In [1]:

```
#split images into flooded and non flooded folders
```

In [3]:

```
import json
import os
import cv2
import pandas as pd
import shutil
import numpy as np
import random
```

In [4]:

```
imgflist = []
rootdir = 'E:\Minor_project\dataset\sen12floods_s1_source'
for file in os.listdir(rootdir):
    d = os.path.join(rootdir, file)
    if os.path.isdir(d):
        imgflist.append(d)


print(f"The number of folders are currently = {len(imgflist)}")

labelslist = []
labeldir = 'E:\Minor_project\dataset\sen12floods_s1_labels'
for lfile in os.listdir(labeldir):
    d = os.path.join(labeldir, lfile)
    if os.path.isdir(d):
        labelslist.append(d)

print(f"The number of label folders are currently = {len(labelslist)}")
```

```
The number of folders are currently = 3331
The number of label folders are currently = 3331
```

In [5]:

```
#in the above code section, we create 2 lists:
# imgflist : stores location of image folder
# labelslist : stores location of each label folder
```

In [6]:

```
#flood and non flood seperation images
#copying flooded images in flooded folder
i=0
for lpath in labelslist:
    json_data=open(lpath+"/stac.json", "rb")
    jdata = json.load(json_data)
    flood = jdata["properties"]["FLOODING"]
    if(flood=='True'):
        src=imgflist[i]+'\VV.tif'
        dst=r'E:\sar_flood_proj\Flood_imgs\VV'+str(i)+'.tif'
        shutil.copy(src,dst)
#copying non flooded images in non flooded folder by accessing the list : imglist[i]
    else:
        src=imgflist[i]+'\VV.tif'
        dst=r'E:\sar_flood_proj\Non_flood_imgs\VV'+str(i)+'.tif'
        shutil.copy(src,dst)
    i=i+1
```

In [8]:

```
#split labels into flooded and non flooded folders
i=0
for lpath in labelslist:
    json_data=open(lpath+"\stac.json", "rb")
    jdata = json.load(json_data)
    flood = jdata["properties"]["FLOODING"]
    if(flood=='True'):
        src=lpath+'\stac.json'
        dst=r'E:\sar_flood_proj\Flood_labels\stac'+str(i)+'.json'
        shutil.copy(src,dst)

    else:
        src=lpath+'\stac.json'
        dst=r'E:\sar_flood_proj\Non_flood_labels\stac'+str(i)+'.json'
        shutil.copy(src,dst)
    i=i+1
```

In [20]:

```
#checking number of images copied into flood and non flood folders
Flood_img_list = []
rootdir = r'E:\sar_flood_proj\Flood_imgs'
for file in os.listdir(rootdir):
    if file.endswith(".tif"):
        d = os.path.join(rootdir, file)
        Flood_img_list.append(d)


print("The number of Flooded images are currently =", {len(Flood_img_list)})

Non_flood_img_list = []
labeldir = r'E:\sar_flood_proj\Non_Flood_imgs'
for lfile in os.listdir(labeldir):
    if file.endswith(".tif"):
        d = os.path.join(rootdir, file)
        Non_flood_img_list.append(d)

print("The number of non flood labels are currently =", {len(Non_flood_img_list)})

#checking number of labels copied into flood and non flood folders

Flood_labels_list = []
rootdir = r'E:\sar_flood_proj\Flood_labels'
for file in os.listdir(rootdir):
    if file.endswith(".json"):
        d = os.path.join(rootdir, file)
        Flood_labels_list.append(d)


print("The number of Flooded labels are currently =", {len(Flood_labels_list)})

Non_flood_labels_list = []
labeldir = r'E:\sar_flood_proj\Non_Flood_labels'
for lfile in os.listdir(labeldir):
    if file.endswith(".json"):
        d = os.path.join(rootdir, file)
        Non_flood_labels_list.append(d)


print("The number of non flood labels are currently =", {len(Non_flood_labels_list)})
```

```
The number of Flooded images are currently = {1031}
The number of non flood labels are currently = {2300}
The number of Flooded labels are currently = {1031}
The number of non flood labels are currently = {2300}
```

