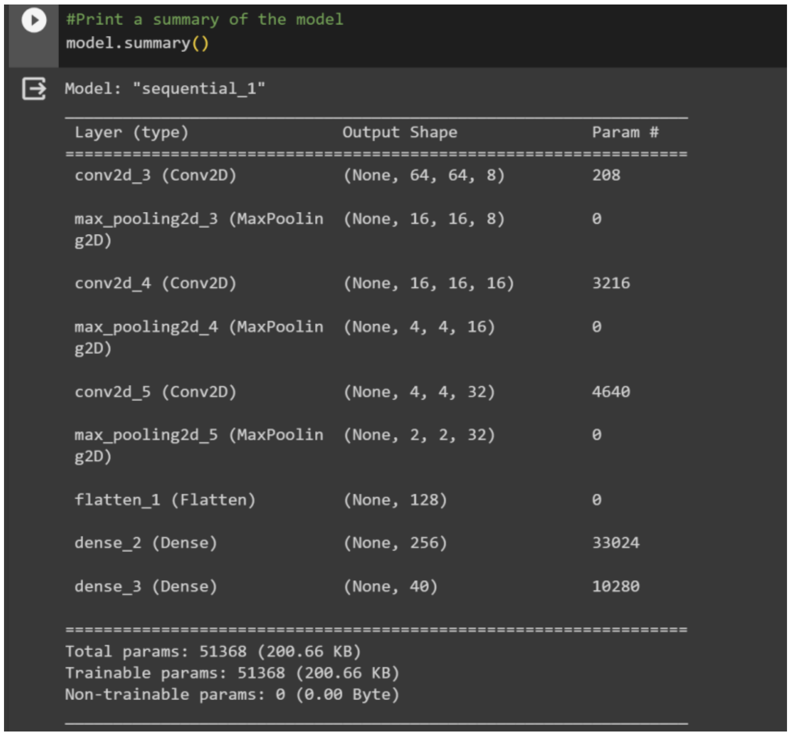
Artificial Neural Networks Project

Face recognition using Convolutional Neural Networks

Tasks

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

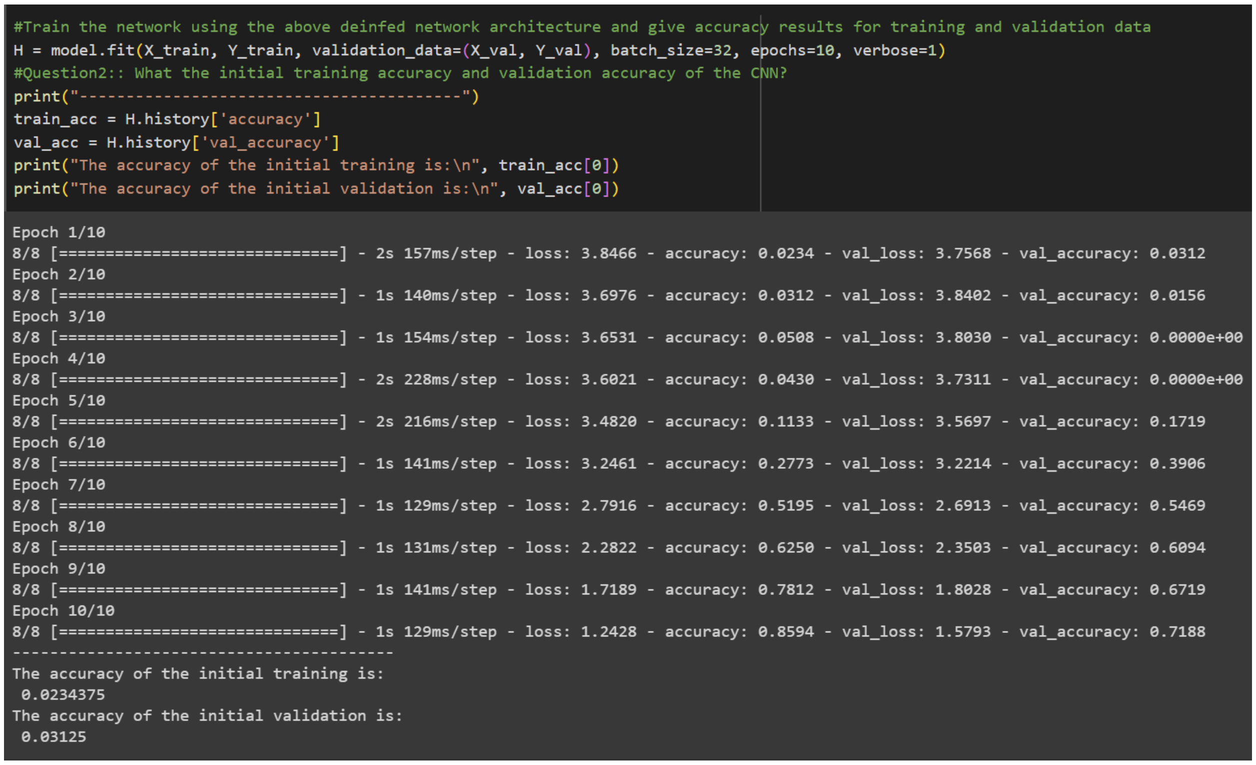
The model.summary() function provides a summary of the architecture of the CNN model. It displays information about each layer in the model, including the layer type, output shape, and number of parameters.



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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 64, 64, 8)         208       
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 16, 16, 8)         0         
 g2D)                                                            
                                                                 
 conv2d_4 (Conv2D)           (None, 16, 16, 16)        3216      
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 4, 4, 16)          0         
 g2D)                                                            
                                                                 
 conv2d_5 (Conv2D)           (None, 4, 4, 32)          4640      
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 2, 2, 32)          0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 256)               33024     
                                                                 
 dense_3 (Dense)             (None, 40)                10280     
                                                                 
=================================================================
Total params: 51368 (200.66 KB)
Trainable params: 51368 (200.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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



```
#Train the network using the above deinfed network architecture and give accuracy results for training and validation data
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
#Question2:: What the initial training accuracy and validation accuracy of the CNN?
print("-----------------------------------------")
train_acc = H.history['accuracy']
val_acc = H.history['val_accuracy']
print("The accuracy of the initial training is:\n", train_acc[0])
print("The accuracy of the initial validation is:\n", val_acc[0])
```

```
Epoch 1/10
8/8 [==============================] - 2s 157ms/step - loss: 3.8466 - accuracy: 0.0234 - val_loss: 3.7568 - val_accuracy: 0.0312
Epoch 2/10
8/8 [==============================] - 1s 140ms/step - loss: 3.6976 - accuracy: 0.0312 - val_loss: 3.8402 - val_accuracy: 0.0156
Epoch 3/10
8/8 [==============================] - 1s 154ms/step - loss: 3.6531 - accuracy: 0.0508 - val_loss: 3.8030 - val_accuracy: 0.0000e+00
Epoch 4/10
8/8 [==============================] - 2s 228ms/step - loss: 3.6021 - accuracy: 0.0430 - val_loss: 3.7311 - val_accuracy: 0.0000e+00
Epoch 5/10
8/8 [==============================] - 2s 216ms/step - loss: 3.4820 - accuracy: 0.1133 - val_loss: 3.5697 - val_accuracy: 0.1719
Epoch 6/10
8/8 [==============================] - 1s 141ms/step - loss: 3.2461 - accuracy: 0.2773 - val_loss: 3.2214 - val_accuracy: 0.3906
Epoch 7/10
8/8 [==============================] - 1s 129ms/step - loss: 2.7916 - accuracy: 0.5195 - val_loss: 2.6913 - val_accuracy: 0.5469
Epoch 8/10
8/8 [==============================] - 1s 131ms/step - loss: 2.2822 - accuracy: 0.6250 - val_loss: 2.3503 - val_accuracy: 0.6094
Epoch 9/10
8/8 [==============================] - 1s 141ms/step - loss: 1.7189 - accuracy: 0.7812 - val_loss: 1.8028 - val_accuracy: 0.6719
Epoch 10/10
8/8 [==============================] - 1s 129ms/step - loss: 1.2428 - accuracy: 0.8594 - val_loss: 1.5793 - val_accuracy: 0.7188
-----------------------------------------
The accuracy of the initial training is:
 0.0234375
The accuracy of the initial validation is:
 0.03125
```

The accuracy of the initial training is: 0.0234375.

