LeNet

LeNet5 is a small network, it contains the basic modules of deep learning: convolutional layer, pooling layer, and full link layer. It is the basis of other deep learning models. Here we analyze LeNet5 in depth. At the same time, through example analysis, deepen the understanding of the convolutional layer and pooling layer.

In [39]:

```
import keras
from keras.datasets import mnist
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Dense, Flatten
from keras.models import Sequential
```

1. Loading Images

In [40]:

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   validation_split=0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [41]:

```
training_set = train_datagen.flow_from_directory(r'C:\Users\roshan.gupta\Downloads\Family\train',
                                                 target_size = (64, 64),
                                                 batch_size = 8,
                                                 subset="training",
                                                 class_mode = 'categorical')

validation_set = train_datagen.flow_from_directory(r'C:\Users\roshan.gupta\Downloads\Family\train'
,
                                                 target_size = (64, 64),
                                                 batch_size = 8,
                                                subset="validation",
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(r'C:\Users\roshan.gupta\Downloads\Family\test',
                                            target_size = (64, 64),
                                            batch_size = 8,
                                            class_mode = 'categorical')

#color_mode = "grayscale"
```

```
Found 70 images belonging to 10 classes.
Found 10 images belonging to 10 classes.
Found 80 images belonging to 10 classes.
```

In [42]:

```
STEP_SIZE_TRAIN=training_set.n//training_set.batch_size
STEP_SIZE_VALID=validation_set.n//validation_set.batch_size
STEP_SIZE_TEST=test_set.n//test_set.batch_size
```

1. Building a sequential model

1.1 Implementation with LeNet architechture - Tanh and Average Pooling

In [43]:

```
model1 = Sequential()

# Select 6 feature convolution kernels with a size of 5 * 5 (without offset), and get 6 feature maps. The size of each feature map is 64-5 + 1 = 59 + 1 = 60
# Parameters between input layer and C1 layer: 6 * (5 * 5 + 1)
model1.add(Conv2D(6, kernel_size=(5, 5), activation='tanh', input_shape=(64, 64, 3)))

# The input of this layer is the output of the first layer, which is a 60 * 60 * 6 node matrix.
# The size of the filter used in this layer is 2 * 2, and the step length and width are both 2, so the output matrix size of this layer is 60 * 60 * 6.
model1.add(AveragePooling2D(pool_size=(2, 2)))


# The input matrix size of this layer is 30 * 30 * 6, the filter size used is 5 * 5, and the depth is 16. This layer does not use all 0 padding, and the step size is 1.
# The output matrix size of this layer is 26 * 26 * 16. This layer has 5 * 5 * 6 * 16 + 16 = 2416 parameters
model1.add(Conv2D(16, kernel_size=(5, 5), activation='tanh'))

# The input matrix size of this layer is 26 * 26 * 16. The size of the filter used in this layer is 2 * 2, and the length and width steps are both 2,
# so the output matrix size of this layer is 13 * 13 * 16.
model1.add(AveragePooling2D(pool_size=(2, 2)))

# The input matrix size of this layer is 13 * 13 * 16. This layer is called a convolution layer in the LeNet-5 paper, but because the size of the filter is 5 * 5, #
# So it is not different from the fully connected layer. If the nodes in the 5 * 5 * 16 matrix are pulled into a vector, then this layer is the same as the fully connected layer.
# The number of output nodes in this layer is 120, with a total of 13 * 13 * 16 * 120 + 120 = 324600 parameters.
model1.add(Flatten())

model1.add(Dense(120, activation='tanh'))
# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total parameter is 120 * 84 + 84 = 10164 (w + b)

model1.add(Dense(84, activation='tanh'))
# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total parameter is 84 * 10 + 10 = 850

model1.add(Dense(10, activation='softmax'))

model1.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
#model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(x_test, y_test))
```

In [44]:

```
model1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_5 (Conv2D)            (None, 60, 60, 6)         456       
_________________________________________________________________
average_pooling2d_5 (Average (None, 30, 30, 6)         0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 26, 26, 16)        2416      
_________________________________________________________________
average_pooling2d_6 (Average (None, 13, 13, 16)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 2704)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 120)               324600    
_________________________________________________________________
dense_8 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_9 (Dense)              (None, 10)                850       
```

```
dense_9 (Dense)              (None, 10)                850
=================================================================
Total params: 338,486
Trainable params: 338,486
Non-trainable params: 0
_________________________________________________________________
```

In [45]:

```
model1.fit_generator(training_set,
                     steps_per_epoch=STEP_SIZE_TRAIN,
                     epochs = 18, verbose=5,
                     validation_data = validation_set,
                     validation_steps = STEP_SIZE_VALID)
```

```
Epoch 1/18
Epoch 2/18
Epoch 3/18
Epoch 4/18
Epoch 5/18
Epoch 6/18
Epoch 7/18
Epoch 8/18
Epoch 9/18
Epoch 10/18
Epoch 11/18
Epoch 12/18
Epoch 13/18
Epoch 14/18
Epoch 15/18
Epoch 16/18
Epoch 17/18
Epoch 18/18
```

Out[45]:

```
<keras.callbacks.callbacks.History at 0x24a8d5bd7c8>
```

In [46]:

```
score = model1.evaluate(test_set)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
10/10 [==============================] - 1s 69ms/step
Test Loss: 0.920493483543396
Test accuracy: 0.887499988079071
```

In [58]:

```
# Part 3 - Making new predictions
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('C:/Users/roshan.gupta/Downloads/Family/Test/Piyush/2.jpg',
target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = model1.predict(test_image)
training_set.class_indices
print(result)
res = np.argmax(result)

dict1 = {0 : 'Bush', 1: 'Cats', 2: 'Dogs', 3: 'Hrithik', 4: 'Modi', 5 : 'Obama', 6 : 'Piyush', 7 : '
Roshan',
         8: 'Salman', 9: 'Shah'}

print("The predicted output is :",dict1[res])
```

```
[[0.03454989 0.037281   0.00540133 0.02462806 0.04111099 0.01344202
  0.8051072  0.00727952 0.01273685 0.01846326]]
The predicted output is : Piyush
```

1.2 Implementation with LeNet architecture - Tanh and Max Pooling

In [61]:

```
model2 = Sequential()
# Select 6 feature convolution kernels with a size of 5 * 5 (without offset), and get 6 feature maps. The size of each feature map is 64-5 + 1 = 59 + 1 = 60
# Parameters between input layer and C1 layer: 6 * (5 * 5 + 1)
model2.add(Conv2D(6, kernel_size=(5, 5), activation='tanh', input_shape=(64, 64, 3)))

# The input of this layer is the output of the first layer, which is a 60 * 60 * 6 node matrix.
# The size of the filter used in this layer is 2 * 2, and the step length and width are both 2, so the output matrix size of this layer is 60 * 60 * 6.
model2.add(MaxPooling2D(pool_size=(2, 2)))


# The input matrix size of this layer is 30 * 30 * 6, the filter size used is 5 * 5, and the depth is 16. This layer does not use all 0 padding, and the step size is 1.
# The output matrix size of this layer is 26 * 26 * 16. This layer has 5 * 5 * 6 * 16 + 16 = 2416 parameters
model2.add(Conv2D(16, kernel_size=(5, 5), activation='tanh'))

# The input matrix size of this layer is 26 * 26 * 16. The size of the filter used in this layer is 2 * 2, and the length and width steps are both 2,
# so the output matrix size of this layer is 13 * 13 * 16.
model2.add(MaxPooling2D(pool_size=(2, 2)))

# The input matrix size of this layer is 13 * 13 * 16. This layer is called a convolution layer in the LeNet-5 paper, but because the size of the filter is 5 * 5, #
# So it is not different from the fully connected layer. If the nodes in the 5 * 5 * 16 matrix are pulled into a vector, then this layer is the same as the fully connected layer.
# The number of output nodes in this layer is 120, with a total of 13 * 13 * 16 * 120 + 120 = 324600 parameters.
model2.add(Flatten())

model2.add(Dense(120, activation='tanh'))
# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total parameter is 120 * 84 + 84 = 10164 (w + b)

model2.add(Dense(84, activation='tanh'))
# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total parameter is 84 * 10 + 10 = 850

model2.add(Dense(10, activation='softmax'))

model2.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

model2.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_9 (Conv2D)            (None, 60, 60, 6)         456       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 30, 30, 6)         0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 26, 26, 16)        2416      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 16)        0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 2704)              0         
_________________________________________________________________
dense_13 (Dense)             (None, 120)               324600    
_________________________________________________________________
dense_14 (Dense)             (None, 84)                10164     
_________________________________________________________________
dense_15 (Dense)             (None, 10)                850       
=================================================================
Total params: 338,486
Trainable params: 338,486
Non-trainable params: 0
_________________________________________________________________
```

In [62]:

```
model2.fit_generator(training_set,
                     steps_per_epoch=STEP_SIZE_TRAIN,
                     epochs = 18, verbose=5,
                     validation_data = validation_set,
                     validation_steps = STEP_SIZE_VALID)
```

```
Epoch 1/18
Epoch 2/18
Epoch 3/18
Epoch 4/18
Epoch 5/18
Epoch 6/18
Epoch 7/18
Epoch 8/18
Epoch 9/18
Epoch 10/18
Epoch 11/18
Epoch 12/18
Epoch 13/18
Epoch 14/18
Epoch 15/18
Epoch 16/18
Epoch 17/18
Epoch 18/18
```

Out[62]:

```
<keras.callbacks.callbacks.History at 0x24a93f38188>
```

In [63]:

```
score = model2.evaluate(test_set)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
10/10 [==============================] - 0s 46ms/step
Test Loss: 0.7357671856880188
Test accuracy: 0.9125000238418579
```

In [71]:

```
# Part 3 - Making new predictions
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('C:/Users/roshan.gupta/Downloads/Family/Test/Modi/2.jpg', target_size
= (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = model1.predict(test_image)
training_set.class_indices
print(result)
res = np.argmax(result)

dict1 = {0 : 'Bush', 1: 'Cats', 2: 'Dogs', 3: 'Hrithik', 4: 'Modi', 5 : 'Obama', 6 : 'Piyush', 7 : '
Roshan',
         8: 'Salman', 9: 'Shah'}

print("The predicted output is :",dict1[res])
```

```
[[2.7456319e-02 1.5266129e-02 2.5604759e-05 2.8343530e-02 7.9575914e-01
  5.3124208e-02 2.9448282e-03 1.6396380e-03 2.5418893e-02 5.0021779e-02]]
The predicted output is : Modi
```

1.3 Implementation with LeNet architecture - Relu and Max Pooling

In [74]:

```
model3 = Sequential()
# Select 6 feature convolution kernels with a size of 5 * 5 (without offset), and get 6 feature ma
ps. The size of each feature map is 64-5 + 1 = 59 + 1 = 60
# Parameters between input layer and C1 layer: 6 * (5 * 5 + 1)
model3.add(Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(64, 64, 3)))

# The input of this layer is the output of the first layer, which is a 60 * 60 * 6 node matrix.
# The size of the filter used in this layer is 2 * 2, and the step length and width are both 2, so
the output matrix size of this layer is 60 * 60 * 6.
model3.add(MaxPooling2D(pool_size=(2, 2)))


# The input matrix size of this layer is 30 * 30 * 6, the filter size used is 5 * 5, and the depth
is 16. This layer does not use all 0 padding, and the step size is 1.
# The output matrix size of this layer is 26 * 26 * 16. This layer has 5 * 5 * 6 * 16 + 16 = 2416
parameters
model3.add(Conv2D(16, kernel_size=(5, 5), activation='relu'))

# The input matrix size of this layer is 26 * 26 * 16. The size of the filter used in this layer i
s 2 * 2, and the length and width steps are both 2,
# so the output matrix size of this layer is 13 * 13 * 16.
model3.add(MaxPooling2D(pool_size=(2, 2)))

# The input matrix size of this layer is 13 * 13 * 16. This layer is called a convolution layer in
the LeNet-5 paper, but because the size of the filter is 5 * 5, #
# So it is not different from the fully connected layer. If the nodes in the 5 * 5 * 16 matrix are
pulled into a vector, then this layer is the same as the fully connected layer.
# The number of output nodes in this layer is 120, with a total of 13 * 13 * 16 * 120 + 120 = 3246
00 parameters.
model3.add(Flatten())

model3.add(Dense(120, activation='relu'))
# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total p
arameter is 120 * 84 + 84 = 10164 (w + b)

model3.add(Dense(84, activation='relu'))
# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total pa
rameter is 84 * 10 + 10 = 850

model3.add(Dense(10, activation='softmax'))

model3.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])

model3.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_13 (Conv2D)           (None, 60, 60, 6)         456       
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 30, 30, 6)         0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 26, 26, 16)        2416      
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 16)        0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 2704)              0         
_________________________________________________________________
dense_19 (Dense)             (None, 120)               324600    
_________________________________________________________________
dense_20 (Dense)             (None, 84)                10164     
_________________________________________________________________
dense_21 (Dense)             (None, 10)                850       
=================================================================
Total params: 338,486
Trainable params: 338,486
Non-trainable params: 0
_________________________________________________________________
```

In [75]:

```
model3.fit_generator(training_set,
                    steps_per_epoch=STEP_SIZE_TRAIN,
```

```
                     epochs = 18, verbose=5,
                     validation_data = validation_set,
                     validation_steps = STEP_SIZE_VALID)
```

```
Epoch 1/18
Epoch 2/18
Epoch 3/18
Epoch 4/18
Epoch 5/18
Epoch 6/18
Epoch 7/18
Epoch 8/18
Epoch 9/18
Epoch 10/18
Epoch 11/18
Epoch 12/18
Epoch 13/18
Epoch 14/18
Epoch 15/18
Epoch 16/18
Epoch 17/18
Epoch 18/18
```

Out[75]:

```
<keras.callbacks.callbacks.History at 0x24a916e8108>
```

In [76]:

```
score = model3.evaluate(test_set)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
10/10 [==============================] - 1s 137ms/step
Test Loss: 2.189748764038086
Test accuracy: 0.824999988079071
```

In [80]:

```
# Part 3 - Making new predictions
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('C:/Users/roshan.gupta/Downloads/Family/Test/Hrithik/6.jpg',
target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = model1.predict(test_image)
training_set.class_indices
print(result)
res = np.argmax(result)

dict1 = {0 : 'Bush', 1: 'Cats', 2: 'Dogs', 3: 'Hrithik', 4: 'Modi', 5 : 'Obama', 6 : 'Piyush', 7 : '
Roshan',
         8: 'Salman', 9: 'Shah'}

print("The predicted output is :",dict1[res])
```

```
[[2.4552407e-02 6.1611593e-02 1.3586821e-03 7.1935385e-01 1.8925961e-02
  1.6382116e-01 5.6543257e-03 1.9961232e-04 3.5745027e-03 9.4804092e-04]]
The predicted output is : Hrithik
```

1.4 Implementation with LeNet architecture - Relu and Average Pooling

In [81]:

```
model4 = Sequential()
# Select 6 feature convolution kernels with a size of 5 * 5 (without offset), and get 6 feature ma
ps.
#The size of each feature map is 64-5 + 1 = 59 + 1 = 60
# Parameters between input layer and C1 layer: 6 * (5 * 5 + 1)
```

```
model4.add(Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(64, 64, 3)))

# The input of this layer is the output of the first layer, which is a 60 * 60 * 6 node matrix.
# The size of the filter used in this layer is 2 * 2, and the step length and width are both 2, so
the output matrix size of this layer is 60 * 60 * 6.
model4.add(AveragePooling2D(pool_size=(2, 2)))


# The input matrix size of this layer is 30 * 30 * 6, the filter size used is 5 * 5, and the depth
is 16. This layer does not use all 0 padding, and the step size is 1.
# The output matrix size of this layer is 26 * 26 * 16. This layer has 5 * 5 * 6 * 16 + 16 = 2416
parameters
model4.add(Conv2D(16, kernel_size=(5, 5), activation='relu'))

# The input matrix size of this layer is 26 * 26 * 16. The size of the filter used in this layer i
s 2 * 2, and the length and width steps are both 2,
# so the output matrix size of this layer is 13 * 13 * 16.
model4.add(AveragePooling2D(pool_size=(2, 2)))

# The input matrix size of this layer is 13 * 13 * 16. This layer is called a convolution layer in
the LeNet-5 paper, but because the size of the filter is 5 * 5, #
# So it is not different from the fully connected layer. If the nodes in the 5 * 5 * 16 matrix are
pulled into a vector, then this layer is the same as the fully connected layer.
# The number of output nodes in this layer is 120, with a total of 13 * 13 * 16 * 120 + 120 = 3246
00 parameters.
model4.add(Flatten())

model4.add(Dense(120, activation='relu'))
# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total p
arameter is 120 * 84 + 84 = 10164 (w + b)

model4.add(Dense(84, activation='relu'))
# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total pa
rameter is 84 * 10 + 10 = 850

model4.add(Dense(10, activation='softmax'))

model4.compile(loss=keras.metrics.categorical_crossentropy, optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])

model4.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 60, 60, 6)         456
_________________________________________________________________
average_pooling2d_7 (Average (None, 30, 30, 6)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 26, 26, 16)        2416
_________________________________________________________________
average_pooling2d_8 (Average (None, 13, 13, 16)        0
_________________________________________________________________
flatten_8 (Flatten)          (None, 2704)              0
_________________________________________________________________
dense_22 (Dense)             (None, 120)               324600
_________________________________________________________________
dense_23 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_24 (Dense)             (None, 10)                850
=================================================================
Total params: 338,486
Trainable params: 338,486
Non-trainable params: 0
_________________________________________________________________
```

