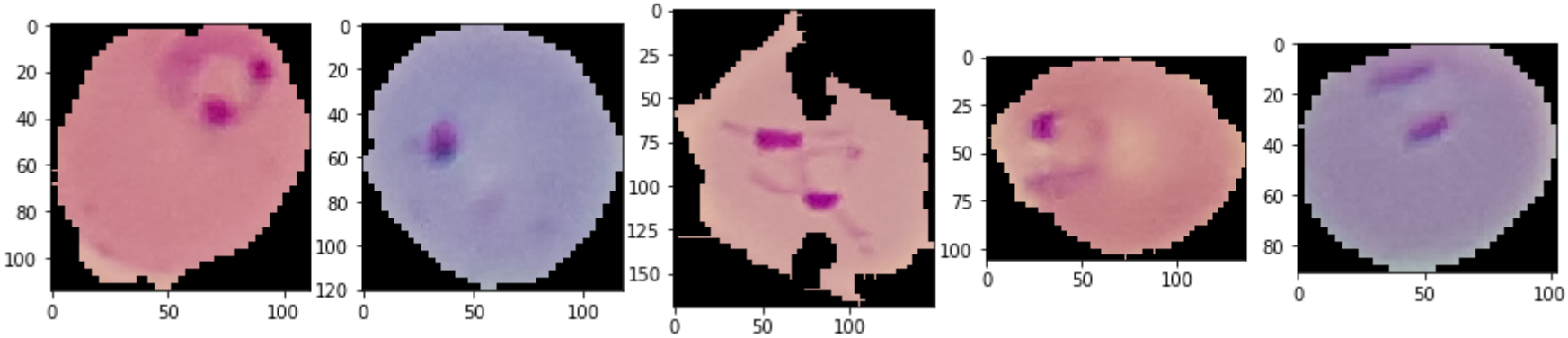
Malaria :

The disease which effects billions and kills Millions.
Here we have blood samples downloaded from https://ceb.nlm.nih.gov/proj/malaria/cell_images.zip (https://ceb.nlm.nih.gov/proj/malaria/cell_images.zip)
Now we will use these pictures to create a deep learning model to identify a person suffering from malaria.

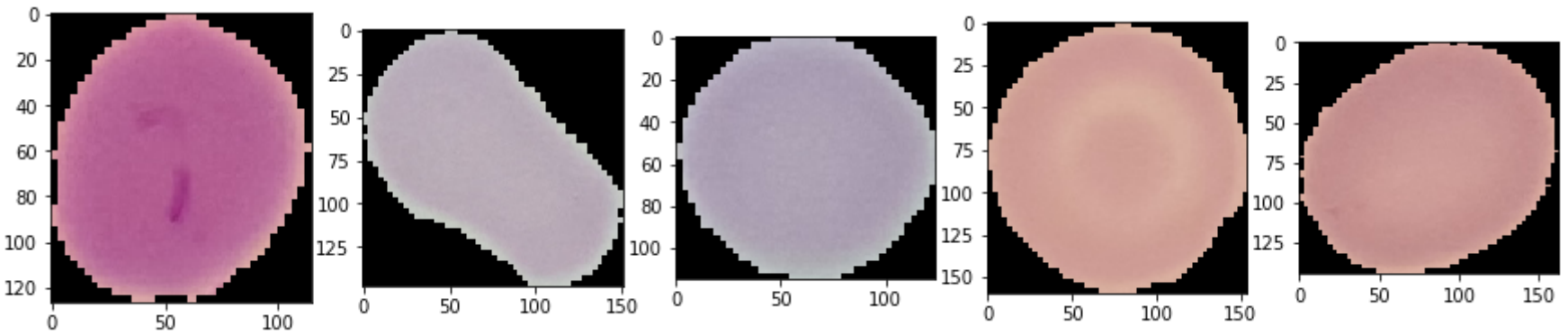
In [5]:
```
import os
import numpy as np
import cv2
from imutils import paths
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from keras.activations import relu, sigmoid
from keras.layers import Conv2D, BatchNormalization, Dropout, Dense
from keras.optimizers import Adam, SGD
from keras.models import Model,Sequential
import random
import shutil
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from sklearn.metrics import classification_report
```

In [6]:
```
directory=os.listdir('dataset2/testing/')
for each in directory:
    plt.figure()
    currentFolder = "dataset/" + each + "/"
#     print(currentFolder)
    plt.figure(figsize=(15,10))
    for i, file in enumerate(os.listdir(currentFolder)[0:5]):
        fullpath = currentFolder + file
        img=mpimg.imread(fullpath)
        plt.subplot(2, 5, i+1)
        plt.imshow(img)
```

```
<Figure size 432x288 with 0 Axes>
```



```
<Figure size 432x288 with 0 Axes>
```



In [12]:
```

# initialize the path to the *original* input directory of images
ORIG_INPUT_DATASET = "dataset"

# initialize the base path to the *new* directory that will contain
# our images after computing the training and testing split
BASE_PATH = "dataset2"

# derive the training, validation, and testing directories
TRAIN_PATH = os.path.sep.join([BASE_PATH, "training"])
VAL_PATH = os.path.sep.join([BASE_PATH, "validation"])
TEST_PATH = os.path.sep.join([BASE_PATH, "testing"])

# define the amount of data that will be used training
TRAIN_SPLIT = 0.8

# the amount of validation data will be a percentage of the
# *training* data
VAL_SPLIT = 0.1
```

In [13]:
```
imagePaths = list(paths.list_images(ORIG_INPUT_DATASET))
random.seed(42)
random.shuffle(imagePaths)
```

In [14]:
```
# compute the training and testing split
i = int(len(imagePaths) * TRAIN_SPLIT)
trainPaths = imagePaths[:i]
testPaths = imagePaths[i:]

# we'll be using part of the training data for validation
i = int(len(trainPaths) * VAL_SPLIT)
valPaths = trainPaths[:i]
trainPaths = trainPaths[i:]
```

In [15]:

```
# define the datasets that we'll be building
datasets = [
    ("training", trainPaths, TRAIN_PATH),
    ("validation", valPaths, VAL_PATH),
    ("testing", testPaths, TEST_PATH)
]
```

In [16]:
```
for (dType, imagePaths, baseOutput) in datasets:
    # show which data split we are creating
    print("[INFO] building '{}' split".format(dType))

    # if the output base output directory does not exist, create it
    if not os.path.exists(baseOutput):
        print("[INFO] 'creating {}' directory".format(baseOutput))
        os.makedirs(baseOutput)

    # loop over the input image paths
    for inputPath in imagePaths:
        # extract the filename of the input image along with its
        # corresponding class label
        filename = inputPath.split(os.path.sep)[-1]
        label = inputPath.split(os.path.sep)[-2]

        # build the path to the label directory
        labelPath = os.path.sep.join([baseOutput, label])

        # if the label output directory does not exist, create it
        if not os.path.exists(labelPath):
            print("[INFO] 'creating {}' directory".format(labelPath))
            os.makedirs(labelPath)

        # construct the path to the destination image and then copy
        # the image itself
        p = os.path.sep.join([labelPath, filename])
        shutil.copy2(inputPath, p)
```

```
[INFO] building 'training' split
[INFO] 'creating dataset2/training' directory
[INFO] 'creating dataset2/training/Parasitized' directory
[INFO] 'creating dataset2/training/Uninfected' directory
[INFO] building 'validation' split
[INFO] 'creating dataset2/validation' directory
[INFO] 'creating dataset2/validation/Parasitized' directory
[INFO] 'creating dataset2/validation/Uninfected' directory
[INFO] building 'testing' split
[INFO] 'creating dataset2/testing' directory
[INFO] 'creating dataset2/testing/Parasitized' directory
[INFO] 'creating dataset2/testing/Uninfected' directory
```

In [17]:
```
NUM_EPOCHS = 50
INIT_LR = 1e-1
BS = 32

def poly_decay(epoch):
    # initialize the maximum number of epochs, base learning rate,
    # and power of the polynomial
    maxEpochs = NUM_EPOCHS
    baseLR = INIT_LR
    power = 1.0

    # compute the new learning rate based on polynomial decay
    alpha = baseLR * (1 - (epoch / float(maxEpochs))) ** power

    # return the new learning rate
    return alpha
```

In [18]:
```
# determine the total number of image paths in training, validation,
# and testing directories
totalTrain = len(list(paths.list_images(TRAIN_PATH)))
totalVal = len(list(paths.list_images(VAL_PATH)))
totalTest = len(list(paths.list_images(TEST_PATH)))
```

In [19]:
```
trainAug = ImageDataGenerator(
    rescale=1 / 255.0,
    rotation_range=20,
    zoom_range=0.05,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.05,
    horizontal_flip=True,
    fill_mode="nearest")

# initialize the validation (and testing) data augmentation object
valAug = ImageDataGenerator(rescale=1 / 255.0)
```

In [20]:

```
trainGen = trainAug.flow_from_directory(
    TRAIN_PATH,
    class_mode="categorical",
    target_size=(64, 64),
    color_mode="rgb",
    shuffle=True,
    batch_size=BS)

# initialize the validation generator
valGen = valAug.flow_from_directory(
    VAL_PATH,
    class_mode="categorical",
    target_size=(64, 64),
    color_mode="rgb",
    shuffle=False,
    batch_size=BS)

# initialize the testing generator
testGen = valAug.flow_from_directory(
    TEST_PATH,
    class_mode="categorical",
    target_size=(64, 64),
    color_mode="rgb",
    shuffle=False,
    batch_size=BS)
```

```
Found 19842 images belonging to 2 classes.
Found 2204 images belonging to 2 classes.
Found 5512 images belonging to 2 classes.
```

In [21]:
```
from resnet import ResNet
model = ResNet.build(64, 64, 3, 2, (3, 4, 6),
    (64, 128, 256, 512), reg=0.0005)
opt = SGD(lr=INIT_LR, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

```
WARNING: Logging before flag parsing goes to stderr.
W1029 07:02:17.933146 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W1029 07:02:18.203471 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W1029 07:02:18.355821 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:174: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

W1029 07:02:18.356955 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:181: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

W1029 07:02:18.357681 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:186: The name tf.Session is deprecated. Please use tf.compat.v1.Session instead.