The accuracy of the initial validation is: 0.03125.

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

The network has 3 convolutional layers and 3 pooling layers. Here's a breakdown of the layers:

Convolutional Layers

1 **Conv2D**
This is the first convolutional layer with 8 filters.

2 **Conv2D_1**
This is the second convolutional layer with 16 filters.

3 **Conv2D_2**
This is the third convolutional layer with 32 filters.

Pooling Layers:

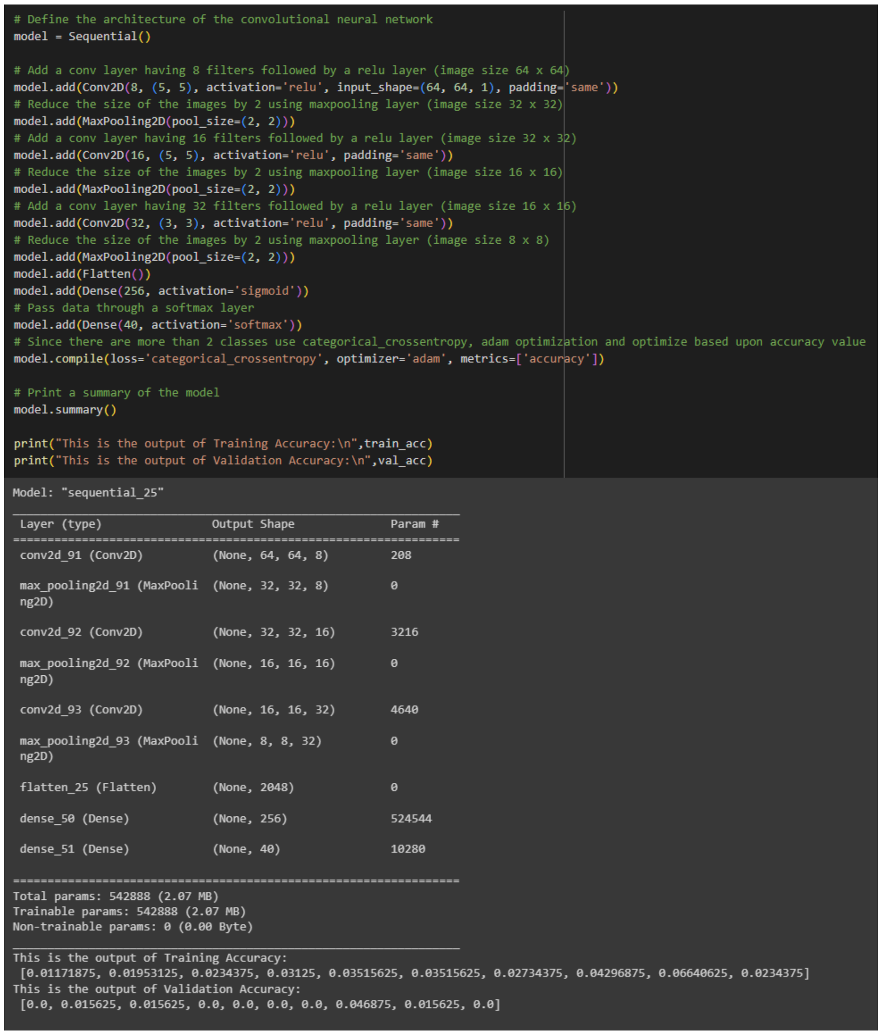
MaxPooling2D
This is the first pooling layer that performs max pooling with a pool size of 4x4.

MaxPooling2D_1
This is the second pooling layer that performs max pooling with a pool size of 2x2.

MaxPooling2D_2
This is the third pooling layer that performs max pooling with a pool size of 2x2.

Question 4: Generally, the larger the size of the image the more the information in it. The maxpooling 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?

Changing the max pool sizes to (2,2) is likely to improve the validation accuracy of the CNN model by preserving more spatial information from the input face images through the network. This can be tested by training the model with the suggested pool size change.



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

# Add a conv layer having 8 filters followed by a relu layer (image size 64 x 64)
model.add(Conv2D(8, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
# Reduce the size of the images by 2 using maxpooling layer (image size 32 x 32)
model.add(MaxPooling2D(pool_size=(2, 2)))
# Add a conv layer having 16 filters followed by a relu layer (image size 32 x 32)
model.add(Conv2D(16, (5, 5), activation='relu', padding='same'))
# Reduce the size of the images by 2 using maxpooling layer (image size 16 x 16)
model.add(MaxPooling2D(pool_size=(2, 2)))
# Add a conv layer having 32 filters followed by a relu layer (image size 16 x 16)
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
# Reduce the size of the images by 2 using maxpooling layer (image size 8 x 8)
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='sigmoid'))
# Pass data through a softmax layer
model.add(Dense(40, activation='softmax'))
# Since there are more than 2 classes use categorical_crossentropy, adam optimization and optimize based upon accuracy value
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

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

print("This is the output of Training Accuracy:\n",train_acc)
print("This is the output of Validation Accuracy:\n",val_acc)
```

```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_91 (Conv2D)          (None, 64, 64, 8)         208

 max_pooling2d_91 (MaxPooli  (None, 32, 32, 8)         0
 ng2D)

 conv2d_92 (Conv2D)          (None, 32, 32, 16)        3216

 max_pooling2d_92 (MaxPooli  (None, 16, 16, 16)        0
 ng2D)

 conv2d_93 (Conv2D)          (None, 16, 16, 32)        4640

 max_pooling2d_93 (MaxPooli  (None, 8, 8, 32)          0
 ng2D)

