loss: 0.5531 - val_accuracy: 0.7
Epoch 16/100
20/20 [==============================] - 681s 34s/step - loss: 0.5031 - accuracy: 0.7441 - val_loss: 0.5441 - val_accuracy: 0.7
Epoch 17/100
20/20 [==============================] - 685s 34s/step - loss: 0.4786 - accuracy: 0.7717 - val_loss: 0.5183 - val_accuracy: 0.7
Epoch 18/100
20/20 [==============================] - 695s 35s/step - loss: 0.4876 - accuracy: 0.7709 - val_loss: 0.5172 - val_accuracy: 0.7
Epoch 19/100
20/20 [==============================] - 693s 35s/step - loss: 0.4900 - accuracy: 0.7654 - val_loss: 0.5763 - val_accuracy: 0.6
Epoch 20/100
20/20 [==============================] - 709s 35s/step - loss: 0.4758 - accuracy: 0.7681 - val_loss: 0.5389 - val_accuracy: 0.7
Epoch 21/100
20/20 [==============================] - 688s 34s/step - loss: 0.4581 - accuracy: 0.7796 - val_loss: 0.4600 - val_accuracy: 0.7
Epoch 22/100
20/20 [==============================] - 677s 34s/step - loss: 0.4496 - accuracy: 0.7906 - val_loss: 0.5538 - val_accuracy: 0.7
Epoch 23/100
20/20 [==============================] - 1052s 54s/step - loss: 0.4628 - accuracy: 0.7812 - val_loss: 0.4696 - val_accuracy: 0.
Epoch 24/100
20/20 [==============================] - 1410s 71s/step - loss: 0.4504 - accuracy: 0.7942 - val_loss: 0.5459 - val_accuracy: 0.
Epoch 25/100
20/20 [==============================] - 1416s 71s/step - loss: 0.4555 - accuracy: 0.7800 - val_loss: 0.5061 - val_accuracy: 0.
Epoch 26/100
20/20 [==============================] - 1290s 64s/step - loss: 0.4346 - accuracy: 0.7981 - val_loss: 0.4688 - val_accuracy: 0.
```

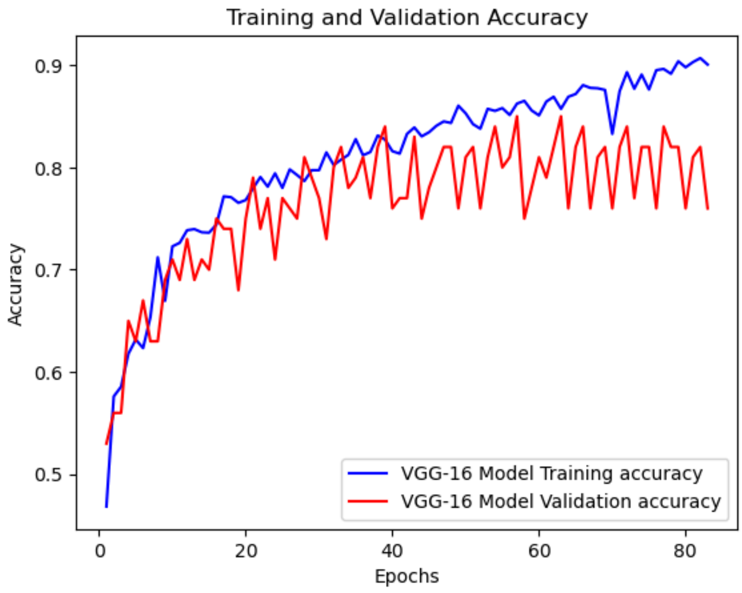
```
from keras.models import load_model

loaded_model = load_model("vgg_model50.h5")
```

```
vgg_model.evaluate(test_data)
```

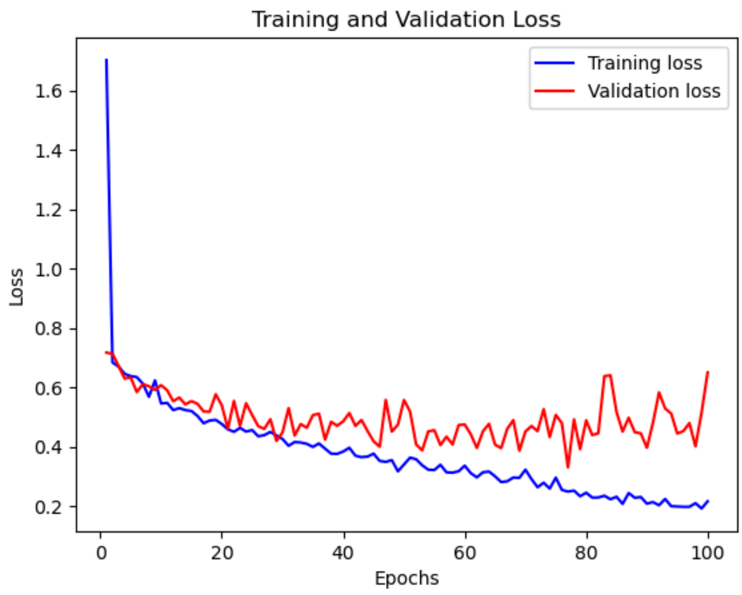
```
import pickle

# Save training history
with open('vgghist.pkl', 'wb') as f:
    pickle.dump(vgghist.history, f)

# Repeat for other models (inception_model, efficient_net)
```

```
import matplotlib.pyplot as plt

# Get the training history
training_loss = vgghist.history['loss']
validation_loss = vgghist.history['val_loss']
training_accuracy = vgghist.history['accuracy']
validation_accuracy = vgghist.history['val_accuracy']

# Plot the training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
import matplotlib.pyplot as plt

# Truncate the train_accuracy list to match the length of val_accuracy list
train_accuracy_truncated = train_accuracy[:len(val_accuracy)]

epochs = range(1, len(train_accuracy_truncated) + 1)

# Plotting the training and validation accuracy
plt.plot(epochs, train_accuracy_truncated, 'b', label='VGG-16 Model Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='VGG-16 Model Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```




```
import matplotlib.pyplot as plt

# Extracted loss values from the provided output
train_loss = [1.7027, 0.6836, 0.6698, 0.6450, 0.6373, 0.6341, 0.6117, 0.5677, 0.6230, 0.5453,
              0.5472, 0.5229, 0.5295, 0.5228, 0.5198, 0.5031, 0.4786, 0.4876, 0.4900, 0.4758,
              0.4581, 0.4496, 0.4628, 0.4504, 0.4555, 0.4346, 0.4384, 0.4493, 0.4371, 0.4254,
              0.4025, 0.4152, 0.4136, 0.4091, 0.3988, 0.4103, 0.3930, 0.3761, 0.3753, 0.3832,
              0.3958, 0.3692, 0.3642, 0.3661, 0.3760, 0.3514, 0.3484, 0.3541, 0.3165, 0.3395,
              0.3626, 0.3572, 0.3368, 0.3220, 0.3208, 0.3386, 0.3131, 0.3120, 0.3166, 0.3354,
              0.3103, 0.2961, 0.3133, 0.3157, 0.2995, 0.2804, 0.2828, 0.2955, 0.2945, 0.3219,
              0.2901, 0.2629, 0.2780, 0.2584, 0.2951, 0.2538, 0.2482, 0.2515, 0.2324, 0.2442,
              0.2280, 0.2282, 0.2342, 0.2223, 0.2307, 0.2067, 0.2433, 0.2273, 0.2298, 0.2075,
              0.2127, 0.2022, 0.2233, 0.1989, 0.1979, 0.1969, 0.1969, 0.2090, 0.1912, 0.2153]

val_loss = [0.7166, 0.7125, 0.6717, 0.6290, 0.6331, 0.5837, 0.6103, 0.6033, 0.5910, 0.6068,
            0.5899, 0.5528, 0.5650, 0.5422, 0.5531, 0.5441, 0.5183, 0.5172, 0.5763, 0.5389,
            0.4600, 0.5538, 0.4696, 0.5459, 0.5061, 0.4688, 0.4596, 0.4919, 0.4190, 0.4477,
            0.5297, 0.4373, 0.4757, 0.4631, 0.5061, 0.5105, 0.4235, 0.4833, 0.4700, 0.4853,
            0.5130, 0.4697, 0.4896, 0.4522, 0.4176, 0.3995, 0.5571, 0.4504, 0.4736, 0.5569,
            0.5179, 0.4072, 0.3870, 0.4511, 0.4550, 0.4051, 0.4334, 0.4069, 0.4723, 0.4744,
            0.4412, 0.3950, 0.4513, 0.4766, 0.4065, 0.3947, 0.4595, 0.4889, 0.3858, 0.4509,
            0.4692, 0.4519, 0.5261, 0.4320, 0.5065, 0.4789, 0.3300, 0.4914, 0.3909, 0.4883,
            0.4386, 0.4448, 0.6374, 0.6400, 0.5163, 0.4505, 0.4970, 0.4484, 0.4444, 0.3960,
            0.4801, 0.5824, 0.5285, 0.5112, 0.4439, 0.4520, 0.4790, 0.4006, 0.5140, 0.6498]

# Number of epochs
epochs = range(1, len(train_loss) + 1)

# Plotting the training and validation loss
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Inception Net

