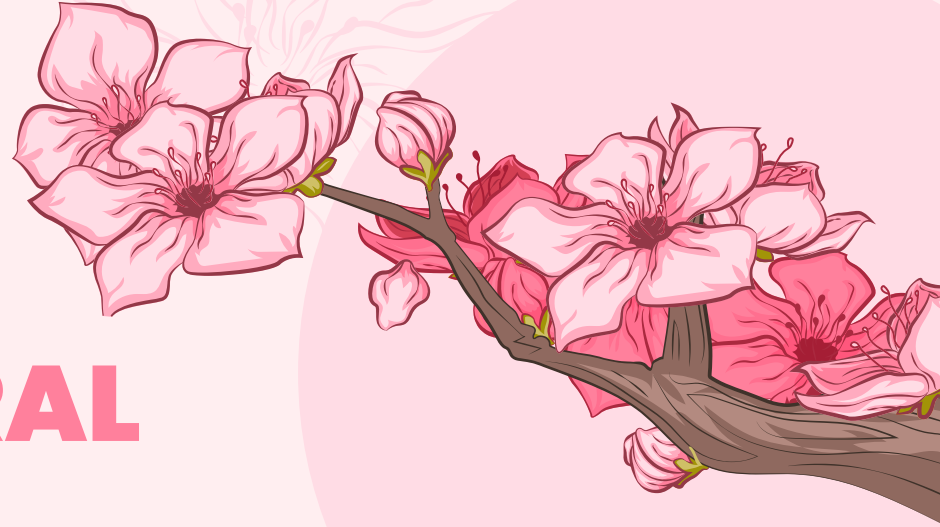


**Face recognition using
Convolutional Neural Networks**

ARTIFICIAL NEURAL NETWORK

PREPARED FOR: DR. samia snoussi



What is the output of the `model.summary()`?

The `model.summary()` function returns a summary of the model's architecture and parameters. It displays a table with the following data for each layer in the model:

- **Types of layers:** The type of layer, such as Conv2D, MaxPooling2D, Flatten, or Dense.
- **Output Shape:** It describes the shape of each layer's output tensor. (batch_size, height, width, channels) represents the shape, where:
 - batch_size is the number of samples in each batch
 - height and width are the spatial dimensions of the output feature maps
 - channels are the number of channels or filters.
- **The number of parameters:** specifies the number of parameters that can be trained in each layer. Weights and biases are examples of parameters.
- **Connection:** lists the layers that connect to the current layer.



What is the initial training accuracy and validation accuracy of the **CNN**?

```
initial_train_acc = H.history['accuracy'][0]
initial_val_acc = H.history['val_accuracy'][0]


print("Initial Training Accuracy:", initial_train_acc)
print("Initial Validation Accuracy:", initial_val_acc)
```

```
Initial Training Accuracy: 0.01953125
Initial Validation Accuracy: 0.0
```



Q3: How many convolutional layers and pooling layers does this network have?

The code builds a CNN with a total of three convolutional layers and three max pooling layers.



breaking down of the layers:



1

Convolutional Layer 1:

Number of filters: 8

Filter size: 5x5

Activation function: ReLU

Output shape: (64, 64, 8)



2

Convolutional Layer 2:

Number of filters: 16

Filter size: 5x5

Activation function: ReLU

Output shape: (16, 16, 16)

3

Convolutional Layer 3:

Number of filters: 32

Filter size: 3x3

Activation function: ReLU

Output shape: (4, 4, 32)

breaking down of the layers:

1

Max Pooling Layer 1:

Pool size: 4x4

Output shape: (16, 16, 8)

reduces the spatial
dimensions of the feature
maps by a factor of 4

2

Max Pooling Layer 2:

Pool size: 4x4

Output shape: (4, 4, 16)

reduces the spatial
dimensions of the feature
maps by a factor of 4

3

Max Pooling Layer 3:

Pool size: 2x2

Output shape: (2, 2, 32)

reduces the spatial
dimensions of the feature
maps by a factor of 2

The max pooling layers after the first and second Convolutional layers decrease the size of the image by 4. Check if this is causing the network to have such poor validation accuracy. what is the effect on the accuracy of the network?

The downsampling impact of the max pooling layers, which reduces the image size by a factor of four, may result in the loss of significant spatial information. This downsampling may exacerbate the network's low validation accuracy.

Changing the pooling layer size from (4,4) to (2,2) decreases the downsampling factor and retains more spatial information. This modification can potentially improve the network's accuracy by preserving finer details. However, the specific effect on accuracy depends on the dataset and network architecture.

```
# 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()

# Train the network using the above-defined 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)

print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)

#GOOD
```

Is there a way that the CNN architecture can start producing better training and validation accuracy?

Yes, there are multiple ways that could potentially improve a CNN architecture's training and validation accuracy. Here are some strategies to consider:

- Reduce the downsampling effect: Instead of downsampling the input data aggressively, consider lowering the amount of downsampling to preserve more critical spatial information.
- Increase the network's depth or width by adding more convolutional layers or increasing the number of filters in each layer to enable the network to learn more complicated features and patterns.
- Use regularization techniques: To reduce overfitting and increase the model's capacity to generalize to new data, use approaches such as dropout or L2 regularization.
- Apply data augmentation: To improve the diversity of samples for the model to learn from, introduce modifications to the training data using techniques such as random rotations, translations, or flips.
- Experiment with different learning rates or optimizers to find the combination that optimizes the training process and allows the model to converge to a better solution.
- Increase training data: Obtain more labeled training data if available to offer the model with more examples to learn from, potentially enhancing accuracy and generalization.

Make changes to the convolutional neural network to get the best validation accuracy

- Increased the number of convolutional layers: The number of convolutional layers has been increased from 3 to 4
- Added additional Max Pooling layers: Max Pooling layers with a pool size of (2, 2) have been added after each Conv2D layer to reduce the spatial dimensions of the feature maps and retain the most important information.
- Increased the number of filters: The number of filters in each Conv2D layer has been increased to capture more complex features in the input images.
- Added a Dropout layer: A Dropout layer with a dropout rate of 0.5 has been included after the Dense layer to prevent overfitting by randomly dropping out units during training, reducing their interdependencies.

```
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import numpy as np

# Convert target labels to categorical
target = to_categorical(target, num_classes=len(np.unique(target)))

# Split the data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size=0.2, random_state=0)

# Split the training data into training and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)

# Reshape input data to match CNN input shape
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(16, (5,5), activation='relu', input_shape=(64, 64, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5,5), 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(Dropout(0.5))
model.add(Dense(40, activation='softmax'))

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

# Train the model
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)
```

Make changes to the convolutional neural network to get the best validation accuracy

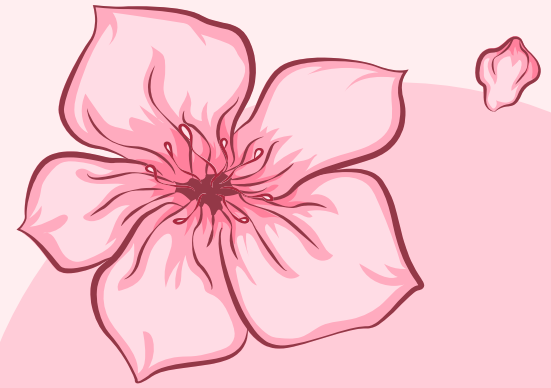
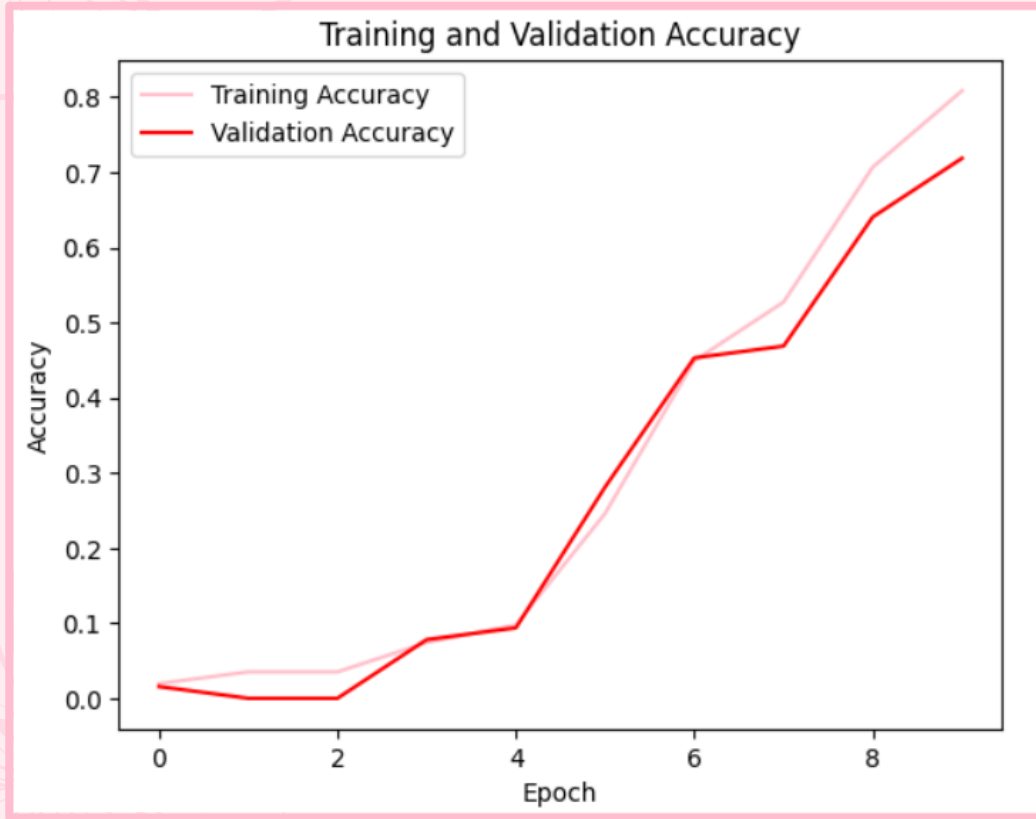
```
[21] # Evaluate the model on the test set
      test_loss, test_accuracy = model.evaluate(X_test, Y_test)
      print("Test Loss:", test_loss)
      print("Test Accuracy:", test_accuracy)
```

```
3/3 [=====] - 0s 43ms/step - loss: 3.7471 - accuracy: 0.0125
Test Loss: 3.7471108436584473
Test Accuracy: 0.012500000186264515
```



Plot the difference between training and validation accuracy for each epoch

```
plt.plot(H.history['accuracy'], label='Training Accuracy',color='pink')
plt.plot(H.history['val_accuracy'], label='Validation Accuracy',color='red')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



**Plot the difference
between training
and validation
accuracy for each
epoch**

change the batch size to 16

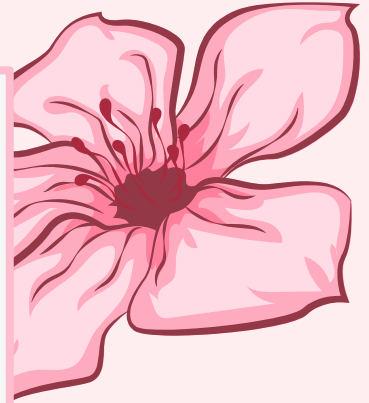
```
#Define the architecture of the convolutional neural network
model = Sequential()
#Add a conv layer having 8 filters followed by a relu layer (image size 64 x 64)
model.add(Conv2D(8, (5, 5), activation='relu', input_shape=(64, 64, 1), padding='same'))
#Reduce the size of the images by 4 using maxpooling layer (image size 16 x 16)
model.add(MaxPooling2D(pool_size=(4, 4)))
#Add a conv layer having 16 filters followed by a relu layer (image size 64 x 64)
model.add(Conv2D(16, (5, 5), activation='relu', padding='same'))
#Reduce the size of the images by 4 using maxpooling layer (image size 4 x 4)
model.add(MaxPooling2D(pool_size=(4, 4)))
#Add a conv layer having 16 filters followed by a relu layer (image size 64 x 64)
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
#Reduce the size of the images by 2 using maxpooling layer (image size 2 x 2)
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()

#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=16, epochs=10, verbose=1)