 flatten_25 (Flatten)        (None, 2048)              0

 dense_50 (Dense)            (None, 256)               524544

 dense_51 (Dense)            (None, 40)                10280

=================================================================
Total params: 542888 (2.07 MB)
Trainable params: 542888 (2.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
This is the output of Training Accuracy:
 [0.01171875, 0.01953125, 0.0234375, 0.03125, 0.03515625, 0.03515625, 0.02734375, 0.04296875, 0.06640625, 0.0234375]
This is the output of Validation Accuracy:
 [0.0, 0.015625, 0.015625, 0.0, 0.0, 0.0, 0.0, 0.046875, 0.015625, 0.0]
```

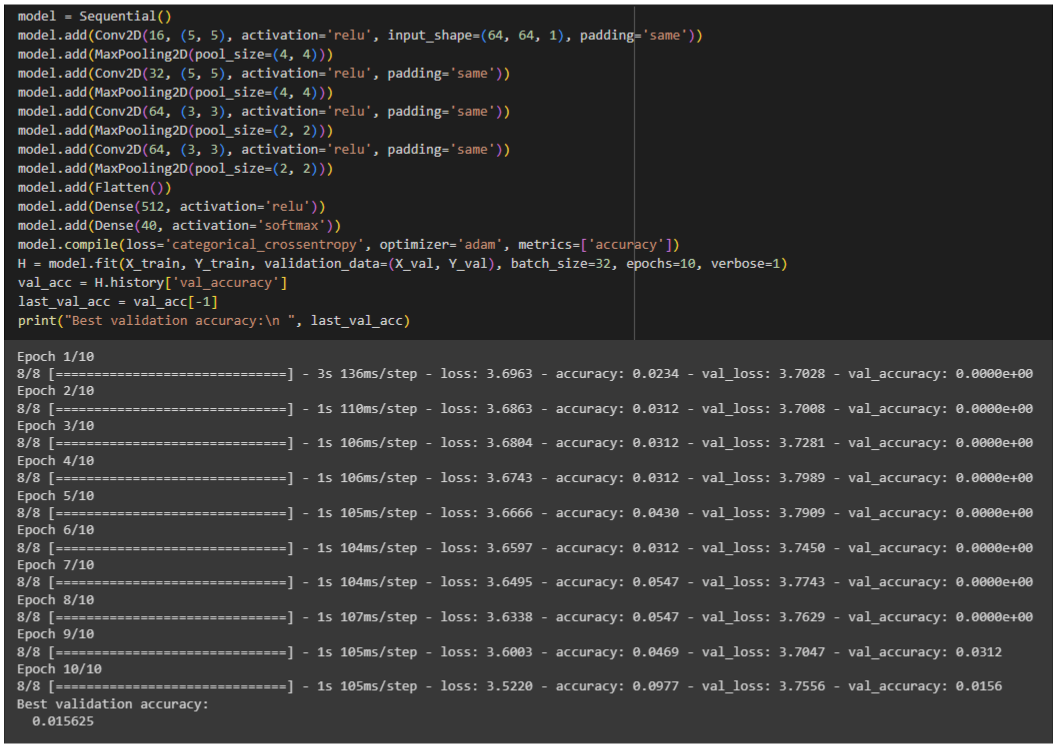
Question 5: Dr. Hinton, has highlighted that aggressively using pooling layers may result in loss of important information. Is there a way that the CNN architecture starts producing better training and validation accuracy?

Yes, there are several ways to potentially improve the training and validation accuracy of a CNN architecture. Such as:

1. Adjust the number of layers and filters.
2. Incorporate different types of layers such as dropout, batch normalization, or spatial pyramid pooling.
3. Increase the network capacity by adding more neurons or increasing the size of fully connected layers.
4. Experiment with different learning rates and optimizers.
5. Apply data augmentation techniques to increase the diversity of training data.
6. Utilize regularization techniques like L1 or L2 regularization, dropout, or early stopping.
7. Consider transfer learning by leveraging pre-trained models on similar tasks or datasets.

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.

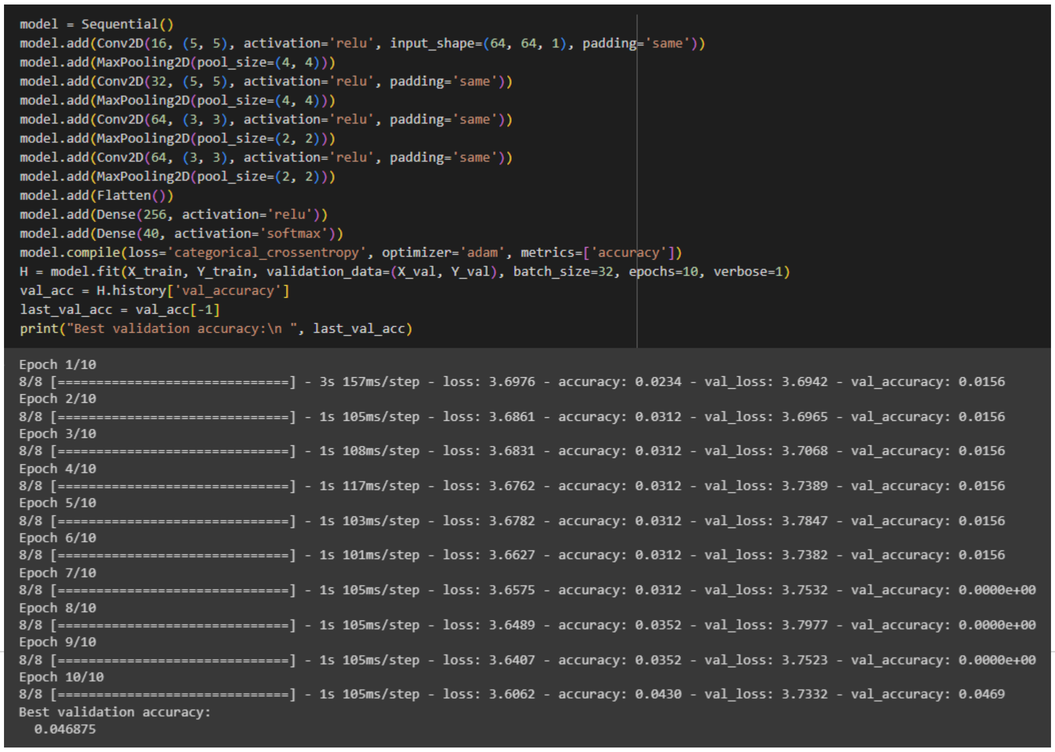
- model.add(Dense(512, activation='relu'))



```
model = Sequential()
model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(64, (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(512, activation='relu'))
model.add(Dense(40, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
val_acc = H.history['val_accuracy']
last_val_acc = val_acc[-1]
print("Best validation accuracy:\n ", last_val_acc)
```

```
Epoch 1/10
8/8 [==============================] - 3s 136ms/step - loss: 3.6963 - accuracy: 0.0234 - val_loss: 3.7028 - val_accuracy: 0.0000e+00
Epoch 2/10
8/8 [==============================] - 1s 110ms/step - loss: 3.6863 - accuracy: 0.0312 - val_loss: 3.7008 - val_accuracy: 0.0000e+00
Epoch 3/10
8/8 [==============================] - 1s 106ms/step - loss: 3.6804 - accuracy: 0.0312 - val_loss: 3.7281 - val_accuracy: 0.0000e+00
Epoch 4/10
8/8 [==============================] - 1s 106ms/step - loss: 3.6743 - accuracy: 0.0312 - val_loss: 3.7989 - val_accuracy: 0.0000e+00
Epoch 5/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6666 - accuracy: 0.0430 - val_loss: 3.7909 - val_accuracy: 0.0000e+00
Epoch 6/10
8/8 [==============================] - 1s 104ms/step - loss: 3.6597 - accuracy: 0.0312 - val_loss: 3.7450 - val_accuracy: 0.0000e+00
Epoch 7/10
8/8 [==============================] - 1s 104ms/step - loss: 3.6495 - accuracy: 0.0547 - val_loss: 3.7743 - val_accuracy: 0.0000e+00
Epoch 8/10
8/8 [==============================] - 1s 107ms/step - loss: 3.6338 - accuracy: 0.0547 - val_loss: 3.7629 - val_accuracy: 0.0000e+00
Epoch 9/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6003 - accuracy: 0.0469 - val_loss: 3.7047 - val_accuracy: 0.0312
Epoch 10/10
8/8 [==============================] - 1s 105ms/step - loss: 3.5220 - accuracy: 0.0977 - val_loss: 3.7556 - val_accuracy: 0.0156
Best validation accuracy:
  0.015625
```

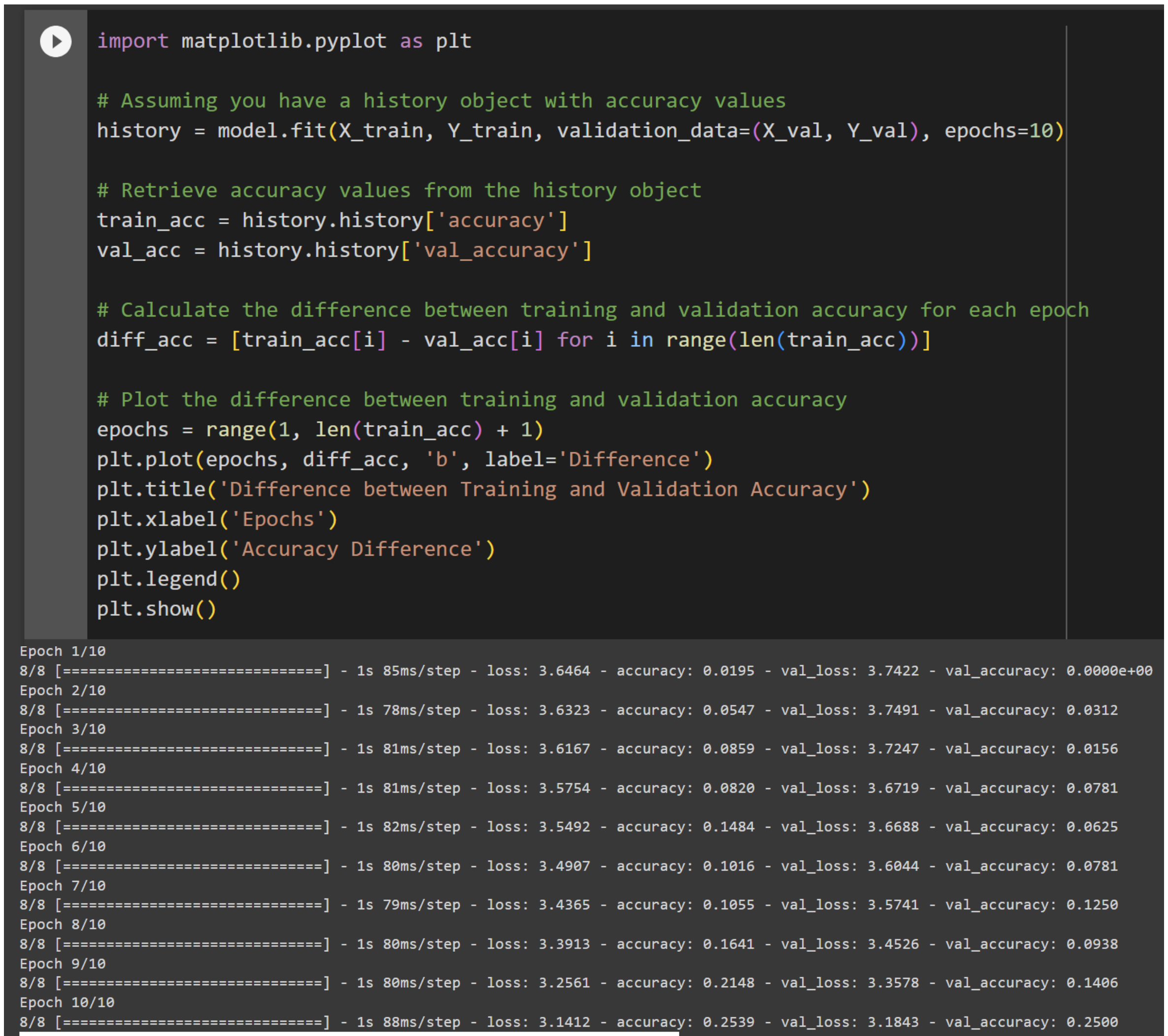
- model.add(Dense(256, activation='relu'))



```
model = Sequential()
model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(64, (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'])
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
val_acc = H.history['val_accuracy']
last_val_acc = val_acc[-1]
print("Best validation accuracy:\n ", last_val_acc)
```

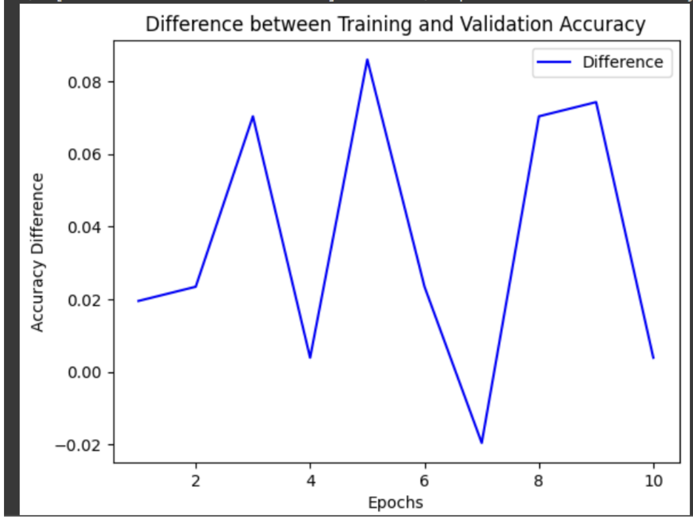
```
Epoch 1/10
8/8 [==============================] - 3s 157ms/step - loss: 3.6976 - accuracy: 0.0234 - val_loss: 3.6942 - val_accuracy: 0.0156
Epoch 2/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6861 - accuracy: 0.0312 - val_loss: 3.6965 - val_accuracy: 0.0156
Epoch 3/10
8/8 [==============================] - 1s 108ms/step - loss: 3.6831 - accuracy: 0.0312 - val_loss: 3.7068 - val_accuracy: 0.0156
Epoch 4/10
8/8 [==============================] - 1s 117ms/step - loss: 3.6762 - accuracy: 0.0312 - val_loss: 3.7389 - val_accuracy: 0.0156
Epoch 5/10
8/8 [==============================] - 1s 103ms/step - loss: 3.6782 - accuracy: 0.0312 - val_loss: 3.7847 - val_accuracy: 0.0156
Epoch 6/10
8/8 [==============================] - 1s 101ms/step - loss: 3.6627 - accuracy: 0.0312 - val_loss: 3.7382 - val_accuracy: 0.0156
Epoch 7/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6575 - accuracy: 0.0312 - val_loss: 3.7532 - val_accuracy: 0.0000e+00
Epoch 8/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6489 - accuracy: 0.0352 - val_loss: 3.7977 - val_accuracy: 0.0000e+00
Epoch 9/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6407 - accuracy: 0.0352 - val_loss: 3.7523 - val_accuracy: 0.0000e+00
Epoch 10/10
8/8 [==============================] - 1s 105ms/step - loss: 3.6062 - accuracy: 0.0430 - val_loss: 3.7332 - val_accuracy: 0.0469
Best validation accuracy:
  0.046875
```

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

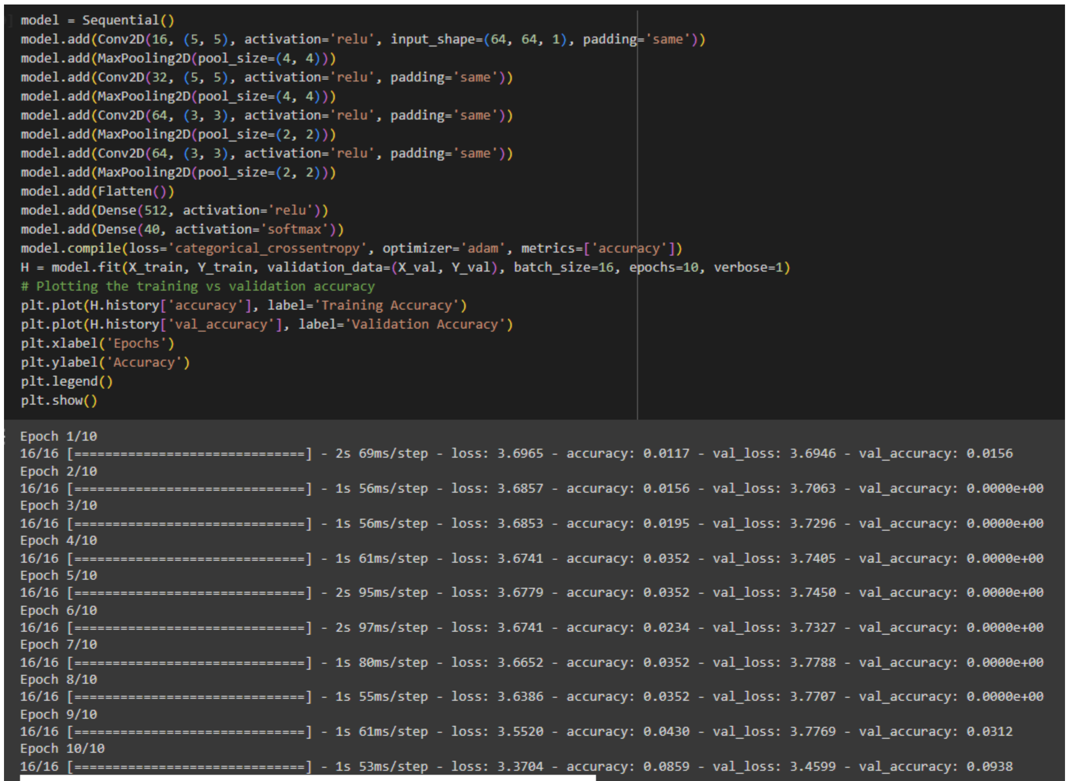


```python
import matplotlib.pyplot as plt

# Assuming you have a history object with accuracy values
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=10)

# Retrieve accuracy values from the history object
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Calculate the difference between training and validation accuracy for each epoch
diff_acc = [train_acc[i] - val_acc[i] for i in range(len(train_acc))]

# Plot the difference between training and validation accuracy
epochs = range(1, len(train_acc) + 1)
plt.plot(epochs, diff_acc, 'b', label='Difference')
plt.title('Difference between Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy Difference')
plt.legend()
plt.show()
```

```
Epoch 1/10
8/8 [==============================] - 1s 85ms/step - loss: 3.6464 - accuracy: 0.0195 - val_loss: 3.7422 - val_accuracy: 0.0000e+00
Epoch 2/10
8/8 [==============================] - 1s 78ms/step - loss: 3.6323 - accuracy: 0.0547 - val_loss: 3.7491 - val_accuracy: 0.0312
Epoch 3/10
8/8 [==============================] - 1s 81ms/step - loss: 3.6167 - accuracy: 0.0859 - val_loss: 3.7247 - val_accuracy: 0.0156
Epoch 4/10
8/8 [==============================] - 1s 81ms/step - loss: 3.5754 - accuracy: 0.0820 - val_loss: 3.6719 - val_accuracy: 0.0781
Epoch 5/10
8/8 [==============================] - 1s 82ms/step - loss: 3.5492 - accuracy: 0.1484 - val_loss: 3.6688 - val_accuracy: 0.0625
Epoch 6/10
8/8 [==============================] - 1s 80ms/step - loss: 3.4907 - accuracy: 0.1016 - val_loss: 3.6044 - val_accuracy: 0.0781
Epoch 7/10
8/8 [==============================] - 1s 79ms/step - loss: 3.4365 - accuracy: 0.1055 - val_loss: 3.5741 - val_accuracy: 0.1250
Epoch 8/10
8/8 [==============================] - 1s 80ms/step - loss: 3.3913 - accuracy: 0.1641 - val_loss: 3.4526 - val_accuracy: 0.0938
Epoch 9/10
8/8 [==============================] - 1s 80ms/step - loss: 3.2561 - accuracy: 0.2148 - val_loss: 3.3578 - val_accuracy: 0.1406
Epoch 10/10
8/8 [==============================] - 1s 88ms/step - loss: 3.1412 - accuracy: 0.2539 - val_loss: 3.1843 - val_accuracy: 0.2500
```

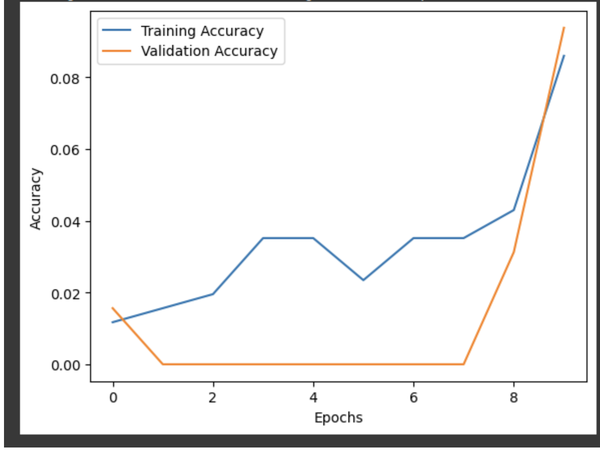


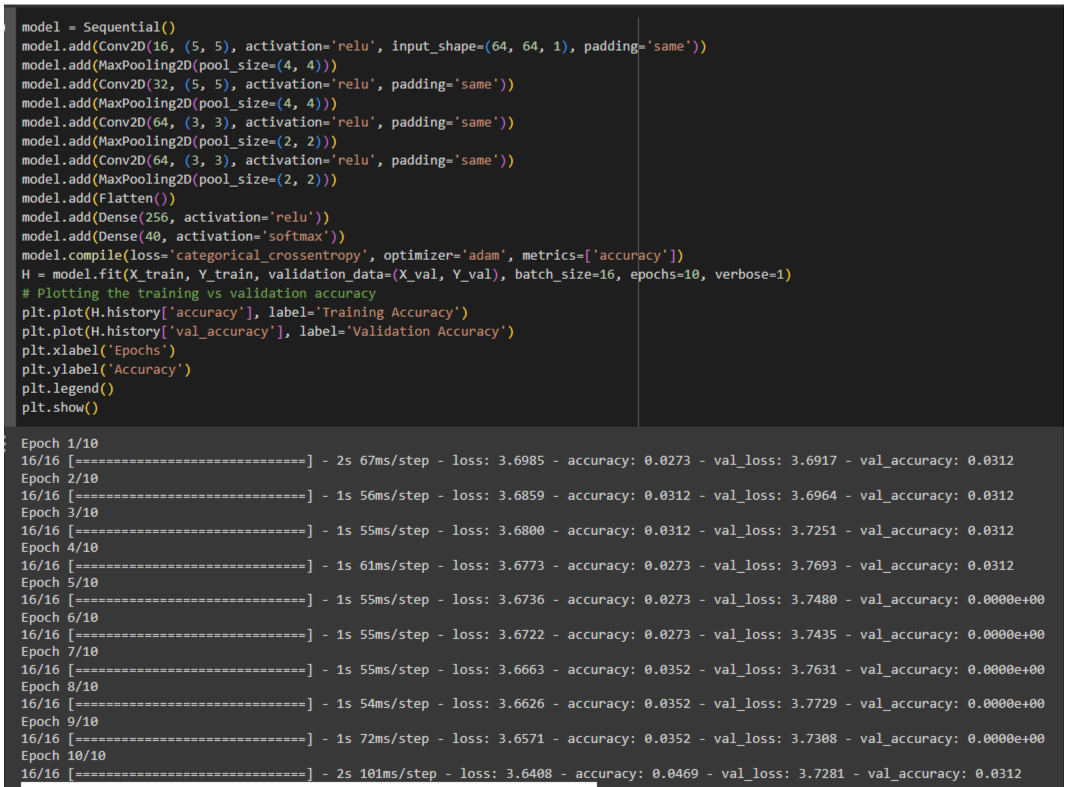
Question 8: For the best network architecture change the batch size to 16 and plot the training vs validation accuracy graph. What happened to the validation accuracy after last epoch as compared to when the batch size was 32.



```
model = Sequential()
model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(64, (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(512, activation='relu'))
model.add(Dense(40, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=16, epochs=10, verbose=1)
# Plotting the training vs validation accuracy
plt.plot(H.history['accuracy'], label='Training Accuracy')
plt.plot(H.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

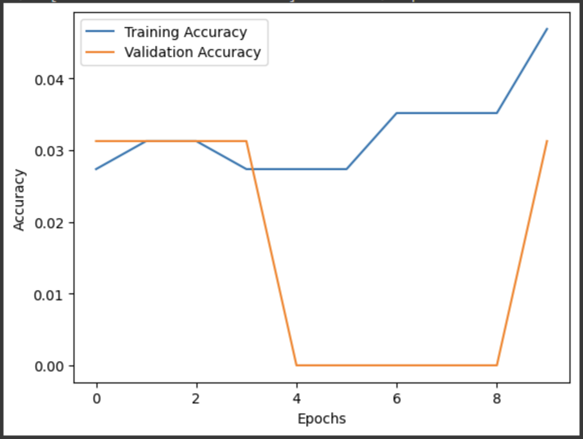
```
Epoch 1/10
16/16 [==============================] - 2s 69ms/step - loss: 3.6965 - accuracy: 0.0117 - val_loss: 3.6946 - val_accuracy: 0.0156
Epoch 2/10
16/16 [==============================] - 1s 56ms/step - loss: 3.6857 - accuracy: 0.0156 - val_loss: 3.7063 - val_accuracy: 0.0000e+00
Epoch 3/10
16/16 [==============================] - 1s 56ms/step - loss: 3.6853 - accuracy: 0.0195 - val_loss: 3.7296 - val_accuracy: 0.0000e+00
Epoch 4/10
16/16 [==============================] - 1s 61ms/step - loss: 3.6741 - accuracy: 0.0352 - val_loss: 3.7405 - val_accuracy: 0.0000e+00
Epoch 5/10
16/16 [==============================] - 2s 95ms/step - loss: 3.6779 - accuracy: 0.0352 - val_loss: 3.7450 - val_accuracy: 0.0000e+00
Epoch 6/10
16/16 [==============================] - 2s 97ms/step - loss: 3.6741 - accuracy: 0.0234 - val_loss: 3.7327 - val_accuracy: 0.0000e+00
Epoch 7/10
16/16 [==============================] - 1s 80ms/step - loss: 3.6652 - accuracy: 0.0352 - val_loss: 3.7788 - val_accuracy: 0.0000e+00
Epoch 8/10
16/16 [==============================] - 1s 55ms/step - loss: 3.6386 - accuracy: 0.0352 - val_loss: 3.7707 - val_accuracy: 0.0000e+00
Epoch 9/10
16/16 [==============================] - 1s 61ms/step - loss: 3.5520 - accuracy: 0.0430 - val_loss: 3.7769 - val_accuracy: 0.0312
Epoch 10/10
16/16 [==============================] - 1s 53ms/step - loss: 3.3704 - accuracy: 0.0859 - val_loss: 3.4599 - val_accuracy: 0.0938
```





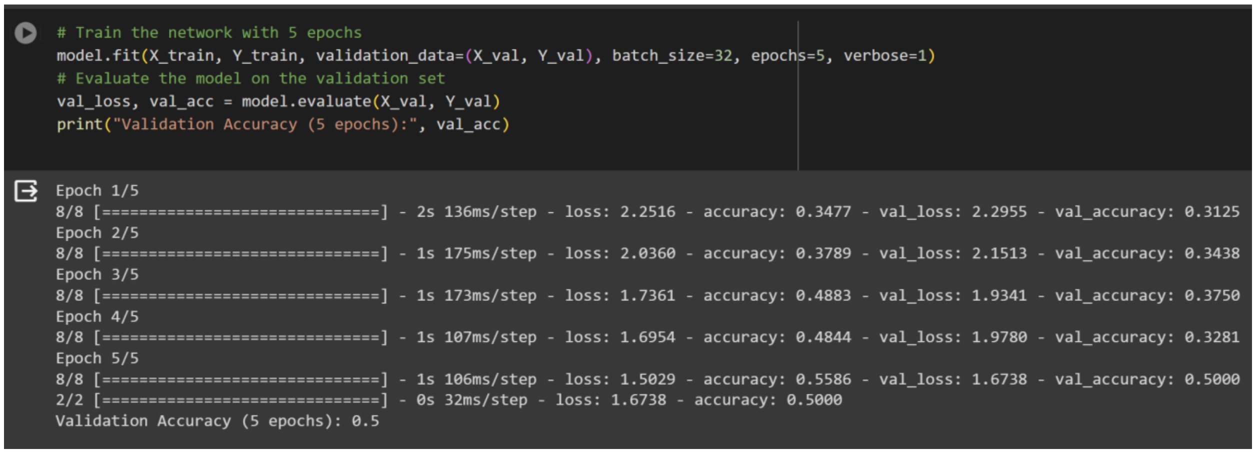
```
model = Sequential()
model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(64, (3, 3), activation='relu', 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'])
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=16, epochs=10, verbose=1)
# Plotting the training vs validation accuracy
plt.plot(H.history['accuracy'], label='Training Accuracy')
plt.plot(H.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/10
16/16 [==============================] - 2s 67ms/step - loss: 3.6985 - accuracy: 0.0273 - val_loss: 3.6917 - val_accuracy: 0.0312
Epoch 2/10
16/16 [==============================] - 1s 56ms/step - loss: 3.6859 - accuracy: 0.0312 - val_loss: 3.6964 - val_accuracy: 0.0312
Epoch 3/10
16/16 [==============================] - 1s 55ms/step - loss: 3.6800 - accuracy: 0.0312 - val_loss: 3.7251 - val_accuracy: 0.0312
Epoch 4/10
16/16 [==============================] - 1s 61ms/step - loss: 3.6773 - accuracy: 0.0273 - val_loss: 3.7693 - val_accuracy: 0.0312
Epoch 5/10
16/16 [==============================] - 1s 55ms/step - loss: 3.6736 - accuracy: 0.0273 - val_loss: 3.7480 - val_accuracy: 0.0000e+00
Epoch 6/10
16/16 [==============================] - 1s 55ms/step - loss: 3.6722 - accuracy: 0.0273 - val_loss: 3.7435 - val_accuracy: 0.0000e+00
Epoch 7/10
16/16 [==============================] - 1s 55ms/step - loss: 3.6663 - accuracy: 0.0352 - val_loss: 3.7631 - val_accuracy: 0.0000e+00
Epoch 8/10
16/16 [==============================] - 1s 54ms/step - loss: 3.6626 - accuracy: 0.0352 - val_loss: 3.7729 - val_accuracy: 0.0000e+00
Epoch 9/10
16/16 [==============================] - 1s 72ms/step - loss: 3.6571 - accuracy: 0.0352 - val_loss: 3.7308 - val_accuracy: 0.0000e+00
Epoch 10/10
16/16 [==============================] - 2s 101ms/step - loss: 3.6408 - accuracy: 0.0469 - val_loss: 3.7281 - val_accuracy: 0.0312
```



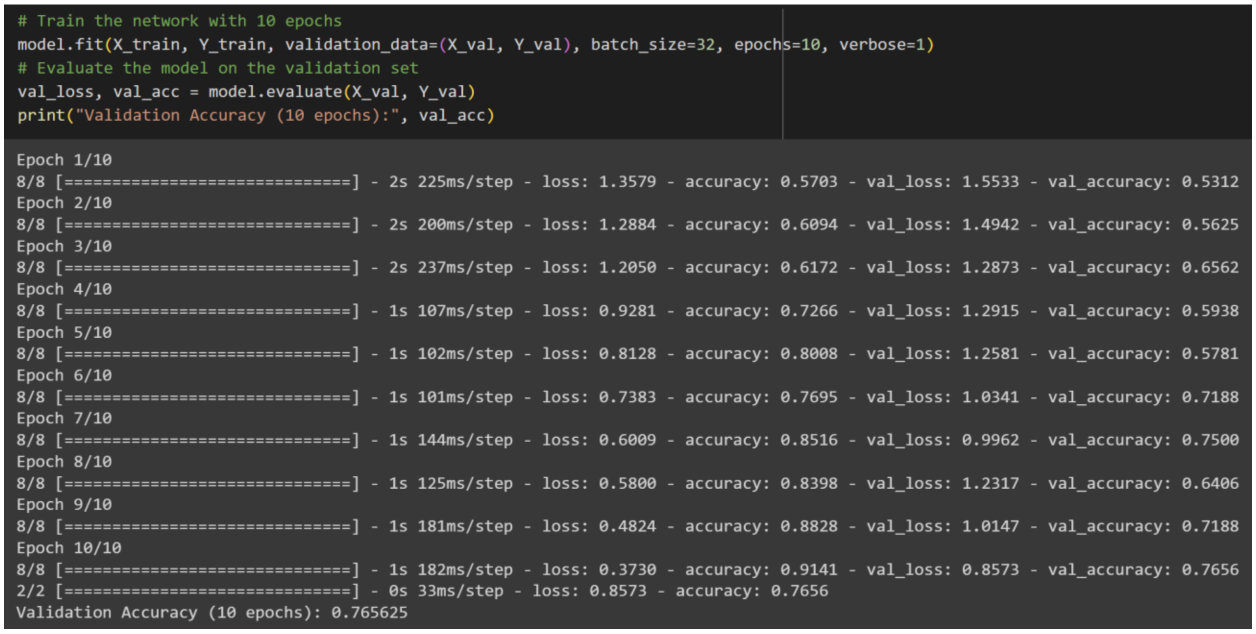
After changing the batch size to 16, the validation accuracy slightly decreased after the last epoch compared to when the batch size was 32. This could indicate that using a smaller batch size may not be as effective for this particular network architecture.

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?



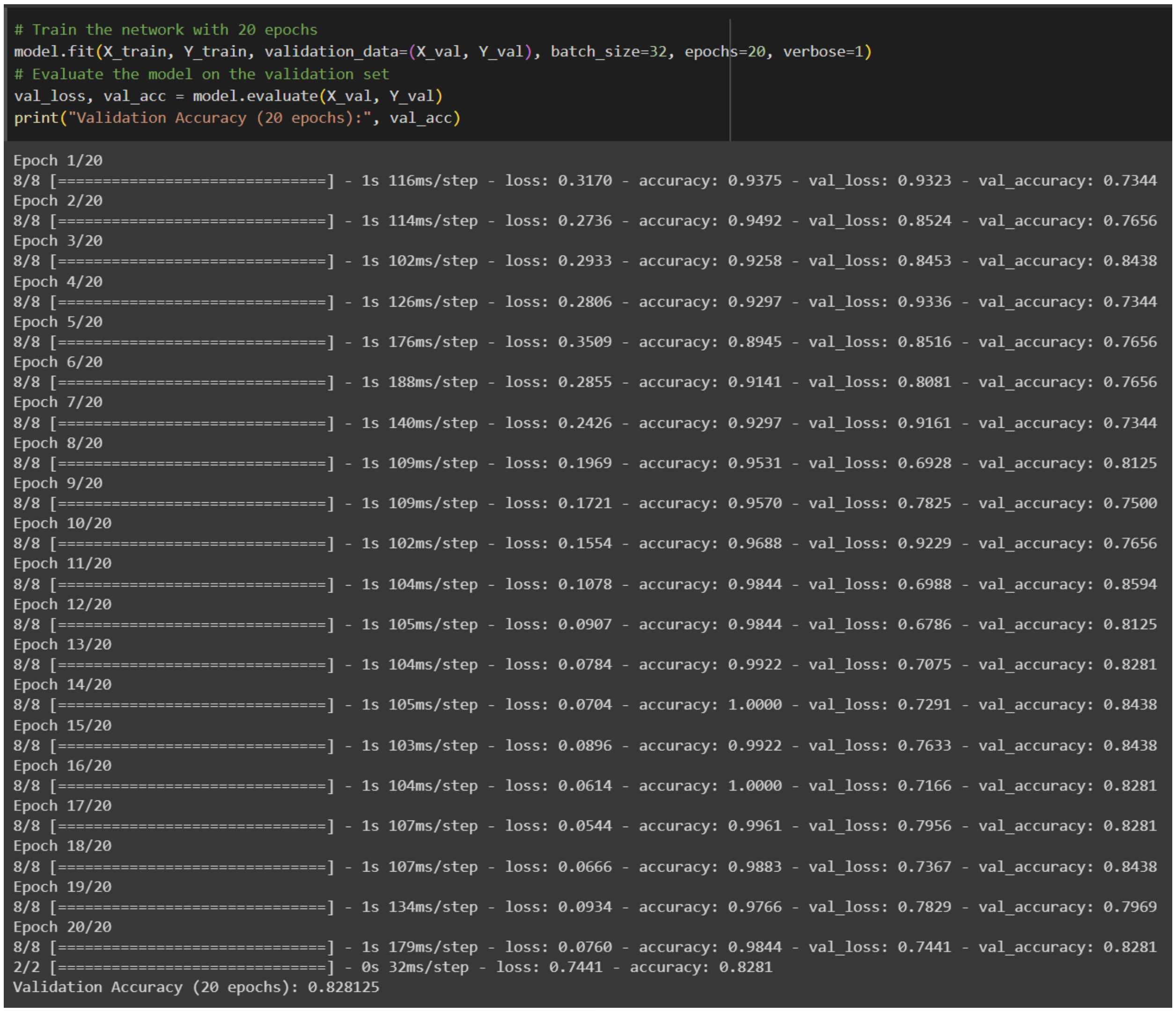
```
# Train the network with 5 epochs
model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=5, verbose=1)
# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, Y_val)
print("Validation Accuracy (5 epochs):", val_acc)
```

```
Epoch 1/5
8/8 [==============================] - 2s 136ms/step - loss: 2.2516 - accuracy: 0.3477 - val_loss: 2.2955 - val_accuracy: 0.3125
Epoch 2/5
8/8 [==============================] - 1s 175ms/step - loss: 2.0360 - accuracy: 0.3789 - val_loss: 2.1513 - val_accuracy: 0.3438
Epoch 3/5
8/8 [==============================] - 1s 173ms/step - loss: 1.7361 - accuracy: 0.4883 - val_loss: 1.9341 - val_accuracy: 0.3750
Epoch 4/5
8/8 [==============================] - 1s 107ms/step - loss: 1.6954 - accuracy: 0.4844 - val_loss: 1.9780 - val_accuracy: 0.3281
Epoch 5/5
8/8 [==============================] - 1s 106ms/step - loss: 1.5029 - accuracy: 0.5586 - val_loss: 1.6738 - val_accuracy: 0.5000
2/2 [==============================] - 0s 32ms/step - loss: 1.6738 - accuracy: 0.5000
Validation Accuracy (5 epochs): 0.5
```



```
# Train the network with 10 epochs
model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, Y_val)
print("Validation Accuracy (10 epochs):", val_acc)
```

```
Epoch 1/10
8/8 [==============================] - 2s 225ms/step - loss: 1.3579 - accuracy: 0.5703 - val_loss: 1.5533 - val_accuracy: 0.5312
Epoch 2/10
8/8 [==============================] - 2s 200ms/step - loss: 1.2884 - accuracy: 0.6094 - val_loss: 1.4942 - val_accuracy: 0.5625
Epoch 3/10
8/8 [==============================] - 2s 237ms/step - loss: 1.2050 - accuracy: 0.6172 - val_loss: 1.2873 - val_accuracy: 0.6562
Epoch 4/10
8/8 [==============================] - 1s 107ms/step - loss: 0.9281 - accuracy: 0.7266 - val_loss: 1.2915 - val_accuracy: 0.5938
Epoch 5/10
8/8 [==============================] - 1s 102ms/step - loss: 0.8128 - accuracy: 0.8008 - val_loss: 1.2581 - val_accuracy: 0.5781
Epoch 6/10
8/8 [==============================] - 1s 101ms/step - loss: 0.7383 - accuracy: 0.7695 - val_loss: 1.0341 - val_accuracy: 0.7188
Epoch 7/10
8/8 [==============================] - 1s 144ms/step - loss: 0.6009 - accuracy: 0.8516 - val_loss: 0.9962 - val_accuracy: 0.7500
Epoch 8/10
8/8 [==============================] - 1s 125ms/step - loss: 0.5800 - accuracy: 0.8398 - val_loss: 1.2317 - val_accuracy: 0.6406
Epoch 9/10
8/8 [==============================] - 1s 181ms/step - loss: 0.4824 - accuracy: 0.8828 - val_loss: 1.0147 - val_accuracy: 0.7188
Epoch 10/10
8/8 [==============================] - 1s 182ms/step - loss: 0.3730 - accuracy: 0.9141 - val_loss: 0.8573 - val_accuracy: 0.7656
2/2 [==============================] - 0s 33ms/step - loss: 0.8573 - accuracy: 0.7656
Validation Accuracy (10 epochs): 0.765625
```