```
def create_inception_model():
    from tensorflow.keras.applications.inception_v3 import InceptionV3
    base_model = InceptionV3(input_shape = (photo_size, photo_size, 3), include_top = False, weights = 'imagenet')
    for layer in base_model.layers:
        layer.trainable = False
    # for layer_idx in range(len(pretrained_model.layers)):
    #     if layer_idx not in [1,2,3,305,306,307,308,309,310]:
    #         pretrained_model.layers[layer_idx].trainable = False
    from tensorflow.keras.optimizers import RMSprop
    from tensorflow.keras import layers
    x = layers.Flatten()(base_model.output)
    x = layers.Dense(512, activation='relu')(x)
    x = layers.Dropout(0.2)(x)

    # Add a final sigmoid layer with 1 node for classification output
    x = layers.Dense(95, activation='softmax')(x)
    model = tf.keras.models.Model(base_model.input, x)
    model.compile(optimizer = RMSprop(learning_rate=0.0001), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
    return model
```

```
inception_model=create_inception_model()
inception_model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]        0         []

 conv2d (Conv2D)             (None, 111, 111, 32)         864       ['input_2[0][0]']

 batch_normalization (Batch  (None, 111, 111, 32)         96        ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, 111, 111, 32)         0         ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)           (None, 109, 109, 32)         9216      ['activation[0][0]']

 batch_normalization_1 (Bat  (None, 109, 109, 32)         96        ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 109, 109, 32)         0         ['batch_normalization_1[0][0]'
                                                                    ]

 conv2d_2 (Conv2D)           (None, 109, 109, 64)         18432     ['activation_1[0][0]']

 batch_normalization_2 (Bat  (None, 109, 109, 64)         192       ['conv2d_2[0][0]']
 chNormalization)

 activation_2 (Activation)   (None, 109, 109, 64)         0         ['batch_normalization_2[0][0]'
                                                                    ]

 max_pooling2d (MaxPooling2  (None, 54, 54, 64)           0         ['activation_2[0][0]']
 D)

 conv2d_3 (Conv2D)           (None, 54, 54, 80)           5120      ['max_pooling2d[0][0]']
```

```
 batch_normalization_3 (Bat  (None, 54, 54, 80)          240       ['conv2d_3[0][0]']
 chNormalization)

 activation_3 (Activation)   (None, 54, 54, 80)          0         ['batch_normalization_3[0][0]'
                                                                    ]

 conv2d_4 (Conv2D)           (None, 52, 52, 192)         138240    ['activation_3[0][0]']

 batch_normalization_4 (Bat  (None, 52, 52, 192)         576       ['conv2d_4[0][0]']
 chNormalization)

 activation_4 (Activation)   (None, 52, 52, 192)         0         ['batch_normalization_4[0][0]'
                                                                    ]

 max_pooling2d_1 (MaxPoolin  (None, 25, 25, 192)         0         ['activation_4[0][0]']
 g2D)

 conv2d_8 (Conv2D)           (None, 25, 25, 64)          12288     ['max_pooling2d_1[0][0]']

 batch_normalization_8 (Bat  (None, 25, 25, 64)          192       ['conv2d_8[0][0]']
 chNormalization)

 activation_8 (Activation)   (None, 25, 25, 64)          0         ['batch_normalization_8[0][0]'
                                                                    ]
```

```
incephist = inception_model.fit(
    train_data,
    epochs=100,
    validation_data=validation_data
)
inception_model.save("inception_model.h5")
```

```
Epoch 1/100
20/20 [==============================] - 84s 4s/step - loss: 0.4384 - accuracy: 0.7827 - val_loss: 0.5027 - val_accuracy: 0.740
Epoch 2/100
20/20 [==============================] - 96s 5s/step - loss: 0.4559 - accuracy: 0.7741 - val_loss: 0.5094 - val_accuracy: 0.700
Epoch 3/100
20/20 [==============================] - 118s 6s/step - loss: 0.4525 - accuracy: 0.7772 - val_loss: 0.5451 - val_accuracy: 0.73
Epoch 4/100
20/20 [==============================] - 109s 5s/step - loss: 0.4382 - accuracy: 0.7843 - val_loss: 0.5177 - val_accuracy: 0.70
Epoch 5/100
20/20 [==============================] - 105s 5s/step - loss: 0.4434 - accuracy: 0.7784 - val_loss: 0.5991 - val_accuracy: 0.74
Epoch 6/100
20/20 [==============================] - 104s 5s/step - loss: 0.4421 - accuracy: 0.7776 - val_loss: 0.4990 - val_accuracy: 0.74
Epoch 7/100
20/20 [==============================] - 109s 5s/step - loss: 0.4384 - accuracy: 0.7961 - val_loss: 0.5295 - val_accuracy: 0.74
Epoch 8/100
20/20 [==============================] - 109s 5s/step - loss: 0.4509 - accuracy: 0.7808 - val_loss: 0.5195 - val_accuracy: 0.72
Epoch 9/100
20/20 [==============================] - 102s 5s/step - loss: 0.4379 - accuracy: 0.7815 - val_loss: 0.4594 - val_accuracy: 0.75
Epoch 10/100
20/20 [==============================] - 110s 5s/step - loss: 0.4349 - accuracy: 0.7969 - val_loss: 0.4830 - val_accuracy: 0.82
Epoch 11/100
20/20 [==============================] - 105s 5s/step - loss: 0.4380 - accuracy: 0.7910 - val_loss: 0.4959 - val_accuracy: 0.75
Epoch 12/100
20/20 [==============================] - 108s 5s/step - loss: 0.4282 - accuracy: 0.7930 - val_loss: 0.4741 - val_accuracy: 0.84
Epoch 13/100
20/20 [==============================] - 108s 5s/step - loss: 0.4320 - accuracy: 0.7875 - val_loss: 0.4980 - val_accuracy: 0.74
Epoch 14/100
20/20 [==============================] - 104s 5s/step - loss: 0.4200 - accuracy: 0.7946 - val_loss: 0.5343 - val_accuracy: 0.72
Epoch 15/100
20/20 [==============================] - 108s 5s/step - loss: 0.4235 - accuracy: 0.7981 - val_loss: 0.5539 - val_accuracy: 0.73
Epoch 16/100
20/20 [==============================] - 113s 6s/step - loss: 0.4232 - accuracy: 0.7930 - val_loss: 0.5314 - val_accuracy: 0.77
Epoch 17/100
20/20 [==============================] - 96s 5s/step - loss: 0.4152 - accuracy: 0.8021 - val_loss: 0.4975 - val_accuracy: 0.750
Epoch 18/100
20/20 [==============================] - 110s 5s/step - loss: 0.4255 - accuracy: 0.7997 - val_loss: 0.5186 - val_accuracy: 0.75
Epoch 19/100
20/20 [==============================] - 105s 5s/step - loss: 0.4150 - accuracy: 0.7946 - val_loss: 0.6189 - val_accuracy: 0.72
Epoch 20/100
20/20 [==============================] - 106s 5s/step - loss: 0.4220 - accuracy: 0.7993 - val_loss: 0.4831 - val_accuracy: 0.73
Epoch 21/100
20/20 [==============================] - 103s 5s/step - loss: 0.4298 - accuracy: 0.7953 - val_loss: 0.4633 - val_accuracy: 0.73
Epoch 22/100
20/20 [==============================] - 106s 5s/step - loss: 0.4218 - accuracy: 0.8013 - val_loss: 0.5487 - val_accuracy: 0.76
Epoch 23/100
20/20 [==============================] - 105s 5s/step - loss: 0.4352 - accuracy: 0.7882 - val_loss: 0.4383 - val_accuracy: 0.80
Epoch 24/100
20/20 [==============================] - 105s 5s/step - loss: 0.4185 - accuracy: 0.8021 - val_loss: 0.4659 - val_accuracy: 0.76
Epoch 25/100
20/20 [==============================] - 108s 5s/step - loss: 0.4078 - accuracy: 0.8166 - val_loss: 0.5467 - val_accuracy: 0.72
Epoch 26/100
20/20 [==============================] - 101s 5s/step - loss: 0.4175 - accuracy: 0.7985 - val_loss: 0.4297 - val_accuracy: 0.77
Epoch 27/100
20/20 [==============================] - 114s 6s/step - loss: 0.4063 - accuracy: 0.8111 - val_loss: 0.4988 - val_accuracy: 0.77
Epoch 28/100
20/20 [==============================] - 107s 5s/step - loss: 0.4025 - accuracy: 0.8119 - val_loss: 0.4450 - val_accuracy: 0.79
```

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Extract the epoch values from the history
epochs = np.arange(1, len(incephist.history['accuracy']) + 1)

# Plot training accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.plot(epochs, incephist.history['accuracy'], label="Training Accuracy", linestyle="-", color='blue')
plt.legend()

# Plot training loss
plt.subplot(1, 2, 2)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(epochs, incephist.history['loss'], label="Training Loss", linestyle="-", color='red')
plt.legend()

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

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Extract the epoch values from the history
epochs = np.arange(1, len(incephist.history['accuracy']) + 1)

# Plot lines
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.plot(epochs, incephist.history['accuracy'], label="InceptionV3 Model Training", linestyle="-")
plt.plot(epochs, incephist.history['val_accuracy'], label="InceptionV3 Model Validation", linestyle="-")
plt.legend()
plt.show()
```


```
inception_model.evaluate(test_data)
```

```
3/3 [==============================] - 9s 3s/step - loss: 0.3711 - accuracy: 0.8133
[0.3711494207382202, 0.8133333325386047]
```

Efficient Net - B7

```
!pip install efficientnet
```

```
Collecting efficientnet
  Downloading efficientnet-1.1.1-py3-none-any.whl.metadata (6.4 kB)
Collecting keras-applications<=1.0.8,>=1.0.7 (from efficientnet)
```

```
  Downloading Keras_Applications-1.0.8-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: scikit-image in e:\apps\anaconda\lib\site-packages (from efficientnet) (0.22.0)
Requirement already satisfied: numpy>=1.9.1 in e:\apps\anaconda\lib\site-packages (from keras-applications<=1.0.8,>=1.0.7->efficient
Requirement already satisfied: h5py in e:\apps\anaconda\lib\site-packages (from keras-applications<=1.0.8,>=1.0.7->efficientnet) (3
Requirement already satisfied: scipy>=1.8 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (1.11.4)
Requirement already satisfied: networkx>=2.8 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (3.1)
Requirement already satisfied: pillow>=9.0.1 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (10.2.0)
Requirement already satisfied: imageio>=2.27 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (2.33.1)
Requirement already satisfied: tifffile>=2022.8.12 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (2023.4.1
Requirement already satisfied: packaging>=21 in c:\users\chada\appdata\roaming\python\python311\site-packages (from scikit-image->ef
Requirement already satisfied: lazy_loader>=0.3 in e:\apps\anaconda\lib\site-packages (from scikit-image->efficientnet) (0.3)
Downloading efficientnet-1.1.1-py3-none-any.whl (18 kB)
Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
   ---------------------------------------- 0.0/50.7 kB ? eta -:--:--
   ------------------------ --------------- 30.7/50.7 kB 640.0 kB/s eta 0:00:01
   ---------------------------------------- 50.7/50.7 kB 518.8 kB/s eta 0:00:00
Installing collected packages: keras-applications, efficientnet
Successfully installed efficientnet-1.1.1 keras-applications-1.0.8
```

```python
from tensorflow.keras.optimizers import RMSprop
from keras.models import Model
import efficientnet.tfkeras as efn
from tensorflow.keras.optimizers import RMSprop

def use_efficient_net(model_type='B0'):
    if model_type == 'B0':
        efn_model = efn.EfficientNetB0(input_shape=(photo_size, photo_size, 3), include_top=False, weights='imagenet')
    else:
        efn_model = efn.EfficientNetB7(input_shape=(photo_size, photo_size, 3), include_top=False, weights='imagenet')

    for layer in efn_model.layers:
        layer.trainable = False

    x = efn_model.output
    x = Flatten()(x)
    x = Dense(256, activation="relu")(x)
    x = Dense(256, activation="relu")(x)
    x = Dropout(0.5)(x)
    predictions = Dense(1, activation="sigmoid")(x)

    efficient_net = Model(efn_model.input, predictions)
    efficient_net.compile(RMSprop(learning_rate=0.0001, rho=0.9), loss='binary_crossentropy', metrics=['accuracy'])

    return efficient_net

efficient_net = use_efficient_net('B0')
efficient_net.summary()
```

```
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_fun

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]        0         []

 stem_conv (Conv2D)          (None, 112, 112, 32)         864       ['input_1[0][0]']

 stem_bn (BatchNormalizatio  (None, 112, 112, 32)         128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 112, 112, 32)         0         ['stem_bn[0][0]']
 n)

 block1a_dwconv (DepthwiseC  (None, 112, 112, 32)         288       ['stem_activation[0][0]']
 onv2D)

 block1a_bn (BatchNormaliza  (None, 112, 112, 32)         128       ['block1a_dwconv[0][0]']
 tion)

 block1a_activation (Activa  (None, 112, 112, 32)         0         ['block1a_bn[0][0]']
 tion)

 block1a_se_squeeze (Global  (None, 32)                   0         ['block1a_activation[0][0]']
 AveragePooling2D)

 block1a_se_reshape (Reshap  (None, 1, 1, 32)             0         ['block1a_se_squeeze[0][0]']
 e)

 block1a_se_reduce (Conv2D)  (None, 1, 1, 8)              264       ['block1a_se_reshape[0][0]']
```

```
block1a_se_expand (Conv2D)  (None, 1, 1, 32)          288       ['block1a_se_reduce[0][0]']

block1a_se_excite (Multipl  (None, 112, 112, 32)      0         ['block1a_activation[0][0]',
y)                                                               'block1a_se_expand[0][0]']

block1a_project_conv (Conv  (None, 112, 112, 16)      512       ['block1a_se_excite[0][0]']
2D)

block1a_project_bn (BatchN  (None, 112, 112, 16)      64        ['block1a_project_conv[0][0]']
ormalization)

block2a_expand_conv (Conv2  (None, 112, 112, 96)      1536      ['block1a_project_bn[0][0]']
D)

block2a_expand_bn (BatchNo  (None, 112, 112, 96)      384       ['block2a_expand_conv[0][0]']
rmalization)

block2a_expand_activation   (None, 112, 112, 96)      0         ['block2a_expand_bn[0][0]']
(Activation)
```

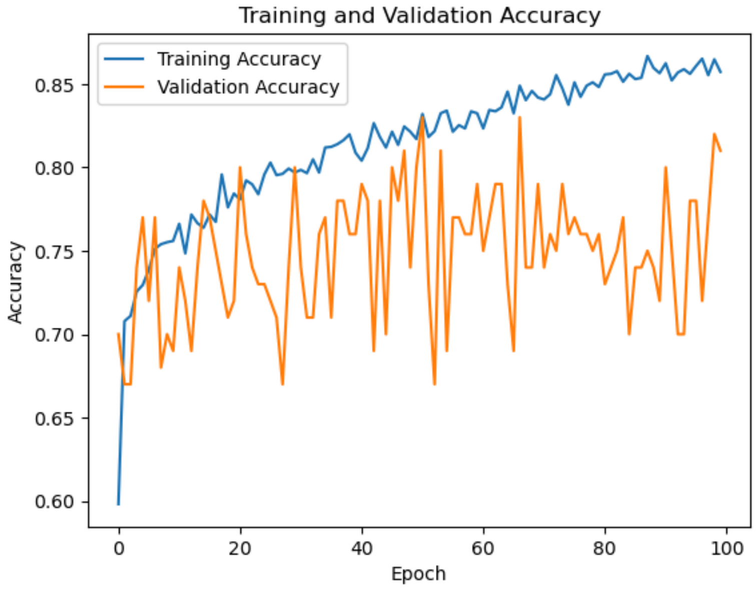
```
effb0_history = efficient_net.fit(train_data, validation_data = validation_data, epochs = 100)
efficient_net.save("efficient_net_B0_model.h5")
```

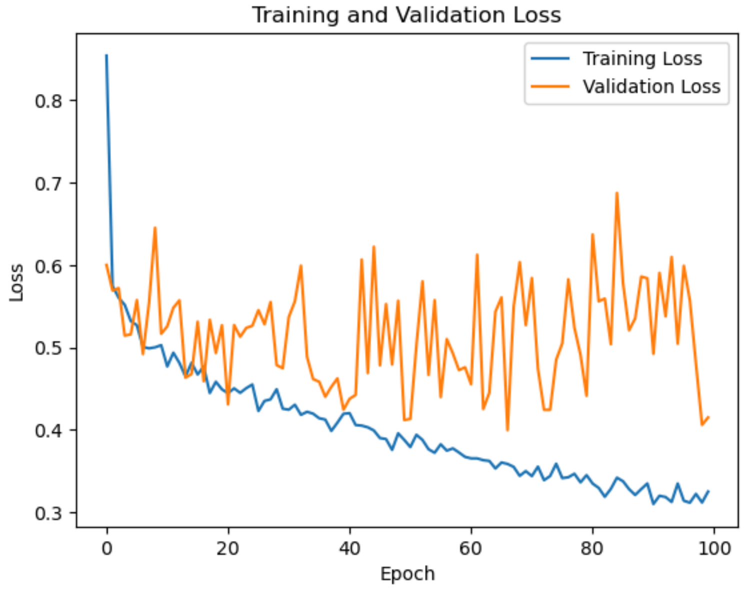
```
Epoch 1/100
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValu

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eage

20/20 [==============================] - 126s 6s/step - loss: 0.8543 - accuracy: 0.5982 - val_loss: 0.6002 - val_accuracy: 0.70
Epoch 2/100
20/20 [==============================] - 75s 4s/step - loss: 0.5746 - accuracy: 0.7078 - val_loss: 0.5691 - val_accuracy: 0.670
Epoch 3/100
20/20 [==============================] - 75s 4s/step - loss: 0.5603 - accuracy: 0.7110 - val_loss: 0.5720 - val_accuracy: 0.670
Epoch 4/100
20/20 [==============================] - 76s 4s/step - loss: 0.5517 - accuracy: 0.7256 - val_loss: 0.5143 - val_accuracy: 0.740
Epoch 5/100
20/20 [==============================] - 75s 4s/step - loss: 0.5320 - accuracy: 0.7295 - val_loss: 0.5160 - val_accuracy: 0.770
Epoch 6/100
20/20 [==============================] - 75s 4s/step - loss: 0.5270 - accuracy: 0.7386 - val_loss: 0.5575 - val_accuracy: 0.720
Epoch 7/100
20/20 [==============================] - 75s 4s/step - loss: 0.5007 - accuracy: 0.7512 - val_loss: 0.4920 - val_accuracy: 0.770
Epoch 8/100
20/20 [==============================] - 75s 4s/step - loss: 0.4989 - accuracy: 0.7539 - val_loss: 0.5540 - val_accuracy: 0.680
Epoch 9/100
20/20 [==============================] - 75s 4s/step - loss: 0.5001 - accuracy: 0.7551 - val_loss: 0.6453 - val_accuracy: 0.700
Epoch 10/100
20/20 [==============================] - 76s 4s/step - loss: 0.5028 - accuracy: 0.7559 - val_loss: 0.5166 - val_accuracy: 0.690
Epoch 11/100
20/20 [==============================] - 75s 4s/step - loss: 0.4770 - accuracy: 0.7662 - val_loss: 0.5257 - val_accuracy: 0.740
Epoch 12/100
20/20 [==============================] - 74s 4s/step - loss: 0.4936 - accuracy: 0.7484 - val_loss: 0.5482 - val_accuracy: 0.720
Epoch 13/100
20/20 [==============================] - 75s 4s/step - loss: 0.4812 - accuracy: 0.7717 - val_loss: 0.5572 - val_accuracy: 0.690
Epoch 14/100
20/20 [==============================] - 75s 4s/step - loss: 0.4649 - accuracy: 0.7666 - val_loss: 0.4633 - val_accuracy: 0.740
Epoch 15/100
20/20 [==============================] - 84s 4s/step - loss: 0.4814 - accuracy: 0.7638 - val_loss: 0.4678 - val_accuracy: 0.780
Epoch 16/100
20/20 [==============================] - 76s 4s/step - loss: 0.4672 - accuracy: 0.7717 - val_loss: 0.5313 - val_accuracy: 0.770
Epoch 17/100
20/20 [==============================] - 74s 4s/step - loss: 0.4769 - accuracy: 0.7674 - val_loss: 0.4590 - val_accuracy: 0.750
Epoch 18/100
20/20 [==============================] - 74s 4s/step - loss: 0.4446 - accuracy: 0.7957 - val_loss: 0.5336 - val_accuracy: 0.730
Epoch 19/100
20/20 [==============================] - 74s 4s/step - loss: 0.4584 - accuracy: 0.7760 - val_loss: 0.4932 - val_accuracy: 0.710
Epoch 20/100
20/20 [==============================] - 74s 4s/step - loss: 0.4491 - accuracy: 0.7843 - val_loss: 0.5270 - val_accuracy: 0.720
Epoch 21/100
20/20 [==============================] - 75s 4s/step - loss: 0.4445 - accuracy: 0.7808 - val_loss: 0.4310 - val_accuracy: 0.800
Epoch 22/100
20/20 [==============================] - 74s 4s/step - loss: 0.4504 - accuracy: 0.7922 - val_loss: 0.5272 - val_accuracy: 0.760
Epoch 23/100
20/20 [==============================] - 74s 4s/step - loss: 0.4449 - accuracy: 0.7898 - val_loss: 0.5130 - val_accuracy: 0.740
Epoch 24/100
20/20 [==============================] - 74s 4s/step - loss: 0.4506 - accuracy: 0.7839 - val_loss: 0.5237 - val_accuracy: 0.730
Epoch 25/100
20/20 [==============================] - 74s 4s/step - loss: 0.4550 - accuracy: 0.7957 - val_loss: 0.5263 - val_accuracy: 0.730
Epoch 26/100
20/20 [==============================] - 74s 4s/step - loss: 0.4229 - accuracy: 0.8028 - val_loss: 0.5453 - val_accuracy: 0.720
```

```
efficient_net.evaluate(test_data)
```

```
3/3 [==============================] - 11s 4s/step - loss: 0.4204 - accuracy: 0.8300
[0.42044728994369507, 0.8299999833106995]
```

```
import matplotlib.pyplot as plt

# Extracting training history
history = effb0_history.history

# Plotting training and validation accuracy
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting training and validation loss
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Efficient Net - B0 (Hybrid Model)

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import efficientnet.tfkeras as efn
import pickle

# Define custom EfficientNet B0 model with additional layers
def create_custom_efficientnet_b0():
    # Load pre-trained EfficientNet B0 model
    base_model = efn.EfficientNetB0(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

    # Freeze all layers in the base model
    base_model.trainable = False

    # Add custom classification head
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation='sigmoid')(x)

    # Combine base model and custom classification head into a new model
    custom_model = Model(inputs=base_model.input, outputs=output)

    # Compile the model
    custom_model.compile(optimizer=RMSprop(learning_rate=0.0001, rho=0.9),
                         loss='binary_crossentropy',
                         metrics=['accuracy'])

    return custom_model

# Create custom EfficientNet B0 model
efficientnet_b0_custom = create_custom_efficientnet_b0()

# Define directories for training and validation data
train_dir = 'E:/Apps/VS Code/Projects Data Set/ASD Prediction/train'
validation_dir = 'E:/Apps/VS Code/Projects Data Set/ASD Prediction/valid'

# Define image data generators with data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# For validation, only rescale the pixel values
validation_datagen = ImageDataGenerator(rescale=1./255)

# Define batch size
batch_size = 64

# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # Resize images to match input shape of EfficientNet B0
    batch_size=batch_size,
    class_mode='binary'  # Assuming binary classification (change if necessary)
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary'
)

# Unfreeze some layers in the base model for fine-tuning
for layer in efficientnet_b0_custom.layers[-20:]:
    layer.trainable = True

# Lower the learning rate
custom_optimizer = RMSprop(learning_rate=0.0001)

# Compile the model with the new optimizer
efficientnet_b0_custom.compile(optimizer=custom_optimizer,
                               loss='binary_crossentropy',
                               metrics=['accuracy'])

# Train the model with increased epochs
```

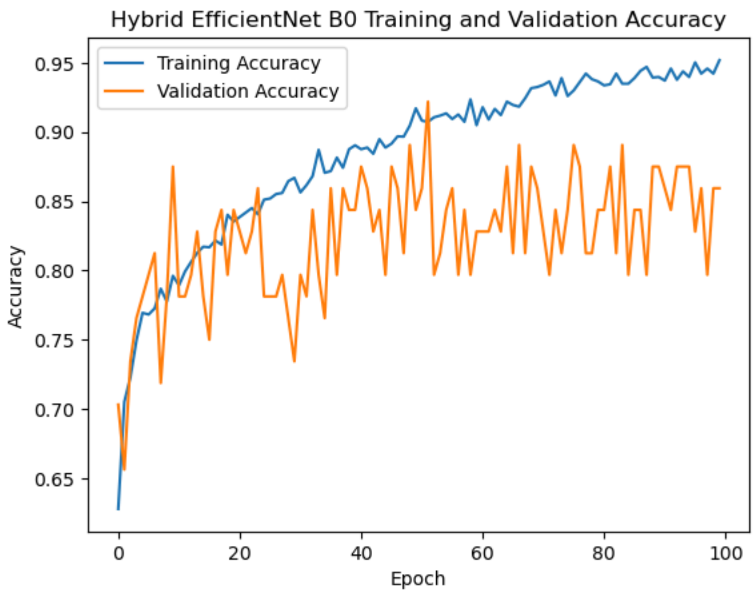
```
history = efficientnet_b0_custom.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=100,  # Increase number of epochs
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size

)
# Save the trained model as a pickle file
with open('efficientnet_b0_custom_model.pkl', 'wb') as f:
  pickle.dump(efficientnet_b0_custom, f)

# Save the trained model as 'hybrid_model.h5'
efficientnet_b0_custom.save('hybrid_model.h5')
from tensorflow import keras
loaded_model = keras.models.load_model('hybrid_model.h5')
efficientnet_b0_custom.save('hybrid_model.keras')
```

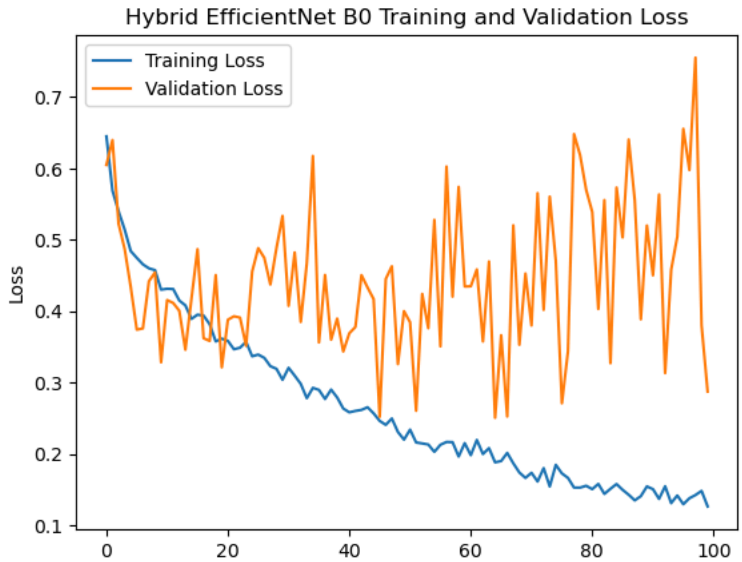
```
Found 2536 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Epoch 1/100
39/39 [==============================] - 80s 2s/step - loss: 0.6449 - accuracy: 0.6278 - val_loss: 0.6049 - val_accuracy: 0.703
Epoch 2/100
39/39 [==============================] - 67s 2s/step - loss: 0.5687 - accuracy: 0.7051 - val_loss: 0.6397 - val_accuracy: 0.656
Epoch 3/100
39/39 [==============================] - 71s 2s/step - loss: 0.5400 - accuracy: 0.7229 - val_loss: 0.5226 - val_accuracy: 0.734
Epoch 4/100
39/39 [==============================] - 76s 2s/step - loss: 0.5144 - accuracy: 0.7496 - val_loss: 0.4873 - val_accuracy: 0.765
Epoch 5/100
39/39 [==============================] - 77s 2s/step - loss: 0.4843 - accuracy: 0.7694 - val_loss: 0.4341 - val_accuracy: 0.781
Epoch 6/100
39/39 [==============================] - 76s 2s/step - loss: 0.4747 - accuracy: 0.7682 - val_loss: 0.3742 - val_accuracy: 0.796
Epoch 7/100
39/39 [==============================] - 74s 2s/step - loss: 0.4657 - accuracy: 0.7727 - val_loss: 0.3758 - val_accuracy: 0.812
Epoch 8/100
39/39 [==============================] - 73s 2s/step - loss: 0.4600 - accuracy: 0.7868 - val_loss: 0.4420 - val_accuracy: 0.718
Epoch 9/100
39/39 [==============================] - 75s 2s/step - loss: 0.4575 - accuracy: 0.7775 - val_loss: 0.4543 - val_accuracy: 0.781
Epoch 10/100
39/39 [==============================] - 74s 2s/step - loss: 0.4302 - accuracy: 0.7961 - val_loss: 0.3284 - val_accuracy: 0.875
Epoch 11/100
39/39 [==============================] - 74s 2s/step - loss: 0.4314 - accuracy: 0.7892 - val_loss: 0.4156 - val_accuracy: 0.781
Epoch 12/100
39/39 [==============================] - 74s 2s/step - loss: 0.4310 - accuracy: 0.7994 - val_loss: 0.4115 - val_accuracy: 0.781
Epoch 13/100
39/39 [==============================] - 73s 2s/step - loss: 0.4150 - accuracy: 0.8062 - val_loss: 0.4006 - val_accuracy: 0.796
Epoch 14/100
39/39 [==============================] - 74s 2s/step - loss: 0.4082 - accuracy: 0.8127 - val_loss: 0.3461 - val_accuracy: 0.828
Epoch 15/100
39/39 [==============================] - 74s 2s/step - loss: 0.3892 - accuracy: 0.8172 - val_loss: 0.4173 - val_accuracy: 0.781
Epoch 16/100
39/39 [==============================] - 74s 2s/step - loss: 0.3951 - accuracy: 0.8167 - val_loss: 0.4869 - val_accuracy: 0.750
Epoch 17/100
39/39 [==============================] - 73s 2s/step - loss: 0.3938 - accuracy: 0.8216 - val_loss: 0.3622 - val_accuracy: 0.828
Epoch 18/100
39/39 [==============================] - 75s 2s/step - loss: 0.3818 - accuracy: 0.8188 - val_loss: 0.3583 - val_accuracy: 0.843
Epoch 19/100
39/39 [==============================] - 74s 2s/step - loss: 0.3576 - accuracy: 0.8402 - val_loss: 0.4506 - val_accuracy: 0.796
Epoch 20/100
39/39 [==============================] - 130s 3s/step - loss: 0.3616 - accuracy: 0.8354 - val_loss: 0.3214 - val_accuracy: 0.84
Epoch 21/100
39/39 [==============================] - 85s 2s/step - loss: 0.3584 - accuracy: 0.8386 - val_loss: 0.3880 - val_accuracy: 0.828
Epoch 22/100
39/39 [==============================] - 62s 2s/step - loss: 0.3467 - accuracy: 0.8418 - val_loss: 0.3929 - val_accuracy: 0.812
Epoch 23/100
39/39 [==============================] - 63s 2s/step - loss: 0.3488 - accuracy: 0.8451 - val_loss: 0.3910 - val_accuracy: 0.828
Epoch 24/100
39/39 [==============================] - 60s 2s/step - loss: 0.3571 - accuracy: 0.8406 - val_loss: 0.3516 - val_accuracy: 0.859
Epoch 25/100
39/39 [==============================] - 62s 2s/step - loss: 0.3368 - accuracy: 0.8511 - val_loss: 0.4555 - val_accuracy: 0.781
Epoch 26/100
39/39 [==============================] - 63s 2s/step - loss: 0.3392 - accuracy: 0.8519 - val_loss: 0.4883 - val_accuracy: 0.781
Epoch 27/100
39/39 [==============================] - 58s 1s/step - loss: 0.3346 - accuracy: 0.8552 - val_loss: 0.4747 - val_accuracy: 0.781
```

```
# Initialize lists to store epoch, loss, and accuracy values
epochs_list = []
loss_list = []
accuracy_list = []
val_loss_list = []
val_accuracy_list = []

# Loop through each epoch
for epoch in range(1, 101):  # Assuming you have 100 epochs
```

```
    # Save the epoch value to the list
    epochs_list.append(epoch)

    # Save the epoch value to a file
    with open('last_epoch.txt', 'w') as f:
        f.write(str(epoch) + '\n')

    # Output the epoch value
    print("Epoch:", epoch)

    # Retrieve loss and accuracy values from history
    loss = history.history['loss'][epoch - 1]
    accuracy = history.history['accuracy'][epoch - 1]
    val_loss = history.history['val_loss'][epoch - 1]
    val_accuracy = history.history['val_accuracy'][epoch - 1]

    # Save the values to respective lists
    loss_list.append(loss)
    accuracy_list.append(accuracy)
    val_loss_list.append(val_loss)
    val_accuracy_list.append(val_accuracy)
```

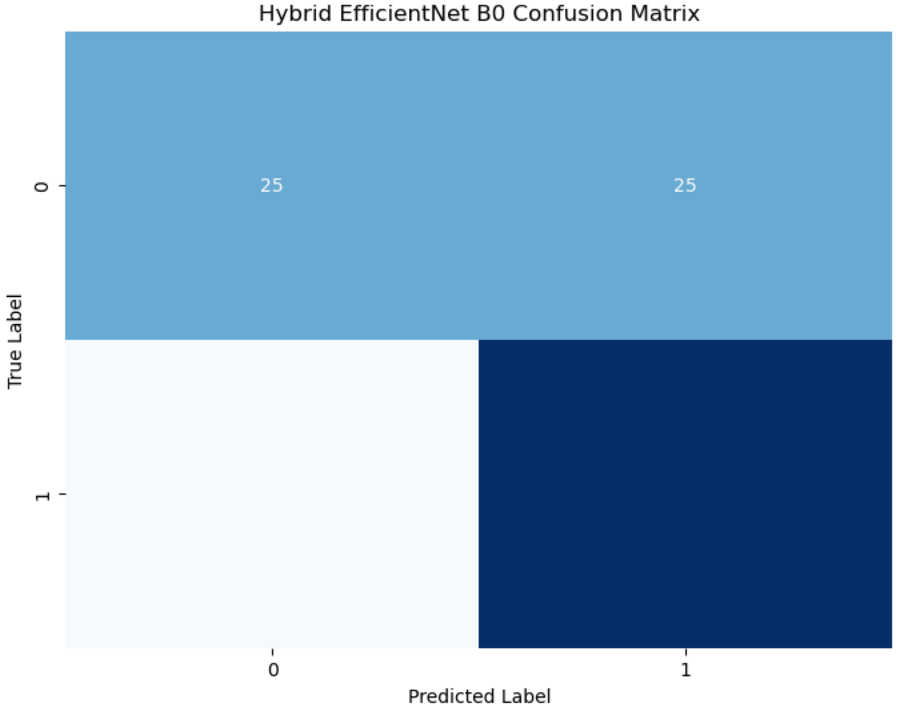
```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Hybrid EfficientNet B0 Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Hybrid EfficientNet B0 Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```






