In [1]:

```
import os
import glob

base_dir = os.path.join('./cell_images')
infected_dir = os.path.join(base_dir,'Parasitized')
healthy_dir = os.path.join(base_dir,'Uninfected')

infected_files = glob.glob(infected_dir+'/*.png')
healthy_files = glob.glob(healthy_dir+'/*.png')
len(infected_files), len(healthy_files)
```

Out[1]:

```
(13779, 13779)
```

In [2]:

```
import numpy as np
import pandas as pd

np.random.seed(42)

files_df = pd.DataFrame({
    'filename': infected_files + healthy_files,
    'label': ['malaria'] * len(infected_files) + ['healthy'] * len(healthy_files)
}).sample(frac=1, random_state=42).reset_index(drop=True)

files_df.head()
```

Out[2]:

	filename	label
0	./cell_images\Parasitized\C130P91ThinF_IMG_201...	malaria
1	./cell_images\Parasitized\C188P149ThinF_IMG_20...	malaria
2	./cell_images\Uninfected\C173P134NThinF_IMG_20...	healthy
3	./cell_images\Uninfected\C78P39ThinF_IMG_20150...	healthy
4	./cell_images\Uninfected\C107P68ThinF_IMG_2015...	healthy

In [3]:

```
from sklearn.model_selection import train_test_split
from collections import Counter

train_files, test_files, train_labels, test_labels = train_test_split(files_df['filename
'].values,
                                                                      files_df['label']
.values,
                                                                      test_size=0.3, ra
ndom_state=42)
train_files, val_files, train_labels, val_labels = train_test_split(train_files,
                                                                    train_labels,
                                                                    test_size=0.1, rand
om_state=42)

print(train_files.shape, val_files.shape, test_files.shape)
print('Train:', Counter(train_labels), '\nVal:', Counter(val_labels), '\nTest:', Counter
(test_labels))
```

```
(17361,) (1929,) (8268,)
Train: Counter({'healthy': 8734, 'malaria': 8627})
Val: Counter({'healthy': 970, 'malaria': 959})
Test: Counter({'malaria': 4193, 'healthy': 4075})
```

In [5]:

```
import os
os.sys.path
```

Out[5]:

```
['C:\\Users\\sidsa\\Anaconda3\\python36.zip',
 'C:\\Users\\sidsa\\Anaconda3\\DLLs',
 'C:\\Users\\sidsa\\Anaconda3\\lib',
 'C:\\Users\\sidsa\\Anaconda3',
 '',
 'C:\\Users\\sidsa\\Anaconda3\\lib\\site-packages',
 'C:\\Users\\sidsa\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\Users\\sidsa\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\sidsa\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\sidsa\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\sidsa\\.ipython']
```

In [7]:

```
import cv2
from concurrent import futures
import threading
```

In [8]:

```
def get_img_shape_parallel(idx, img, total_imgs):
    if idx % 5000 == 0 or idx == (total_imgs - 1):
        print('{}: working on img num: {}'.format(threading.current_thread().name,
                                                  idx))
    return cv2.imread(img).shape

ex = futures.ThreadPoolExecutor(max_workers=None)
data_inp = [(idx, img, len(train_files)) for idx, img in enumerate(train_files)]
print('Starting Img shape computation:')
train_img_dims_map = ex.map(get_img_shape_parallel,
                            [record[0] for record in data_inp],
                            [record[1] for record in data_inp],
                            [record[2] for record in data_inp])
train_img_dims = list(train_img_dims_map)
print('Min Dimensions:', np.min(train_img_dims, axis=0))
print('Avg Dimensions:', np.mean(train_img_dims, axis=0))
print('Median Dimensions:', np.median(train_img_dims, axis=0))
print('Max Dimensions:', np.max(train_img_dims, axis=0))
```

```
Starting Img shape computation:
ThreadPoolExecutor-0_0: working on img num: 0
ThreadPoolExecutor-0_19: working on img num: 5000
ThreadPoolExecutor-0_1: working on img num: 10000
ThreadPoolExecutor-0_13: working on img num: 15000
ThreadPoolExecutor-0_12: working on img num: 17360
Min Dimensions: [46 49  3]
Avg Dimensions: [132.89856575 132.50751685   3.        ]
Median Dimensions: [130. 130.   3.]
Max Dimensions: [382 394   3]
```