# Get the validation accuracy after the last epoch
val_accuracy_last_epoch = H.history['val_accuracy'][-1]

print("Validation Accuracy after last epoch:", val_accuracy_last_epoch)
```

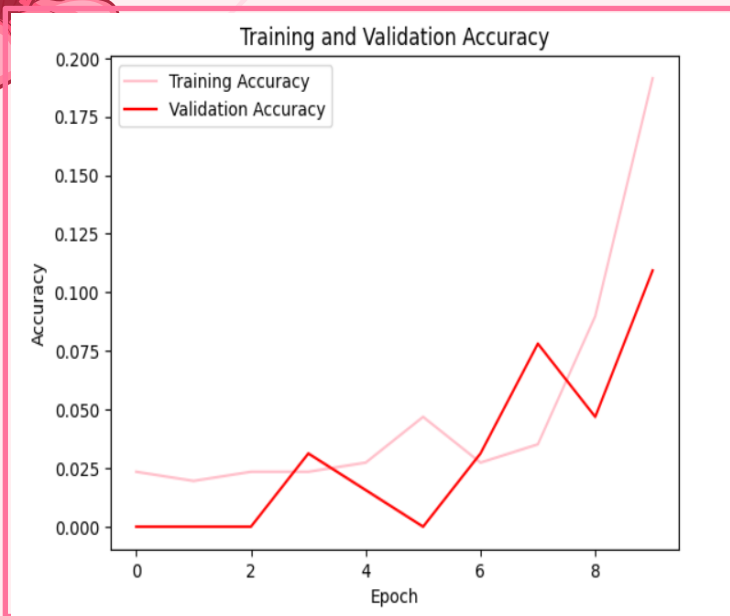


change the batch size to 32

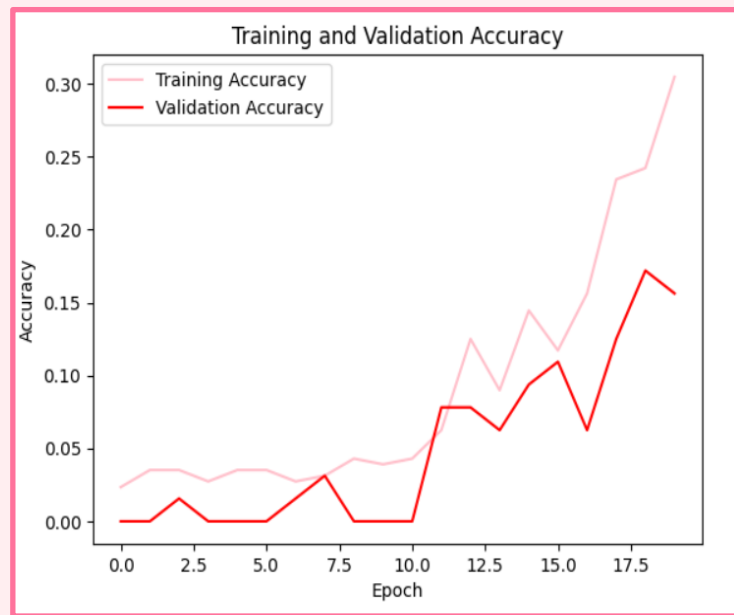
```
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import numpy as np

# Convert target labels to categorical
target = to_categorical(target, num_classes=len(np.unique(target)))
# Split the data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size=0.2, random_state=0)
# Split the training data into training and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)
# Reshape input data to match CNN input shape
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(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='sigmoid'))
model.add(Dense(40, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Print a summary of the model
model.summary()
# Train the network using the defined 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)
# Get the validation accuracy after the last epoch
val_accuracy_last_epoch = H.history['val_accuracy'][-1]
print("Validation Accuracy after last epoch:", val_accuracy_last_epoch)
```

validation accuracy after changing the batch size to 16



validation accuracy after changing the batch size to 32

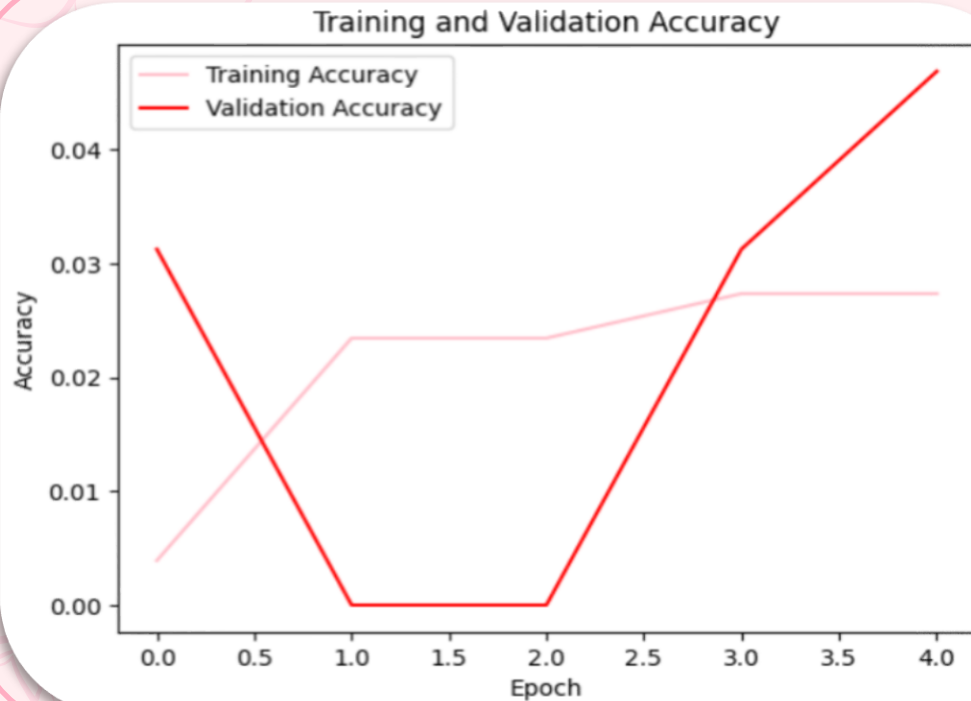


change the number of epochs

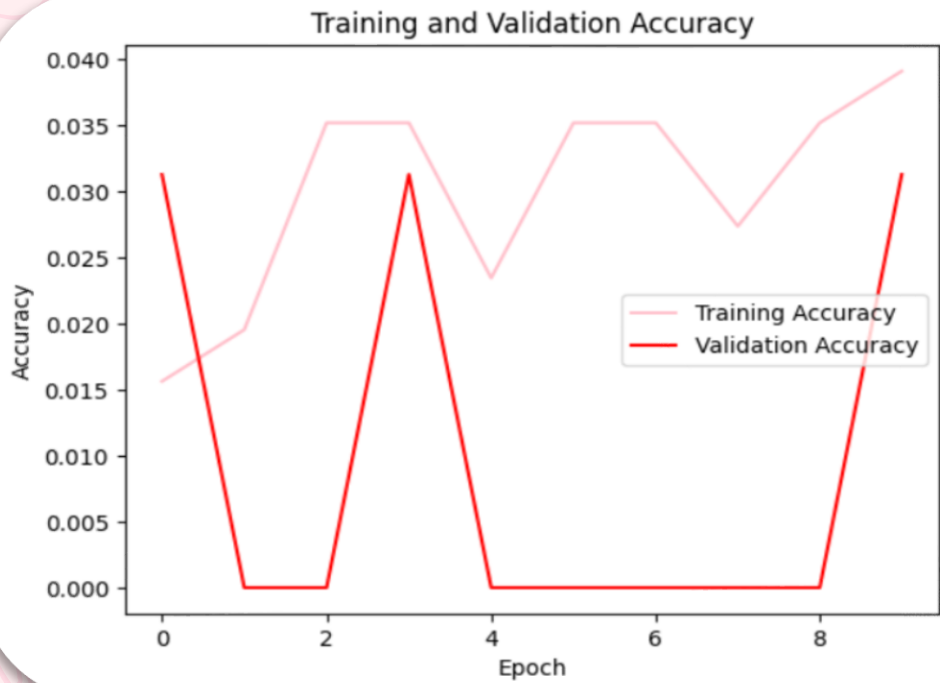
By repeatedly sending the full dataset through the network, increasing the number of epochs in training a neural network allows the model to learn more from the data. As the model's parameters are adjusted over time, this can lead to improved performance.

However, it lengthens training time and increases the risk of overfitting, in which the model becomes overly specialized to the training data.

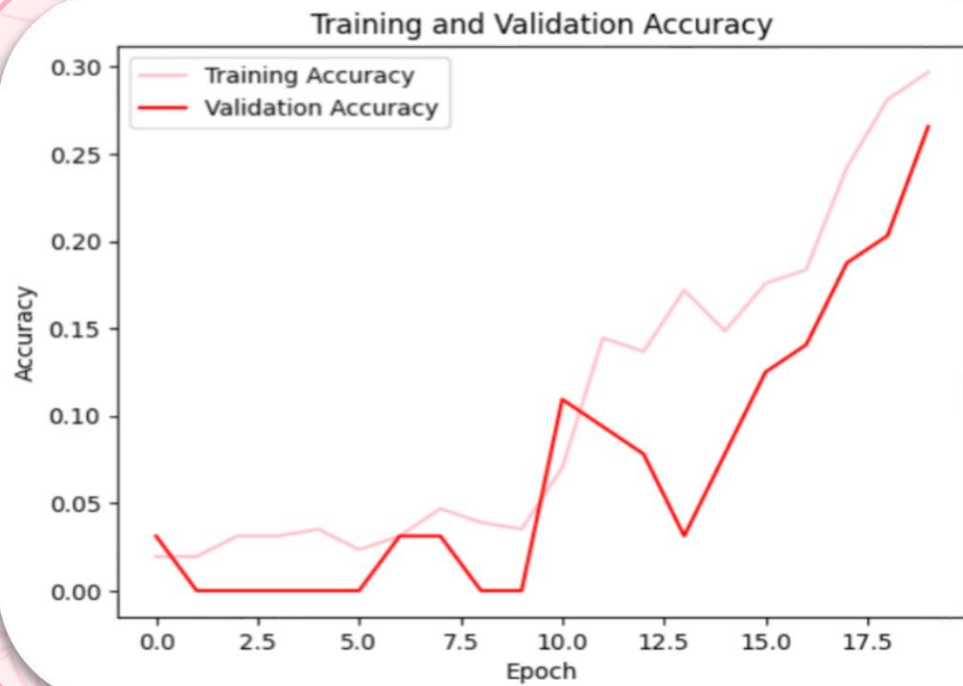




**When
Epoch is 5:**



**When
Epoch is 10:**



**When
Epoch is 20:**

change the test data size to 40%

```
# Set the new test data size to 40%
test_data_size = 0.4

# Determine the number of samples for the test data
num_test_samples = int(len(X_val) * test_data_size)

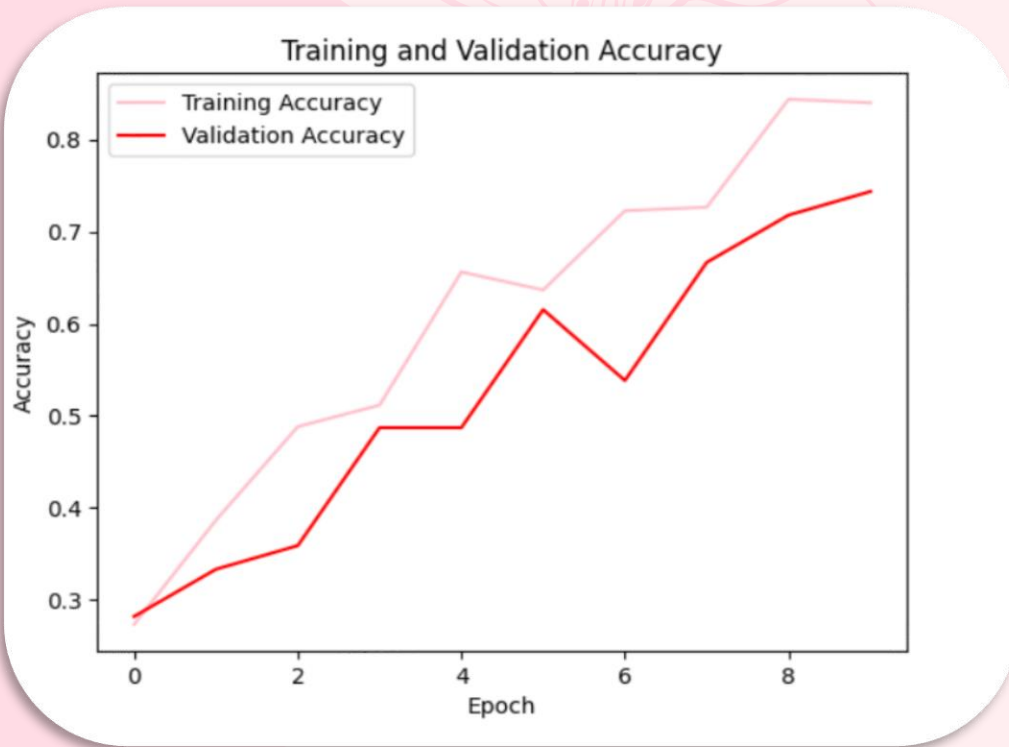
# Split the validation data into new validation and test data
X_new_val = X_val[:-num_test_samples]
Y_new_val = Y_val[:-num_test_samples]
X_new_test = X_val[-num_test_samples:]
Y_new_test = Y_val[-num_test_samples:]

# Train the model on the original train data
H = model.fit(X_train, Y_train, validation_data=(X_new_val, Y_new_val), batch_size=16, epochs=10, verbose=1)

# Evaluate the model on the new test data
scores = model.evaluate(X_new_test, Y_new_test, verbose=0)

# Get the validation accuracy from the evaluation scores
final_val_acc = scores[1]

# Print the final validation accuracy
print(f"Final validation accuracy with 40% test data size: {final_val_acc}")
```



**The Effect On
The Validation
Accuracy Of The
Algorithm?**