In [33]:

```
#fli=flood imgs fll=flood labels 80% for training and 20% for testing

fli = int(0.8 * len((Flood_img_list)))
fll = int(0.8 * len((Flood_labels_list)))
nfli = int(0.8 * len((Non_flood_img_list)))
nfll = int(0.8 * len((Non_flood_labels_list)))

print(fli,fll,nfli,nfll)
```

```
824 824 1840 1840
```

In [39]:

```
#copying into training and testing
#we are doing * 9 for good shuffling of flood and non flood images

#train img ds destination
tridst='E:\sar_flood_proj\_train_and_test\_train_imgs'
#train labels ds destination
trldst='E:\sar_flood_proj\_train_and_test\_train_labels'

#test img ds destination
tsidst='E:\sar_flood_proj\_train_and_test\_test_imgs'
#test labels ds destination
tsldst='E:\sar_flood_proj\_train_and_test\_test_labels'


#training data
for i in range(0,92):
    for j in range(i*9,i*9+9):
        shutil.copy(Flood_img_list[j],tridst+'\_fl'+str(j)+'.tif')
        shutil.copy(Flood_labels_list[j],trldst+'\_fl'+str(j)+'.json')
        if(j>824):
            break
    for k in range(i*20,i*20+20):
        shutil.copy(Non_flood_img_list[k],tridst+'\_nfl'+str(k)+'.tif')
        shutil.copy(Non_flood_labels_list[k],trldst+'\_nfl'+str(k)+'.json')

#testing data

#flooded
for i in range(fli+1,len(Flood_img_list)):
    shutil.copy(Flood_img_list[i],tsidst+'\_fl'+str(i)+'.tif')
    shutil.copy(Flood_labels_list[i],tsldst+'\_fl'+str(i)+'.json')
#non flood
for k in range(nfli+1,len(Non_flood_img_list)):
        shutil.copy(Non_flood_img_list[k],tsidst+'\_nfl'+str(k)+'.tif')
        shutil.copy(Non_flood_labels_list[k],tsldst+'\_nfl'+str(k)+'.json')
```

In [40]:

```
# training images = 2666
# testing images = 665
```

In [66]:

```
#understanding image
imglist=[]
rootdir=r"E:\sar_flood_proj\_png_train_and_test\_png_train_imgs"
for file in os.listdir(rootdir):
    if file.endswith(".png"):
        file=str(file)
        file=file[2:]
        d = os.path.join(rootdir, file)
        imglist.append(d)
for i in range(0,9):

    img = cv2.imread(imglist[i])

    # counting the number of pixels
    number_of_white_pix = np.sum(img == 255)
    number_of_black_pix = np.sum(img == 0)

    print('Number of white pixels:', number_of_white_pix)
    print('Number of black pixels:', number_of_black_pix)

    print(img.size)

    dimensions = img.shape

    # height, width, number of channels in image
    height = img.shape[0]
    width = img.shape[1]
    channels = img.shape[2]

    print('Image Dimension    : ',dimensions)
    print('Image Height       : ',height)
    print('Image Width        : ',width)
    print('Number of Channels : ',channels)

    print('-------------------------------------------')
```

```
Number of white pixels: 861
Number of black pixels: 0
859650
Image Dimension    :  (521, 550, 3)
Image Height       :  521
Image Width        :  550
Number of Channels :  3
-------------------------------------------
Number of white pixels: 1734
Number of black pixels: 1182
861213
Image Dimension    :  (521, 551, 3)
Image Height       :  521
Image Width        :  551
Number of Channels :  3
-------------------------------------------
Number of white pixels: 0
Number of black pixels: 0
861213
Image Dimension    :  (521, 551, 3)
Image Height       :  521
Image Width        :  551
```

```
Number of Channels :  3
------------------------------------------
Number of white pixels: 24249
Number of black pixels: 0
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
Number of white pixels: 24624
Number of black pixels: 51
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
Number of white pixels: 16569
Number of black pixels: 6
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
Number of white pixels: 13488
Number of black pixels: 99
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
Number of white pixels: 12120
Number of black pixels: 0
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
Number of white pixels: 11937
Number of black pixels: 180
862866
Image Dimension    :  (522, 551, 3)
Image Height       :  522
Image Width        :  551
Number of Channels :  3
------------------------------------------
```

In [67]:

```
img = cv2.imread(imglist[0])

print('-------------------------------------------')

print(np.array(img))
```

```
-------------------------------------------
[[[36 36 36]
  [48 48 48]
  [44 44 44]
  ...
  [26 26 26]
  [35 35 35]
  [27 27 27]]

 [[28 28 28]
  [46 46 46]
  [46 46 46]
  ...
  [41 41 41]
  [41 41 41]
  [30 30 30]]

 [[22 22 22]
  [33 33 33]
  [43 43 43]
  ...
  [51 51 51]
  [37 37 37]
  [26 26 26]]

 ...

 [[31 31 31]
  [36 36 36]
  [44 44 44]
  ...
  [21 21 21]
  [33 33 33]
  [37 37 37]]

 [[31 31 31]
  [31 31 31]
  [53 53 53]
  ...
  [20 20 20]
  [30 30 30]
  [42 42 42]]

 [[29 29 29]
  [28 28 28]
  [45 45 45]
  ...
  [22 22 22]
  [26 26 26]
  [36 36 36]]]
```

In [77]:

```
def load_data(rootdir):

    data = []
    images = []
    labels = []
    fl=0
    nfl=0
    print("started")
    ilist = [] #images location list
    for file in os.listdir(rootdir):
        if file.endswith(".png"):
            file=str(file)
            file=file[2:]
            d = os.path.join(rootdir, file)
            print(d)
            ilist.append(d)
    random.shuffle(ilist)
    for folder in ilist:
        print("processing ",folder)
        strfolder=str(folder)
        if("_fl" in strfolder):
            label=1
            fl=fl+1
        else:
            label=0
            nfl=nfl+1
        # Open the img
        image = cv2.imread(folder)
        #resizing images
        image.resize(500,500,3)
        # Append the image and its corresponding label to the output
        images.append(image)
        labels.append(label)


    npimages= np.array([np.array(xi) for xi in images])
    nplabels = np.array(labels)
    print("number of flooded images are ",fl)
    print("number of non-flooded images are ",nfl)

    data.append([images, labels])
    return npimages, nplabels
```

In [73]:

```
train_images, train_labels = load_data(r'E:\png_train_test\_train_imgs_png_')
```

```
started
E:\png_train_test\_train_imgs_png_\_fl101.png
E:\png_train_test\_train_imgs_png_\_fl114.png
E:\png_train_test\_train_imgs_png_\_fl136.png
E:\png_train_test\_train_imgs_png_\_fl144.png
E:\png_train_test\_train_imgs_png_\_fl187.png
E:\png_train_test\_train_imgs_png_\_fl195.png
E:\png_train_test\_train_imgs_png_\_fl197.png
E:\png_train_test\_train_imgs_png_\_fl207.png
E:\png_train_test\_train_imgs_png_\_fl209.png
E:\png_train_test\_train_imgs_png_\_fl225.png
E:\png_train_test\_train_imgs_png_\_fl23.png
E:\png_train_test\_train_imgs_png_\_fl243.png
E:\png_train_test\_train_imgs_png_\_fl246.png
E:\png_train_test\_train_imgs_png_\_fl247.png
E:\png_train_test\_train_imgs_png_\_fl248.png
E:\png_train_test\_train_imgs_png_\_fl250.png
E:\png_train_test\_train_imgs_png_\_fl251.png
E:\png_train_test\_train_imgs_png_\_fl252.png
```

