In [1]:
```
import numpy as np
import pandas as pd
import h5py
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split, KFold
print(tf.__version__)
```

```
2.18.0
```

In [2]:
```
data_path = "TCIR-ALL_2017.h5"
data_info = pd.read_hdf(data_path, key="info", mode='r')
with h5py.File(data_path, 'r') as hf:
    data_matrix = hf['matrix'][:]
```

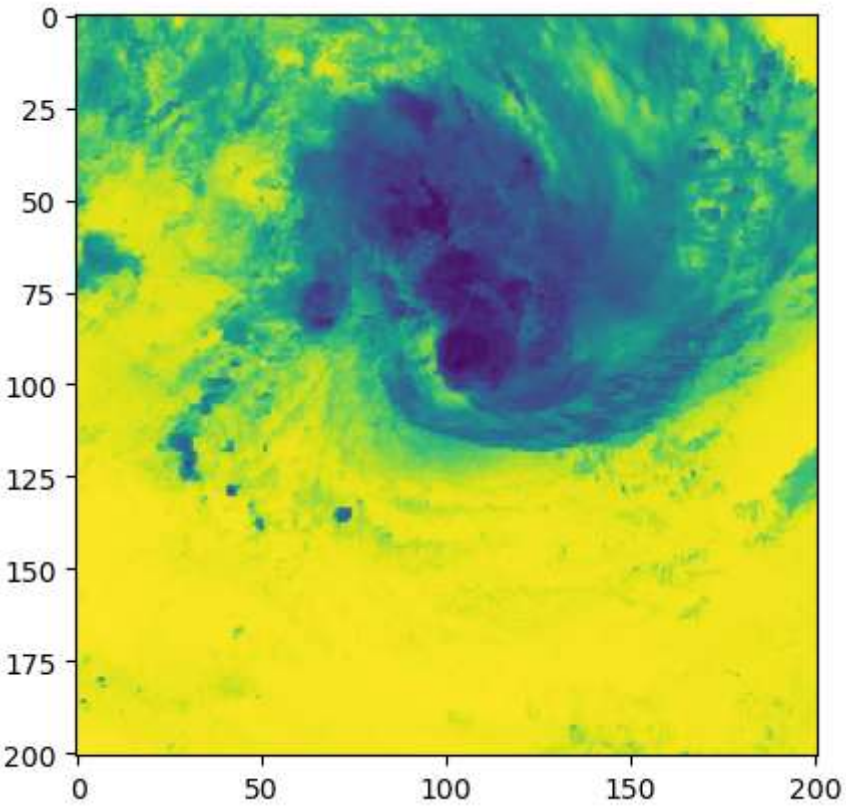
In [3]:
```
print("Min Intensity ",data_info.Vmax.min())
print("Min Intensity ",data_info.Vmax.max())
print("Min Intensity ",round(data_info.Vmax.mean(),2))
```

```
Min Intensity  15.0
Min Intensity  155.0
Min Intensity  48.14
```

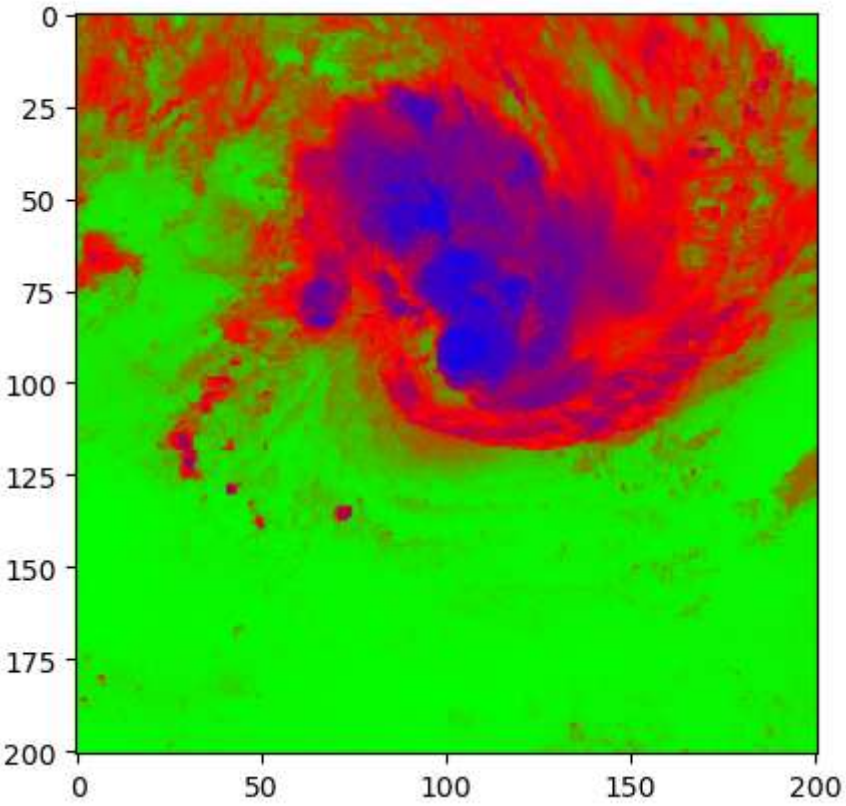
In [4]:
```
np.shape(data_matrix)
```

Out[4]: (4580, 201, 201, 4)

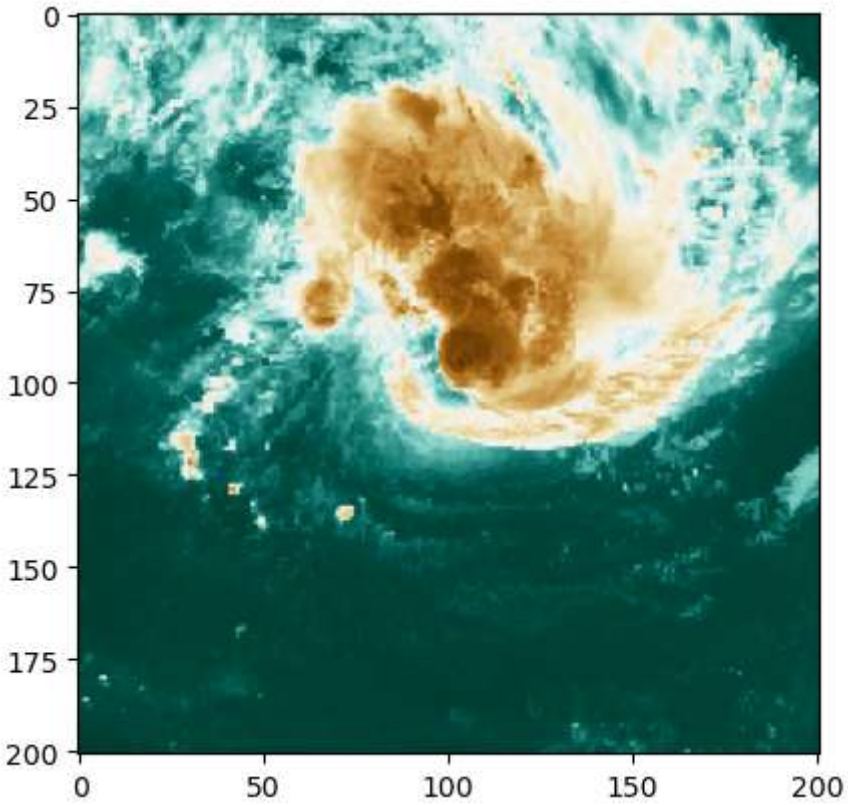
In [5]:
```
img = data_matrix[4000,:,:,0].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img)
```