In [9]:

```
IMG_DIMS = (125, 125)

def get_img_data_parallel(idx, img, total_imgs):
    if idx % 5000 == 0 or idx == (total_imgs - 1):
        print('{}: working on img num: {}'.format(threading.current_thread().name,
                                                  idx))
    img = cv2.imread(img)
    img = cv2.resize(img, dsize=IMG_DIMS,
                     interpolation=cv2.INTER_CUBIC)
    img = np.array(img, dtype=np.float32)
    return img

ex = futures.ThreadPoolExecutor(max_workers=None)
```

```
train_data_inp = [(idx, img, len(train_files)) for idx, img in enumerate(train_files)]
val_data_inp = [(idx, img, len(val_files)) for idx, img in enumerate(val_files)]
test_data_inp = [(idx, img, len(test_files)) for idx, img in enumerate(test_files)]
```

In [10]:

```
print('Loading Train Images:')
train_data_map = ex.map(get_img_data_parallel,
                        [record[0] for record in train_data_inp],
                        [record[1] for record in train_data_inp],
                        [record[2] for record in train_data_inp])
train_data = np.array(list(train_data_map))
```

```
Loading Train Images:
ThreadPoolExecutor-1_0: working on img num: 0
ThreadPoolExecutor-1_6: working on img num: 5000
ThreadPoolExecutor-1_8: working on img num: 10000
ThreadPoolExecutor-1_14: working on img num: 15000
ThreadPoolExecutor-1_9: working on img num: 17360
```

In [11]:

```
print('\nLoading Validation Images:')
val_data_map = ex.map(get_img_data_parallel,
                        [record[0] for record in val_data_inp],
                        [record[1] for record in val_data_inp],
                        [record[2] for record in val_data_inp])
val_data = np.array(list(val_data_map))
```

```
Loading Validation Images:
ThreadPoolExecutor-1_10: working on img num: 0
ThreadPoolExecutor-1_7: working on img num: 1928
```

In [12]:

```
print('\nLoading Test Images:')
test_data_map = ex.map(get_img_data_parallel,
                        [record[0] for record in test_data_inp],
                        [record[1] for record in test_data_inp],
                        [record[2] for record in test_data_inp])
test_data = np.array(list(test_data_map))
```

```
Loading Test Images:
ThreadPoolExecutor-1_15: working on img num: 0
ThreadPoolExecutor-1_15: working on img num: 5000
ThreadPoolExecutor-1_18: working on img num: 8267
```

In [13]:

```
2+2
```

Out[13]:

```
4
```

In [14]:

```
train_data.shape, val_data.shape, test_data.shape
```

Out[14]:

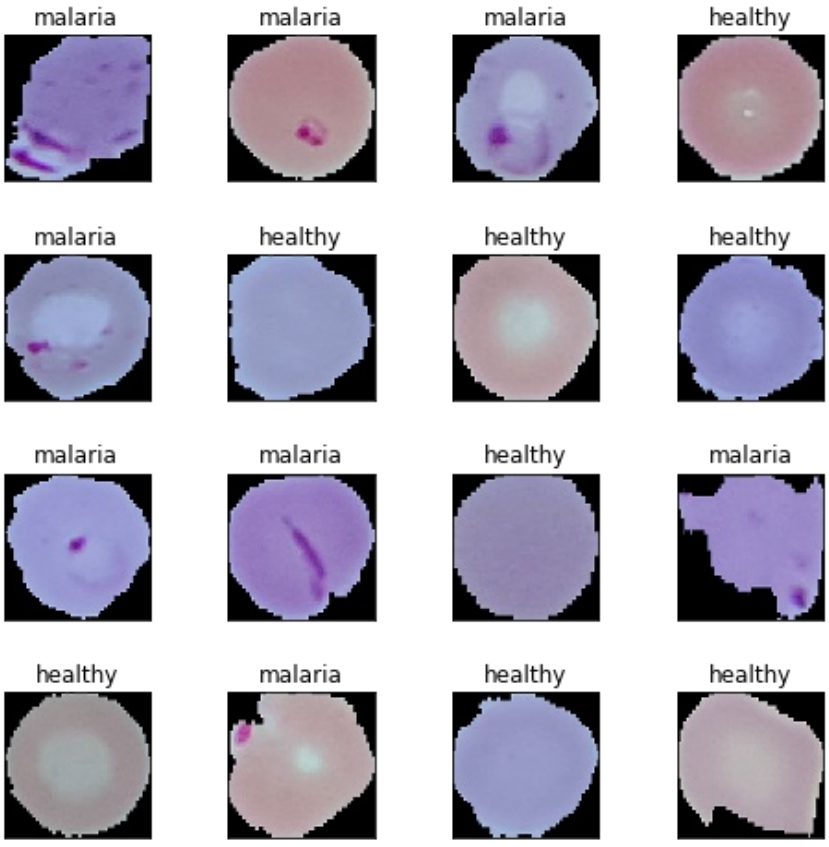
```
((17361, 125, 125, 3), (1929, 125, 125, 3), (8268, 125, 125, 3))
```

In [15]:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(1 , figsize = (8 , 8))
n = 0
for i in range(16):
    n += 1
```

```
    r = np.random.randint(0 , train_data.shape[0] , 1)
    plt.subplot(4 , 4 , n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
    plt.imshow(train_data[r[0]]/255.)
    plt.title('{}'.format(train_labels[r[0]]))
    plt.xticks([]) , plt.yticks([])
```



In [16]:

```
BATCH_SIZE = 64
NUM_CLASSES = 2
EPOCHS = 25
INPUT_SHAPE = (125, 125, 3)

train_imgs_scaled = train_data / 255.
val_imgs_scaled = val_data / 255.

# encode text category labels
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(train_labels)
train_labels_enc = le.transform(train_labels)
val_labels_enc = le.transform(val_labels)

print(train_labels[:6], train_labels_enc[:6])
```

```
['malaria' 'malaria' 'malaria' 'healthy' 'healthy' 'malaria'] [1 1 1 0 0 1]
```

In [21]:

```
import tensorflow as tf
# Load the TensorBoard notebook extension (optional)
%load_ext tensorboard

tf.random.set_seed(42)
tf.__version__
```

Out[21]:

```
'2.1.0'
```

In [22]:

```
inp = tf.keras.layers.Input(shape=INPUT_SHAPE)

conv1 = tf.keras.layers.Conv2D(32, kernel_size=(3, 3),
                               activation='relu', padding='same')(inp)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = tf.keras.layers.Conv2D(64, kernel_size=(3, 3),
                               activation='relu', padding='same')(pool1)
pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = tf.keras.layers.Conv2D(128, kernel_size=(3, 3),
                               activation='relu', padding='same')(pool2)
pool3 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv3)

flat = tf.keras.layers.Flatten()(pool3)

hidden1 = tf.keras.layers.Dense(512, activation='relu')(flat)
drop1 = tf.keras.layers.Dropout(rate=0.3)(hidden1)
hidden2 = tf.keras.layers.Dense(512, activation='relu')(drop1)
drop2 = tf.keras.layers.Dropout(rate=0.3)(hidden2)

out = tf.keras.layers.Dense(1, activation='sigmoid')(drop2)

model = tf.keras.Model(inputs=inp, outputs=out)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 125, 125, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 125, 125, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 62, 62, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 31, 31, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 28800)             0
_________________________________________________________________
dense (Dense)                (None, 512)               14746112
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 15,102,529
Trainable params: 15,102,529
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
import datetime

logdir = os.path.join('tensorboard_logs',
                      datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                              patience=2, min_lr=0.000001)
```

```
callbacks = [reduce_lr, tensorboard_callback]

history = model.fit(x=train_imgs_scaled, y=train_labels_enc,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(val_imgs_scaled, val_labels_enc),
                    callbacks=callbacks,
                    verbose=1)
```

```
Train on 17361 samples, validate on 1929 samples
Epoch 1/25
17361/17361 [==============================] - 287s 17ms/sample - loss: 0.4056 - accuracy
: 0.8048 - val_loss: 0.1835 - val_accuracy: 0.9430
Epoch 2/25
17361/17361 [==============================] - 288s 17ms/sample - loss: 0.1710 - accuracy
: 0.9474 - val_loss: 0.1460 - val_accuracy: 0.9565
Epoch 3/25
17361/17361 [==============================] - 279s 16ms/sample - loss: 0.1460 - accuracy
: 0.9533 - val_loss: 0.1388 - val_accuracy: 0.9606
Epoch 4/25
17361/17361 [==============================] - 272s 16ms/sample - loss: 0.1281 - accuracy
: 0.9574 - val_loss: 0.1597 - val_accuracy: 0.9580
Epoch 5/25
17361/17361 [==============================] - 272s 16ms/sample - loss: 0.1145 - accuracy
: 0.9608 - val_loss: 0.1287 - val_accuracy: 0.9606
Epoch 6/25
17361/17361 [==============================] - 270s 16ms/sample - loss: 0.0967 - accuracy
: 0.9658 - val_loss: 0.1556 - val_accuracy: 0.9596
Epoch 7/25
17361/17361 [==============================] - 274s 16ms/sample - loss: 0.0812 - accuracy
: 0.9706 - val_loss: 0.1356 - val_accuracy: 0.9585
Epoch 8/25
17361/17361 [==============================] - 277s 16ms/sample - loss: 0.0439 - accuracy
: 0.9854 - val_loss: 0.1962 - val_accuracy: 0.9590
Epoch 9/25
17361/17361 [==============================] - 291s 17ms/sample - loss: 0.0307 - accuracy
: 0.9897 - val_loss: 0.2211 - val_accuracy: 0.9575
Epoch 10/25
17361/17361 [==============================] - 271s 16ms/sample - loss: 0.0173 - accuracy
: 0.9949 - val_loss: 0.2498 - val_accuracy: 0.9585
Epoch 11/25
17361/17361 [==============================] - 229s 13ms/sample - loss: 0.0101 - accuracy
: 0.9974 - val_loss: 0.2764 - val_accuracy: 0.9590
Epoch 12/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0054 - accuracy
: 0.9989 - val_loss: 0.2901 - val_accuracy: 0.9590
Epoch 13/25
17361/17361 [==============================] - 241s 14ms/sample - loss: 0.0046 - accuracy
: 0.9990 - val_loss: 0.3007 - val_accuracy: 0.9580
Epoch 14/25