In [74]:

```
test_images, test_labels = load_data(r'E:\png_train_test\_test_imgs_png_')
```

```
started
E:\png_train_test\_test_imgs_png_\_fl1000.png
E:\png_train_test\_test_imgs_png_\_fl1001.png
E:\png_train_test\_test_imgs_png_\_fl1002.png
E:\png_train_test\_test_imgs_png_\_fl1003.png
E:\png_train_test\_test_imgs_png_\_fl1004.png
E:\png_train_test\_test_imgs_png_\_fl1005.png
E:\png_train_test\_test_imgs_png_\_fl1006.png
E:\png_train_test\_test_imgs_png_\_fl1007.png
E:\png_train_test\_test_imgs_png_\_fl1008.png
E:\png_train_test\_test_imgs_png_\_fl1009.png
E:\png_train_test\_test_imgs_png_\_fl1010.png
E:\png_train_test\_test_imgs_png_\_fl1011.png
E:\png_train_test\_test_imgs_png_\_fl1012.png
E:\png_train_test\_test_imgs_png_\_fl1013.png
E:\png_train_test\_test_imgs_png_\_fl1014.png
E:\png_train_test\_test_imgs_png_\_fl1015.png
E:\png_train_test\_test_imgs_png_\_fl1016.png
E:\png_train_test\_test_imgs_png_\_fl1017.png
```

In [75]:

```
#Model creation

# Import the Deep Learing modules
import matplotlib.pyplot as plt
import seaborn as sns

import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.utils.vis_utils import plot_model
import pydot

import tensorflow as tf

import cv2
import os

import numpy as np
```

In [43]:

```
#ANN
ann = keras.models.Sequential([
 keras.layers.Flatten(input_shape = (500, 500, 3)),
 keras.layers.Dense(300, activation='relu'),
 keras.layers.Dense(100, activation='relu'),
 keras.layers.Dense(2, activation='sigmoid'),
])
```

In [44]:

```
ann.compile(optimizer='SGD',loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
```

In [45]:

```
ann.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 750000)            0         
                                                                 
 dense (Dense)               (None, 300)               225000300 
                                                                 
 dense_1 (Dense)             (None, 100)               30100     
                                                                 
 dense_2 (Dense)             (None, 2)                 202       
                                                                 
=================================================================
Total params: 225,030,602
Trainable params: 225,030,602
Non-trainable params: 0
_________________________________________________________________
```

In [46]:

```
history = ann.fit(train_images,train_labels, batch_size=28, epochs=10,validation_split=0.2)
```

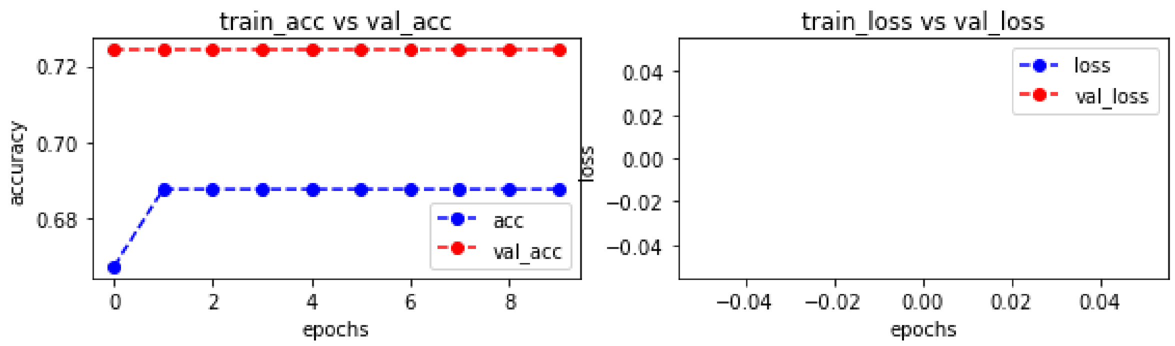
```
Epoch 1/10
55/55 [==============================] - 29s 523ms/step - loss: nan - accura
cy: 0.6673 - val_loss: nan - val_accuracy: 0.7244
Epoch 2/10
55/55 [==============================] - 32s 578ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 3/10
55/55 [==============================] - 25s 460ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 4/10
55/55 [==============================] - 25s 463ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 5/10
55/55 [==============================] - 26s 468ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 6/10
55/55 [==============================] - 25s 458ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 7/10
55/55 [==============================] - 23s 427ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 8/10
55/55 [==============================] - 25s 458ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 9/10
55/55 [==============================] - 25s 460ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
Epoch 10/10
55/55 [==============================] - 25s 454ms/step - loss: nan - accura
cy: 0.6877 - val_loss: nan - val_accuracy: 0.7244
```

In [47]:

```
def plot_accuracy_loss(history):
    fig = plt.figure(figsize=(10,5))
    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'],'bo--', label = "acc")
    plt.plot(history.history['val_accuracy'], 'ro--', label = "val_acc")
    plt.title("train_acc vs val_acc")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()
    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'],'bo--', label = "loss")
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")
    plt.legend()
    plt.show()
```

In [48]:

```
plot_accuracy_loss(history)
```



In [49]:

```
test_loss = ann.evaluate(test_images,test_labels)
```

```
21/21 [==============================] - 3s 139ms/step - loss: nan - accurac
y: 0.6884
```

In [50]:

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (500, 500, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(16, (3, 3), activation = 'relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])
```

In [51]:

```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accur
```

In [52]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 498, 498, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 249, 249, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 247, 247, 16)      4624      
                                                                 
 flatten_1 (Flatten)         (None, 976144)            0         
                                                                 
 dense_3 (Dense)             (None, 128)               124946560 
                                                                 
 dense_4 (Dense)             (None, 2)                 258       
                                                                 
=================================================================
Total params: 124,952,338
Trainable params: 124,952,338
Non-trainable params: 0
_________________________________________________________________
```

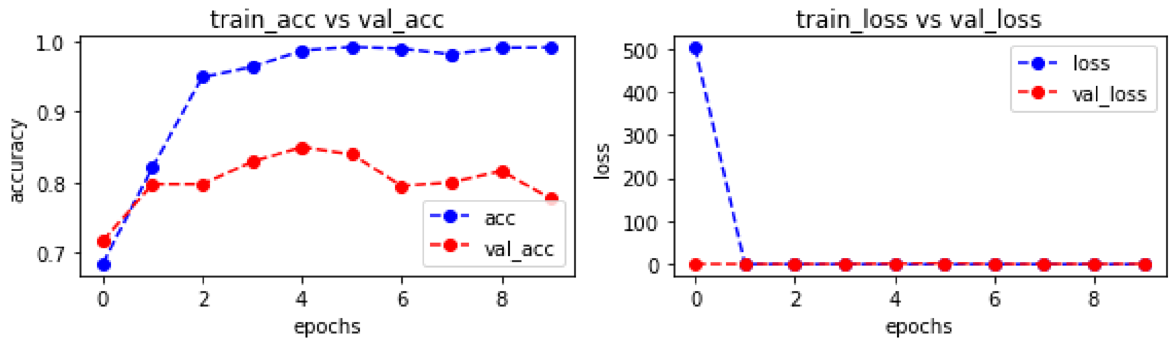
In [53]:

```
history = model.fit(train_images,train_labels, batch_size=28, epochs=10,validation_split=0.
```

```
Epoch 1/10
55/55 [==============================] - 165s 3s/step - loss: 503.2566 - acc
uracy: 0.6844 - val_loss: 0.6490 - val_accuracy: 0.7165
Epoch 2/10
55/55 [==============================] - 157s 3s/step - loss: 0.4005 - accur
acy: 0.8212 - val_loss: 0.5950 - val_accuracy: 0.7979
Epoch 3/10
55/55 [==============================] - 163s 3s/step - loss: 0.2305 - accur
acy: 0.9494 - val_loss: 0.5910 - val_accuracy: 0.7979
Epoch 4/10
55/55 [==============================] - 146s 3s/step - loss: 0.1331 - accur
acy: 0.9638 - val_loss: 0.7380 - val_accuracy: 0.8294
Epoch 5/10
55/55 [==============================] - 147s 3s/step - loss: 0.0514 - accur
acy: 0.9869 - val_loss: 1.4167 - val_accuracy: 0.8504
Epoch 6/10
55/55 [==============================] - 146s 3s/step - loss: 0.0237 - accur
acy: 0.9921 - val_loss: 2.2727 - val_accuracy: 0.8399
Epoch 7/10
55/55 [==============================] - 148s 3s/step - loss: 0.1707 - accur
acy: 0.9895 - val_loss: 0.5574 - val_accuracy: 0.7953
Epoch 8/10
55/55 [==============================] - 146s 3s/step - loss: 0.1008 - accur
acy: 0.9816 - val_loss: 0.6615 - val_accuracy: 0.8005
Epoch 9/10
55/55 [==============================] - 146s 3s/step - loss: 0.0406 - accur
acy: 0.9908 - val_loss: 0.7661 - val_accuracy: 0.8163
Epoch 10/10
55/55 [==============================] - 146s 3s/step - loss: 0.0301 - accur
acy: 0.9915 - val_loss: 0.8913 - val_accuracy: 0.7769
```

In [54]:

```
plot_accuracy_loss(history)
```



In [79]:

```
test_loss = model.evaluate(test_images, test_labels)
```

```
21/21 [==============================] - 15s 685ms/step - loss: 0.6943 - acc
uracy: 0.7292
```

In [80]:

```
# New Layers
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(8, (3, 3), activation = 'relu', input_shape = (500, 500, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(16, (3, 3), activation = 'relu'),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])
```

In [81]:

```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accur
```

In [82]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 498, 498, 8)       224       
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 249, 249, 8)      0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 247, 247, 16)      1168      
                                                                 
 conv2d_4 (Conv2D)           (None, 245, 245, 32)      4640      
                                                                 
 conv2d_5 (Conv2D)           (None, 243, 243, 64)      18496     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 121, 121, 64)     0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 937024)            0         
                                                                 
 dense_5 (Dense)             (None, 128)               119939200 
                                                                 
 dense_6 (Dense)             (None, 2)                 258       
                                                                 
=================================================================
Total params: 119,963,986
Trainable params: 119,963,986
Non-trainable params: 0
_________________________________________________________________
```

In [83]:

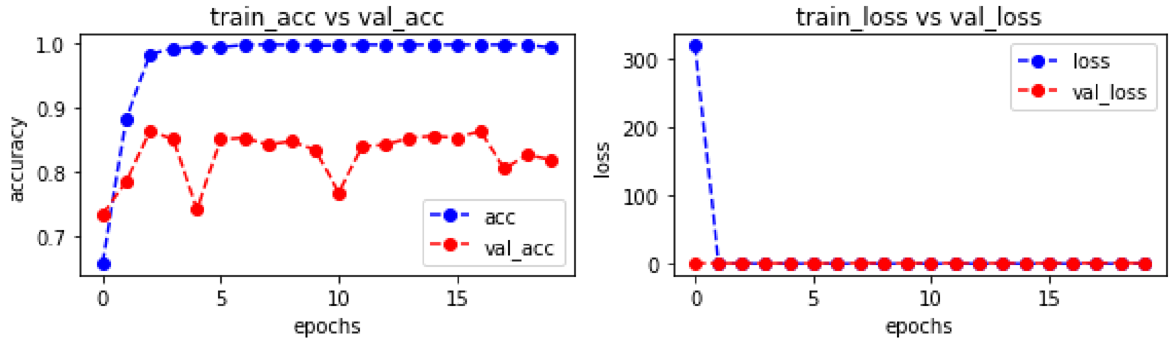
```
history = model.fit(train_images,train_labels, batch_size=28, epochs=20,validation_split=0.
```

