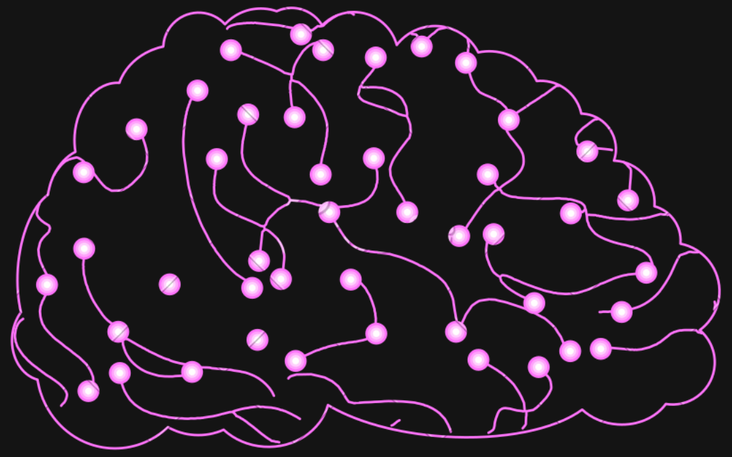
Artificial Neural Network

Face recognition using Convolutional Neural Networks

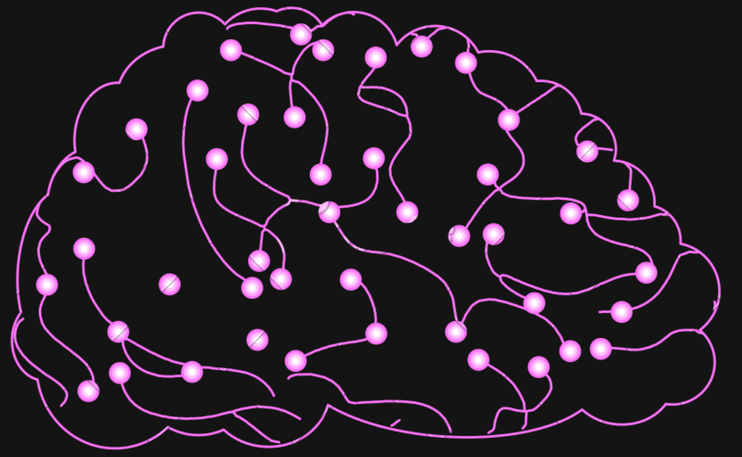


Question 1: What is the output of the model.summary()?

```
#Print a summary of the model
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 64, 64, 8)         208

 max_pooling2d_12 (MaxPooli  (None, 16, 16, 8)         0
 ng2D)

 conv2d_13 (Conv2D)          (None, 16, 16, 16)        3216

 max_pooling2d_13 (MaxPooli  (None, 4, 4, 16)          0
 ng2D)

 conv2d_14 (Conv2D)          (None, 4, 4, 32)          4640

 max_pooling2d_14 (MaxPooli  (None, 2, 2, 32)          0
 ng2D)

 flatten_4 (Flatten)         (None, 128)               0

 dense_8 (Dense)             (None, 256)               33024

 dense_9 (Dense)             (None, 40)                10280

=================================================================
Total params: 51368 (200.66 KB)
Trainable params: 51368 (200.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Information about all the layers in the model, the information includes layer type, output shape, and number of parameters. Total params is the trainable parameters in the model.



Question 2: What the initial training accuracy and validation accuracy of the CNN?

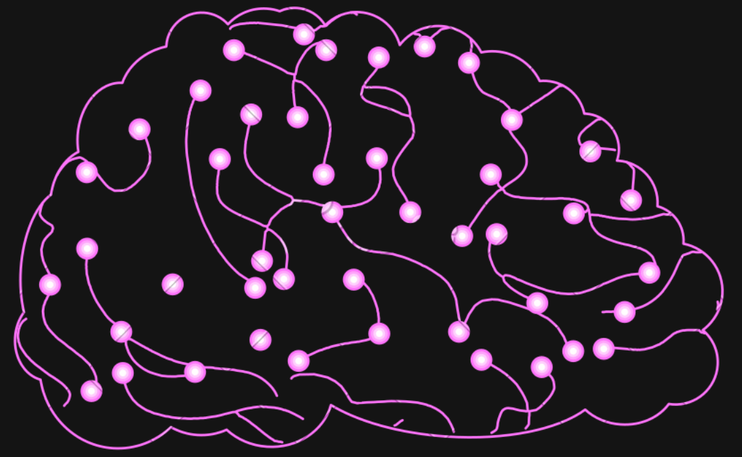
```
initial_training_accuracy = H.history['accuracy'][0]
initial_validation_accuracy = H.history['val_accuracy'][0]

print(f'Initial Training Accuracy: {initial_training_accuracy}')
print(f'Initial Validation Accuracy: {initial_validation_accuracy}')
```

```
Initial Training Accuracy: 0.00390625
Initial Validation Accuracy: 0.015625
```

The initial training accuracy and validation accuracy are very low

```
[89] training_accuracy = H.history['accuracy']
     validation_accuracy = H.history['val_accuracy']

     print("Training Accuracy:\n",training_accuracy)
     print("\nValidation Accuracy:\n",validation_accuracy)
```