```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get the true labels from the validation generator
true_labels = validation_generator.classes

# Get the predicted labels using the trained model
predicted_scores = efficientnet_b0_custom.predict(validation_generator)
predicted_labels = np.where(predicted_scores > 0.5, 1, 0)  # Assuming binary classification

# Create the confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Hybrid EfficientNet B0 Confusion Matrix')
plt.show()
```

```
2/2 [==============================] - 3s 512ms/step
```




```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and F1 score
precision = precision_score(true_labels, predicted_labels)
recall = recall_score(true_labels, predicted_labels)
f1 = f1_score(true_labels, predicted_labels)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Precision: 0.5192307692307693
Recall: 0.54
F1 Score: 0.5294117647058824
```

```
pip install flask
```

```
Requirement already satisfied: flask in e:\apps\anaconda\lib\site-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in e:\apps\anaconda\lib\site-packages (from flask) (3.0.2)
Requirement already satisfied: Jinja2>=3.1.2 in e:\apps\anaconda\lib\site-packages (from flask) (3.1.3)
Requirement already satisfied: itsdangerous>=2.1.2 in e:\apps\anaconda\lib\site-packages (from flask) (2.1.2)
Requirement already satisfied: click>=8.1.3 in e:\apps\anaconda\lib\site-packages (from flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in e:\apps\anaconda\lib\site-packages (from flask) (1.6.2)
Requirement already satisfied: colorama in e:\apps\anaconda\lib\site-packages (from click>=8.1.3->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in e:\apps\anaconda\lib\site-packages (from Jinja2>=3.1.2->flask) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

```
from flask import Flask, render_template, request, redirect, url_for
import pickle
import cv2  # Assuming you're using OpenCV for image processing

app = Flask(__name__)

# Load the trained model
try:
    with open('efficientnet_b0_custom_model.pkl', 'rb') as f:
        model = pickle.load(f)
        print("Model loaded successfully!")
except FileNotFoundError:
    print("Error: Model file not found. Please check the path.")
    exit(1)
except Exception as e:
    print(f"Unexpected error loading model: {e}")
    exit(1)

# Define route for the upload page (index.html)
@app.route('/')
def upload_page():
    return render_template('index.html')
```

```
# Define route to handle form submission and predict output
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get the uploaded image file from the request
        try:
            img = request.files['file']
        except Exception as e:
            print(f"Error accessing uploaded file: {e}")
            return render_template('error.html', error_message="Failed to access uploaded image.")

        # Preprocess the image (adjust based on your model's requirements)
        try:
            # Read the image as RGB
            img_array = cv2.imdecode(np.fromstring(img.read(), np.uint8), cv2.IMREAD_COLOR)
            # Resize (adjust dimensions as needed)
            img_array = cv2.resize(img_array, (224, 224))  # Example for EfficientNet B0
            # Normalize pixel values (common practice)
            img_array = img_array / 255.0
        except cv2.error as e:
            print(f"Error reading image: {e}")
            return render_template('error.html', error_message="Failed to read uploaded image. Please try again.")
        except Exception as e:
            print(f"Unexpected error during preprocessing: {e}")
            return render_template('error.html', error_message="An error occurred during image processing. Please try again.")

        # Make prediction using your custom EfficientNet B0 model logic
        try:
            # Assuming your model expects a batch dimension (modify based on your model's input format)
            prediction = model.predict(np.expand_dims(img_array, axis=0))
            # Extract the predicted class (modify based on your model's output format)
            predicted_class = np.argmax(prediction)
            # Access class labels from your model training process (replace with your actual labels)
            class_labels = ["Class 1", "Class 2"]  # Replace with your class labels
            prediction_label = class_labels[predicted_class]
        except Exception as e:
            print(f"Error making prediction: {e}")
            return render_template('error.html', error_message="An error occurred during prediction. Please try again.")

        # Redirect to data_page.html with prediction result
        return redirect(url_for('show_result', result=prediction_label))

# Define route to render data_page.html with prediction result
@app.route('/result/<result>')
def show_result(result):
    return render_template('data_page.html', result=result)

# Define route for error page (error.html)
@app.route('/error')
def error():
    return render_template('error.html')

if __name__ == '__main__':
    app.run(debug=True)
```

```
Model loaded successfully!
 * Serving Flask app '__main__'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
An exception has occurred, use %tb to see the full traceback.

SystemExit: 1

C:\Users\chada\AppData\Roaming\Python\Python311\site-packages\IPython\core\interactiveshell.py:3534: UserWarning: To exit: use 'exi
```

Efficient Net - B7

```
pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in e:\apps\anaconda\lib\site-packages (from keras-tuner) (2.15.0)
Requirement already satisfied: packaging in c:\users\chada\appdata\roaming\python\python311\site-packages (from keras-tuner) (23.2)
Requirement already satisfied: requests in e:\apps\anaconda\lib\site-packages (from keras-tuner) (2.31.0)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: charset-normalizer<4,>=2 in e:\apps\anaconda\lib\site-packages (from requests->keras-tuner) (2.0.4)
```

```
Requirement already satisfied: idna<4,>=2.5 in e:\apps\anaconda\lib\site-packages (from requests->keras-tuner) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in e:\apps\anaconda\lib\site-packages (from requests->keras-tuner) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in e:\apps\anaconda\lib\site-packages (from requests->keras-tuner) (2024.2.2)
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
   ---------------------------------------- 0.0/129.1 kB ? eta -:--:--
   --- ------------------------------------ 10.2/129.1 kB ? eta -:--:--
   ------------ ------------------------- 41.0/129.1 kB 487.6 kB/s eta 0:00:01
   ---------------------------------- ---- 112.6/129.1 kB 930.9 kB/s eta 0:00:01
   -------------------------------------- 129.1/129.1 kB 951.3 kB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
Note: you may need to restart the kernel to use updated packages.
```

```python
from keras.preprocessing.image import ImageDataGenerator

def prepare_dataset(data_dir, photo_size=224, batch_size=128):
    # Original dataset size
    original_datagen = ImageDataGenerator(rescale=1./255)
    original_generator = original_datagen.flow_from_directory(
        data_dir,
        target_size=(photo_size, photo_size),
        batch_size=batch_size,
        class_mode='binary',
        shuffle=False  # Ensure order is maintained for accurate count
    )
    original_dataset_size = original_generator.samples

    # Data augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        brightness_range=[0.7, 1.3],
        channel_shift_range=50.0,
        fill_mode='nearest'
    )

    augmented_generator = train_datagen.flow_from_directory(
        data_dir,
        target_size=(photo_size, photo_size),
        batch_size=batch_size,
        class_mode='binary',
        shuffle=False  # Ensure order is maintained for accurate count
    )
    augmented_dataset_size = augmented_generator.samples

    # Validation data generator (no augmentation)
    validation_datagen = ImageDataGenerator(rescale=1./255)
    validation_generator = validation_datagen.flow_from_directory(
        data_dir,
        target_size=(photo_size, photo_size),
        batch_size=batch_size,
        class_mode='binary',
        shuffle=False  # Ensure order is maintained for accurate count
    )
    validation_dataset_size = validation_generator.samples

    print("Original dataset size:", original_dataset_size)
    print("Augmented dataset size:", augmented_dataset_size)

    return augmented_generator, validation_generator

# Specify the directory containing your dataset
data_dir = 'E:/Apps/VS Code/Projects Data Set/ASD Prediction/train'

# Call the prepare_dataset function
train_data, validation_data = prepare_dataset(data_dir)
```

```
Found 2536 images belonging to 2 classes.
Found 2536 images belonging to 2 classes.
Found 2536 images belonging to 2 classes.
Original dataset size: 2536
Augmented dataset size: 2536
```

```python
def use_efficient_net(model_type='B0'):
    from tensorflow.keras.optimizers import RMSprop
```

```
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import Flatten, Dense, Dropout
    import efficientnet.tfkeras as efn

    if model_type == 'B0':
        efn_model = efn.EfficientNetB0(input_shape=(photo_size, photo_size, 3), include_top=False, weights='imagenet')
    else:
        efn_model = efn.EfficientNetB7(input_shape=(photo_size, photo_size, 3), include_top=False, weights='imagenet')

    for layer in efn_model.layers:
        layer.trainable = False

    x = efn_model.output
    x = Flatten()(x)
    x = Dense(256, activation="relu")(x)
    x = Dense(256, activation="relu")(x)
    x = Dropout(0.5)(x)
    predictions = Dense(1, activation="sigmoid")(x)
    efficient_net = Model(efn_model.input, predictions)

    efficient_net.compile(tf.keras.optimizers.legacy.RMSprop(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
    return efficient_net

efficient_net = use_efficient_net('B7')
efficient_net.summary()
```

```
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_fun

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]        0         []

 stem_conv (Conv2D)          (None, 112, 112, 64)         1728      ['input_1[0][0]']

 stem_bn (BatchNormalizatio  (None, 112, 112, 64)         256       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 112, 112, 64)         0         ['stem_bn[0][0]']
 n)

 block1a_dwconv (DepthwiseC  (None, 112, 112, 64)         576       ['stem_activation[0][0]']
 onv2D)

 block1a_bn (BatchNormaliza  (None, 112, 112, 64)         256       ['block1a_dwconv[0][0]']
 tion)

 block1a_activation (Activa  (None, 112, 112, 64)         0         ['block1a_bn[0][0]']
 tion)

 block1a_se_squeeze (Global  (None, 64)                   0         ['block1a_activation[0][0]']
 AveragePooling2D)

 block1a_se_reshape (Reshap  (None, 1, 1, 64)             0         ['block1a_se_squeeze[0][0]']
 e)

 block1a_se_reduce (Conv2D)  (None, 1, 1, 16)             1040      ['block1a_se_reshape[0][0]']

 block1a_se_expand (Conv2D)  (None, 1, 1, 64)             1088      ['block1a_se_reduce[0][0]']

 block1a_se_excite (Multipl  (None, 112, 112, 64)         0         ['block1a_activation[0][0]',
 y)                                                                  'block1a_se_expand[0][0]']

 block1a_project_conv (Conv  (None, 112, 112, 32)         2048      ['block1a_se_excite[0][0]']
 2D)

 block1a_project_bn (BatchN  (None, 112, 112, 32)         128       ['block1a_project_conv[0][0]']
 ormalization)

 block1b_dwconv (DepthwiseC  (None, 112, 112, 32)         288       ['block1a_project_bn[0][0]']
 onv2D)

 block1b_bn (BatchNormaliza  (None, 112, 112, 32)         128       ['block1b_dwconv[0][0]']
 tion)

 block1b_activation (Activa  (None, 112, 112, 32)         0         ['block1b_bn[0][0]']
 tion)

 block1b_se_squeeze (Global  (None, 32)                   0         ['block1b_activation[0][0]']
```

```
effb7_history = efficient_net.fit(train_data, validation_data = validation_data, epochs = 50)
efficient_net.save("efficient_net_B7_model.h5")
```

```
Epoch 1/50
WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValu

WARNING:tensorflow:From e:\Apps\Anaconda\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eage

20/20 [==============================] - 574s 27s/step - loss: 1.9938 - accuracy: 0.5548 - val_loss: 0.6257 - val_accuracy: 0.6
Epoch 2/50
20/20 [==============================] - 531s 27s/step - loss: 0.9772 - accuracy: 0.5205 - val_loss: 0.6799 - val_accuracy: 0.5
Epoch 3/50
20/20 [==============================] - 530s 27s/step - loss: 0.9207 - accuracy: 0.5422 - val_loss: 0.5865 - val_accuracy: 0.6
Epoch 4/50
20/20 [==============================] - 534s 27s/step - loss: 0.8837 - accuracy: 0.5564 - val_loss: 0.5829 - val_accuracy: 0.7
Epoch 5/50
20/20 [==============================] - 511s 26s/step - loss: 0.8309 - accuracy: 0.5304 - val_loss: 0.5925 - val_accuracy: 0.6
Epoch 6/50
20/20 [==============================] - 512s 26s/step - loss: 0.8531 - accuracy: 0.6526 - val_loss: 0.7775 - val_accuracy: 0.6
Epoch 7/50
20/20 [==============================] - 513s 26s/step - loss: 0.7974 - accuracy: 0.5714 - val_loss: 0.5565 - val_accuracy: 0.7
Epoch 8/50
20/20 [==============================] - 512s 26s/step - loss: 0.7582 - accuracy: 0.6238 - val_loss: 0.5622 - val_accuracy: 0.7
Epoch 9/50
20/20 [==============================] - 512s 26s/step - loss: 0.7595 - accuracy: 0.5864 - val_loss: 0.5517 - val_accuracy: 0.7
Epoch 10/50
20/20 [==============================] - 513s 26s/step - loss: 0.6862 - accuracy: 0.6144 - val_loss: 0.5727 - val_accuracy: 0.7
Epoch 11/50
20/20 [==============================] - 512s 26s/step - loss: 0.6914 - accuracy: 0.6092 - val_loss: 0.5773 - val_accuracy: 0.6
Epoch 12/50
20/20 [==============================] - 516s 26s/step - loss: 0.6815 - accuracy: 0.6317 - val_loss: 0.5365 - val_accuracy: 0.7
Epoch 13/50
20/20 [==============================] - 513s 26s/step - loss: 0.6708 - accuracy: 0.6439 - val_loss: 0.5759 - val_accuracy: 0.6
Epoch 14/50
20/20 [==============================] - 514s 26s/step - loss: 0.7752 - accuracy: 0.6672 - val_loss: 0.5635 - val_accuracy: 0.6
Epoch 15/50
20/20 [==============================] - 536s 27s/step - loss: 0.6791 - accuracy: 0.6353 - val_loss: 0.5339 - val_accuracy: 0.7
Epoch 16/50
20/20 [==============================] - 547s 28s/step - loss: 0.6665 - accuracy: 0.6459 - val_loss: 0.5437 - val_accuracy: 0.6
Epoch 17/50
20/20 [==============================] - 553s 28s/step - loss: 0.6474 - accuracy: 0.6530 - val_loss: 0.5272 - val_accuracy: 0.7
Epoch 18/50
20/20 [==============================] - 553s 28s/step - loss: 0.6499 - accuracy: 0.6522 - val_loss: 0.5757 - val_accuracy: 0.6
Epoch 19/50
20/20 [==============================] - 555s 28s/step - loss: 0.6720 - accuracy: 0.6577 - val_loss: 0.5187 - val_accuracy: 0.7
Epoch 20/50
20/20 [==============================] - 556s 28s/step - loss: 0.6577 - accuracy: 0.6502 - val_loss: 0.5301 - val_accuracy: 0.7
Epoch 21/50
20/20 [==============================] - 559s 29s/step - loss: 0.5880 - accuracy: 0.6834 - val_loss: 0.6058 - val_accuracy: 0.6
Epoch 22/50
20/20 [==============================] - 562s 29s/step - loss: 0.6532 - accuracy: 0.6723 - val_loss: 0.5404 - val_accuracy: 0.7
Epoch 23/50
20/20 [==============================] - 571s 29s/step - loss: 0.6632 - accuracy: 0.6652 - val_loss: 0.5076 - val_accuracy: 0.7
Epoch 24/50
20/20 [==============================] - 594s 30s/step - loss: 0.6315 - accuracy: 0.6711 - val_loss: 0.7028 - val_accuracy: 0.6
Epoch 25/50
20/20 [==============================] - 696s 36s/step - loss: 0.6430 - accuracy: 0.6305 - val_loss: 0.5208 - val_accuracy: 0.7
Epoch 26/50
20/20 [==============================] - 687s 35s/step - loss: 0.6235 - accuracy: 0.6656 - val_loss: 0.5987 - val_accuracy: 0.6
```

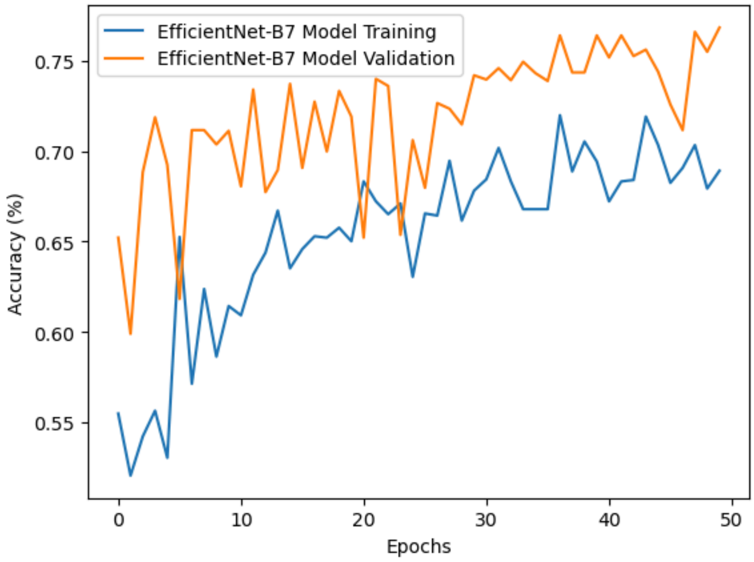
```
efficient_net.evaluate(test_data)
```

```
3/3 [==============================] - 35s 10s/step - loss: 1.5080 - accuracy: 0.2167
[1.5079811811447144, 0.21666666865348816]
```