```
Epoch 1/20
55/55 [==============================] - 239s 4s/step - loss: 319.2124 - acc
uracy: 0.6588 - val_loss: 0.5051 - val_accuracy: 0.7349
Epoch 2/20
55/55 [==============================] - 231s 4s/step - loss: 0.3083 - accur
acy: 0.8817 - val_loss: 0.3988 - val_accuracy: 0.7848
Epoch 3/20
55/55 [==============================] - 222s 4s/step - loss: 0.0864 - accur
acy: 0.9822 - val_loss: 0.4004 - val_accuracy: 0.8635
Epoch 4/20
55/55 [==============================] - 258s 5s/step - loss: 0.0398 - accur
acy: 0.9915 - val_loss: 0.3545 - val_accuracy: 0.8504
Epoch 5/20
55/55 [==============================] - 210s 4s/step - loss: 0.0587 - accur
acy: 0.9934 - val_loss: 0.5597 - val_accuracy: 0.7428
Epoch 6/20
55/55 [==============================] - 216s 4s/step - loss: 0.0513 - accur
acy: 0.9928 - val_loss: 0.3406 - val_accuracy: 0.8504
Epoch 7/20
55/55 [==============================] - 251s 5s/step - loss: 0.0254 - accur
acy: 0.9967 - val_loss: 0.5348 - val_accuracy: 0.8530
Epoch 8/20
55/55 [==============================] - 223s 4s/step - loss: 0.0162 - accur
acy: 0.9967 - val_loss: 0.6840 - val_accuracy: 0.8425
Epoch 9/20
55/55 [==============================] - 225s 4s/step - loss: 0.0133 - accur
acy: 0.9967 - val_loss: 0.6205 - val_accuracy: 0.8478
Epoch 10/20
55/55 [==============================] - 216s 4s/step - loss: 0.0275 - accur
acy: 0.9954 - val_loss: 0.3898 - val_accuracy: 0.8346
Epoch 11/20
55/55 [==============================] - 201s 4s/step - loss: 0.0177 - accur
acy: 0.9961 - val_loss: 0.8157 - val_accuracy: 0.7690
Epoch 12/20
55/55 [==============================] - 205s 4s/step - loss: 0.0157 - accur
acy: 0.9967 - val_loss: 0.5869 - val_accuracy: 0.8399
Epoch 13/20
55/55 [==============================] - 206s 4s/step - loss: 0.0161 - accur
acy: 0.9967 - val_loss: 0.5945 - val_accuracy: 0.8425
Epoch 14/20
55/55 [==============================] - 200s 4s/step - loss: 0.0134 - accur
acy: 0.9967 - val_loss: 0.5605 - val_accuracy: 0.8530
Epoch 15/20
55/55 [==============================] - 195s 4s/step - loss: 0.0149 - accur
acy: 0.9967 - val_loss: 0.5732 - val_accuracy: 0.8556
Epoch 16/20
55/55 [==============================] - 194s 4s/step - loss: 0.0131 - accur
acy: 0.9967 - val_loss: 0.5721 - val_accuracy: 0.8530
Epoch 17/20
55/55 [==============================] - 194s 4s/step - loss: 0.0140 - accur
acy: 0.9967 - val_loss: 0.5754 - val_accuracy: 0.8635
Epoch 18/20
55/55 [==============================] - 194s 4s/step - loss: 0.0161 - accur
acy: 0.9967 - val_loss: 0.9842 - val_accuracy: 0.8058
Epoch 19/20
55/55 [==============================] - 197s 4s/step - loss: 0.0103 - accur
acy: 0.9967 - val_loss: 0.7934 - val_accuracy: 0.8268
```

```
Epoch 20/20
55/55 [==============================] - 196s 4s/step - loss: 0.0488 - accur
acy: 0.9921 - val_loss: 0.6178 - val_accuracy: 0.8189
```

In [84]:

```
plot_accuracy_loss(history)
```



In [87]:

```
test_loss = model.evaluate(test_images, test_labels)
```

```
21/21 [==============================] - 14s 658ms/step - loss: 1.0043 - acc
uracy: 0.5900
```

In [90]:

```
history = model.fit(train_images,train_labels, batch_size=20, epochs=17,validation_split=0.
```

```
Epoch 1/17
77/77 [==============================] - 203s 3s/step - loss: 0.0154 - accur
acy: 0.9967 - val_loss: 0.8414 - val_accuracy: 0.8241
Epoch 2/17
77/77 [==============================] - 202s 3s/step - loss: 0.0173 - accur
acy: 0.9967 - val_loss: 0.8750 - val_accuracy: 0.8215
Epoch 3/17
77/77 [==============================] - 205s 3s/step - loss: 0.0128 - accur
acy: 0.9967 - val_loss: 0.8917 - val_accuracy: 0.8189
Epoch 4/17
77/77 [==============================] - 205s 3s/step - loss: 0.0139 - accur
acy: 0.9967 - val_loss: 0.9568 - val_accuracy: 0.8084
Epoch 5/17
77/77 [==============================] - 206s 3s/step - loss: 0.0138 - accur
acy: 0.9967 - val_loss: 1.0397 - val_accuracy: 0.8058
Epoch 6/17
77/77 [==============================] - 205s 3s/step - loss: 0.0132 - accur
acy: 0.9967 - val_loss: 1.0618 - val_accuracy: 0.8058
Epoch 7/17
77/77 [==============================] - 205s 3s/step - loss: 0.0135 - accur
acy: 0.9967 - val_loss: 1.1398 - val_accuracy: 0.8031
Epoch 8/17
77/77 [==============================] - 205s 3s/step - loss: 0.0175 - accur
acy: 0.9967 - val_loss: 1.1476 - val_accuracy: 0.8005
Epoch 9/17
77/77 [==============================] - 204s 3s/step - loss: 0.0136 - accur
acy: 0.9967 - val_loss: 0.7969 - val_accuracy: 0.8320
Epoch 10/17
77/77 [==============================] - 205s 3s/step - loss: 0.0123 - accur
acy: 0.9967 - val_loss: 0.9882 - val_accuracy: 0.8163
Epoch 11/17
77/77 [==============================] - 205s 3s/step - loss: 0.0133 - accur
acy: 0.9967 - val_loss: 1.0114 - val_accuracy: 0.8189
Epoch 12/17
77/77 [==============================] - 204s 3s/step - loss: 0.0125 - accur
acy: 0.9967 - val_loss: 0.9587 - val_accuracy: 0.8294
Epoch 13/17
77/77 [==============================] - 204s 3s/step - loss: 0.0119 - accur
acy: 0.9967 - val_loss: 1.1040 - val_accuracy: 0.8163
Epoch 14/17
77/77 [==============================] - 206s 3s/step - loss: 0.0126 - accur
acy: 0.9967 - val_loss: 1.1476 - val_accuracy: 0.8241
Epoch 15/17
77/77 [==============================] - 204s 3s/step - loss: 0.0126 - accur
acy: 0.9967 - val_loss: 1.2542 - val_accuracy: 0.8110
Epoch 16/17
77/77 [==============================] - 204s 3s/step - loss: 0.0126 - accur
acy: 0.9967 - val_loss: 1.3064 - val_accuracy: 0.8110
Epoch 17/17
77/77 [==============================] - 204s 3s/step - loss: 0.0119 - accur
acy: 0.9967 - val_loss: 1.4038 - val_accuracy: 0.8031
```

In [91]:

```
test_loss = model.evaluate(test_images, test_labels)
```

```
21/21 [==============================] - 19s 909ms/step - loss: 1.7319 - acc
uracy: 0.5855
```

In [ ]:

```

```