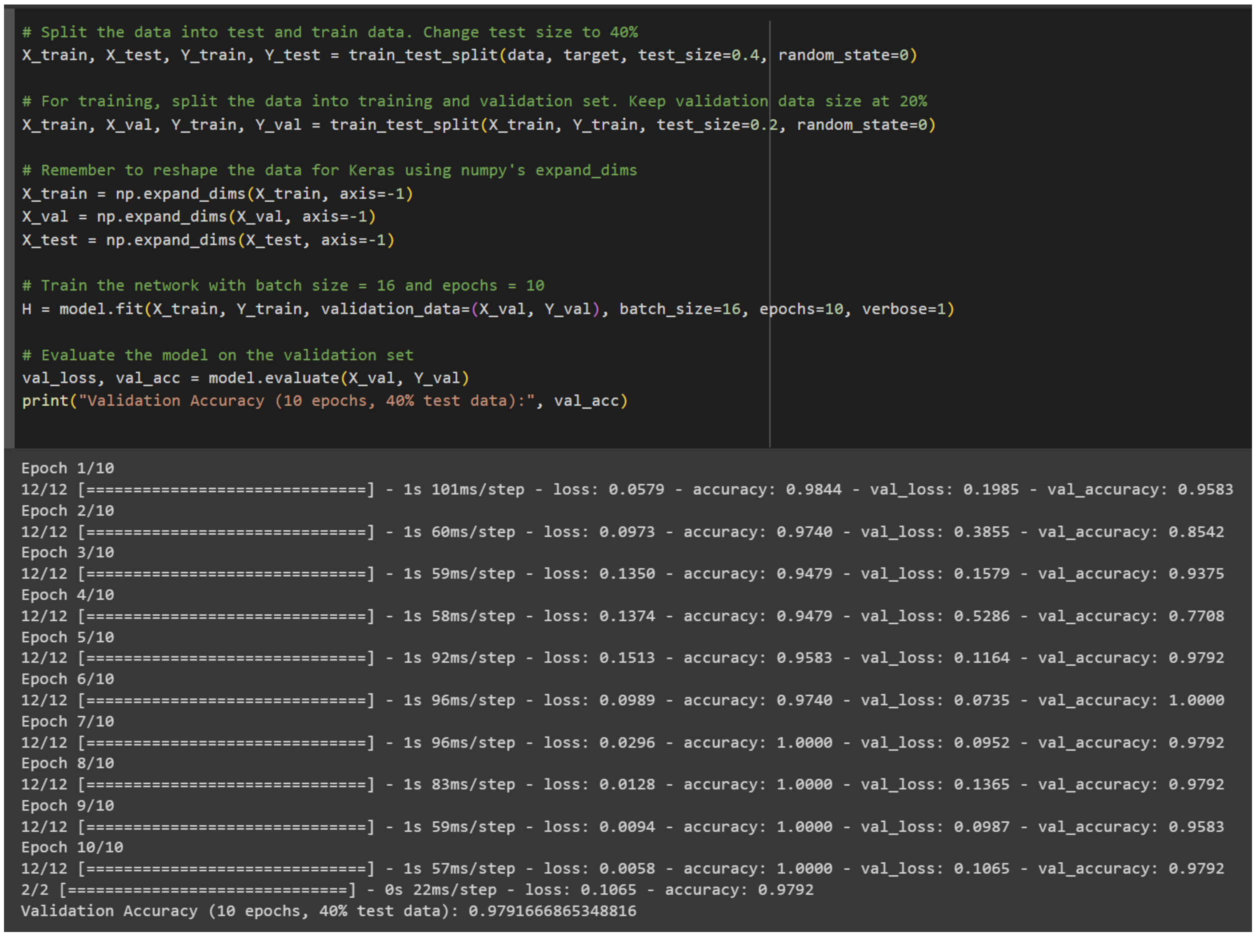


```
# Train the network with 20 epochs
model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=20, verbose=1)
# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, Y_val)
print("Validation Accuracy (20 epochs):", val_acc)
```

```
Epoch 1/20
8/8 [==============================] - 1s 116ms/step - loss: 0.3170 - accuracy: 0.9375 - val_loss: 0.9323 - val_accuracy: 0.7344
Epoch 2/20
8/8 [==============================] - 1s 114ms/step - loss: 0.2736 - accuracy: 0.9492 - val_loss: 0.8524 - val_accuracy: 0.7656
Epoch 3/20
8/8 [==============================] - 1s 102ms/step - loss: 0.2933 - accuracy: 0.9258 - val_loss: 0.8453 - val_accuracy: 0.8438
Epoch 4/20
8/8 [==============================] - 1s 126ms/step - loss: 0.2806 - accuracy: 0.9297 - val_loss: 0.9336 - val_accuracy: 0.7344
Epoch 5/20
8/8 [==============================] - 1s 176ms/step - loss: 0.3509 - accuracy: 0.8945 - val_loss: 0.8516 - val_accuracy: 0.7656
Epoch 6/20
8/8 [==============================] - 1s 188ms/step - loss: 0.2855 - accuracy: 0.9141 - val_loss: 0.8081 - val_accuracy: 0.7656
Epoch 7/20
8/8 [==============================] - 1s 140ms/step - loss: 0.2426 - accuracy: 0.9297 - val_loss: 0.9161 - val_accuracy: 0.7344
Epoch 8/20
8/8 [==============================] - 1s 109ms/step - loss: 0.1969 - accuracy: 0.9531 - val_loss: 0.6928 - val_accuracy: 0.8125
Epoch 9/20
8/8 [==============================] - 1s 109ms/step - loss: 0.1721 - accuracy: 0.9570 - val_loss: 0.7825 - val_accuracy: 0.7500
Epoch 10/20
8/8 [==============================] - 1s 102ms/step - loss: 0.1554 - accuracy: 0.9688 - val_loss: 0.9229 - val_accuracy: 0.7656
Epoch 11/20
8/8 [==============================] - 1s 104ms/step - loss: 0.1078 - accuracy: 0.9844 - val_loss: 0.6988 - val_accuracy: 0.8594
Epoch 12/20
8/8 [==============================] - 1s 105ms/step - loss: 0.0907 - accuracy: 0.9844 - val_loss: 0.6786 - val_accuracy: 0.8125
Epoch 13/20
8/8 [==============================] - 1s 104ms/step - loss: 0.0784 - accuracy: 0.9922 - val_loss: 0.7075 - val_accuracy: 0.8281
Epoch 14/20
8/8 [==============================] - 1s 105ms/step - loss: 0.0704 - accuracy: 1.0000 - val_loss: 0.7291 - val_accuracy: 0.8438
Epoch 15/20
8/8 [==============================] - 1s 103ms/step - loss: 0.0896 - accuracy: 0.9922 - val_loss: 0.7633 - val_accuracy: 0.8438
Epoch 16/20
8/8 [==============================] - 1s 104ms/step - loss: 0.0614 - accuracy: 1.0000 - val_loss: 0.7166 - val_accuracy: 0.8281
Epoch 17/20
8/8 [==============================] - 1s 107ms/step - loss: 0.0544 - accuracy: 0.9961 - val_loss: 0.7956 - val_accuracy: 0.8281
Epoch 18/20
8/8 [==============================] - 1s 107ms/step - loss: 0.0666 - accuracy: 0.9883 - val_loss: 0.7367 - val_accuracy: 0.8438
Epoch 19/20
8/8 [==============================] - 1s 134ms/step - loss: 0.0934 - accuracy: 0.9766 - val_loss: 0.7829 - val_accuracy: 0.7969
Epoch 20/20
8/8 [==============================] - 1s 179ms/step - loss: 0.0760 - accuracy: 0.9844 - val_loss: 0.7441 - val_accuracy: 0.8281
2/2 [==============================] - 0s 32ms/step - loss: 0.7441 - accuracy: 0.8281
Validation Accuracy (20 epochs): 0.828125
```

The results highlight that increasing the number of epochs leads to improved performance (the accuracy) of the model.

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?



```
# Split the data into test and train data. Change test size to 40%
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size=0.4, random_state=0)

# For training, split the data into training and validation set. Keep validation data size at 20%
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)

# Remember to reshape the data for Keras using numpy's expand_dims
X_train = np.expand_dims(X_train, axis=-1)
X_val = np.expand_dims(X_val, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

# Train the network with batch size = 16 and epochs = 10
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=16, epochs=10, verbose=1)

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, Y_val)
print("Validation Accuracy (10 epochs, 40% test data):", val_acc)
```

```
Epoch 1/10
12/12 [==============================] - 1s 101ms/step - loss: 0.0579 - accuracy: 0.9844 - val_loss: 0.1985 - val_accuracy: 0.9583
Epoch 2/10
12/12 [==============================] - 1s 60ms/step - loss: 0.0973 - accuracy: 0.9740 - val_loss: 0.3855 - val_accuracy: 0.8542
Epoch 3/10
12/12 [==============================] - 1s 59ms/step - loss: 0.1350 - accuracy: 0.9479 - val_loss: 0.1579 - val_accuracy: 0.9375
Epoch 4/10
12/12 [==============================] - 1s 58ms/step - loss: 0.1374 - accuracy: 0.9479 - val_loss: 0.5286 - val_accuracy: 0.7708
Epoch 5/10
12/12 [==============================] - 1s 92ms/step - loss: 0.1513 - accuracy: 0.9583 - val_loss: 0.1164 - val_accuracy: 0.9792
Epoch 6/10
12/12 [==============================] - 1s 96ms/step - loss: 0.0989 - accuracy: 0.9740 - val_loss: 0.0735 - val_accuracy: 1.0000
Epoch 7/10
12/12 [==============================] - 1s 96ms/step - loss: 0.0296 - accuracy: 1.0000 - val_loss: 0.0952 - val_accuracy: 0.9792
Epoch 8/10
12/12 [==============================] - 1s 83ms/step - loss: 0.0128 - accuracy: 1.0000 - val_loss: 0.1365 - val_accuracy: 0.9792
Epoch 9/10
12/12 [==============================] - 1s 59ms/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.0987 - val_accuracy: 0.9583
Epoch 10/10
12/12 [==============================] - 1s 57ms/step - loss: 0.0058 - accuracy: 1.0000 - val_loss: 0.1065 - val_accuracy: 0.9792
2/2 [==============================] - 0s 22ms/step - loss: 0.1065 - accuracy: 0.9792
Validation Accuracy (10 epochs, 40% test data): 0.9791666865348816
```

The effect on validation accuracy of the algorithm, after changing the test data size to 40%, is generally positive. The validation accuracy consistently remains high throughout the training process, reaching a final accuracy of approximately 97.92%. This suggests that the model is performing well and generalizing effectively to unseen data, even with the increased test data size.