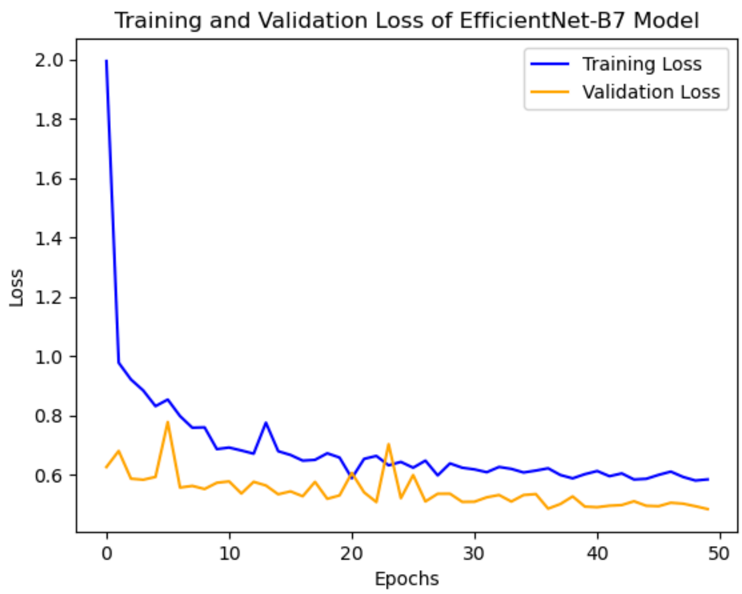
```
import pickle

# Save training history
with open('effb7_history.pkl', 'wb') as f:
    pickle.dump(effb7_history.history, f)

# Load training history
with open('effb7_history.pkl', 'rb') as f:
    effb7_history_loaded = pickle.load(f)

# Now you can plot the accuracies using the loaded training history (effb7_history_loaded)
```

```
import matplotlib.pyplot as plt

# Define the number of epochs
epochs = 50

# Create data for x-axis (epochs)
x = range(epochs)
```

```
# Plot lines
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.plot(x, effb7_history.history['accuracy'], label="EfficientNet-B7 Model Training", linestyle="-")
plt.plot(x, effb7_history.history['val_accuracy'], label="EfficientNet-B7 Model Validation", linestyle="-")
plt.legend()
plt.show()
```




```
import matplotlib.pyplot as plt

# Define the number of epochs
epochs = 50

# Create data for x-axis (epochs)
x = range(epochs)

# Plot training loss
plt.plot(x, effb7_history.history['loss'], label="Training Loss", linestyle="-", color='blue')

# Plot validation loss
plt.plot(x, effb7_history.history['val_loss'], label="Validation Loss", linestyle="-", color='orange')

# Add labels and legend
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss of EfficientNet-B7 Model")
plt.legend()

# Show plot
plt.show()
```



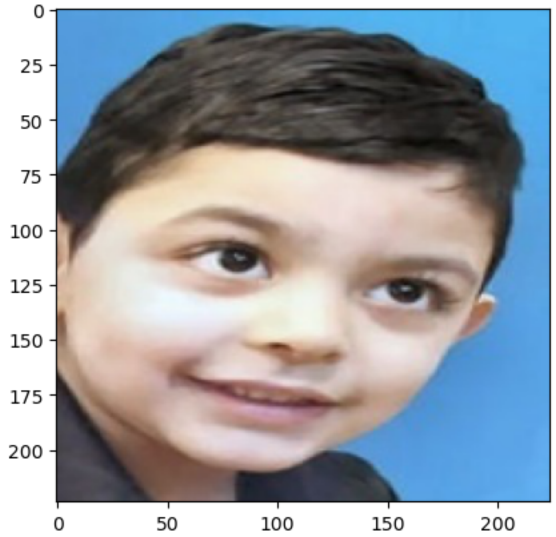
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# create data
x=[]
for i in range(0, 1):
    x.append(i)

# plot lines
plt.xlabel("Epochs")
plt.ylabel("Validation Accuracy (%)")
plt.plot(x, vgghist.history['val_accuracy'], label = "VGG-16 Model", linestyle="-")
plt.plot(x, incephist.history['val_accuracy'], label = "InceptionV3 Model", linestyle="-")
plt.plot(x, effb0_history.history['val_accuracy'], label = "EfficientNet-B0 Model", linestyle="-")
plt.plot(x, effb7_history.history['accuracy'], label = "EfficientNet-B7 Model", linestyle="-")
plt.legend()
plt.show()
```

```
from efficientnet.tfkeras import EfficientNetB0
from keras.models import load_model
efficientnet_b0_model= load_model("E:/Apps/VS Code/Project Files/ASD Project Files/efficient_net_B0_model.h5")
efficientnet_b7_model= load_model("E:/Apps/VS Code/Project Files/ASD Project Files/efficient_net_B7_model.h5")
vgg_model = load_model('E:/Apps/VS Code/Project Files/ASD Project Files/vgg_model50.h5')
inception_v3_model = load_model("E:/Apps/VS Code/Project Files/ASD Project Files/inception_model.h5")
#efficient_net_model.summary()
```

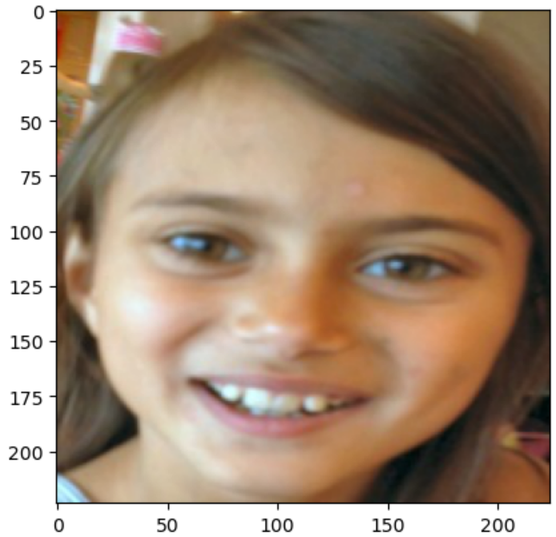
```
from PIL import Image
import numpy as np
from skimage import transform
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
photo_size=224
def load_image_from_path(filename):
    img = mpimg.imread(filename)
    imgplot = plt.imshow(img)
    plt.show()
    np_image = Image.open(filename)
    np_image = np.array(np_image).astype('float32') / 255
    np_image = transform.resize(np_image, (photo_size, photo_size, 3))
    np_image = np.expand_dims(np_image, axis=0)
    return np_image
```

```
import os
mTestPath = 'E:/Apps/VS Code/Projects Data Set/ASD Prediction/test/autistic'
for test in os.listdir(mTestPath):
    print(test)
    img = load_image_from_path(os.path.join(mTestPath, test))
    res = vgg_model.predict(img).argmax()
    if(res==1):
        print("VGG-16\t\tAutistic")
    else:
        print("VGG-16\t\tNon-Autistic")
    res = inception_v3_model.predict(img).argmax()
    if(res==1):
        print("Inception-V3\tAutistic")
    else:
        print("Inception-V3\tNon-Autistic")
    res = efficientnet_b0_model.predict(img).argmax()
    if(res==1):
        print("Efficient-NetB0\tNon-Autistic")
    else:
        print("Efficient-NetB0\tAutistic")
    res = efficientnet_b7_model.predict(img).argmax()
    if(res==1):
        print("Efficient-NetB7\tNon-Autistic")
    else:
        print("Efficient-NetB7\tAutistic")
```

```
001.jpg
```




```
1/1 [==============================] - 1s 837ms/step
VGG-16          Autistic
1/1 [==============================] - 4s 4s/step
Inception-V3    Autistic
1/1 [==============================] - 4s 4s/step
Efficient-NetB0 Autistic
1/1 [==============================] - 9s 9s/step
Efficient-NetB7 Autistic
002.jpg
```




```
1/1 [==============================] - 0s 238ms/step
VGG-16          Autistic
1/1 [==============================] - 0s 105ms/step
Inception-V3    Autistic
1/1 [==============================] - 0s 99ms/step
Efficient-NetB0 Autistic
1/1 [==============================] - 0s 351ms/step
Efficient-NetB7 Autistic
003.jpg
```