```
Training Accuracy:
 [0.00390625, 0.01953125, 0.03125, 0.03515625, 0.03515625, 0.046875, 0.046875, 0.0390625, 0.06640625, 0.078125]

Validation Accuracy:
 [0.015625, 0.015625, 0.0, 0.0, 0.0, 0.015625, 0.0, 0.0, 0.046875, 0.046875]
```



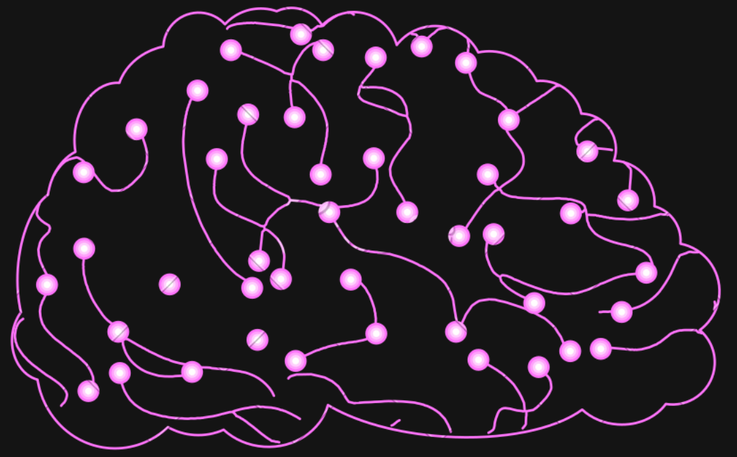
Question 3: How many convolutional layers and pooling layers does this network have?

```
[90] num_conv_layers = sum(1 for layer in model.layers if isinstance(layer, Conv2D))
     num_pooling_layers = sum(1 for layer in model.layers if isinstance(layer, MaxPooling2D))
```

```
print(f"Number of Convolutional Layers: {num_conv_layers}")
print(f"Number of Pooling Layers: {num_pooling_layers}")
```

```
Number of Convolutional Layers: 3
Number of Pooling Layers: 3
```

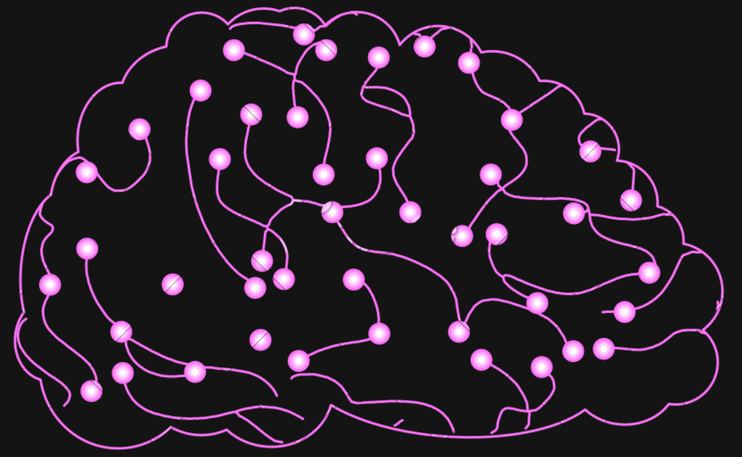
It contain 3 convolutional layers
And 3 pooling layers



Question 4: Generally, the larger the size of the image the more the information in it. The max- pooling layers after first and second Convolutional layer decrease the size of the image by 4. Check if this is causing the network to have such a poor validation accuracy? If the size of pooling layers size is changed from (4,4) to (2,2) what is the effect on accuracy of the network?

Max Pooling (4,4)

```
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)

Epoch 1/10
8/8 [==============================] - 2s 113ms/step - loss: 3.8280 - accuracy: 0.0273 - val_loss: 3.8543 - val_accuracy: 0.0156
Epoch 2/10
8/8 [==============================] - 1s 87ms/step - loss: 3.6948 - accuracy: 0.0312 - val_loss: 3.8018 - val_accuracy: 0.0000e+00
Epoch 3/10
8/8 [==============================] - 1s 87ms/step - loss: 3.6877 - accuracy: 0.0273 - val_loss: 3.7969 - val_accuracy: 0.0000e+00
Epoch 4/10
8/8 [==============================] - 1s 86ms/step - loss: 3.6768 - accuracy: 0.0234 - val_loss: 3.8161 - val_accuracy: 0.0156
Epoch 5/10
8/8 [==============================] - 1s 85ms/step - loss: 3.6754 - accuracy: 0.0312 - val_loss: 3.8257 - val_accuracy: 0.0000e+00
Epoch 6/10
8/8 [==============================] - 1s 85ms/step - loss: 3.6709 - accuracy: 0.0273 - val_loss: 3.8027 - val_accuracy: 0.0000e+00
Epoch 7/10
8/8 [==============================] - 1s 86ms/step - loss: 3.6659 - accuracy: 0.0273 - val_loss: 3.7674 - val_accuracy: 0.0312
Epoch 8/10
8/8 [==============================] - 1s 84ms/step - loss: 3.6621 - accuracy: 0.0312 - val_loss: 3.7749 - val_accuracy: 0.0312
Epoch 9/10
8/8 [==============================] - 1s 86ms/step - loss: 3.6547 - accuracy: 0.0391 - val_loss: 3.7712 - val_accuracy: 0.0000e+00
Epoch 10/10
8/8 [==============================] - 1s 86ms/step - loss: 3.6429 - accuracy: 0.0273 - val_loss: 3.7547 - val_accuracy: 0.0156
```



Question 4 cont:

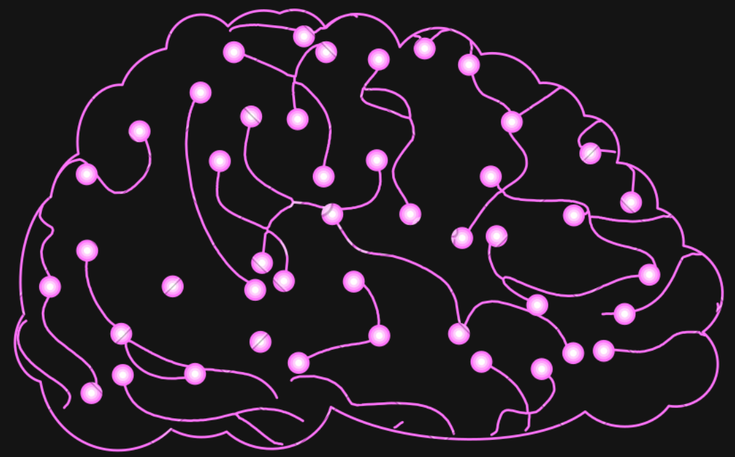
Max Pooling (2,2)

```
model2.add(Conv2D(8, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(16, (5, 5), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(256, activation='sigmoid'))

model2.add(Dense(40, activation='softmax'))

model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
H_modified = model2.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
```

```
Epoch 1/10
8/8 [==============================] - 3s 180ms/step - loss: 3.8217 - accuracy: 0.0352 - val_loss: 3.7862 - val_accuracy: 0.0000e+00
Epoch 2/10
8/8 [==============================] - 1s 163ms/step - loss: 3.6821 - accuracy: 0.0312 - val_loss: 3.7900 - val_accuracy: 0.0312
Epoch 3/10
8/8 [==============================] - 1s 149ms/step - loss: 3.6528 - accuracy: 0.0312 - val_loss: 3.7675 - val_accuracy: 0.0312
Epoch 4/10
8/8 [==============================] - 1s 144ms/step - loss: 3.6158 - accuracy: 0.1016 - val_loss: 3.7324 - val_accuracy: 0.0938
Epoch 5/10
8/8 [==============================] - 1s 132ms/step - loss: 3.5521 - accuracy: 0.1133 - val_loss: 3.6406 - val_accuracy: 0.1094
Epoch 6/10
8/8 [==============================] - 1s 138ms/step - loss: 3.4137 - accuracy: 0.2500 - val_loss: 3.4370 - val_accuracy: 0.1562
Epoch 7/10
8/8 [==============================] - 1s 137ms/step - loss: 3.1304 - accuracy: 0.4062 - val_loss: 3.1434 - val_accuracy: 0.2812
Epoch 8/10
8/8 [==============================] - 1s 137ms/step - loss: 2.6825 - accuracy: 0.4297 - val_loss: 2.6613 - val_accuracy: 0.4062
Epoch 9/10
8/8 [==============================] - 2s 213ms/step - loss: 2.2125 - accuracy: 0.5469 - val_loss: 2.2845 - val_accuracy: 0.5156
Epoch 10/10
8/8 [==============================] - 2s 230ms/step - loss: 1.6434 - accuracy: 0.7617 - val_loss: 1.7271 - val_accuracy: 0.6562
```

After Reducing Max Pooling to (2,2):
Training loss decreased at a faster rate compared to before reducing Max Pooling, indicating that the model was learning more effectively.
Training accuracy increased, 76% .
Validation loss decreased, which means the model was generalizing better.
Validation accuracy improved, over 65%



Question 5: Dr. Hinton, has highlighted that aggressively using pooling layers may result in loss of important information. Is there a way that the CNN architecture starts producing better training and validation accuracy?

```
model3=Sequential()
#1.Reduce Pooling Size or Stride
#2.Hyperparameter Tuning
#3.Utilize Strided Convolutions
#4.Add more convolutional layer

model3.add(Conv2D(8, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
#1
model3.add(MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))


model3.add(Conv2D(16, (5, 5), activation='relu', padding='same'))
#1
model3.add(MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))

#3
model3.add(Conv2D(32, (3, 3), activation='relu', padding='same', strides=(2, 2)))
#1
model3.add(MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))
#4
model3.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model3.add(Flatten())
model3.add(Dense(256, activation='sigmoid'))

model3.add(Dense(40, activation='softmax'))

#2
model3.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=['accuracy'])
```

```
[149] H_model3 = model3.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)

Epoch 1/10
8/8 [==============================] - 10s 976ms/step - loss: 3.9421 - accuracy: 0.0078 - val_loss: 3.8706 - val_accuracy: 0.0156
Epoch 2/10
8/8 [==============================] - 5s 685ms/step - loss: 3.6701 - accuracy: 0.0234 - val_loss: 3.7567 - val_accuracy: 0.0000e+00
Epoch 3/10
8/8 [==============================] - 7s 973ms/step - loss: 3.4742 - accuracy: 0.1367 - val_loss: 3.3302 - val_accuracy: 0.3438
Epoch 4/10
8/8 [==============================] - 6s 748ms/step - loss: 2.6672 - accuracy: 0.5156 - val_loss: 2.2515 - val_accuracy: 0.4844
Epoch 5/10
8/8 [==============================] - 8s 987ms/step - loss: 1.3089 - accuracy: 0.7852 - val_loss: 1.5699 - val_accuracy: 0.6094
Epoch 6/10
8/8 [==============================] - 6s 695ms/step - loss: 0.7284 - accuracy: 0.9141 - val_loss: 1.0334 - val_accuracy: 0.8281
Epoch 7/10
8/8 [==============================] - 6s 699ms/step - loss: 0.3884 - accuracy: 0.9805 - val_loss: 0.8111 - val_accuracy: 0.8750
Epoch 8/10
8/8 [==============================] - 8s 916ms/step - loss: 0.2301 - accuracy: 1.0000 - val_loss: 0.7377 - val_accuracy: 0.8750
Epoch 9/10
8/8 [==============================] - 5s 674ms/step - loss: 0.1394 - accuracy: 1.0000 - val_loss: 0.6555 - val_accuracy: 0.8438
Epoch 10/10
8/8 [==============================] - 8s 1000ms/step - loss: 0.0957 - accuracy: 1.0000 - val_loss: 0.6414 - val_accuracy: 0.8594
```

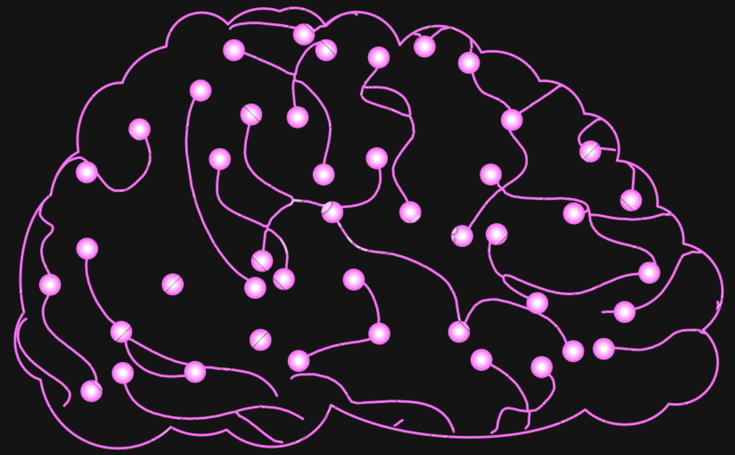
```
test_loss, test_accuracy = model3.evaluate(X_test, Y_test, verbose=0)
print(f"\nTest Accuracy of Modified Model: {test_accuracy}")
```

```
Test Accuracy of Modified Model: 0.887499988079071
```

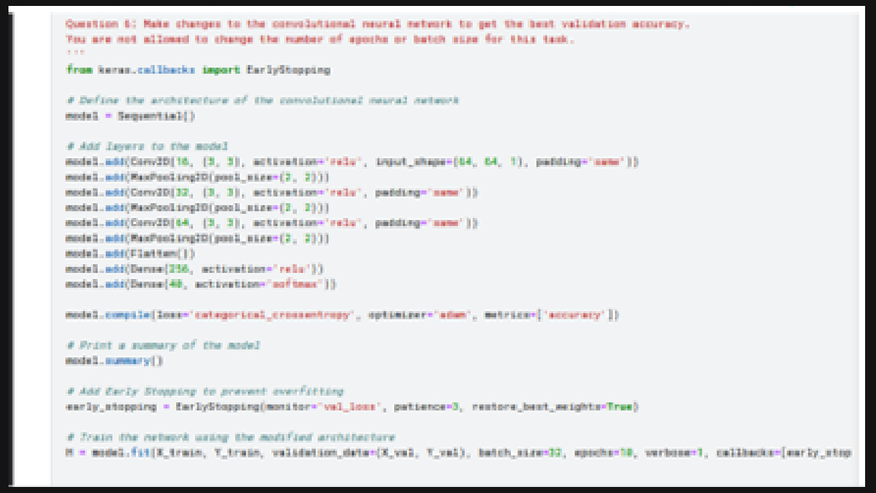
We used four techniques that can provide better training and validation accuracy

1-Reduce Pooling Size or Stride
2-Hyperparameter Tuning
3-Utilize Strided Convolutions
4-Use Different Architectures

Which imroved the accuracy significantly



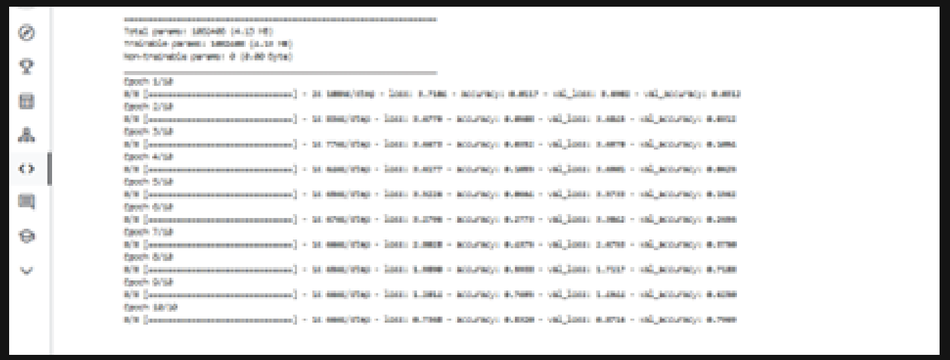
Question 6: Make changes to the convolutional neural network to get the best validation accuracy.
You are not allowed to change the number of epochs or batch size for this task.

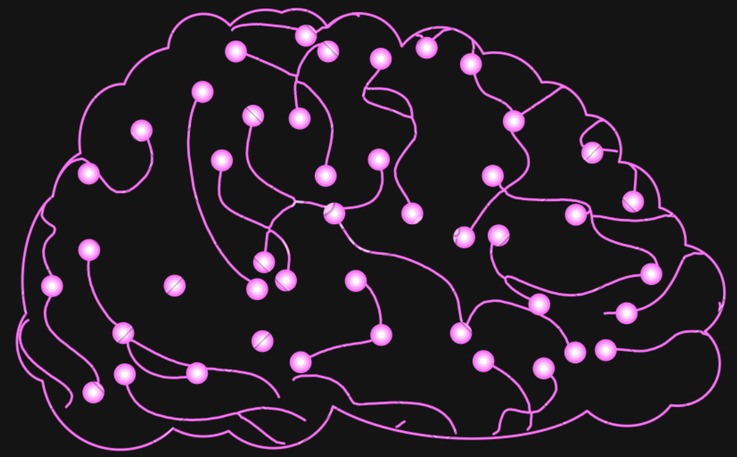


```
Question 6: Make changes to the convolutional neural network to get the best validation accuracy.
You are not allowed to change the number of epochs or batch size for this task.
'''
from keras.callbacks import EarlyStopping

# Define the architecture of the convolutional neural network
model = Sequential()

# Add layers to the model
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(40, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print a summary of the model
model.summary()

# Add Early Stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the network using the modified architecture
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=30, verbose=1, callbacks=[early_stop
```

I.Increased Depth:
Added an extra convolutional layer (Conv2D(64, (3, 3), activation='relu', padding='same')) to capture more complex features.

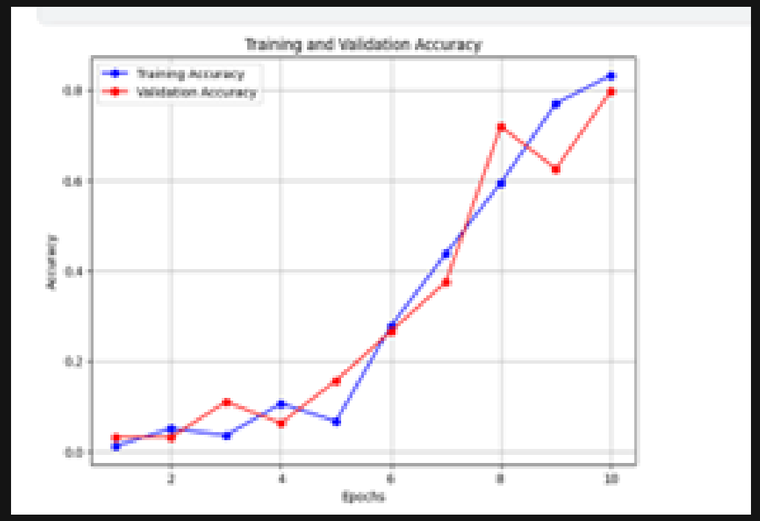
.



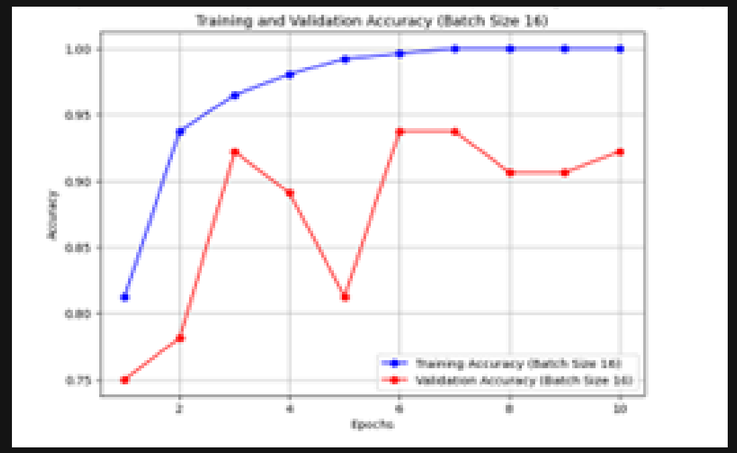


Question 7: Plot the difference between training and validation accuracy for each epoch.

```
'''
Question 7: Plot the difference between training and validation accuracy for each epoch.
'''
# Extracting accuracy values from the history object
train_accuracy = H.history['accuracy']
val_accuracy = H.history['val_accuracy']

# Plotting training and validation accuracy
plt.figure(figsize=(8, 6))
epochs = range(1, len(train_accuracy) + 1)
plt.plot(epochs, train_accuracy, 'bo-', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'ro-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()
```



Question 8: For the best network architecture change the batch size to 16 and plot the training vs validation accuracy graph. What happened to the validation accuracy after last epoch as compared to when the batch size was 32



For batch size 32:
Validation accuracy after the last epoch: 0.7969 (Epoch 10/10 val_accuracy: 0.7969)
For batch size 16:
Validation accuracy after the last epoch: 0.9219 (Epoch 10/10 val_accuracy: 0.9219)
The validation accuracy after the last epoch increased significantly when the batch size was changed from 32 to 16. This suggests that with a smaller batch size, the model might generalize better and achieve .higher accuracy on unseen data

Question 8: For the best network architecture change the batch size to 16 and plot the training vs validation accuracy graph. What happened to the validation accuracy after last epoch as compared to when the batch size was 32

```
'''
Question 8: For the best network architecture change the batch size to 16 and plot the training vs
validation accuracy graph. What happened to the validation accuracy after last epoch as compared
to when the batch size was 32.
'''
# Change batch size to 16 and recompile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Retrain the network using the updated batch size
H_batch_16 = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=16, epochs=10, verbose=1)

# Extracting accuracy values from the history object for batch size 16
train_accuracy_batch_16 = H_batch_16.history['accuracy']
val_accuracy_batch_16 = H_batch_16.history['val_accuracy']

# Plotting training and validation accuracy for batch size 16
plt.figure(figsize=(8, 6))
epochs_batch_16 = range(1, len(train_accuracy_batch_16) + 1)
plt.plot(epochs_batch_16, train_accuracy_batch_16, 'bo-', label='Training Accuracy (Batch Size 16)')
plt.plot(epochs_batch_16, val_accuracy_batch_16, 'ro-', label='Validation Accuracy (Batch Size 16)')
plt.title('Training and Validation Accuracy (Batch Size 16)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()
```

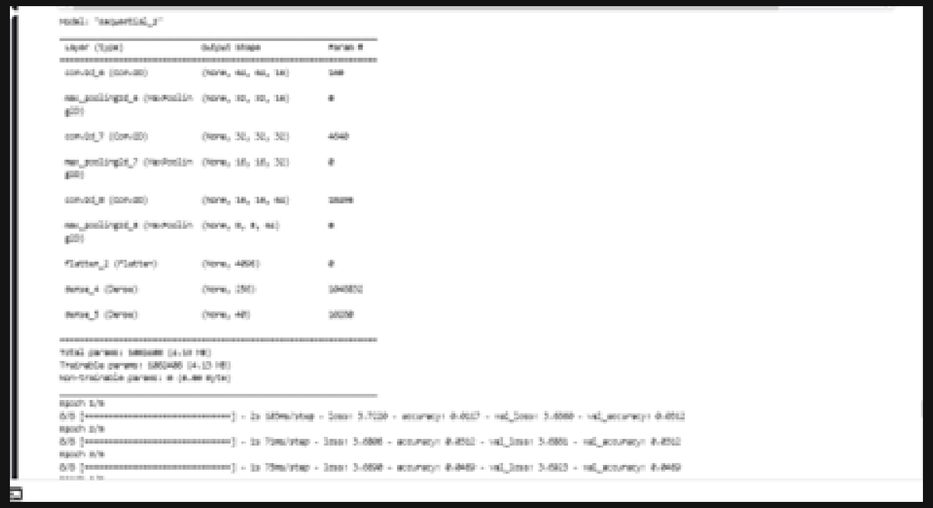
```
Epoch 1/10
16/16 [==============================] - 2s 50ms/step - loss: 0.7989 - accuracy: 0.8125 - val_loss: 0.9736 - val_accuracy: 0.7500
Epoch 2/10
16/16 [==============================] - 1s 38ms/step - loss: 0.3618 - accuracy: 0.9375 - val_loss: 0.7733 - val_accuracy: 0.7852
Epoch 3/10
16/16 [==============================] - 1s 37ms/step - loss: 0.1792 - accuracy: 0.9648 - val_loss: 0.5377 - val_accuracy: 0.9219
Epoch 4/10
16/16 [==============================] - 1s 37ms/step - loss: 0.0826 - accuracy: 0.9805 - val_loss: 0.3753 - val_accuracy: 0.8906
Epoch 5/10
16/16 [==============================] - 1s 39ms/step - loss: 0.0513 - accuracy: 0.9902 - val_loss: 0.4736 - val_accuracy: 0.8125
Epoch 6/10
16/16 [==============================] - 1s 38ms/step - loss: 0.0282 - accuracy: 0.9961 - val_loss: 0.1879 - val_accuracy: 0.9375
Epoch 7/10
16/16 [==============================] - 1s 40ms/step - loss: 0.0157 - accuracy: 1.0000 - val_loss: 0.2077 - val_accuracy: 0.9375
Epoch 8/10
16/16 [==============================] - 1s 38ms/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.2336 - val_accuracy: 0.9062
Epoch 9/10
16/16 [==============================] - 1s 39ms/step - loss: 0.0149 - accuracy: 1.0000 - val_loss: 0.2340 - val_accuracy: 0.9062
Epoch 10/10
16/16 [==============================] - 1s 39ms/step - loss: 0.0087 - accuracy: 1.0000 - val_loss: 0.1854 - val_accuracy: 0.9219
```

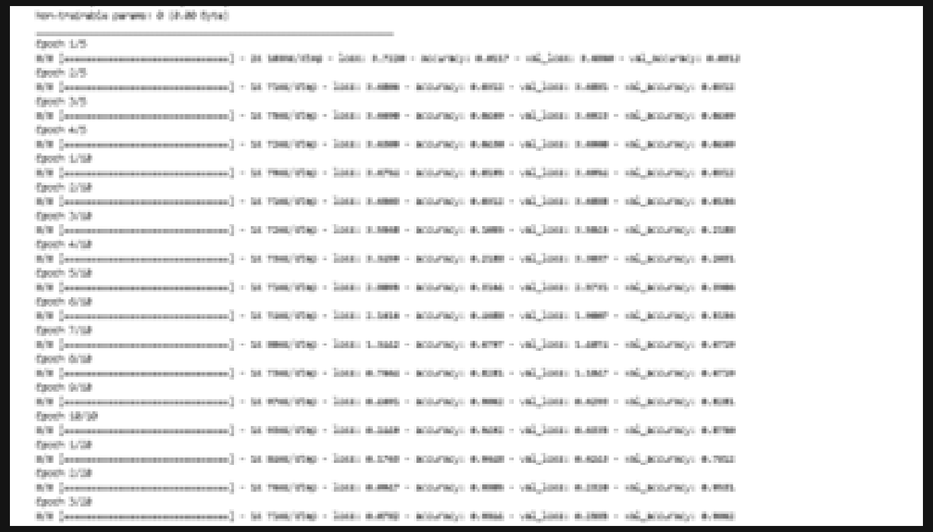
Question 9: For the best network architecture change the number of epochs to 5 and 20 and share the final validation accuracy for 5, 10 and 20 epochs. What do the results highlight?

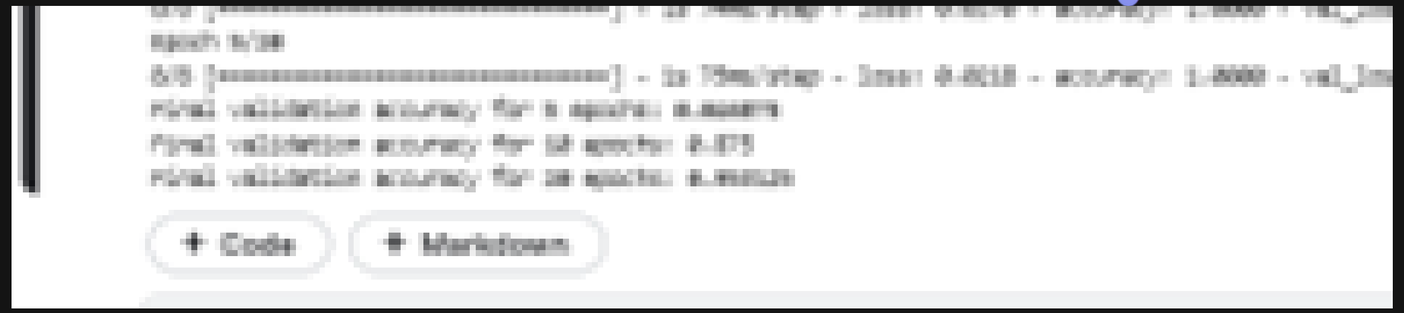
```
'''
Question 9: For the best network architecture change the number of epochs to 5 and 20 and
share the final validation accuracy for 5, 10 and 20 epochs. What do the results highlight?
'''
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.callbacks import EarlyStopping

# Load data, face images, and their target i.e., person number
data = np.load('/kaggle/input/d/sahilyagnik/olivetti-faces/olivetti_faces.npy')
target = np.load('/kaggle/input/d/sahilyagnik/olivetti-faces/olivetti_faces_target.npy')

# Convert target labels to one-hot encoded format using to_categorical
target_encoded = to_categorical(target, num_classes=len(np.unique(target)))

# Use sklearn library to split data into training, validation, and testing set
X_train, X_test, Y_train, Y_test = train_test_split(data, target_encoded, test_size=0.2, random_state=0)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)

# Reshape data for CNN input
X_train = X_train.reshape(X_train.shape[0], 64, 64, 1)
X_val = X_val.reshape(X_val.shape[0], 64, 64, 1)
X_test = X_test.reshape(X_test.shape[0], 64, 64, 1)
# Define the architecture of the convolutional neural network (unchanged)
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
```

```
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(40, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the network with 5 epochs
H_5_epochs = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=5, verbose=1, callbacks=[early

# Train the network with 10 epochs (as per original code)
H_10_epochs = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1, callbacks=[ear

# Train the network with 20 epochs
H_20_epochs = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=20, verbose=1, callbacks=[ear

# Obtain final validation accuracies
val_accuracy_5_epochs = H_5_epochs.history['val_accuracy'][-1]
val_accuracy_10_epochs = H_10_epochs.history['val_accuracy'][-1]
val_accuracy_20_epochs = H_20_epochs.history['val_accuracy'][-1]

print(f"Final validation accuracy for 5 epochs: {val_accuracy_5_epochs}")
print(f"Final validation accuracy for 10 epochs: {val_accuracy_10_epochs}")
```

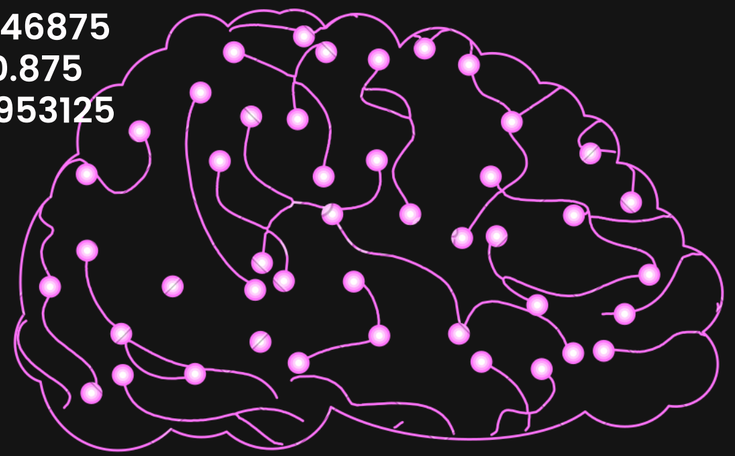
Question 9: For the best network architecture change the number of epochs to 5 and 20 and share the final validation accuracy for 5, 10 and 20 epochs. What do the results highlight?







Final validation accuracy for 5 epochs: 0.046875
Final validation accuracy for 10 epochs: 0.875
Final validation accuracy for 20 epochs: 0.953125



Question 10: For the best network architecture and batch size =16 and epochs =10, change the test data size to 40% and share what is the effect on validation accuracy of the algorithm?

```
Question 10: For the best network architecture and batch size =16 and epochs =10,
change the test data size to 40% and share what is the effect on validation accuracy of the algorithm?
'''
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Load data, face images, and their target i.e., person number
data = np.load('/kaggle/input/d/sahilyagnik/olivetti-faces/olivetti_faces.npy')
target = np.load('/kaggle/input/d/sahilyagnik/olivetti-faces/olivetti_faces_target.npy')

# Convert target labels to one-hot encoded format using to_categorical
target_encoded = to_categorical(target, num_classes=len(np.unique(target)))

# Use sklearn library to split data into training, validation, and testing set (60-20-20 split)
X_train, X_test, Y_train, Y_test = train_test_split(data, target_encoded, test_size=0.4, random_state=0)
X_val, X_test, Y_val, Y_test = train_test_split(X_test, Y_test, test_size=0.5, random_state=0)  # Change test size to 40%

# Reshape data for CNN input
X_train = X_train.reshape(X_train.shape[0], 64, 64, 1)
X_val = X_val.reshape(X_val.shape[0], 64, 64, 1)
X_test = X_test.reshape(X_test.shape[0], 64, 64, 1)

# Define the architecture of the convolutional neural network
model = Sequential()
model.add(Conv2D(8, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(16, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
```

```
model.add(Dense(40, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the network using the provided architecture and dataset
model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=16, epochs=10, verbose=1)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, Y_test, verbose=1)
print(f"Test accuracy with 40% test data size: {test_accuracy}")
```

```
Epoch 1/10
13/13 [==============================] - 2s 42ms/step - loss: 3.9113 - accuracy: 0.0083 - val_loss: 3.6941 - val_accuracy: 0.0000e+00
Epoch 2/10
13/13 [==============================] - 0s 28ms/step - loss: 3.7137 - accuracy: 0.0450 - val_loss: 3.6048 - val_accuracy: 0.0000e+00
Epoch 3/10
13/13 [==============================] - 0s 28ms/step - loss: 3.7017 - accuracy: 0.0250 - val_loss: 3.6407 - val_accuracy: 0.0000e+00
Epoch 4/10
13/13 [==============================] - 0s 28ms/step - loss: 3.6897 - accuracy: 0.0250 - val_loss: 3.6258 - val_accuracy: 0.0000e+00
Epoch 5/10
13/13 [==============================] - 0s 28ms/step - loss: 3.6854 - accuracy: 0.0333 - val_loss: 3.6304 - val_accuracy: 0.0000e+00
Epoch 6/10
13/13 [==============================] - 0s 27ms/step - loss: 3.6806 - accuracy: 0.0250 - val_loss: 3.6223 - val_accuracy: 0.0000e+00
Epoch 7/10
13/13 [==============================] - 0s 28ms/step - loss: 3.6671 - accuracy: 0.0167 - val_loss: 3.6258 - val_accuracy: 0.0250
Epoch 8/10
13/13 [==============================] - 0s 28ms/step - loss: 3.6594 - accuracy: 0.0375 - val_loss: 3.7008 - val_accuracy: 0.0000e+00
Epoch 9/10
13/13 [==============================] - 0s 27ms/step - loss: 3.6468 - accuracy: 0.0500 - val_loss: 3.7018 - val_accuracy: 0.0000e+00
Epoch 10/10
13/13 [==============================] - 0s 28ms/step - loss: 3.6222 - accuracy: 0.0625 - val_loss: 3.7328 - val_accuracy: 0.0250
8/8 [==============================] - 0s 1ms/step - loss: 3.6811 - accuracy: 0.0625
Test accuracy with 40% test data size: 0.0625
```

Test accuracy with 40% test data size: 0.0625

THE END