In [82]:

```
model4.fit_generator(training_set,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    epochs = 18, verbose=5,
                    validation_data = validation_set,
                    validation_steps = STEP_SIZE_VALID)
```

```
Epoch 1/18
Epoch 2/18
Epoch 3/18
Epoch 4/18
Epoch 5/18
Epoch 6/18
Epoch 7/18
Epoch 8/18
Epoch 9/18
Epoch 10/18
Epoch 11/18
Epoch 12/18
Epoch 13/18
Epoch 14/18
Epoch 15/18
Epoch 16/18
Epoch 17/18
Epoch 18/18
```

Out[82]:

```
<keras.callbacks.callbacks.History at 0x24a970fa348>
```

In [83]:

```python
score = model4.evaluate(test_set)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
10/10 [==============================] - 1s 70ms/step
Test Loss: 1.9003064632415771
Test accuracy: 0.699999988079071
```

In [92]:

```python
# Part 3 - Making new predictions
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('C:/Users/roshan.gupta/Downloads/Family/Test/Salman/3.jpg',
target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = model1.predict(test_image)
training_set.class_indices
print(result)
res = np.argmax(result)

dict1 = {0 : 'Bush', 1: 'Cats', 2: 'Dogs', 3: 'Hrithik', 4: 'Modi', 5 : 'Obama', 6 : 'Piyush', 7 : '
Roshan',
         8: 'Salman', 9: 'Shah'}

print("The predicted output is :",dict1[res])
```

```
[[0.05474975 0.01296161 0.00069671 0.2829544  0.06095781 0.08145274
  0.01436987 0.00335436 0.48727518 0.00122761]]
The predicted output is : Salman
```

In [132]:

```python
#model.evaluate_generator(generator=validation_set, steps=STEP_SIZE_TEST)
```

Out[132]:

```
[3.2621877193450928, 0.16249999403953552]
```

Table (Different models with different activation funcs and Poolings):

In [93]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Activation Funcn", "Pooling", "Test Accuracy"]

x.add_row(["Tanh", "Average Pooling", 0.8874])
x.add_row(["Tanh", "Max Pooling" , 0.9125])
x.add_row(["Relu", "Max Pooling" , 0.8249])
x.add_row(["Relu", "Average Pooling" , 0.6999])
print(x)
```

```
+------------------+-----------------+---------------+
| Activation Funcn |     Pooling     | Test Accuracy |
+------------------+-----------------+---------------+
|       Tanh       | Average Pooling |     0.8874    |
|       Tanh       |   Max Pooling   |     0.9125    |
|       Relu       |   Max Pooling   |     0.8249    |
|       Relu       | Average Pooling |     0.6999    |
+------------------+-----------------+---------------+
```

Conclusions:

1. Here, we can see that architecture with activation function 'Tanh' and Pooling 'Max Pooling' yeild the best results.
2. Architecture with 'Tanh' function giving the better results.

In [ ]:

```

```