17361/17361 [==============================] - 247s 14ms/sample - loss: 0.0033 - accuracy
: 0.9992 - val_loss: 0.3211 - val_accuracy: 0.9590
Epoch 15/25
17361/17361 [==============================] - 257s 15ms/sample - loss: 0.0029 - accuracy
: 0.9996 - val_loss: 0.3296 - val_accuracy: 0.9590
Epoch 16/25
17361/17361 [==============================] - 242s 14ms/sample - loss: 0.0026 - accuracy
: 0.9994 - val_loss: 0.3298 - val_accuracy: 0.9590
Epoch 17/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0021 - accuracy
: 0.9997 - val_loss: 0.3322 - val_accuracy: 0.9596
Epoch 18/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0017 - accuracy
: 0.9997 - val_loss: 0.3390 - val_accuracy: 0.9590
Epoch 19/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0020 - accuracy
: 0.9996 - val_loss: 0.3432 - val_accuracy: 0.9596
Epoch 20/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0018 - accuracy
: 0.9996 - val_loss: 0.3468 - val_accuracy: 0.9596
Epoch 21/25
17361/17361 [==============================] - 229s 13ms/sample - loss: 0.0018 - accuracy
```

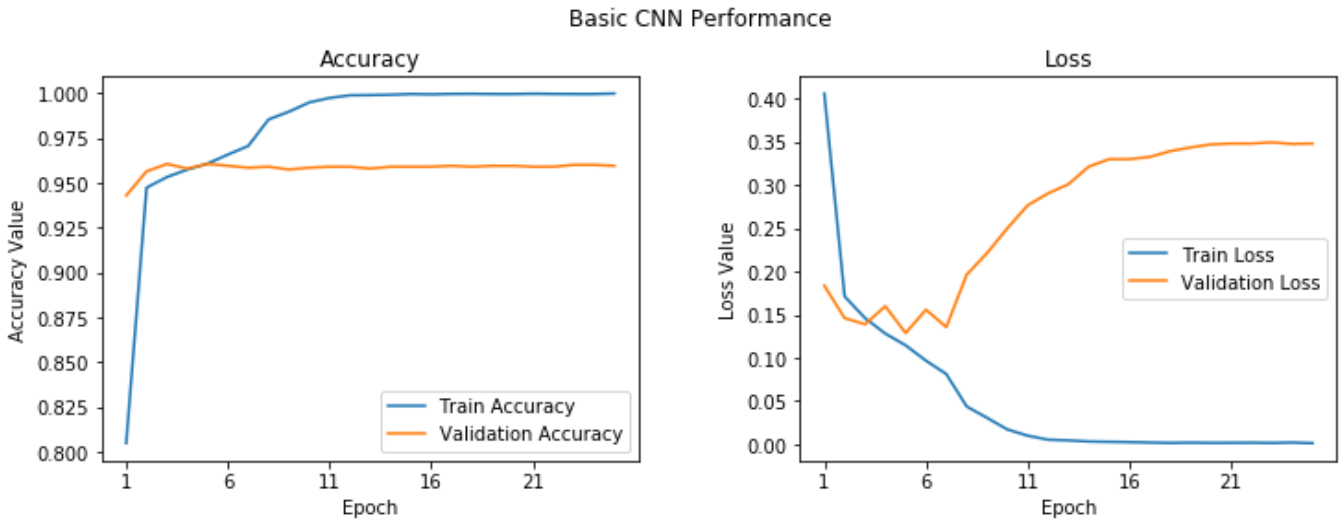
```
: 0.9998 - val_loss: 0.3477 - val_accuracy: 0.9590
Epoch 22/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0019 - accuracy
: 0.9997 - val_loss: 0.3478 - val_accuracy: 0.9590
Epoch 23/25
17361/17361 [==============================] - 231s 13ms/sample - loss: 0.0017 - accuracy
: 0.9996 - val_loss: 0.3491 - val_accuracy: 0.9601
Epoch 24/25
17361/17361 [==============================] - 230s 13ms/sample - loss: 0.0021 - accuracy
: 0.9996 - val_loss: 0.3473 - val_accuracy: 0.9601
Epoch 25/25
17361/17361 [==============================] - 229s 13ms/sample - loss: 0.0014 - accuracy
: 0.9999 - val_loss: 0.3478 - val_accuracy: 0.9596
```

In [28]:

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
t = f.suptitle('Basic CNN Performance', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

max_epoch = len(history.history['accuracy'])+1
epoch_list = list(range(1,max_epoch))
ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accuracy')
ax1.set_xticks(np.arange(1, max_epoch, 5))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(1, max_epoch, 5))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```



In [29]:

```
model.save('malaria.h5')
```

In [ ]:

```
```