W1029 07:02:19.908818 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

W1029 07:02:20.271513 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:3976: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

W1029 07:02:24.357822 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:3980: The name tf.nn.avg_pool is deprecated. Please use tf.nn.avg_pool2d instead.

W1029 07:02:24.399907 140221142517504 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-package
```

```
s/keras/optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer
instead.

W1029 07:02:24.409409 140221142517504 deprecation.py:323] From /usr/local/lib/python3.5/dist-packages/tensorf
low/python/ops/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) i
s deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [22]:
```
model.summary()
```

conv2d_1 (Conv2D)	(None, 64, 64, 64)	4800	batch_normalization_1[0][0]
batch_normalization_2 (BatchNor	(None, 64, 64, 64)	256	conv2d_1[0][0]
activation_1 (Activation)	(None, 64, 64, 64)	0	batch_normalization_2[0][0]
zero_padding2d_1 (ZeroPadding2D	(None, 66, 66, 64)	0	activation_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0	zero_padding2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 32, 32, 64)	256	max_pooling2d_1[0][0]
activation_2 (Activation)	(None, 32, 32, 64)	0	batch_normalization_3[0][0]
conv2d_2 (Conv2D)	(None, 32, 32, 32)	2048	activation_2[0][0]
batch_normalization_4 (BatchNor	(None, 32, 32, 32)	128	conv2d_2[0][0]
activation_3 (Activation)	(None, 32, 32, 32)	0	batch_normalization_4[0][0]

In [24]:
```
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint("weights{epoch:03d}.h5", monitor='val_acc', verbose=1, save_best_only=True, mode
```

In [ ]:
```
callbacks = [LearningRateScheduler(poly_decay), checkpoint]
H = model.fit_generator(
    trainGen,
    steps_per_epoch=32,
    validation_data=valGen,
    validation_steps=32,
    epochs=NUM_EPOCHS,
    callbacks=callbacks
)
```

```
Epoch 1/50
32/32 [==============================] - 146s 5s/step - loss: 1.3650 - acc: 0.8779 - val_loss: 1.2775 - val_a
cc: 0.8926

Epoch 00001: val_acc improved from -inf to 0.89258, saving model to weights001.h5
Epoch 2/50
32/32 [==============================] - 138s 4s/step - loss: 1.2702 - acc: 0.9053 - val_loss: 1.2013 - val_a
cc: 0.9422

Epoch 00002: val_acc improved from 0.89258 to 0.94216, saving model to weights002.h5
Epoch 3/50
32/32 [==============================] - 121s 4s/step - loss: 1.2431 - acc: 0.9053 - val_loss: 1.2697 - val_a
cc: 0.8633

Epoch 00003: val_acc did not improve from 0.94216
Epoch 4/50
32/32 [==============================] - 137s 4s/step - loss: 1.1983 - acc: 0.9180 - val_loss: 1.1177 - val_a
cc: 0.9608
```

In [29]:
```
model.fit_generator(
    trainGen,
    steps_per_epoch=32,
    validation_data=valGen,
    validation_steps=32,
    epochs=1,
    callbacks=callbacks
)
```

```
Epoch 1/1
32/32 [==============================] - 120s 4s/step - loss: 0.8077 - acc: 0.9434 - val_loss: 0.7531 - val_a
cc: 0.9805

Epoch 00001: val_acc improved from 0.97451 to 0.98047, saving model to weights001.h5
```

Out[29]:
```
<keras.callbacks.History at 0x7f858cff74a8>
```

In [34]:
```
model.save("Final.h5")
```

In [30]:
```
# reset the testing generator and then use our trained model to
# make predictions on the data
print("[INFO] evaluating network...")
testGen.reset()
predIdxs = model.predict_generator(testGen,
    steps=(totalTest // BS) + 1)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testGen.classes, predIdxs,
    target_names=testGen.class_indices.keys()))
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

  Uninfected       0.98      0.91      0.94      2726
 Parasitized       0.92      0.98      0.95      2786

    accuracy                           0.95      5512
   macro avg       0.95      0.95      0.95      5512
weighted avg       0.95      0.95      0.95      5512
```

In [28]:
```
# plot the training loss and accuracy
N = NUM_EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")

```

Out[28]: <matplotlib.legend.Legend at 0x7f8687fb3828>

In [33]:
```
from sklearn.metrics import accuracy_score
accuracy_score(testGen.classes, predIdxs)
```

Out[33]: 0.9457547169811321

Summary :

The main parameter while classifying something related to Medics is F1 Score.
In Medics we don't want to classify someone as uninfected but in real the person is infected i.e False Positives.
Our precision should be high.
As i have limited resources so unfortunately and i was training on CPU, So I had to reduce Step size and trained for only 50 epochs. As we are seeing a decline in loss, so we can train it further with more step size and we will get better results.