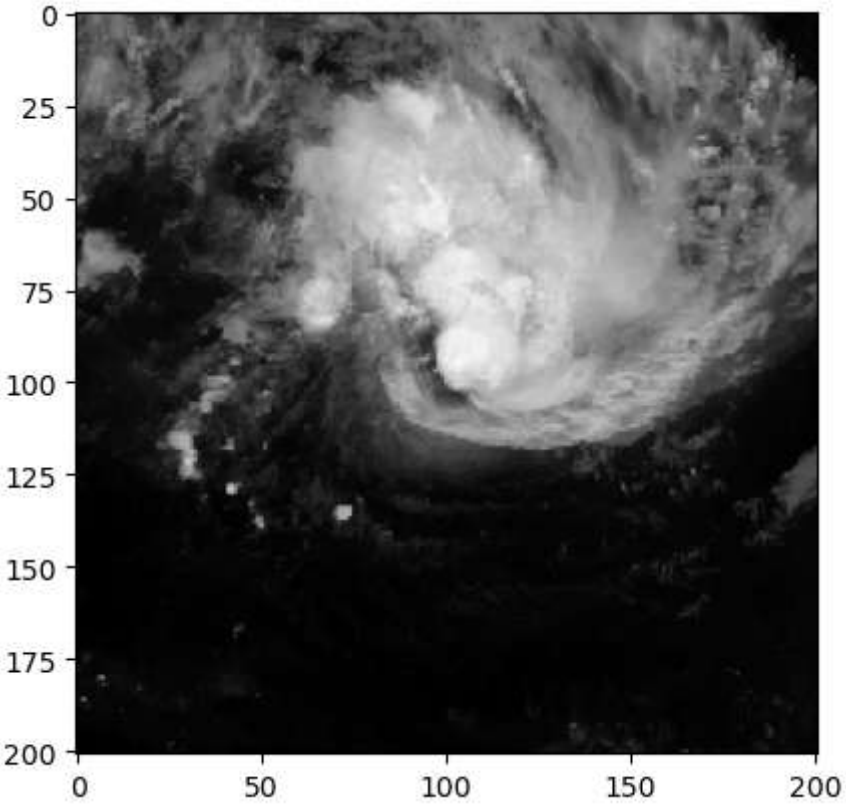
In [6]:
```
img = data_matrix[4000,:,:,0].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img,plt.cm.brg)
```



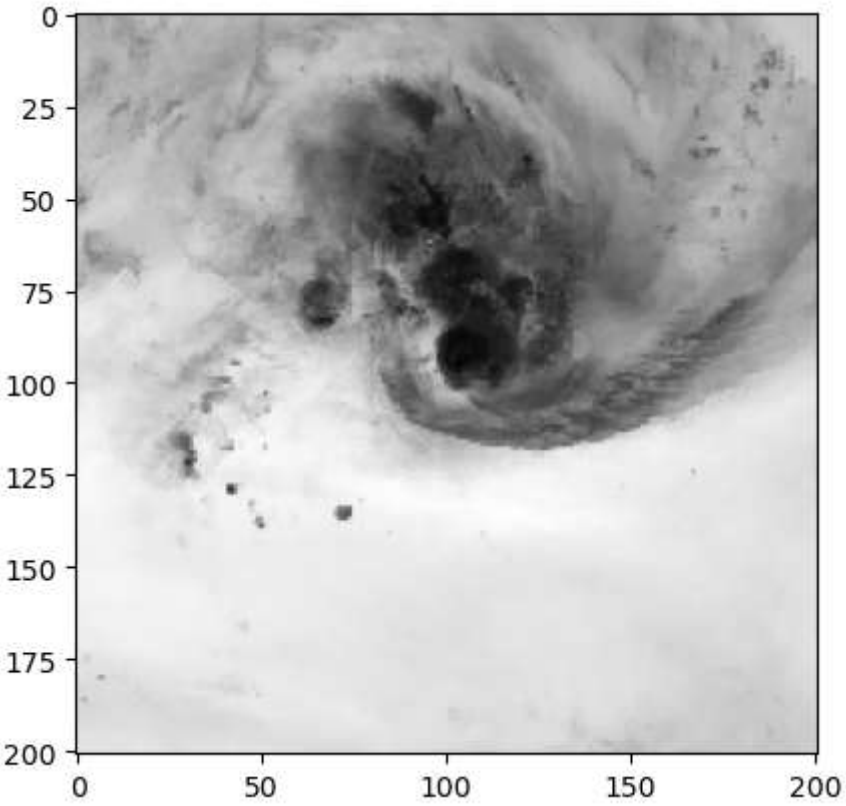
In [7]:
```
img = data_matrix[4000,:,:,0].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img,plt.cm.BrBG)
```



In [8]:
```
img = data_matrix[4000,:,:,0].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img,plt.cm.binary)
```



In [9]:
```
img = data_matrix[4000,:,:,1].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img, plt.cm.gray)
```

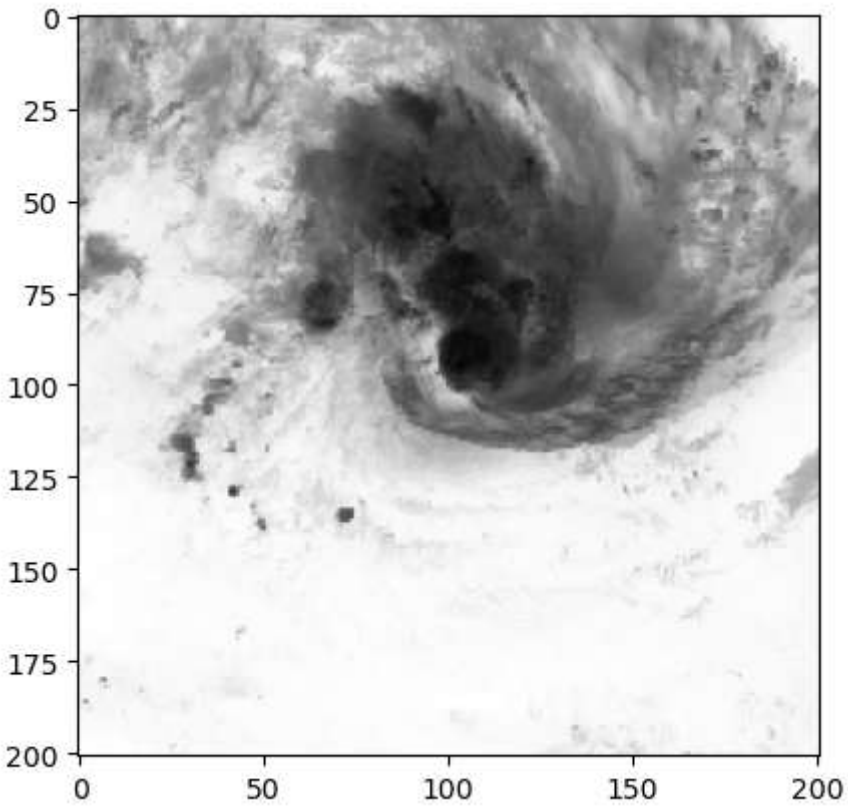


In [10]:
```
data_info = data_info.assign(time=pd.to_datetime(data_info.time, format=r'%Y%m%d%H'
```

In [11]:
```
## keep only IR and PMW
X_irpmw = data_matrix[:,:,:,0::3]
y = data_info['Vmax'].values[:,np.newaxis]
```

In [12]:
```
X_irpmw[np.isnan(X_irpmw)] = 0
X_irpmw[X_irpmw > 1000] = 0
```

In [13]:
```
# X_std = tf.image.per_image_standardization(X_irpmw)
img = data_matrix[4000,:,:,0].copy()
fig, ax = plt.subplots()
pos = ax.imshow(img, plt.cm.gray)
```



In [14]:
```
class Preprocessing(keras.layers.Layer):
    def __init__(self):
        super(Preprocessing, self).__init__()
    def call(self, inputs, training=None):
        if training:
            inputs = tf.image.rot90(inputs, k=np.random.randint(4))
        return tf.image.central_crop(inputs, 0.5)
```

Alexnet CNN

In [16]:
```
def train_val_model(train_x,train_y,val_x,val_y, n_epochs, batch_size):
    reg_param = 1e-5
```

```
    train_X = tf.convert_to_tensor(train_x)
    train_Y = tf.convert_to_tensor(train_y)

    val_X = tf.convert_to_tensor(val_x)
    val_Y = tf.convert_to_tensor(val_y)

    weights_initializer = keras.initializers.GlorotUniform()


    model = keras.models.Sequential([
        Preprocessing(),
        keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=4,padding='val
        keras.layers.MaxPool2D(pool_size=(3, 3),strides=2),
        keras.layers.Conv2D(filters=256, kernel_size=(5,5), padding='same', activat
        keras.layers.MaxPool2D(pool_size=(3, 3),strides=2),
        keras.layers.Conv2D(filters=384, kernel_size=(3,3), padding='same', activat
        keras.layers.Conv2D(filters=384, kernel_size=(3,3), padding='same', activat
        keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding='same', activat
        keras.layers.MaxPool2D(pool_size=(3, 3),strides=2),
        keras.layers.Flatten(),
        keras.layers.Dense(4096, activation='relu'),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(4096, activation='relu'),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(1, activation='relu'),
    ])

    #Compiling the model
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=5e-4, beta_1=0.99,
                  loss='mean_squared_error',
                  metrics=['mean_squared_error'],
                 )

    #Training the network
    history = model.fit(train_X,train_Y,
         epochs=n_epochs,
         batch_size=batch_size,
         verbose=1
        )

    val_score = model.evaluate(val_X, val_Y)
    print("Val Score: ",val_score)
    return history,val_score,model
```

In [17]:
```
model_history=[]
val_scores=[]
n_epochs=20
batch_size=256
train_x, val_x, train_y, val_y = train_test_split(X_irpmw, y, random_state = 101, t
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, random_state
history,val_score,model = train_val_model(train_x,train_y,val_x,val_y, n_epochs, ba
model_history.append(history)
val_scores.append(val_score)
```

```
Epoch 1/20
```

```
C:\Users\shrey\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.
py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the first laye
r in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
WARNING:tensorflow:From C:\Users\shrey\anaconda3\Lib\site-packages\keras\src\backend
\tensorflow\core.py:216: The name tf.placeholder is deprecated. Please use tf.compa
t.v1.placeholder instead.

15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 761ms/step - loss: 94932.2578 - mean_squared_error: 9
4932.2578
Epoch 2/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 759ms/step - loss: 885.1139 - mean_squared_error: 88
5.1139
Epoch 3/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 750ms/step - loss: 842.9739 - mean_squared_error: 84
2.9739
Epoch 4/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 740ms/step - loss: 784.3317 - mean_squared_error: 78
4.3317
Epoch 5/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 763ms/step - loss: 793.9986 - mean_squared_error: 79
3.9986
Epoch 6/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 757ms/step - loss: 760.4693 - mean_squared_error: 76
0.4693
Epoch 7/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 743ms/step - loss: 717.9434 - mean_squared_error: 71
7.9434
Epoch 8/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 767ms/step - loss: 712.7525 - mean_squared_error: 71
2.7525
Epoch 9/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 786ms/step - loss: 679.1050 - mean_squared_error: 67
9.1050
Epoch 10/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 783ms/step - loss: 592.2773 - mean_squared_error: 59
2.2773
Epoch 11/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 786ms/step - loss: 561.6824 - mean_squared_error: 56
1.6824
Epoch 12/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 808ms/step - loss: 469.5116 - mean_squared_error: 46
9.5116
Epoch 13/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 814ms/step - loss: 454.0175 - mean_squared_error: 45
4.0175
Epoch 14/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 13s 829ms/step - loss: 437.2028 - mean_squared_error: 43
7.2028
Epoch 15/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 808ms/step - loss: 406.8280 - mean_squared_error: 40
6.8280
Epoch 16/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 810ms/step - loss: 383.6246 - mean_squared_error: 38
3.6246
Epoch 17/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 13s 826ms/step - loss: 357.3449 - mean_squared_error: 35
7.3449
Epoch 18/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 817ms/step - loss: 351.3412 - mean_squared_error: 35
```

```
1.3412
Epoch 19/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 810ms/step - loss: 324.3003 - mean_squared_error: 32
4.3003
Epoch 20/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 805ms/step - loss: 316.1648 - mean_squared_error: 31
6.1648
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 410.5630 - mean_squared_error: 410.5
630
Val Score:  [380.9270935058594, 380.9270935058594]
```

In [18]:
```
y_pred = model.predict(test_x)
print('Testing...')
score = model.evaluate(test_x,test_y,
                       batch_size=16, verbose=1)

print('Test accuracy:', score[1])
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step
Testing...
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 338.6914 - mean_squared_error: 338.6
914
Test accuracy: 336.8890686035156
```

In [19]:
```
abcd = []
for x in y_pred:
    abcd.append(int(x))
```

```
C:\Users\shrey\AppData\Local\Temp\ipykernel_13688\2544756279.py:3: DeprecationWarnin
g: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before performing this o
peration. (Deprecated NumPy 1.25.)
  abcd.append(int(x))
```

In [20]:
```
cate = []
for x in abcd:
    if x <=33:
        cate.append('Tropical Depression')
    elif x>33 and x<=63:
        cate.append('Tropical Storm')
    elif x>63 and x<=129:
        cate.append('Typhoon')
    elif x>129:
        cate.append('Super Typhoon')
```

