pathy-kaggle-cnn-inception-vgg-95

May 1, 2025

[1]:

[2]:

len(os.listdir(r"/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/

𝗌Healthy"))

**import matplotlib.pyplot as plt import numpy as np**

**import cv2 import os import PIL**

**import tensorflow as tf**

**from tensorflow import** keras

**from tensorflow.keras import** layers

**from tensorflow.keras.models import** Sequential

**from keras.preprocessing.image import** img\_to\_array, load\_img

**from tensorflow.keras.preprocessing.image import** ImageDataGenerator

**import pathlib**

**from xgboost import** XGBClassifier

**from sklearn.ensemble import**␣

𝗌RandomForestClassifier,AdaBoostClassifier,BaggingClassifier,ExtraTreesClassifier,GradientBo

[2]: 3326

[3]:

len(os.listdir(r"/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe␣

𝗌DR"))

[3]: 3096

[4]:

healthy\_images = os.listdir(r"/kaggle/input/

𝗌diabetic-retinopathyhealthy-and-severe-dr/Healthy")

[5]:

healthy\_images[:5]

1. : ['aug-\_0\_5525.png', 'aug-\_0\_3515.png', 'Healthy\_490.png', 'aug-\_0\_3549.png', 'aug-\_0\_7330.png']
2. :

*# augmented\_healthy\_images = 'augmented\_healthy\_images' # augmented\_severe\_images = 'augmented\_severe\_images'*

*# os.makedirs(augmented\_healthy\_images, exist\_ok=True) # os.makedirs(augmented\_severe\_images, exist\_ok=True)*

1. :

*# datagen = ImageDataGenerator( # rotation\_range=40,*

*# zoom\_range=0.2,*

*# horizontal\_flip=True, # vertical\_flip=True*

*# )*

1. :

*# for image in healthy\_images[:200]:*

*# # #*

*# #*

*image\_path = 'retinopathy\_images/healthy/' + image img = cv2.imread(str(image\_path))*

*img = cv2.resize(img,(180,180))*

*x = img\_to\_array(img)*

*x = x.reshape((1,) + x.shape)*

*# # # # # # # #*

*i = 0*

*for batch in datagen.flow(x, batch\_size=1,*

*save\_to\_dir=augmented\_healthy\_images, save\_prefix=f'aug-', save\_format='png'):*

*i += 1*

*if i > 20:*

*break*

1. :

severe = os.listdir(r"/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/

𝗌Severe DR")

1. :

*# for image in severe:*

*# # #*

*image\_path = 'retinopathy\_images/severe DR/' + image img = cv2.imread(str(image\_path))*

*img = cv2.resize(img,(180,180))*

*# #*

*# # # #*

*x = img\_to\_array(img)*

*x = x.reshape((1,) + x.shape)*

*i = 0*

*for batch in datagen.flow(x, batch\_size=1,*

*save\_to\_dir=augmented\_severe\_images, save\_prefix=f'aug-',*

*# # # #*

*save\_format='png'):*

*i += 1*

*if i > 20:*

*break*

1. :

data\_dir = r"/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr"

1. :

data\_dir=pathlib.Path(data\_dir) data\_dir

1. : PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr')
2. :

health = list(data\_dir.glob("Healthy/\*"))

1. :

len(health)

[14]: 3326

1. :

health[:5]

1. : [PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe- dr/Healthy/aug-\_0\_5525.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe- dr/Healthy/aug-\_0\_3515.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe- dr/Healthy/Healthy\_490.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe- dr/Healthy/aug-\_0\_3549.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe- dr/Healthy/aug-\_0\_7330.png')]

1. :

sev = list(data\_dir.glob("Severe DR/\*"))

1. :

len(sev)

[17]: 3096

1. :

sev[:5]

1. : [PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe DR/aug-\_0\_2319.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe DR/Severe DR\_185.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe DR/aug-\_0\_2434.png'),

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe DR/aug-\_0\_2688.png'),

1. :
2. :
3. :

PosixPath('/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Severe DR/aug-\_0\_734.png')]

image\_dict = {"health":health,"severe":sev}

labels\_dict = {"health":0,"severe":1}

1. :

X = np.array(X) y = np.array(y)

X, y = [], []

**for** name, images **in** image\_dict.items():

**for** image **in** images:

img = cv2.imread(str(image))

**try**:

resized\_img = cv2.resize(img,(224,224)) X.append(resized\_img) y.append(labels\_dict[name])

**except**:

print("image couldn't be loaded")

1. :

len(X)

[23]: 6422

1. :

len(y)

[24]: 6422

1. :

**from sklearn.model\_selection import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,test\_size=0.2,␣

𝗌random\_state=0)

1. :

X\_train\_scaled = X\_train / 255 X\_test\_scaled = X\_test / 255

1. :

X\_train\_scaled.shape

[27]: (5137, 224, 224, 3)

[44]:

**from tensorflow.keras.callbacks import** EarlyStopping

early\_stopper = EarlyStopping( monitor='accuracy', patience=10,

verbose=1, mode='auto',

restore\_best\_weights=**True**

)

[45]:

model1 = Sequential([

layers.Conv2D(64, 3, padding='same',␣

𝗌activation='relu',input\_shape=(224,224,3)),

layers.MaxPooling2D(),

layers.Conv2D(64, 3, padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Conv2D(128, 3, padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(256, activation='relu'), layers.Dense(64, activation='relu'), layers.Dense(2,activation='sigmoid')])

model1.compile(optimizer='adam',

loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])

[46]:

**from tensorflow.keras.utils import** to\_categorical y\_train = to\_categorical(y\_train, num\_classes=2) y\_test = to\_categorical(y\_test, num\_classes=2)

[47]:

model1.fit(X\_train\_scaled, y\_train, epochs=30,callbacks=[early\_stopper])

|  |  |  |
| --- | --- | --- |
| Epoch 1/30 |  | |
| **161/161** | **16s** 84ms/step | - |
| accuracy: 0.6581 | - loss: 0.7510 |  |
| Epoch 2/30 |  |  |
| **161/161** | **12s** 74ms/step | - |
| accuracy: 0.9000 | - loss: 0.2763 |  |
| Epoch 3/30 |  |  |
| **161/161** | **12s** 72ms/step | - |
| accuracy: 0.9159 | - loss: 0.2292 |  |
| Epoch 4/30 |  |  |
| **161/161** | **12s** 71ms/step | - |
| accuracy: 0.9277 | - loss: 0.1876 |  |
| Epoch 5/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9453 | - loss: 0.1564 |  |
| Epoch 6/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9484 | - loss: 0.1347 |  |
| Epoch 7/30 |  |  |

|  |  |  |
| --- | --- | --- |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9713 | - loss: 0.0790 |  |
| Epoch 8/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9727 | - loss: 0.0766 |  |
| Epoch 9/30 |  |  |
| **161/161** | **12s** 71ms/step | - |
| accuracy: 0.9750 | - loss: 0.0721 |  |
| Epoch 10/30 |  |  |
| **161/161** | **12s** 72ms/step | - |
| accuracy: 0.9753 | - loss: 0.0651 |  |
| Epoch 11/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9806 | - loss: 0.0562 |  |
| Epoch 12/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9847 | - loss: 0.0482 |  |
| Epoch 13/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9913 | - loss: 0.0270 |  |
| Epoch 14/30 |  |  |
| **161/161** | **11s** 70ms/step | - |
| accuracy: 0.9921 | - loss: 0.0208 |  |
| Epoch 15/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9924 | - loss: 0.0241 |  |
| Epoch 16/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9973 | - loss: 0.0115 |  |
| Epoch 17/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 0.9982 | - loss: 0.0046 |  |
| Epoch 18/30 |  |  |
| **161/161** | **11s** 71ms/step | - |
| accuracy: 1.0000 | - loss: 0.0016 |  |
| Epoch 19/30 |  |  |
| **161/161** | **11s** 70ms/step | - |
| accuracy: 0.9895 | - loss: 0.0314 |  |
| Epoch 20/30 |  |  |
| **161/161** | **11s** 70ms/step | - |
| accuracy: 0.9751 | - loss: 0.0629 |  |
| Epoch 21/30 |  |  |
| **161/161** | **11s** 70ms/step | - |
| accuracy: 0.9954 | - loss: 0.0166 |  |
| Epoch 22/30 |  |  |
| **161/161** | **11s** 70ms/step | - |
| accuracy: 0.9991 | - loss: 0.0022 |  |
| Epoch 23/30 |  |  |

**161/161 11s** 71ms/step - accuracy: 0.9998 - loss: 9.6908e-04 Epoch 24/30

**161/161 11s** 71ms/step - accuracy: 1.0000 - loss: 2.7776e-04 Epoch 25/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 1.0332e-04 Epoch 26/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 1.3002e-04 Epoch 27/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 6.4663e-05 Epoch 28/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 3.0117e-05 Epoch 29/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 2.7005e-05 Epoch 30/30

**161/161 11s** 70ms/step - accuracy: 1.0000 - loss: 1.6962e-05

Restoring model weights from the end of the best epoch: 24.

1. : <keras.src.callbacks.history.History at 0x7b7e881d8940> [48]:

model1.evaluate(X\_test\_scaled,y\_test)

**41/41 2s** 30ms/step - accuracy: 0.9535 - loss: 0.4117

1. : [0.43084365129470825, 0.9494163393974304]
2. :

**import joblib**

joblib.dump(model1,'model1.h5')

[49]: ['model1.h5']

[52]:

model2 = Sequential([

layers.Conv2D(64, 3, padding='same',␣

𝗌activation='relu',input\_shape=(224,224,3)),

layers.Conv2D(64,3,padding='same',activation='relu'), layers.Dropout(0.2),

layers.MaxPooling2D(),

layers.Conv2D(128, 3, padding='same', activation='relu'), layers.Conv2D(128, 3, padding='same', activation='relu'), layers.Dropout(0.2),

layers.MaxPooling2D(),

layers.Conv2D(256, 3, padding='same', activation='relu'), layers.Conv2D(256, 3, padding='same', activation='relu'), layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(256, activation='relu'), layers.Dense(64, activation='relu'), layers.Dense(2,activation='sigmoid')])

model2.compile(optimizer='adam',

loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])

[53]:

model2.fit(X\_train\_scaled, y\_train, epochs=30,callbacks=[early\_stopper])

Epoch 10/30

|  |  |  |
| --- | --- | --- |
| Epoch 1/30 |  | |
| **161/161** |  | **105s** 443ms/step - |
| accuracy: | 0.5409 | - loss: 0.7466 |
| Epoch 2/30 |  |  |
| **161/161** |  | **49s** 302ms/step - |
| accuracy: | 0.4775 | - loss: 0.6937 |
| Epoch 3/30 |  |  |
| **161/161** |  | **49s** 302ms/step - |
| accuracy: | 0.5155 | - loss: 0.6929 |
| Epoch 4/30 |  |  |
| **161/161** |  | **49s** 303ms/step - |
| accuracy: | 0.5192 | - loss: 0.6926 |
| Epoch 5/30 |  |  |
| **161/161** |  | **49s** 302ms/step - |
| accuracy: | 0.5190 | - loss: 0.6925 |
| Epoch 6/30 |  |  |
| **161/161** |  | **48s** 301ms/step - |
| accuracy: | 0.5210 | - loss: 0.6924 |
| Epoch 7/30 |  |  |
| **161/161** |  | **48s** 301ms/step - |
| accuracy: | 0.5133 | - loss: 0.6928 |
| Epoch 8/30 |  |  |
| **161/161** |  | **49s** 301ms/step - |
| accuracy: | 0.5159 | - loss: 0.6927 |
| Epoch 9/30 |  |  |
| **161/161** |  | **48s** 301ms/step - |
| accuracy: | 0.5178 | - loss: 0.6926 |

**161/161 48s** 301ms/step - accuracy: 0.5188 - loss: 0.6925 Epoch 10: early stopping

Restoring model weights from the end of the best epoch: 1.

[53]: <keras.src.callbacks.history.History at 0x7b7e802a2fe0> [58]:

joblib.dump(model2,'model2.h5')

1. : ['model2.h5']
2. :

**from tensorflow.keras.applications import** InceptionV3, VGG16

**from tensorflow.keras.models import** Model

**from tensorflow.keras.layers import** Dense, GlobalAveragePooling2D, Dropout

1. :

base\_inception = InceptionV3(weights='imagenet', include\_top=**False**,␣

𝗌input\_shape=(224, 224, 3)) base\_inception.trainable = **False**

x = base\_inception.output

x = GlobalAveragePooling2D()(x)

x = Dense(256, activation='relu')(x) x = Dropout(0.3)(x)

x = Dense(64, activation='relu')(x) x = Dropout(0.2)(x)

out\_inception = Dense(2, activation='sigmoid')(x)

model\_inception = Model(inputs=base\_inception.input, outputs=out\_inception) model\_inception.compile(optimizer='adam',

loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])

1. :

print("Training InceptionV3 Model") model\_inception.fit(X\_train\_scaled, y\_train, epochs=30,␣

𝗌callbacks=[early\_stopper])

print("Evaluating InceptionV3 Model") model\_inception.evaluate(X\_test\_scaled, y\_test)

Training InceptionV3 Model Epoch 1/30

**161/161 26s** 94ms/step - accuracy: 0.8924 - loss: 0.2527 Epoch 2/30

**161/161 11s** 68ms/step - accuracy: 0.9753 - loss: 0.0731 Epoch 3/30

**161/161 11s** 69ms/step - accuracy: 0.9797 - loss: 0.0583 Epoch 4/30

**161/161 11s** 68ms/step - accuracy: 0.9805 - loss: 0.0496

|  |  |  |
| --- | --- | --- |
| Epoch 5/30 |  | |
| **161/161** | **11s** 67ms/step | - |
| accuracy: 0.9876 | - loss: 0.0354 |  |
| Epoch 6/30 |  |  |
| **161/161** | **11s** 66ms/step | - |
| accuracy: 0.9878 | - loss: 0.0364 |  |
| Epoch 7/30 |  |  |
| **161/161** | **11s** 66ms/step | - |
| accuracy: 0.9874 | - loss: 0.0296 |  |
| Epoch 8/30 |  |  |
| **161/161** | **11s** 66ms/step | - |
| accuracy: 0.9921 | - loss: 0.0230 |  |
| Epoch 9/30 |  |  |
| **161/161** | **11s** 66ms/step | - |
| accuracy: 0.9956 | - loss: 0.0190 |  |
| Epoch 10/30 |  |  |
| **161/161** | **11s** 67ms/step | - |
| accuracy: 0.9930 | - loss: 0.0205 |  |

Epoch 10: early stopping

Restoring model weights from the end of the best epoch: 1. Evaluating InceptionV3 Model

**41/41 9s** 115ms/step - accuracy: 0.9611 - loss: 0.0983

[61]: [0.10441748797893524, 0.9603112936019897]

1. :

joblib.dump(model\_inception,'model\_inception.h5')

1. : ['model\_inception.h5']
2. :

base\_vgg = VGG16(weights='imagenet', include\_top=**False**, input\_shape=(224, 224,␣

𝗌3))

base\_vgg.trainable = **False**

x\_vgg = base\_vgg.output

x\_vgg = GlobalAveragePooling2D()(x\_vgg) x\_vgg = Dense(256, activation='relu')(x\_vgg) x\_vgg = Dropout(0.3)(x\_vgg)

x\_vgg = Dense(64, activation='relu')(x\_vgg) x\_vgg = Dropout(0.2)(x\_vgg)

out\_vgg = Dense(2, activation='sigmoid')(x\_vgg)

model\_vgg = Model(inputs=base\_vgg.input, outputs=out\_vgg) model\_vgg.compile(optimizer='adam',

loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])

Downloading data from https://storage.googleapis.com/tensorflow/keras-

1. :

applications/vgg16/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

**58889256/58889256 0s**

0us/step

print("Training VGG16 Model")

model\_vgg.fit(X\_train\_scaled, y\_train, epochs=10, callbacks=[early\_stopper])

print("Evaluating VGG16 Model") model\_vgg.evaluate(X\_test\_scaled, y\_test)

Training VGG16 Model Epoch 1/10

**161/161 55s** 235ms/step - accuracy: 0.8133 - loss: 0.4161 Epoch 2/10

**161/161 23s** 142ms/step - accuracy: 0.9422 - loss: 0.1659 Epoch 3/10

**161/161 23s** 143ms/step - accuracy: 0.9518 - loss: 0.1445 Epoch 4/10

**161/161 24s** 148ms/step - accuracy: 0.9686 - loss: 0.0891 Epoch 5/10

**161/161 24s** 146ms/step - accuracy: 0.9701 - loss: 0.0881 Epoch 6/10

**161/161 23s** 144ms/step - accuracy: 0.9689 - loss: 0.0937 Epoch 7/10

**161/161 23s** 145ms/step - accuracy: 0.9732 - loss: 0.0764 Epoch 8/10

**161/161 23s** 146ms/step - accuracy: 0.9781 - loss: 0.0648 Epoch 9/10

**161/161 24s** 146ms/step - accuracy: 0.9703 - loss: 0.0767 Epoch 10/10

**161/161 24s** 147ms/step - accuracy: 0.9761 - loss: 0.0680 Epoch 10: early stopping

Restoring model weights from the end of the best epoch: 1. Evaluating VGG16 Model

**41/41 11s** 242ms/step - accuracy: 0.9421 - loss: 0.1701

[64]: [0.17108404636383057, 0.9416342377662659]

1. :

joblib.dump(model\_vgg,'model\_vgg.h5')

1. : ['model\_vgg.h5']
2. :

**from sklearn.metrics import** classification\_report, confusion\_matrix,␣

𝗌roc\_auc\_score

**import seaborn as sns**

1. :

**def** plot\_metrics(history, model\_name): plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy') plt.plot(history.history['val\_accuracy'], label='Validation Accuracy') plt.title(f'**{**model\_name**}** Accuracy')

plt.xlabel('Epochs') plt.ylabel('Accuracy') plt.legend()

plt.subplot(1, 2, 2) plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss') plt.title(f'**{**model\_name**}** Loss')

plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()

plt.show()

1. :

**def** evaluate\_model(model, X\_test, y\_test, model\_name): y\_pred = (model.predict(X\_test) > 0.5).astype("int32") cm = confusion\_matrix(y\_test, y\_pred)

cr = classification\_report(y\_test, y\_pred, target\_names=['Healthy',␣

𝗌'Severe'])

auc = roc\_auc\_score(y\_test, y\_pred)

print(f"**\n{**model\_name**}** Classification Report:**\n**", cr) print(f"**{**model\_name**}** AUC: **{**auc**:**.4f**}\n**")

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=**True**, fmt='d', cmap='Blues', xticklabels=['Healthy',␣

𝗌'Severe'], yticklabels=['Healthy', 'Severe']) plt.title(f'**{**model\_name**}** Confusion Matrix') plt.xlabel('Predicted')

plt.ylabel('True') plt.show()

[70]:

**def** evaluate\_model(model, X\_test, y\_test, model\_name):

**if** y\_test.shape[1] > 1:

y\_test = np.argmax(y\_test, axis=1)

y\_pred = model.predict(X\_test) y\_pred = np.argmax(y\_pred, axis=1)

cm = confusion\_matrix(y\_test, y\_pred)

cr = classification\_report(y\_test, y\_pred, target\_names=['Healthy',␣

𝗌'Severe'])

auc = roc\_auc\_score(y\_test, y\_pred)

print(f"**\n{**model\_name**}** Classification Report:**\n**", cr) print(f"**{**model\_name**}** AUC: **{**auc**:**.4f**}\n**")

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=**True**, fmt='d', cmap='Blues', xticklabels=['Healthy',␣

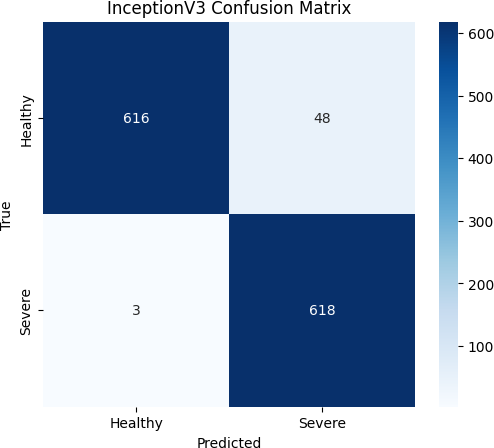
𝗌'Severe'], yticklabels=['Healthy', 'Severe']) plt.title(f'**{**model\_name**}** Confusion Matrix') plt.xlabel('Predicted')

plt.ylabel('True') plt.show()

[ ]:

evaluate\_model(model\_inception, X\_test\_scaled, y\_test, "InceptionV3") evaluate\_model(model\_vgg, X\_test\_scaled, y\_test, "VGG16")

|  |  |  |
| --- | --- | --- |
| **41/41 3s** 63ms/step |  | |
| InceptionV3 Classification Report:  precision recall | f1-score | support |
| Healthy 1.00 0.93 | 0.96 | 664 |
| Severe 0.93 1.00 | 0.96 | 621 |
| accuracy | 0.96 | 1285 |
| macro avg 0.96 0.96 | 0.96 | 1285 |
| weighted avg 0.96 0.96 | 0.96 | 1285 |
| InceptionV3 AUC: 0.9614 |  |  |



**16/41 3s** 150ms/step

[72]:

cnn\_model\_path = "/kaggle/working/model1.h5" vgg\_model\_path = "/kaggle/working/model\_vgg.h5"

inception\_model\_path = "/kaggle/working/model\_inception.h5"

cnn\_model = joblib.load(cnn\_model\_path) vgg\_model = joblib.load(vgg\_model\_path)

inception\_model = joblib.load(inception\_model\_path)

**def** preprocess\_image(image\_path, target\_size=(224, 224)): img = cv2.imread(image\_path)

img = cv2.resize(img, target\_size) img = img / 255.0

img = np.expand\_dims(img, axis=0)

**return** img

**def** predict\_image(model, image\_path, model\_name):

processed\_img = preprocess\_image(image\_path) predictions = model.predict(processed\_img) predicted\_class = np.argmax(predictions, axis=1)[0] confidence = np.max(predictions)

print(f"**{**model\_name**}** Prediction: Class **{**predicted\_class**}**, Confidence:␣

𝗌**{**confidence**:**.2f**}**")

img = cv2.imread(image\_path) plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

plt.title(f"**{**model\_name**}** Prediction: **{**predicted\_class**}**, Confidence:␣

𝗌**{**confidence**:**.2f**}**")

plt.axis("off") plt.show()

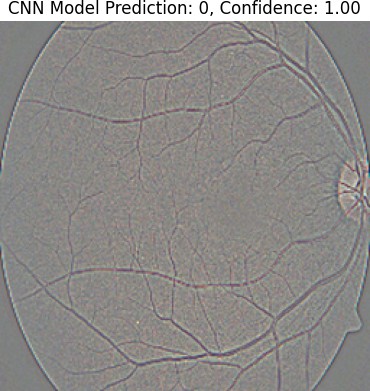
image\_path = "/kaggle/input/diabetic-retinopathyhealthy-and-severe-dr/Healthy/

𝗌Healthy\_100.png"

predict\_image(cnn\_model, image\_path, "CNN Model") predict\_image(vgg\_model, image\_path, "VGG16 Model") predict\_image(inception\_model, image\_path, "Inception Model")

**1/1 0s** 196ms/step

CNN Model Prediction: Class 0, Confidence: 1.00



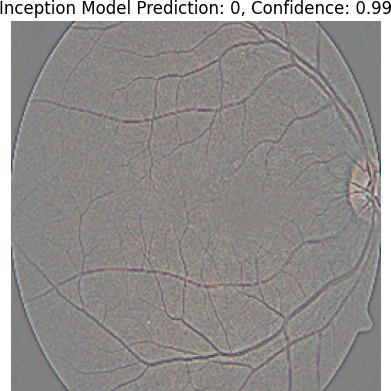
**1/1 2s** 2s/step

VGG16 Model Prediction: Class 1, Confidence: 0.61



**1/1 6s** 6s/step

Inception Model Prediction: Class 0, Confidence: 0.99



[ ]: