Performing Facial Recognition with Deep Learning

Objective: Create a facial recognition tool using a relevant deep learning algorithm, leveraging the provided resources.

Context: You are working for Face2Gene, an American AI company that has developed a healthcare app for doctors. The app utilizes deep learning algorithms to aid in diagnosing patients for genetic disorders and their variants. It converts patient photos into de-identified mathematical facial descriptors, which are then compared to syndrome-specific computational-based classifiers to determine similarity. The app provides a prioritized list of syndromes with similar morphology and suggests phenotypic traits and genes for feature annotation and syndrome prioritization.

Management has given priority to empowering and entrusting the in-house AI team. As a new member of the team, your task is to build a baseline model for facial recognition. The goal is to further enhance the app's existing features and add more value to the business based on this baseline model.

Dataset Details: The ORL Database of Faces consists of 400 images from 40 different subjects. The images were captured at different times, under varying lighting conditions, with different facial expressions (open, closed eyes, smiling, not smiling), and with or without glasses. All the images have a dark homogeneous background, and the subjects are positioned upright and frontal with some tolerance for side movement. Each image has a size of 92x112 pixels and 256 grey levels per pixel.

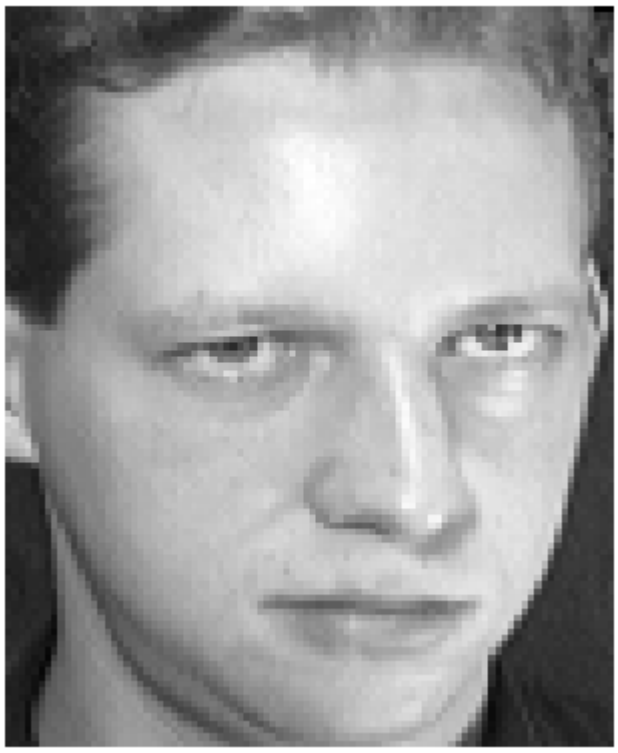
Data can be downloaded from the following link:
https://www.kaggle.com/datasets/kasikrit/att-database-of-faces

In [147…
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from PIL import Image
import os
import glob
```

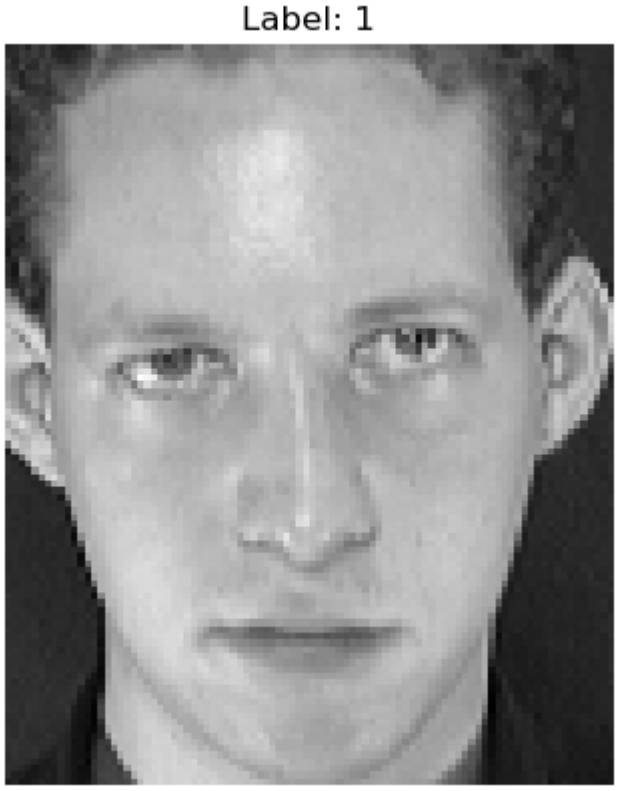
In [365…
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from PIL import Image
import os
import glob

# --- Data Loading and Preparation ---
data_dir = r'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artifical Engineer\

# Collect image paths
image_paths = glob.glob(os.path.join(data_dir, '**/*.pgm'))

# Create empty lists for images and labels
images = []
labels = []
```

```
# Load the images and extract labels
for image_path in image_paths:
    label = int(os.path.split(os.path.dirname(image_path))[-1][1:])  # Extract l
    image = np.array(Image.open(image_path).convert('L'))          # Convert to
    #image = np.array(Image.open(image_path).convert('L').transpose(Image.TRANSP

    images.append(image)
    labels.append(label)


# --- Data Visualization ---
def display_image(image, label):
    plt.imshow(image, cmap='gray')
    plt.title(f"Label: {label}")
    plt.axis('off')
    plt.show()

# Display a few sample images
for i in range(5):
    display_image(images[i], labels[i])
```

Label: 1



Label: 1



In [367…

```
from sklearn.model_selection import train_test_split
# Convert image and label lists to NumPy arrays
images = np.array(images)
labels = np.array(labels)

# Initial split: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.

# Further split: 70% of original training for training, 30% for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
```

In [369…
```
labels
```

Out[369…
```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 10, 10, 10, 10, 10, 10, 10,
       10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12,
       12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13, 13, 13, 13, 14,
       14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 15, 15,
       15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17,
       17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 19, 19,
       19, 19, 19, 19, 19, 19, 19,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21,
       21, 21, 21, 21, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23,
       23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
       25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26,
       26, 26, 26, 27, 27, 27, 27, 27, 27, 27, 27, 27, 28, 28, 28, 28,
       28, 28, 28, 28, 28, 28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3, 30, 30, 30, 30, 30, 30, 30, 30,
       30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32,
       32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 34, 34,
       34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35,
       35, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37, 37, 37,
       37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 39, 39, 39,
       39, 39, 39, 39, 39,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
       40, 40, 40, 40, 40, 40, 40, 40, 40, 40,  5,  5,  5,  5,  5,  5,  5,
        5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9])
```

In [371…
```
# Reshape the images to add a channel dimension
X_train = X_train.reshape(X_train.shape[0], 92, 112, 1)
X_val = X_val.reshape(X_val.shape[0], 92, 112, 1)
X_test = X_test.reshape(X_test.shape[0], 92, 112, 1)

# Normalize the pixel values to be between 0 and 1
X_train = X_train.astype('float32') / 255.0
X_val = X_val.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode the labels
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)
```

In [373…
```
y_train = y_train[:, :40]
y_test = y_test[:, :40]
```

In [218…
```
from tensorflow.keras.optimizers import Nadam
learning_rate = 0.0005
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=lea
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_val, y_
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/10
4/4 [==============================] - 11s 2s/step - loss: 5.0495 - accuracy: 0.6
250 - val_loss: 0.8026 - val_accuracy: 0.8229
Epoch 2/10
4/4 [==============================] - 9s 2s/step - loss: 4.3931 - accuracy: 0.68
75 - val_loss: 0.7376 - val_accuracy: 0.8333
Epoch 3/10
4/4 [==============================] - 9s 2s/step - loss: 4.0331 - accuracy: 0.70
98 - val_loss: 0.7181 - val_accuracy: 0.8646
Epoch 4/10
4/4 [==============================] - 9s 2s/step - loss: 4.9663 - accuracy: 0.65
18 - val_loss: 0.7318 - val_accuracy: 0.8333
Epoch 5/10
4/4 [==============================] - 9s 2s/step - loss: 4.0301 - accuracy: 0.70
98 - val_loss: 0.7701 - val_accuracy: 0.8125
Epoch 6/10
4/4 [==============================] - 9s 2s/step - loss: 4.6776 - accuracy: 0.66
96 - val_loss: 0.8025 - val_accuracy: 0.8229
Epoch 7/10
4/4 [==============================] - 9s 2s/step - loss: 4.2457 - accuracy: 0.69
64 - val_loss: 0.8271 - val_accuracy: 0.8229
Epoch 8/10
4/4 [==============================] - 8s 2s/step - loss: 4.0298 - accuracy: 0.70
98 - val_loss: 0.8444 - val_accuracy: 0.8229
Epoch 9/10
4/4 [==============================] - 9s 2s/step - loss: 4.8932 - accuracy: 0.65
62 - val_loss: 0.8494 - val_accuracy: 0.8229
Epoch 10/10
4/4 [==============================] - 9s 2s/step - loss: 4.9651 - accuracy: 0.65
18 - val_loss: 0.8494 - val_accuracy: 0.8333
3/3 [==============================] - 1s 225ms/step - loss: 0.5286 - accuracy:
0.8750
Test loss: 0.5286089777946472
Test accuracy: 0.875
```

In [221…

```python
from tensorflow.keras.optimizers import Adamax
learning_rate = 0.00005
model.compile(loss='categorical_crossentropy', optimizer=Adamax(learning_rate=le
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_val, y_
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/10
4/4 [==============================] - 10s 2s/step - loss: 4.3894 - accuracy: 0.6
875 - val_loss: 0.8386 - val_accuracy: 0.8333
Epoch 2/10
4/4 [==============================] - 7s 2s/step - loss: 5.1089 - accuracy: 0.64
29 - val_loss: 0.8357 - val_accuracy: 0.8333
Epoch 3/10
4/4 [==============================] - 8s 2s/step - loss: 4.3174 - accuracy: 0.69
20 - val_loss: 0.8385 - val_accuracy: 0.8333
Epoch 4/10
4/4 [==============================] - 9s 2s/step - loss: 4.8211 - accuracy: 0.66
07 - val_loss: 0.8406 - val_accuracy: 0.8333
Epoch 5/10
4/4 [==============================] - 9s 2s/step - loss: 3.8856 - accuracy: 0.71
88 - val_loss: 0.8409 - val_accuracy: 0.8333
Epoch 6/10
4/4 [==============================] - 10s 2s/step - loss: 4.1015 - accuracy: 0.7
054 - val_loss: 0.8399 - val_accuracy: 0.8333
Epoch 7/10
4/4 [==============================] - 10s 3s/step - loss: 5.1089 - accuracy: 0.6
429 - val_loss: 0.8396 - val_accuracy: 0.8333
Epoch 8/10
4/4 [==============================] - 10s 3s/step - loss: 3.6698 - accuracy: 0.7
321 - val_loss: 0.8395 - val_accuracy: 0.8333
Epoch 9/10
4/4 [==============================] - 11s 3s/step - loss: 4.5332 - accuracy: 0.6
786 - val_loss: 0.8400 - val_accuracy: 0.8333
Epoch 10/10
4/4 [==============================] - 10s 2s/step - loss: 4.6052 - accuracy: 0.6
741 - val_loss: 0.8404 - val_accuracy: 0.8333
3/3 [==============================] - 1s 290ms/step - loss: 0.5087 - accuracy:
0.8625
Test loss: 0.5086637735366821
Test accuracy: 0.862500011920929
```

In [217…

```python
from keras.optimizers import Adam

model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accur
# Train the model for 10 epochs with a batch size of 32 and validation data
model.fit(X_train, y_train, epochs=10, batch_size=16, validation_data=(X_val, y_
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/10
14/14 [==============================] - 11s 732ms/step - loss: 7.2278 - accurac
y: 0.0268 - val_loss: 3.7569 - val_accuracy: 0.0312
Epoch 2/10
14/14 [==============================] - 10s 691ms/step - loss: 7.0449 - accurac
y: 0.0804 - val_loss: 3.5614 - val_accuracy: 0.0938
Epoch 3/10
14/14 [==============================] - 11s 760ms/step - loss: 6.3460 - accurac
y: 0.1786 - val_loss: 3.3391 - val_accuracy: 0.1458
Epoch 4/10
14/14 [==============================] - 10s 696ms/step - loss: 5.3520 - accurac
y: 0.3705 - val_loss: 2.3381 - val_accuracy: 0.3750
Epoch 5/10
14/14 [==============================] - 10s 753ms/step - loss: 4.4576 - accurac
y: 0.5848 - val_loss: 1.8074 - val_accuracy: 0.5833
Epoch 6/10
14/14 [==============================] - 10s 758ms/step - loss: 4.7375 - accurac
y: 0.6116 - val_loss: 1.8918 - val_accuracy: 0.6250
Epoch 7/10
14/14 [==============================] - 10s 747ms/step - loss: 5.4991 - accurac
y: 0.5982 - val_loss: 1.6508 - val_accuracy: 0.7083
Epoch 8/10
14/14 [==============================] - 10s 684ms/step - loss: 4.9166 - accurac
y: 0.6339 - val_loss: 1.8759 - val_accuracy: 0.6458
Epoch 9/10
14/14 [==============================] - 10s 724ms/step - loss: 4.3493 - accurac
y: 0.6696 - val_loss: 1.6525 - val_accuracy: 0.6562
Epoch 10/10
14/14 [==============================] - 10s 738ms/step - loss: 5.4213 - accurac
y: 0.5982 - val_loss: 1.1698 - val_accuracy: 0.7812
3/3 [==============================] - 1s 208ms/step - loss: 0.9638 - accuracy:
0.7875
Test loss: 0.9637966156005859
Test accuracy: 0.7875000238418579
```

In [375…

```
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(224, 92, 112, 1) (96, 92, 112, 1) (80, 92, 112, 1)
```

In [377…

```
import numpy as np

def convert_grayscale_to_rgb(X):
    return np.stack((X[..., 0],)*3, axis=-1)

# Apply the conversion to your datasets
X_train = convert_grayscale_to_rgb(X_train)
X_val = convert_grayscale_to_rgb(X_val)
X_test = convert_grayscale_to_rgb(X_test)
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(224, 92, 112, 3) (96, 92, 112, 3) (80, 92, 112, 3)
```

In [391…

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten

# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
```

```
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_val, y_
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/20
4/4 [==============================] - 20s 4s/step - loss: 3.7501 - accuracy: 0.0
312 - val_loss: 3.8524 - val_accuracy: 0.0312
Epoch 2/20
4/4 [==============================] - 17s 4s/step - loss: 3.3696 - accuracy: 0.1
027 - val_loss: 3.4854 - val_accuracy: 0.0833
Epoch 3/20
4/4 [==============================] - 16s 4s/step - loss: 2.9254 - accuracy: 0.1
920 - val_loss: 3.2122 - val_accuracy: 0.1667
Epoch 4/20
4/4 [==============================] - 17s 4s/step - loss: 2.4829 - accuracy: 0.2
768 - val_loss: 2.6631 - val_accuracy: 0.2708
Epoch 5/20
4/4 [==============================] - 17s 4s/step - loss: 1.6908 - accuracy: 0.5
134 - val_loss: 1.8458 - val_accuracy: 0.5000
Epoch 6/20
4/4 [==============================] - 18s 4s/step - loss: 1.0158 - accuracy: 0.7
054 - val_loss: 1.2996 - val_accuracy: 0.5938
Epoch 7/20
4/4 [==============================] - 18s 4s/step - loss: 0.6361 - accuracy: 0.7
902 - val_loss: 1.1726 - val_accuracy: 0.6875
Epoch 8/20
4/4 [==============================] - 18s 4s/step - loss: 0.4251 - accuracy: 0.8
571 - val_loss: 0.5876 - val_accuracy: 0.8333
Epoch 9/20
4/4 [==============================] - 18s 4s/step - loss: 0.1082 - accuracy: 0.9
375 - val_loss: 0.4868 - val_accuracy: 0.8542
Epoch 10/20
4/4 [==============================] - 18s 5s/step - loss: 0.0366 - accuracy: 0.9
554 - val_loss: 0.4543 - val_accuracy: 0.8646
Epoch 11/20
4/4 [==============================] - 17s 4s/step - loss: 0.0399 - accuracy: 0.9
554 - val_loss: 2.8201 - val_accuracy: 0.5104
Epoch 12/20
4/4 [==============================] - 17s 4s/step - loss: 1.4965 - accuracy: 0.6
830 - val_loss: 1.5446 - val_accuracy: 0.7292
Epoch 13/20
4/4 [==============================] - 17s 4s/step - loss: 0.3650 - accuracy: 0.8
929 - val_loss: 0.9569 - val_accuracy: 0.8542
Epoch 14/20
4/4 [==============================] - 18s 5s/step - loss: 0.1582 - accuracy: 0.9
509 - val_loss: 0.8744 - val_accuracy: 0.8438
Epoch 15/20
4/4 [==============================] - 18s 5s/step - loss: 0.0807 - accuracy: 0.9
554 - val_loss: 1.0679 - val_accuracy: 0.8333
Epoch 16/20
4/4 [==============================] - 18s 5s/step - loss: 0.0932 - accuracy: 0.9
554 - val_loss: 2.0625 - val_accuracy: 0.8229
Epoch 17/20
4/4 [==============================] - 19s 5s/step - loss: 0.8147 - accuracy: 0.8
304 - val_loss: 2.9732 - val_accuracy: 0.5417
Epoch 18/20
4/4 [==============================] - 19s 5s/step - loss: 1.0160 - accuracy: 0.7
812 - val_loss: 0.8746 - val_accuracy: 0.7083
Epoch 19/20
4/4 [==============================] - 19s 5s/step - loss: 0.0787 - accuracy: 0.9
330 - val_loss: 0.9194 - val_accuracy: 0.8646
Epoch 20/20
4/4 [==============================] - 21s 5s/step - loss: 0.1492 - accuracy: 0.9
196 - val_loss: 7.3990 - val_accuracy: 0.4271
```

```
3/3 [==============================] - 3s 816ms/step - loss: 5.7235 - accuracy:
0.5000
Test loss: 5.723528861999512
Test accuracy: 0.5
```

In [405…

```python
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=1, restore_best_weig

from keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2, min_lr
                              verbose=1, mode='max')  # Reduce LR when accuracy

# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=32,  callbacks=[early_stopping
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/20
7/7 [==============================] - 21s 3s/step - loss: 3.6147 - accuracy: 0.0
179 - val_loss: 3.5756 - val_accuracy: 0.0521 - lr: 1.0000e-04
Epoch 2/20
7/7 [==============================] - 19s 3s/step - loss: 3.0756 - accuracy: 0.1
652 - val_loss: 2.9289 - val_accuracy: 0.3333 - lr: 1.0000e-04
Epoch 3/20
7/7 [==============================] - 19s 3s/step - loss: 2.3803 - accuracy: 0.3
616 - val_loss: 2.2051 - val_accuracy: 0.3229 - lr: 1.0000e-04
Epoch 4/20
7/7 [==============================] - 20s 3s/step - loss: 1.2599 - accuracy: 0.6
071 - val_loss: 1.2007 - val_accuracy: 0.6458 - lr: 1.0000e-04
Epoch 5/20
7/7 [==============================] - 20s 3s/step - loss: 0.5556 - accuracy: 0.8
036 - val_loss: 1.3282 - val_accuracy: 0.6458 - lr: 1.0000e-04
3/3 [==============================] - 3s 831ms/step - loss: 1.4630 - accuracy:
0.5875
Test loss: 1.4629908800125122
Test accuracy: 0.5874999761581421
```

In [404…

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint


early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weig
from keras.callbacks import ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2, min_lr
                              verbose=1, mode='max')  # Reduce LR when accuracy

checkpoint_path = "vgg16_orl_faces-{epoch:02d}-{val_accuracy:.2f}.hdf5"  # Filen

checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_accuracy',
                             mode='max',  # Monitor for maximizing validation ac
                             save_best_only=True,  # Save only the best performi
                             verbose=1)


# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=32,  callbacks=[reduce_lr, che

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/20
7/7 [==============================] - ETA: 0s - loss: 3.6966 - accuracy: 0.0312
Epoch 1: val_accuracy improved from -inf to 0.02083, saving model to vgg16_orl_fa
ces-01-0.02.hdf5
C:\anaconda3\Lib\site-packages\keras\src\engine\training.py:3000: UserWarning: Yo
u are saving your model as an HDF5 file via `model.save()`. This file format is c
onsidered legacy. We recommend using instead the native Keras format, e.g. `mode
l.save('my_model.keras')`.
  saving_api.save_model(
```

```
7/7 [==============================] - 19s 2s/step - loss: 3.6966 - accuracy: 0.0
312 - val_loss: 3.6163 - val_accuracy: 0.0208 - lr: 1.0000e-04
Epoch 2/20
7/7 [==============================] - ETA: 0s - loss: 3.2635 - accuracy: 0.1250
Epoch 2: val_accuracy improved from 0.02083 to 0.15625, saving model to vgg16_orl
_faces-02-0.16.hdf5
7/7 [==============================] - 17s 3s/step - loss: 3.2635 - accuracy: 0.1
250 - val_loss: 3.1611 - val_accuracy: 0.1562 - lr: 1.0000e-04
Epoch 3/20
7/7 [==============================] - ETA: 0s - loss: 2.4895 - accuracy: 0.3125
Epoch 3: val_accuracy improved from 0.15625 to 0.28125, saving model to vgg16_orl
_faces-03-0.28.hdf5
7/7 [==============================] - 18s 3s/step - loss: 2.4895 - accuracy: 0.3
125 - val_loss: 2.4571 - val_accuracy: 0.2812 - lr: 1.0000e-04
Epoch 4/20
7/7 [==============================] - ETA: 0s - loss: 1.5065 - accuracy: 0.5402
Epoch 4: val_accuracy improved from 0.28125 to 0.59375, saving model to vgg16_orl
_faces-04-0.59.hdf5
7/7 [==============================] - 19s 3s/step - loss: 1.5065 - accuracy: 0.5
402 - val_loss: 1.5464 - val_accuracy: 0.5938 - lr: 1.0000e-04
Epoch 5/20
7/7 [==============================] - ETA: 0s - loss: 0.7571 - accuracy: 0.7723
Epoch 5: val_accuracy improved from 0.59375 to 0.73958, saving model to vgg16_orl
_faces-05-0.74.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.7571 - accuracy: 0.7
723 - val_loss: 0.9314 - val_accuracy: 0.7396 - lr: 1.0000e-04
Epoch 6/20
7/7 [==============================] - ETA: 0s - loss: 0.2670 - accuracy: 0.8839
Epoch 6: val_accuracy improved from 0.73958 to 0.84375, saving model to vgg16_orl
_faces-06-0.84.hdf5
7/7 [==============================] - 18s 3s/step - loss: 0.2670 - accuracy: 0.8
839 - val_loss: 0.6335 - val_accuracy: 0.8438 - lr: 1.0000e-04
Epoch 7/20
7/7 [==============================] - ETA: 0s - loss: 0.1028 - accuracy: 0.9420
Epoch 7: val_accuracy improved from 0.84375 to 0.87500, saving model to vgg16_orl
_faces-07-0.88.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.1028 - accuracy: 0.9
420 - val_loss: 0.4233 - val_accuracy: 0.8750 - lr: 1.0000e-04
Epoch 8/20
7/7 [==============================] - ETA: 0s - loss: 0.1745 - accuracy: 0.9152
Epoch 8: val_accuracy did not improve from 0.87500
7/7 [==============================] - 18s 3s/step - loss: 0.1745 - accuracy: 0.9
152 - val_loss: 1.3798 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 9/20
7/7 [==============================] - ETA: 0s - loss: 0.6651 - accuracy: 0.8125
Epoch 9: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.

Epoch 9: val_accuracy did not improve from 0.87500
7/7 [==============================] - 18s 3s/step - loss: 0.6651 - accuracy: 0.8
125 - val_loss: 0.5352 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 10/20
7/7 [==============================] - ETA: 0s - loss: 0.1657 - accuracy: 0.9330
Epoch 10: val_accuracy improved from 0.87500 to 0.93750, saving model to vgg16_or
l_faces-10-0.94.hdf5
7/7 [==============================] - 18s 3s/step - loss: 0.1657 - accuracy: 0.9
330 - val_loss: 0.3222 - val_accuracy: 0.9375 - lr: 5.0000e-05
Epoch 11/20
7/7 [==============================] - ETA: 0s - loss: 0.0387 - accuracy: 0.9554
Epoch 11: val_accuracy did not improve from 0.93750
7/7 [==============================] - 17s 3s/step - loss: 0.0387 - accuracy: 0.9
```

```
554 - val_loss: 0.2875 - val_accuracy: 0.9271 - lr: 5.0000e-05
Epoch 12/20
7/7 [==============================] - ETA: 0s - loss: 0.0458 - accuracy: 0.9464
Epoch 12: val_accuracy did not improve from 0.93750
7/7 [==============================] - 18s 3s/step - loss: 0.0458 - accuracy: 0.9
464 - val_loss: 0.9166 - val_accuracy: 0.7083 - lr: 5.0000e-05
Epoch 13/20
7/7 [==============================] - ETA: 0s - loss: 1.3247 - accuracy: 0.7098
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.4999999368446880e-05.

Epoch 13: val_accuracy did not improve from 0.93750
7/7 [==============================] - 18s 3s/step - loss: 1.3247 - accuracy: 0.7
098 - val_loss: 2.4414 - val_accuracy: 0.7917 - lr: 5.0000e-05
Epoch 14/20
7/7 [==============================] - ETA: 0s - loss: 0.5232 - accuracy: 0.8795
Epoch 14: val_accuracy did not improve from 0.93750
7/7 [==============================] - 19s 3s/step - loss: 0.5232 - accuracy: 0.8
795 - val_loss: 1.6026 - val_accuracy: 0.7604 - lr: 2.5000e-05
Epoch 15/20
7/7 [==============================] - ETA: 0s - loss: 0.2061 - accuracy: 0.9330
Epoch 15: ReduceLROnPlateau reducing learning rate to 1.2499999684223440e-05.

Epoch 15: val_accuracy did not improve from 0.93750
7/7 [==============================] - 19s 3s/step - loss: 0.2061 - accuracy: 0.9
330 - val_loss: 2.0498 - val_accuracy: 0.8229 - lr: 2.5000e-05
Epoch 16/20
7/7 [==============================] - ETA: 0s - loss: 0.3620 - accuracy: 0.9196
Epoch 16: val_accuracy did not improve from 0.93750
7/7 [==============================] - 19s 3s/step - loss: 0.3620 - accuracy: 0.9
196 - val_loss: 2.7585 - val_accuracy: 0.8750 - lr: 1.2500e-05
Epoch 17/20
7/7 [==============================] - ETA: 0s - loss: 0.3332 - accuracy: 0.9375
Epoch 17: ReduceLROnPlateau reducing learning rate to 1e-05.

Epoch 17: val_accuracy did not improve from 0.93750
7/7 [==============================] - 18s 3s/step - loss: 0.3332 - accuracy: 0.9
375 - val_loss: 3.3240 - val_accuracy: 0.8229 - lr: 1.2500e-05
Epoch 18/20
7/7 [==============================] - ETA: 0s - loss: 0.3812 - accuracy: 0.9375
Epoch 18: val_accuracy did not improve from 0.93750
7/7 [==============================] - 19s 3s/step - loss: 0.3812 - accuracy: 0.9
375 - val_loss: 4.0545 - val_accuracy: 0.7604 - lr: 1.0000e-05
Epoch 19/20
7/7 [==============================] - ETA: 0s - loss: 0.4686 - accuracy: 0.9196
Epoch 19: val_accuracy did not improve from 0.93750
7/7 [==============================] - 18s 3s/step - loss: 0.4686 - accuracy: 0.9
196 - val_loss: 5.6035 - val_accuracy: 0.6771 - lr: 1.0000e-05
Epoch 20/20
7/7 [==============================] - ETA: 0s - loss: 1.1105 - accuracy: 0.7902
Epoch 20: val_accuracy did not improve from 0.93750
7/7 [==============================] - 18s 3s/step - loss: 1.1105 - accuracy: 0.7
902 - val_loss: 7.7720 - val_accuracy: 0.6250 - lr: 1.0000e-05
3/3 [==============================] - 3s 766ms/step - loss: 3.4259 - accuracy:
0.6250
Test loss: 3.4258506298065186
Test accuracy: 0.625
```

In [408…

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten
```

```
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint


early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weig

from keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2, min_lr
                              verbose=1, mode='max')  # Reduce LR when accuracy

checkpoint_path = "vgg16_orl_faces-{epoch:02d}-{val_accuracy:.2f}.hdf5"  # Filen

checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_accuracy',
                             mode='max',  # Monitor for maximizing validation ac
                             save_best_only=True,  # Save only the best performi
                             verbose=1)


# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=32,  callbacks=[early_stopping

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Epoch 1/20
7/7 [==============================] - ETA: 0s - loss: 3.6753 - accuracy: 0.0357
Epoch 1: val_accuracy improved from -inf to 0.15625, saving model to vgg16_orl_fa
ces-01-0.16.hdf5
7/7 [==============================] - 18s 2s/step - loss: 3.6753 - accuracy: 0.0
357 - val_loss: 3.5079 - val_accuracy: 0.1562 - lr: 1.0000e-04
Epoch 2/20
7/7 [==============================] - ETA: 0s - loss: 3.1032 - accuracy: 0.1830
Epoch 2: val_accuracy did not improve from 0.15625
7/7 [==============================] - 17s 2s/step - loss: 3.1032 - accuracy: 0.1
830 - val_loss: 3.0938 - val_accuracy: 0.1458 - lr: 1.0000e-04
Epoch 3/20
7/7 [==============================] - ETA: 0s - loss: 2.5148 - accuracy: 0.2991
Epoch 3: val_accuracy improved from 0.15625 to 0.40625, saving model to vgg16_orl
_faces-03-0.41.hdf5
7/7 [==============================] - 18s 3s/step - loss: 2.5148 - accuracy: 0.2
991 - val_loss: 2.2152 - val_accuracy: 0.4062 - lr: 1.0000e-04
Epoch 4/20
7/7 [==============================] - ETA: 0s - loss: 1.2836 - accuracy: 0.6786
Epoch 4: val_accuracy improved from 0.40625 to 0.62500, saving model to vgg16_orl
_faces-04-0.62.hdf5
7/7 [==============================] - 18s 3s/step - loss: 1.2836 - accuracy: 0.6
786 - val_loss: 1.4951 - val_accuracy: 0.6250 - lr: 1.0000e-04
Epoch 5/20
7/7 [==============================] - ETA: 0s - loss: 0.4414 - accuracy: 0.8795
Epoch 5: val_accuracy improved from 0.62500 to 0.73958, saving model to vgg16_orl
_faces-05-0.74.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.4414 - accuracy: 0.8
795 - val_loss: 0.8161 - val_accuracy: 0.7396 - lr: 1.0000e-04
Epoch 6/20
7/7 [==============================] - ETA: 0s - loss: 0.1707 - accuracy: 0.9286
Epoch 6: val_accuracy improved from 0.73958 to 0.78125, saving model to vgg16_orl
_faces-06-0.78.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.1707 - accuracy: 0.9
286 - val_loss: 0.6164 - val_accuracy: 0.7812 - lr: 1.0000e-04
Epoch 7/20
7/7 [==============================] - ETA: 0s - loss: 0.1106 - accuracy: 0.9330
Epoch 7: val_accuracy improved from 0.78125 to 0.81250, saving model to vgg16_orl
_faces-07-0.81.hdf5
7/7 [==============================] - 18s 3s/step - loss: 0.1106 - accuracy: 0.9
330 - val_loss: 0.7365 - val_accuracy: 0.8125 - lr: 1.0000e-04
Epoch 8/20
7/7 [==============================] - ETA: 0s - loss: 0.1085 - accuracy: 0.9286
Epoch 8: val_accuracy did not improve from 0.81250
7/7 [==============================] - 19s 3s/step - loss: 0.1085 - accuracy: 0.9
286 - val_loss: 1.2023 - val_accuracy: 0.6771 - lr: 1.0000e-04
3/3 [==============================] - 3s 742ms/step - loss: 0.5728 - accuracy:
0.8375
Test loss: 0.5728174448013306
Test accuracy: 0.8374999761581421
```

In [454…

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.callbacks import Callback

class AccuracyThresholdCallback(Callback):
    def __init__(self, threshold=0.7, filepath=r"D:\OneDrive\Knowledge Center\AI
```

```
        super().__init__()
        self.threshold = threshold
        self.filepath = filepath


    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') >= self.threshold:
            filename = self.filepath.format(accuracy=logs['val_accuracy'])
            self.model.save(filename)  # Change to save the entire model
            print(f"Validation accuracy exceeded threshold! Saving model to: {fi

early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weig
from keras.callbacks import ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2, min_lr
                              verbose=1, mode='max')  # Reduce LR when accuracy
#checkpoint_path = "vgg16_orl_faces-{epoch:02d}-{val_accuracy:.2f}.hdf5"  # File
checkpoint = AccuracyThresholdCallback(threshold=0.6)

# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=32,  callbacks=[early_stopping

filepath=r"'D:\\OneDrive\\Knowledge Center\\AI - ML\\Masters in Artifical Engine
```

```
Epoch 1/20
7/7 [==============================] - 20s 3s/step - loss: 3.6293 - accuracy: 0.0
312 - val_loss: 3.7265 - val_accuracy: 0.0208 - lr: 1.0000e-04
Epoch 2/20
7/7 [==============================] - 20s 3s/step - loss: 3.1913 - accuracy: 0.1
741 - val_loss: 3.2496 - val_accuracy: 0.2188 - lr: 1.0000e-04
Epoch 3/20
7/7 [==============================] - 18s 3s/step - loss: 2.3403 - accuracy: 0.3
795 - val_loss: 2.7203 - val_accuracy: 0.2083 - lr: 1.0000e-04
Epoch 4/20
7/7 [==============================] - ETA: 0s - loss: 1.3451 - accuracy: 0.5982V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.72.hdf5
7/7 [==============================] - 22s 3s/step - loss: 1.3451 - accuracy: 0.5
982 - val_loss: 1.3435 - val_accuracy: 0.7188 - lr: 1.0000e-04
Epoch 5/20
7/7 [==============================] - ETA: 0s - loss: 0.4843 - accuracy: 0.8527V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.82.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.4843 - accuracy: 0.8
527 - val_loss: 0.7590 - val_accuracy: 0.8229 - lr: 1.0000e-04
Epoch 6/20
7/7 [==============================] - ETA: 0s - loss: 0.1441 - accuracy: 0.9420V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.84.hdf5
7/7 [==============================] - 20s 3s/step - loss: 0.1441 - accuracy: 0.9
420 - val_loss: 0.5877 - val_accuracy: 0.8438 - lr: 1.0000e-04
Epoch 7/20
7/7 [==============================] - ETA: 0s - loss: 0.0512 - accuracy: 0.9509V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.77.hdf5
7/7 [==============================] - 20s 3s/step - loss: 0.0512 - accuracy: 0.9
509 - val_loss: 1.0263 - val_accuracy: 0.7708 - lr: 1.0000e-04
Epoch 8/20
7/7 [==============================] - ETA: 0s - loss: 0.5688 - accuracy: 0.8125V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.71.hdf5
7/7 [==============================] - 19s 3s/step - loss: 0.5688 - accuracy: 0.8
125 - val_loss: 1.3656 - val_accuracy: 0.7083 - lr: 1.0000e-04
```

In [414…

```
import os
print(os.getcwd())  # Prints the current working directory
```

```
C:\Users\naseh
```

In [455…

```
import os
from keras.models import load_model

saved_models_dir = 'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artifical En
model_filenames = [f for f in os.listdir(saved_models_dir) if f.endswith('.hdf5'

for model_filename in model_filenames:
    model_path = os.path.join(saved_models_dir, model_filename)
```

```
model = load_model(model_path)
print(f"Model: {model_filename} - Test Loss: {test_loss:.4f} - Test Accuracy: {t
```

```
Model: model_acc_0.91.hdf5 - Test Loss: 0.5489 - Test Accuracy: 0.7875
```

In [456…

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.callbacks import Callback

class AccuracyThresholdCallback(Callback):
    def __init__(self, threshold=0.9, filepath=r"D:\OneDrive\Knowledge Center\AI
        super().__init__()
        self.threshold = threshold
        self.filepath = filepath


    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') > self.threshold:
            filename = self.filepath.format(accuracy=logs['val_accuracy'])
            self.model.save(filename)  # Change to save the entire model
            print(f"Validation accuracy exceeded threshold! Saving model to: {fi

early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weig
from keras.callbacks import ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2, min_lr
                              verbose=1, mode='max')  # Reduce LR when accuracy
#checkpoint_path = "vgg16_orl_faces-{epoch:02d}-{val_accuracy:.2f}.hdf5"  # File
checkpoint = AccuracyThresholdCallback(threshold=0.6)

# Load VGG16 model with ImageNet weights, exclude the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(92, 112,

# Freeze some of the initial layers (optional)
for layer in base_model.layers[:10]:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = Flatten()(x)
predictions = Dense(40, activation='softmax')(x) # 40 classes

model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Nadam(learning_rate=0.0

# Train (consider using a lower learning rate for fine-tuning)
model.fit(X_train, y_train, epochs=20, batch_size=32,  callbacks=[early_stopping

filepath=r"'D:\\OneDrive\\Knowledge Center\\AI - ML\\Masters in Artifical Engine
```

```
Epoch 1/20
7/7 [==============================] - 25s 3s/step - loss: 3.6739 - accuracy: 0.0
134 - val_loss: 3.6225 - val_accuracy: 0.0417
Epoch 2/20
7/7 [==============================] - 21s 3s/step - loss: 3.2160 - accuracy: 0.1
429 - val_loss: 3.0034 - val_accuracy: 0.3646
Epoch 3/20
7/7 [==============================] - 20s 3s/step - loss: 2.3834 - accuracy: 0.3
527 - val_loss: 2.6043 - val_accuracy: 0.3125
Epoch 4/20
7/7 [==============================] - 24s 4s/step - loss: 1.4896 - accuracy: 0.5
938 - val_loss: 1.3733 - val_accuracy: 0.5938
Epoch 5/20
7/7 [==============================] - ETA: 0s - loss: 0.6392 - accuracy: 0.7679V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.75.hdf5
7/7 [==============================] - 24s 4s/step - loss: 0.6392 - accuracy: 0.7
679 - val_loss: 0.7774 - val_accuracy: 0.7500
Epoch 6/20
7/7 [==============================] - ETA: 0s - loss: 0.2434 - accuracy: 0.9062V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.84.hdf5
7/7 [==============================] - 23s 3s/step - loss: 0.2434 - accuracy: 0.9
062 - val_loss: 0.4966 - val_accuracy: 0.8438
Epoch 7/20
7/7 [==============================] - ETA: 0s - loss: 0.0906 - accuracy: 0.9420V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.90.hdf5
7/7 [==============================] - 22s 3s/step - loss: 0.0906 - accuracy: 0.9
420 - val_loss: 0.3397 - val_accuracy: 0.8958
Epoch 8/20
7/7 [==============================] - ETA: 0s - loss: 0.1322 - accuracy: 0.9196V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.74.hdf5
7/7 [==============================] - 23s 3s/step - loss: 0.1322 - accuracy: 0.9
196 - val_loss: 1.1945 - val_accuracy: 0.7396
Epoch 9/20
7/7 [==============================] - ETA: 0s - loss: 1.7123 - accuracy: 0.6875V
alidation accuracy exceeded threshold! Saving model to: D:\OneDrive\Knowledge Cen
ter\AI - ML\Masters in Artifical Engineer\Advanced Deep Learning and Computer Vis
ion\model_acc_0.74.hdf5
7/7 [==============================] - 22s 3s/step - loss: 1.7123 - accuracy: 0.6
875 - val_loss: 1.2917 - val_accuracy: 0.7396
```

In [460…

```python
import os
from keras.models import load_model

saved_models_dir = 'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artifical En
model_filenames = [f for f in os.listdir(saved_models_dir) if f.endswith('.hdf5'

for model_filename in model_filenames:
    model_path = os.path.join(saved_models_dir, model_filename)

model = load_model(model_path)
print(f"Model: {model_filename} - Test Loss: {test_loss:.4f} - Test Accuracy: {t
```

```
Model: model_acc_0.91.hdf5 - Test Loss: 0.5489 - Test Accuracy: 0.7875
```

As we see using the above model we could get accuracy above 90%