In [21]:
```
cate_dataset = list(zip(abcd,cate))
```

In [22]:
```
df = pd.DataFrame(cate_dataset,columns=['Intensity','Category',])
```

In [23]:
```
df
```

Out[23]:

	Intensity	Category
0	52	Tropical Storm
1	35	Tropical Storm
2	27	Tropical Depression
3	65	Typhoon
4	33	Tropical Depression
...	...	...
408	27	Tropical Depression
409	32	Tropical Depression
410	31	Tropical Depression
411	28	Tropical Depression
412	112	Typhoon

413 rows × 2 columns

In [ ]:

Deep CNN

In [29]:
```
def train_val_model(train_x,train_y,val_x,val_y, n_epochs, batch_size):
    reg_param = 1e-5

    train_X = tf.convert_to_tensor(train_x)
    train_Y = tf.convert_to_tensor(train_y)

    val_X = tf.convert_to_tensor(val_x)
    val_Y = tf.convert_to_tensor(val_y)

    weights_initializer = keras.initializers.GlorotUniform()


    model = keras.models.Sequential([
        Preprocessing(),
        keras.layers.Conv2D(filters=16, kernel_size=4, strides=2, padding='valid',
        keras.layers.Conv2D(filters=32, kernel_size=3, strides=2, padding='valid',
        keras.layers.Conv2D(filters=64, kernel_size=3, strides=2, padding='valid',
        keras.layers.Conv2D(filters=128, kernel_size=3, strides=2, padding='valid',
        keras.layers.Flatten(),
        keras.layers.Dense(256, activation='relu', kernel_initializer = weights_ini
        keras.layers.Dense(128, activation='relu', kernel_initializer = weights_ini
        keras.layers.Dense(1, activation='relu', kernel_initializer = weights_initi
    ])
    #Compiling the model
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=5e-4, beta_1=0.99,
```

```
                loss='mean_squared_error',
                metrics=['mean_squared_error'],
               )

    #Training the network
    history = model.fit(train_X,train_Y,
         epochs=n_epochs,
         batch_size=batch_size,
         verbose=1
        )

    val_score = model.evaluate(val_X, val_Y)
    print("Val Score: ",val_score)
    return history,val_score,model
```

In [31]:

```
model_history=[]
val_scores=[]
n_epochs=10
batch_size=64
train_x, val_x, train_y, val_y = train_test_split(X_irpmw, y, random_state = 101, t
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, random_state
history,val_score,model = train_val_model(train_x,train_y,val_x,val_y, n_epochs, ba
model_history.append(history)
val_scores.append(val_score)
```

```
Epoch 1/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - loss: 1505.6150 - mean_squared_error: 150
5.6072
Epoch 2/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 824.8979 - mean_squared_error: 824.8
911
Epoch 3/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 781.0808 - mean_squared_error: 781.0
747
Epoch 4/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 648.4774 - mean_squared_error: 648.4
716
Epoch 5/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 452.9676 - mean_squared_error: 452.9
619
Epoch 6/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 391.4334 - mean_squared_error: 391.4
276
Epoch 7/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 380.8996 - mean_squared_error: 380.8
937
Epoch 8/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 351.0440 - mean_squared_error: 351.0
381
Epoch 9/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 332.1414 - mean_squared_error: 332.1
355
Epoch 10/10
58/58 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 331.3477 - mean_squared_error: 331.3
418
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 329.7191 - mean_squared_error: 329.71
32
Val Score:  [334.5224609375, 334.5164794921875]
```

In [33]:
```
y_pred = model.predict(test_x)
print('Testing...')
score = model.evaluate(test_x,test_y,
                       batch_size=16, verbose=1)
print('Test accuracy:', score[1])
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Testing...
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 297.7839 - mean_squared_error: 297.77
80
Test accuracy: 297.42742919921875
```

In [35]:
```
abcd = []
for x in y_pred:
    abcd.append(int(x))
```

```
C:\Users\shrey\AppData\Local\Temp\ipykernel_13688\2544756279.py:3: DeprecationWarnin
g: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before performing this o
peration. (Deprecated NumPy 1.25.)
  abcd.append(int(x))
```

In [36]:
```
cate = []
for x in abcd:
    if x <=33:
        cate.append('Tropical Depression')
    elif x>33 and x<=63:
        cate.append('Tropical Storm')
    elif x>63 and x<=129:
        cate.append('Typhoon')
    elif x>129:
        cate.append('Super Typhoon')
```

In [37]:
```
cate_dataset = list(zip(abcd,cate))
```

In [38]:
```
df = pd.DataFrame(cate_dataset,columns=['Intensity','Category',])
```

In [39]:
```
df
```

Out[39]:

	Intensity	Category
0	58	Tropical Storm
1	32	Tropical Depression
2	30	Tropical Depression
3	81	Typhoon
4	39	Tropical Storm
...	...	...
408	38	Tropical Storm
409	37	Tropical Storm
410	35	Tropical Storm
411	34	Tropical Storm
412	91	Typhoon

413 rows × 2 columns

Kfold

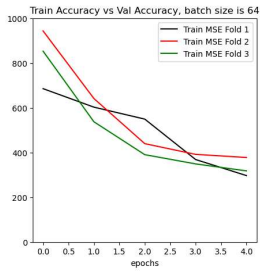
In [41]:
```
n_epochs=5
batch_size=32
model_history = []  #save the model history in a list after fitting so that we can
val_scores=[]
kf = KFold(n_splits=3)

i=0
for train_index, test_index in kf.split(X_irpmw):
    print("Training on Fold: ",i+1)
```

```
    i+=1
    train_x, val_x = X_irpmw[train_index], X_irpmw[test_index]
    train_y, val_y = y[train_index], y[test_index]
    history,val_score,model = train_val_model(train_x,train_y,val_x,val_y, n_epochs
    model_history.append(history)
    val_scores.append(val_score)
    print("=======*12, end="\n\n\n")
```

```
Training on Fold:  1
Epoch 1/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 774.5449 - mean_squared_error: 774.5
374
Epoch 2/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 605.7460 - mean_squared_error: 605.7
405
Epoch 3/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 617.1497 - mean_squared_error: 617.1
449
Epoch 4/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 380.6368 - mean_squared_error: 380.6
320
Epoch 5/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 285.9716 - mean_squared_error: 285.9
668
48/48 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 520.3771 - mean_squared_error: 520.37
24
Val Score:  [563.3367309570312, 563.3320922851562]
================================================================================


Training on Fold:  2
Epoch 1/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 1151.7588 - mean_squared_error: 115
1.7515
Epoch 2/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 745.8410 - mean_squared_error: 745.8
350
Epoch 3/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 456.0321 - mean_squared_error: 456.0
267
Epoch 4/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 385.3188 - mean_squared_error: 385.3
131
Epoch 5/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 365.2140 - mean_squared_error: 365.2
083
48/48 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 254.7952 - mean_squared_error: 254.78
94
Val Score:  [295.8565673828125, 295.850830078125]
================================================================================


Training on Fold:  3
Epoch 1/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - loss: 1100.8359 - mean_squared_error: 110
0.8282
Epoch 2/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 602.2726 - mean_squared_error: 602.2
661
Epoch 3/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 403.3742 - mean_squared_error: 403.3
680
Epoch 4/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 362.3805 - mean_squared_error: 362.3
```

```
741
Epoch 5/5
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 320.8531 - mean_squared_error: 320.8
467
48/48 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 415.1064 - mean_squared_error: 415.10
00
Val Score:  [396.39788818359375, 396.39154052734375]
================================================================================
```

In [42]:
```
plt.figure(figsize=(5,5))
plt.title('Train Accuracy vs Val Accuracy, batch size is 64')
colors=['black','red','green']
for i in range(3):
    plt.plot(model_history[i].history['mean_squared_error'], label='Train MSE Fold
#     plt.plot(model_history[i].history['val_mean_squared_error'], label='Val MSE F

plt.legend(loc='upper right')
plt.xlabel('epochs')
plt.ylim(0,1000)
plt.text(20,800,"MSE on validation sets: "+str([int(v) for v,v2 in val_scores]))
plt.text(20,700,"mean MSE on validation set: "+str(int(np.mean(val_scores,axis=0)[0
plt.show()
```



MSE on validation sets: [563, 295, 396]

mean MSE on validation set: 418

In [ ]: