In this project, we will perform a multi-class classification of medical images of different body parts using 2 deep learning models. First, we will use the **dense neural networks or multi layer perceptron (MLP) model with tensorflow** and analyse how accurately it can classify the images. Later, we will use a simplified **Convolutional Neural Networks (CNN) model with tensorflow** to see if that improves the classification metrics. This dataset has been collected from Kaggle.

In order to do that, let's import all the relevant libraries, i.e, numpy and pandas first.

In [1]:

```
import numpy as np
import pandas as pd
```

Exploratory data analysis and visualization.

Now we need to load the data and do little bit of investigation. This dataset has 5 categories of medical data corresponding to different body parts.

1. Hand
2. Breast
3. Head
4. Abdomen
5. Chest

We will load the images and get a glimpse of some of these images:

In [2]:

```
import glob
data_path = 'med_dataset/*'  #dataset path

data_dirpaths = []
for dir in glob.glob(data_path):
    data_dirpaths.append(dir) #appending each category dataset path
print(data_dirpaths)
```

```
['med_dataset/Hand', 'med_dataset/BreastMRI', 'med_dataset/HeadCT', 'med_datase
t/AbdomenCT', 'med_dataset/CXR']
```

So there are 5 directories containing the images for each category. Now let's go to each cateogory and collect their filenames and category and write into a pandas dataframe

In [3]:

```
import os
image_files = [] # list to store image file paths
category = [] # List to store image categories
cat_name = ['hand', 'breast', 'head', 'abdomen', 'chest']
for i, data_dirpath in enumerate(data_dirpaths):
    data_dirpath += '/*.jpeg'
    print(data_dirpath)
    image_files_cat = sorted(glob.glob(data_dirpath))
    cat_len = len(image_files_cat)
    image_files += image_files_cat
    category += [cat_name[i]]*cat_len

print(f"Number of image files: {len(image_files)}")
print(f" Length of category items : {len(category)}")

# Write everything to a pandas dicitonary
dict = {'image_file_paths' : image_files, 'category': category}
df = pd.DataFrame(dict)
```

```
med_dataset/Hand/*.jpeg
med_dataset/BreastMRI/*.jpeg
med_dataset/HeadCT/*.jpeg
med_dataset/AbdomenCT/*.jpeg
med_dataset/CXR/*.jpeg
Number of image files: 48954
 Length of category items : 48954
```

In [4]:

```
df.head()
```

Out[4]:

	image_file_paths	category
0	med_dataset/Hand/000000.jpeg	hand
1	med_dataset/Hand/000001.jpeg	hand
2	med_dataset/Hand/000002.jpeg	hand
3	med_dataset/Hand/000003.jpeg	hand
4	med_dataset/Hand/000004.jpeg	hand

In [5]:

```
df.describe()
```

Out[5]:

	image_file_paths	category
count	48954	48954
unique	48954	5
top	med_dataset/Hand/000000.jpeg	hand

freq	1	10000

In [6]:

```
df['category'].value_counts()
```

Out[6]:

```
category
hand        10000
head        10000
abdomen     10000
chest       10000
breast       8954
Name: count, dtype: int64
```

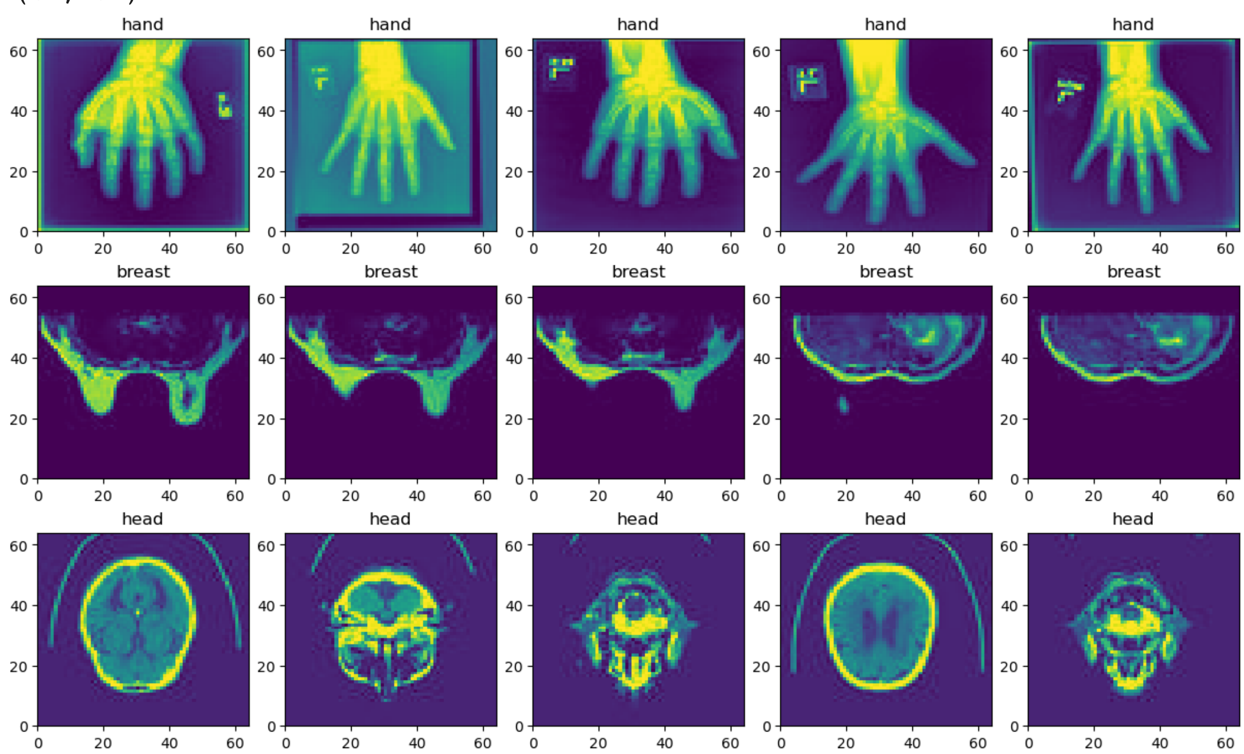
As we can see there are 10000 images in each category except the breast category which has 8954 images in it. So this is a well balanced dataset to conduct the classification task.

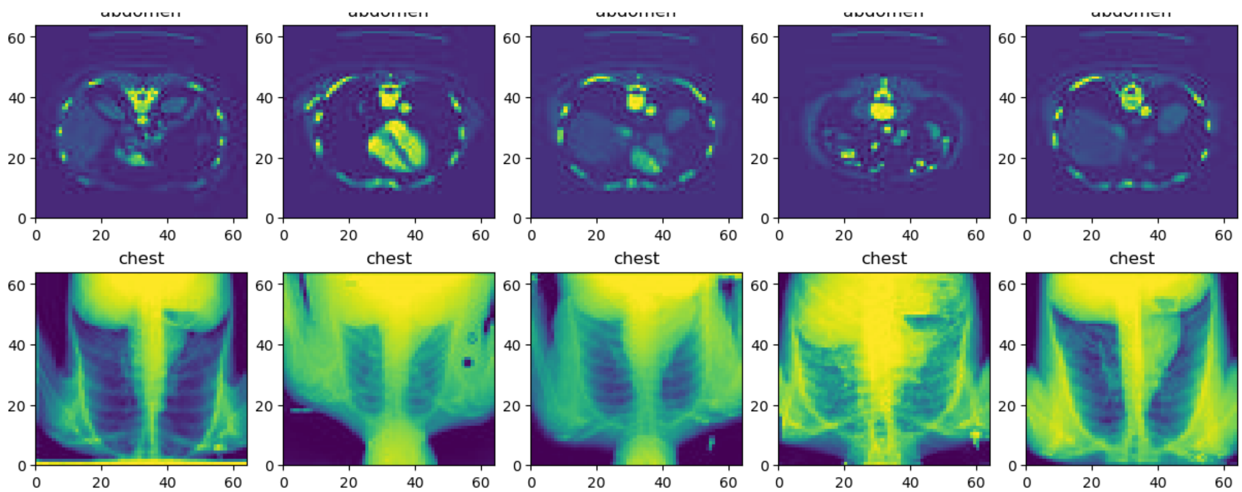
Let's try to read some of the images and visualize them using the matplotlib library.

In [7]:

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

fig, axs = plt.subplots(5,5, constrained_layout=True, figsize = (12,12))
for i in range(5):
    cat = cat_name[i]
    read_files = list(df[df['category'] == cat]['image_file_paths'][:5])
    for j in range(5):
        img = mpimg.imread(read_files[j])
        #print(img.shape)
        axs[i,j].pcolormesh(img)
        axs[i,j].set_title(cat)
print(img.shape)
plt.show()
```

```
(64, 64)
```





Train, validation and test splitting

The plot shows the different categories of medical images. Each image is 60 by 60 pixel == 3600 features when the 2D array is flattened.

The hand, abdomen, chest and breast has more or less same features while the head images vary a lot. Before modelling the data, we need to shuffle the data and split it into train, validation and test set.

In [8]:
```
#shuffling and returning all the data
df_new = df.sample(frac = 1)

#Let's convert the categorical values into numerical values
df_new['category'].replace(cat_name, [0, 1, 2, 3, 4], inplace=True)
df_new.head()
```

Out[8]:

	image_file_paths	category
35042	med_dataset/AbdomenCT/006088.jpeg	3
5085	med_dataset/Hand/005085.jpeg	0
23282	med_dataset/HeadCT/004328.jpeg	2
45424	med_dataset/CXR/006470.jpeg	4
33940	med_dataset/AbdomenCT/004986.jpeg	3

In [9]:
```
from sklearn.model_selection import train_test_split

# Split into train and test first
data_train, data_test = train_test_split(df_new, test_size = 0.2, random_stat

# Split the test into test and validate again
data_val, data_test = train_test_split(data_test, test_size = 0.5, random_sta
print(f" Train data shape : {data_train.shape}, Test data shape : {data_test.
```

```
 Train data shape : (39163, 2), Test data shape : (4896, 2), Validation data sh
ape: (4895, 2)
```

Now before modelling, we need to load the data from all the image into a numpy array

In [10]:

```
X_train = np.zeros((data_train.shape[0], 64, 64))
y_train = np.array(data_train['category']).reshape(-1, 1)

print(y_train.shape)

X_val = np.zeros((data_val.shape[0], 64, 64))
y_val = np.array(data_val['category']).reshape(-1, 1)

X_test = np.zeros((data_test.shape[0], 64, 64))
y_test = np.array(data_test['category']).reshape(-1, 1)
```

```
(39163, 1)
```

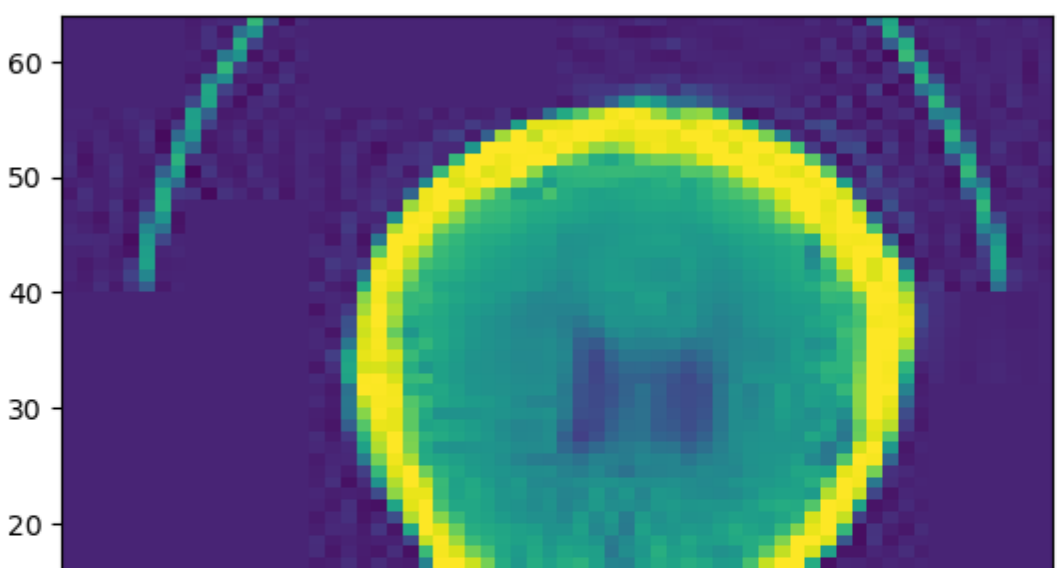
In [11]:

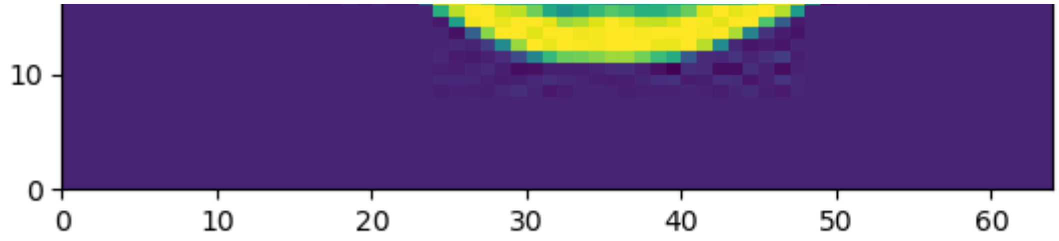
```
#Now let's read the images and fill in the training dataset:
# training data
print("Starting to collect data from images")
for i,filepath_train in enumerate(list(data_train['image_file_paths'])):
    img_array = mpimg.imread(filepath_train)
    X_train[i,:,:] = img_array

# validation data
for j,filepath_val in enumerate(list(data_val['image_file_paths'])):
    img_array = mpimg.imread(filepath_val)
    X_val[j,:,:] = img_array

# testing data
for k,filepath_test in enumerate(list(data_test['image_file_paths'])):
    img_array = mpimg.imread(filepath_test)
    X_test[k,:,:] = img_array
print("Finished collecting images")
```

```
Starting to collect data from images
Finished collecting images
```

In [12]:

```
#plotting a random image to test it
plt.pcolormesh(X_train[100,:,:])
plt.show()
```





Model building with Tensorflow

Dense Neural Network or Multi Layer Perceptron

Now let's build a Multi Layer Perceptron model with tensorflow. The input layer will flatten the array values giving 4096 features. Let's build 2 hidden layers, first with 25 units and second with 15 units, and at the end we will have an output layer with 5 units. We will keep the relu activation units for the hidden layers. In principal we could use softmax regression unit for the output layers. In tensorflow it is better recommended to keep the output layers linear and calculate probabilities of each class later in order to avoid round off errors.

We will also use Flatten layer to flatten the array into a one dimensional array. Using Sequential function, we can connect multiple Dense layers as a deep neural network. Depending the performance of this model, we can build complex models later on.

In [14]:

```
import random
# load the tensorflow libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Let's build the model

tf.random.set_seed(42) # Set the random seed to get same reproducable results
#np.random.seed(4)
#random.seed(4)

model1 = Sequential([
        Flatten(input_shape=[64, 64]),
        Dense(25, activation="relu", name = 'layer1'),
        Dense(15, activation="relu", name = 'layer2'),
        Dense(5, activation="linear", name = 'layer3')
])
```

The first hidden layer will have ((4096 weights)+ 1 bias term) * 25 neural units = 102425 parameters

The second hidden layer will have ((25 weights) + 1 bias term) * 15 neural units = 390 parameters

layer 3 will have ((15 weights) + 1 bias term) * 5 neural units = 80 parameters

Total trainable parameters = 102895

Let's look at the model summary:

In [15]:
```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_1 (Flatten)         (None, 4096)              0         
                                                                 
 layer1 (Dense)              (None, 25)                102425    
                                                                 
 layer2 (Dense)              (None, 15)                390       
                                                                 
 layer3 (Dense)              (None, 5)                 80        
                                                                 
=================================================================
Total params: 102,895
Trainable params: 102,895
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
# Get layer information
model1.layers
```

Out[16]:
```
[<keras.layers.reshaping.flatten.Flatten at 0x7f78c89e2d90>,
 <keras.layers.core.dense.Dense at 0x7f78ca50a100>,
 <keras.layers.core.dense.Dense at 0x7f78ca50a250>,
 <keras.layers.core.dense.Dense at 0x7f78ca4fdf40>]
```

In [17]:
```
# Get each layers
[layer1, layer2, layer3, layer4] = model1.layers

# Get each layer parameters
w1, b1 = layer2.get_weights()
print(w1.shape, b1.shape)

#printing randomly initialized parameters of the first unit
print(w1[:10, 0])
```

```
(4096, 25) (25,)
[-0.02464228  0.00269334 -0.01353711  0.03560334  0.00161467  0.01980854
  0.01158295  0.03724434  0.03488356 -0.02041216]
```

Model compiling and training

Since we set the activation function of the output layers to be linear, we need to do logits = True while compiling the model. Also we will use the sparse_categorical_crossentropy loss function and Adam optimizer for choosing the learning rate for optimizing the cost function. Adam optimizer can adjust the the learning rate of each model parameter adaptively depending on how the cost function changes.

In [18]:
```
model1.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3),
               loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logi
```

```
              metrics = ["accuracy"])
```

In [19]:

```
history1 = model1.fit(X_train, y_train, epochs = 30, validation_data=(X_val,
```

```
Epoch 1/30
1224/1224 [==============================] - 2s 1ms/step - loss: 1.0801 - accur
acy: 0.9460 - val_loss: 0.3364 - val_accuracy: 0.9743
Epoch 2/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.2507 - accur
acy: 0.9714 - val_loss: 0.1606 - val_accuracy: 0.9767
Epoch 3/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1531 - accur
acy: 0.9745 - val_loss: 0.1069 - val_accuracy: 0.9804
Epoch 4/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1161 - accur
acy: 0.9786 - val_loss: 0.1175 - val_accuracy: 0.9736
Epoch 5/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1078 - accur
acy: 0.9781 - val_loss: 0.1173 - val_accuracy: 0.9704
Epoch 6/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1072 - accur
acy: 0.9802 - val_loss: 1.2549 - val_accuracy: 0.8701
Epoch 7/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1070 - accur
acy: 0.9790 - val_loss: 0.0992 - val_accuracy: 0.9767
Epoch 8/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0647 - accur
acy: 0.9846 - val_loss: 0.0656 - val_accuracy: 0.9869
Epoch 9/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0986 - accur
acy: 0.9822 - val_loss: 0.0660 - val_accuracy: 0.9859
Epoch 10/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0915 - accur
acy: 0.9802 - val_loss: 0.1726 - val_accuracy: 0.9624
Epoch 11/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0623 - accur
acy: 0.9852 - val_loss: 0.0657 - val_accuracy: 0.9873
Epoch 12/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.1145 - accur
acy: 0.9792 - val_loss: 0.1549 - val_accuracy: 0.9726
Epoch 13/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0844 - accur
acy: 0.9846 - val_loss: 0.0608 - val_accuracy: 0.9867
Epoch 14/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0516 - accur
acy: 0.9878 - val_loss: 0.1092 - val_accuracy: 0.9804
Epoch 15/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0565 - accur
acy: 0.9879 - val_loss: 0.0549 - val_accuracy: 0.9869
Epoch 16/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0697 - accur
acy: 0.9881 - val_loss: 0.0478 - val_accuracy: 0.9900
Epoch 17/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0434 - accur
acy: 0.9903 - val_loss: 0.0510 - val_accuracy: 0.9906
Epoch 18/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0509 - accur
acy: 0.9886 - val_loss: 0.0635 - val_accuracy: 0.9863
Epoch 19/30
```

```
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0422 - accur
acy: 0.9902 - val_loss: 0.0525 - val_accuracy: 0.9906
Epoch 20/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0639 - accur
acy: 0.9886 - val_loss: 0.0685 - val_accuracy: 0.9830
Epoch 21/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0416 - accur
acy: 0.9910 - val_loss: 0.1450 - val_accuracy: 0.9657
Epoch 22/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0661 - accur
acy: 0.9866 - val_loss: 0.0514 - val_accuracy: 0.9912
Epoch 23/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0363 - accur
acy: 0.9912 - val_loss: 0.0506 - val_accuracy: 0.9902
Epoch 24/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0400 - accur
acy: 0.9913 - val_loss: 0.0577 - val_accuracy: 0.9884
Epoch 25/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0394 - accur
acy: 0.9912 - val_loss: 0.0812 - val_accuracy: 0.9857
Epoch 26/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0515 - accur
acy: 0.9887 - val_loss: 0.0769 - val_accuracy: 0.9828
Epoch 27/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0855 - accur
acy: 0.9887 - val_loss: 0.0764 - val_accuracy: 0.9879
Epoch 28/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0337 - accur
acy: 0.9928 - val_loss: 0.0569 - val_accuracy: 0.9916
Epoch 29/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0435 - accur
acy: 0.9913 - val_loss: 0.0618 - val_accuracy: 0.9916
Epoch 30/30
1224/1224 [==============================] - 1s 1ms/step - loss: 0.0353 - accur
acy: 0.9927 - val_loss: 0.0475 - val_accuracy: 0.9918
```

At a time the model is trained in batches of 32 images and each time the model goes through 1224 batches per epoch to train the data. As we can see we are seeing a 99 % accuracy in the training and validation dataset which is great.

Let's look at the predictions now.

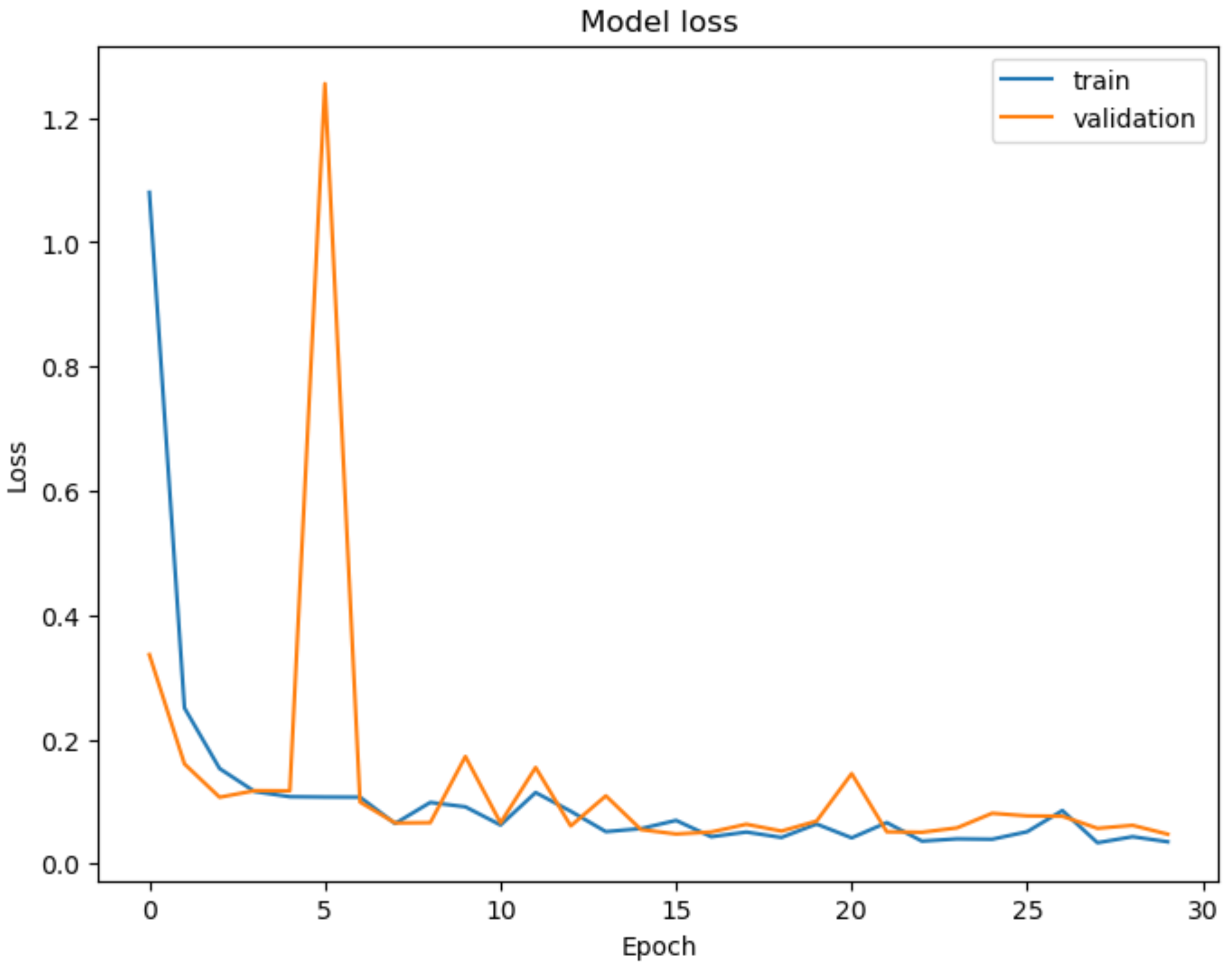
In [20]:

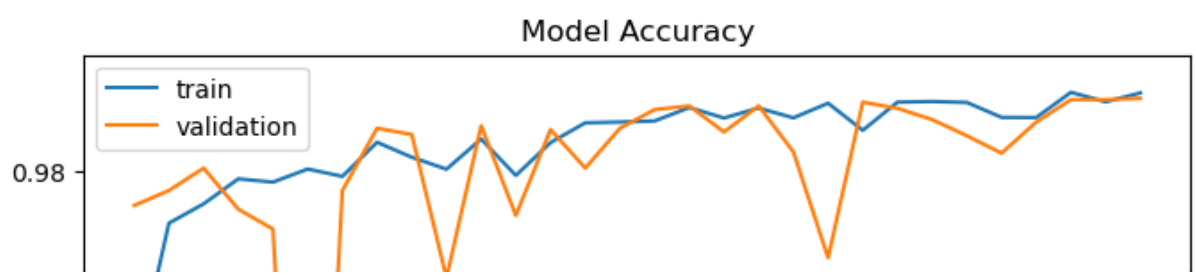
```
# Now let's plot the model loss and accuracy for both the training and valida
# of number of epochs.

# The history.history["loss"] entry is a dictionary with as many values as ep
# model was trained on.
df_loss_acc = pd.DataFrame(history1.history)

# losses data frame
df_loss= df_loss_acc[['loss','val_loss']]
df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)

print(df_loss.shape)

# accuracy data frame
df_acc= df_loss_acc[['accuracy','val_accuracy']]
df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac

# plotting the loss and accuracy
```

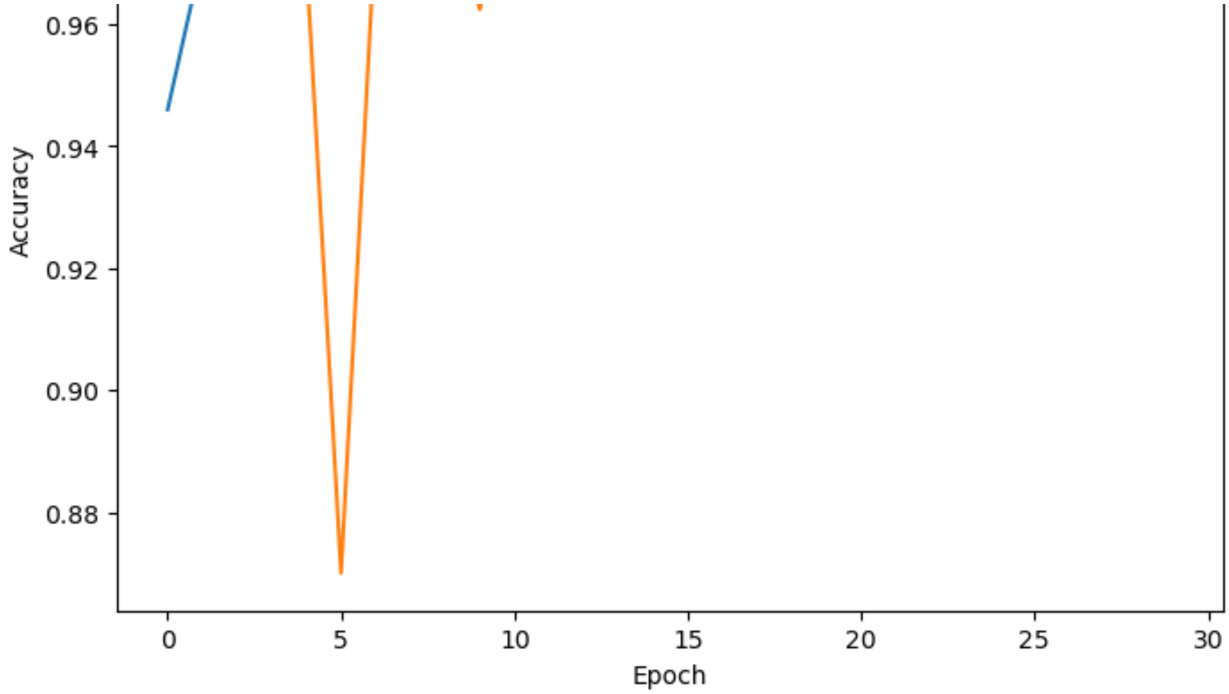
```
# plotting the loss and accuracy
df_loss.plot(title='Model loss',figsize=(8,6)).set(xlabel='Epoch',ylabel='Los
df_acc.plot(title='Model Accuracy',figsize=(8,6)).set(xlabel='Epoch',ylabel='
```

```
(30, 2)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/1848071422.py:1
0: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/1848071422.py:1
6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac
e=True)
```

Out[20]: [Text(0.5, 0, 'Epoch'), Text(0, 0.5, 'Accuracy')]







The loss of both training and validation datasets are decreasing as expected. There is a dip in the accuracy plot in the beginning probably because the gradient descent not converging in the right direction giving rise to high losses at the same time. The accuracy of the training and validation dataset is reaching 99% which is really good metrics. Let's take a look at the other statistics as well.

In [21]:

```
# Calculating the predictions for the training dataset
#Get the linear value from the mode
logits_train = model1.predict(X_train)
pred_train = tf.nn.softmax(logits_train) # this basically gives the probabili
ypred_train = np.argmax(pred_train, axis = 1)
#print(y_pred_train.shape)

#Calculating the predictions for the validation dataset
#Get the linear value from the model
logits_val = model1.predict(X_val)
pred_val = tf.nn.softmax(logits_val) # this basically gives the probability o
ypred_val = np.argmax(pred_val, axis = 1)

# Calculating the predictions for the test dataset
#Get the linear value from the model
logits_test = model1.predict(X_test)
pred_test = tf.nn.softmax(logits_test) # this basically gives the probability
ypred_test = np.argmax(pred_test, axis = 1)

for i in range(10):
    print(y_train[i], ypred_train[i]) # returns the index of maximum probabil
```

```
1224/1224 [==============================] - 1s 414us/step
153/153 [==============================] - 0s 421us/step
153/153 [==============================] - 0s 438us/step
[4] 4
[3] 3
[3] 3
[2] 2
```

```
[4] 4
[2] 2
[0] 2
[4] 4
[1] 1
[4] 4
```

Evaluation of the model and prediction errors.

At least in the 50 cases we have listed above, the prediction of category and actual category matches pretty well. Let's look at number of cases when predictions failed.

In [22]:

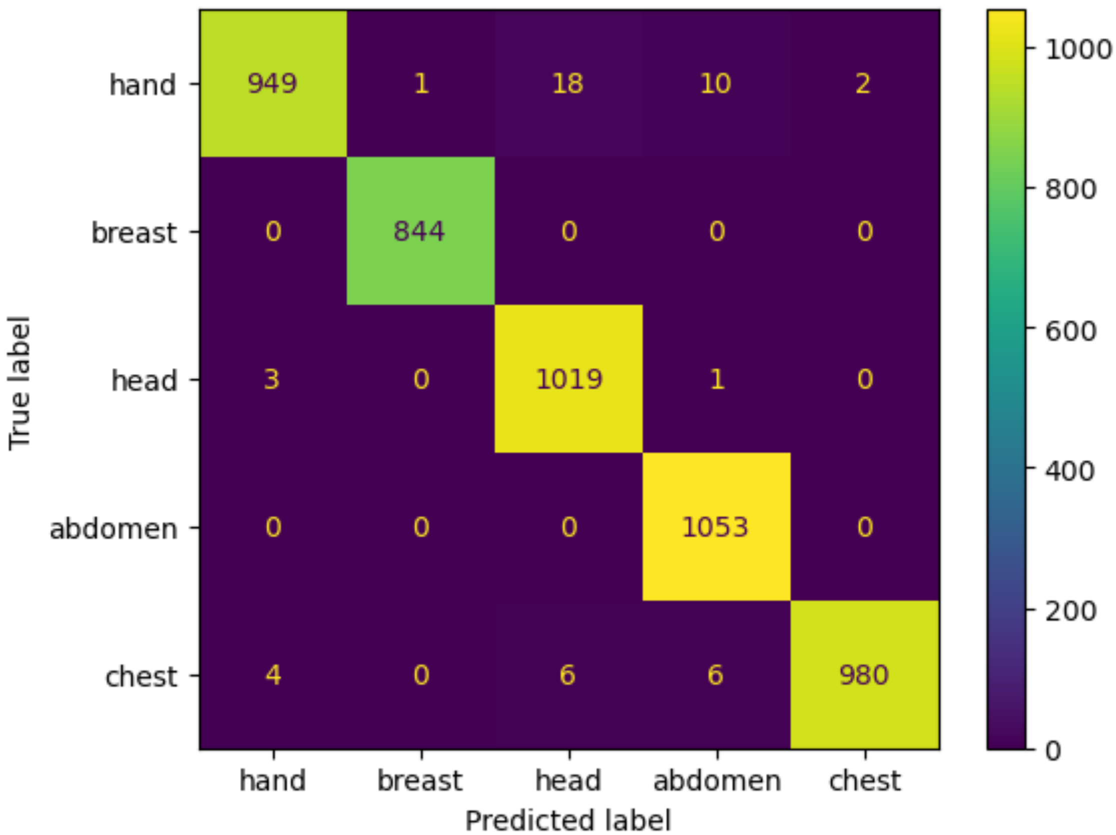
```python
# Let's look at the classification report
from sklearn.metrics import classification_report

print("Report: Train data")
print(classification_report(y_train, ypred_train, target_names=['hand', 'brea

print("Report: Validation data")
print(classification_report(y_val, ypred_val, target_names=['hand', 'breast',

print("Report: Test data")
print(classification_report(y_test, ypred_test, target_names=['hand', 'breast
```

```
Report: Train data
              precision    recall  f1-score   support

        hand       1.00      0.98      0.99      7977
      breast       1.00      1.00      1.00      7217
        head       0.99      1.00      0.99      8016
     abdomen       1.00      1.00      1.00      7957
       chest       1.00      0.99      1.00      7996

    accuracy                           0.99     39163
   macro avg       0.99      0.99      0.99     39163
weighted avg       0.99      0.99      0.99     39163

Report: Validation data
              precision    recall  f1-score   support

        hand       0.99      0.98      0.98      1043
      breast       1.00      1.00      1.00       893
        head       0.99      0.99      0.99       961
     abdomen       0.99      1.00      1.00       990
       chest       1.00      0.99      0.99      1008

    accuracy                           0.99      4895
   macro avg       0.99      0.99      0.99      4895
weighted avg       0.99      0.99      0.99      4895

Report: Test data
              precision    recall  f1-score   support

        hand       0.99      0.97      0.98       980
      breast       1.00      1.00      1.00       844
        head       0.98      1.00      0.99      1023
     abdomen       0.98      1.00      0.99      1053
       chest       1.00      0.98      0.99       996
```

```
    accuracy                           0.99      4896
   macro avg       0.99      0.99      0.99      4896
weighted avg       0.99      0.99      0.99      4896
```

In [23]:

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, ypred_test, display_labels =
plt.show()
```



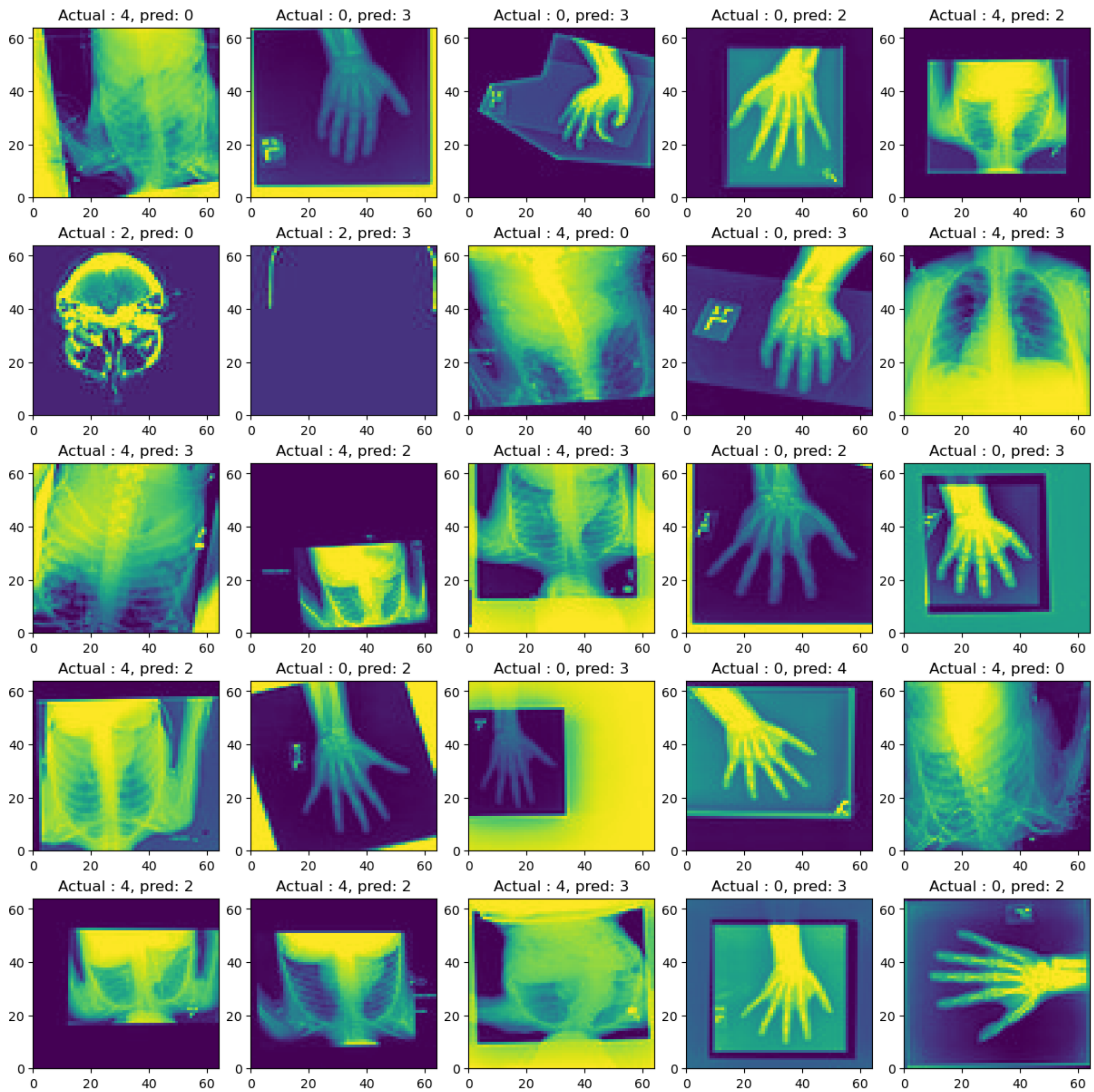
As we can see from the classification report and confusion matrix of the test data, **the MLP classifier is doing a great job in classifying the images of each class with more than 98% accuracy, precision, recall and F1 score.** But there are some hand images predicted as head and abdomen. Otherwise, the algorithm is doing a great job in the classfication process. Now let's look at some of the 50 misclassified images and why it might have happened.

This was our category: ('hand': 0, 'breast': 1, 'head': 2, 'abdomen': 3, 'chest':4]

In [24]:

```
error_ind_test = np.where((y_test[:,0] != ypred_test))[0] # error condidtion


# Plotting the first 25 of the misclassified images:
fig, axs = plt.subplots(5,5, constrained_layout=True, figsize = (12,12))
for i in range(5):
    for j in range(5):
        num = i*5+j
        ind = error_ind_test[num]
        img = X_test[ind,:,:]
```

```
        axs[i,j].pcolormesh(img)
        axs[i,j].set_title(f"Actual : {y_test[ind,0]}, pred: {ypred_test[ind]
plt.show()
```



Interestingly, what we can see is that in most of the misclassified cases are from the chest and hand images. Most of these images are zoomed out or zoomed in versions and appears translated in pixel space. The MLP model is finding hard to classify the wonky images which looks different from the normal class of images showed earlier.

Let's build a little more complex model with 3 hidden layers and more units and see if it can detect features from wonky images:

In [25]:

```
# Building more complex model
model2 = Sequential([
        Flatten(input_shape=[64, 64]),
        Dense(50, activation="relu", name = 'layer1'),
        Dense(25, activation="relu", name = 'layer2'),
        Dense(15, activation="relu", name = 'layer3'),
```

```
        Dense(15, activation="relu", name = 'layer3'),
        Dense(5, activation="linear", name = 'layer4')
])

model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 4096)              0

 layer1 (Dense)              (None, 50)                204850

 layer2 (Dense)              (None, 25)                1275

 layer3 (Dense)              (None, 15)                390

 layer4 (Dense)              (None, 5)                 80

=================================================================
Total params: 206,595
Trainable params: 206,595
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
# Compiling and fitting the data
model2.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3),
               loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logi
               metrics = ["accuracy"])

history2 = model2.fit(X_train, y_train, epochs = 30, validation_data=(X_val,
```

```
Epoch 1/30
1224/1224 [==============================] - 2s 2ms/step - loss: 2.3982 - accur
acy: 0.9403 - val_loss: 0.2791 - val_accuracy: 0.9812
Epoch 2/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.4599 - accur
acy: 0.9664 - val_loss: 0.2243 - val_accuracy: 0.9751
Epoch 3/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.1393 - accur
acy: 0.9818 - val_loss: 0.1518 - val_accuracy: 0.9781
Epoch 4/30
1224/1224 [==============================] - 2s 2ms/step - loss: 0.0964 - accur
acy: 0.9834 - val_loss: 0.0805 - val_accuracy: 0.9886
Epoch 5/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.1149 - accur
acy: 0.9814 - val_loss: 0.0539 - val_accuracy: 0.9896
Epoch 6/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0793 - accur
acy: 0.9856 - val_loss: 0.0899 - val_accuracy: 0.9839
Epoch 7/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0483 - accur
acy: 0.9897 - val_loss: 0.7296 - val_accuracy: 0.9230
Epoch 8/30
1224/1224 [==============================] - 2s 2ms/step - loss: 0.1768 - accur
acy: 0.9789 - val_loss: 0.1652 - val_accuracy: 0.9800
Epoch 9/30
1224/1224 [==============================] - 2s 2ms/step - loss: 0.0610 - accur
acy: 0.9893 - val_loss: 0.0468 - val_accuracy: 0.9912
Epoch 10/30
```

```
1224/1224 [==============================] - 2s 2ms/step - loss: 0.0674 - accur
acy: 0.9871 - val_loss: 0.1055 - val_accuracy: 0.9845
Epoch 11/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0675 - accur
acy: 0.9884 - val_loss: 0.0616 - val_accuracy: 0.9908
Epoch 12/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0543 - accur
acy: 0.9892 - val_loss: 0.0980 - val_accuracy: 0.9824
Epoch 13/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0404 - accur
acy: 0.9911 - val_loss: 0.0322 - val_accuracy: 0.9947
Epoch 14/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0194 - accur
acy: 0.9954 - val_loss: 0.0276 - val_accuracy: 0.9951
Epoch 15/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0226 - accur
acy: 0.9949 - val_loss: 0.0312 - val_accuracy: 0.9937
Epoch 16/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0223 - accur
acy: 0.9952 - val_loss: 0.0993 - val_accuracy: 0.9810
Epoch 17/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0173 - accur
acy: 0.9960 - val_loss: 0.0336 - val_accuracy: 0.9947
Epoch 18/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0558 - accur
acy: 0.9929 - val_loss: 0.0314 - val_accuracy: 0.9951
Epoch 19/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0225 - accur
acy: 0.9954 - val_loss: 0.0327 - val_accuracy: 0.9949
Epoch 20/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0216 - accur
acy: 0.9967 - val_loss: 0.0264 - val_accuracy: 0.9953
Epoch 21/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0306 - accur
acy: 0.9945 - val_loss: 0.0317 - val_accuracy: 0.9943
Epoch 22/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0266 - accur
acy: 0.9958 - val_loss: 0.0261 - val_accuracy: 0.9957
Epoch 23/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0203 - accur
acy: 0.9970 - val_loss: 0.0335 - val_accuracy: 0.9931
Epoch 24/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0134 - accur
acy: 0.9972 - val_loss: 0.0241 - val_accuracy: 0.9965
Epoch 25/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0113 - accur
acy: 0.9977 - val_loss: 0.0337 - val_accuracy: 0.9959
Epoch 26/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0149 - accur
acy: 0.9967 - val_loss: 0.0463 - val_accuracy: 0.9920
Epoch 27/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0123 - accur
acy: 0.9973 - val_loss: 0.0338 - val_accuracy: 0.9949
Epoch 28/30
1224/1224 [==============================] - 2s 1ms/step - loss: 0.0124 - accur
acy: 0.9973 - val_loss: 0.0248 - val_accuracy: 0.9967
Epoch 29/30
1224/1224 [==============================] - 2s 2ms/step - loss: 0.0163 - accur
acy: 0.9965 - val_loss: 0.0304 - val_accuracy: 0.9957
Epoch 30/30
1224/1224 [==============================] - 2s 2ms/step - loss: 0.0131 - accur
```

```
acy: 0.9975 - val_loss: 0.0343 - val_accuracy: 0.9955
```

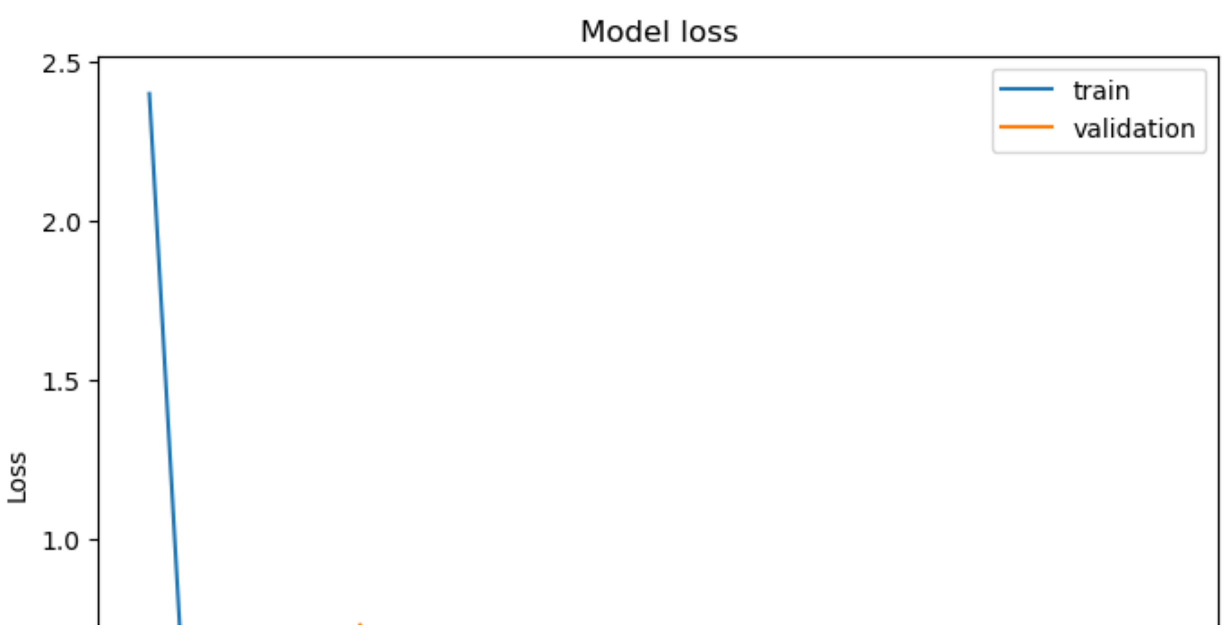
In [27]:

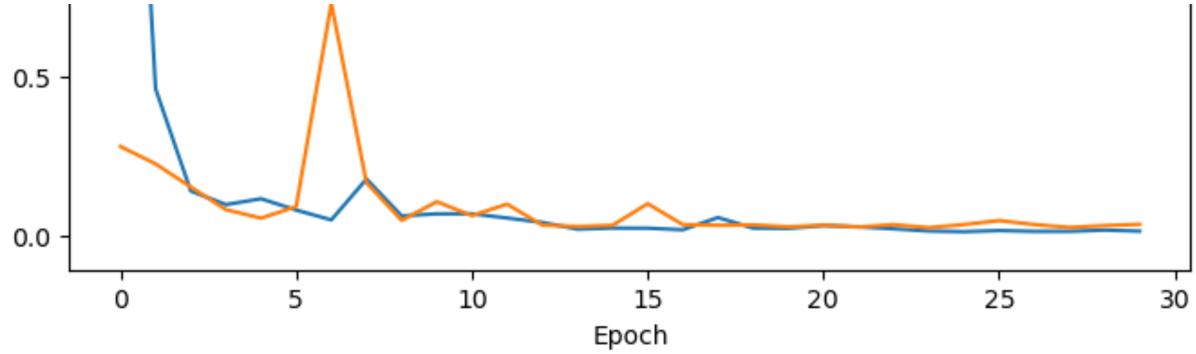
```python
# Now let's plot the model loss and accuracy for both the training and valida
# of number of epochs.

# The history.history["loss"] entry is a dictionary with as many values as ep
# model was trained on.
df_loss_acc = pd.DataFrame(history2.history)

# losses data frame
df_loss= df_loss_acc[['loss','val_loss']]
df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)

print(df_loss.shape)

# accuracy data frame
df_acc= df_loss_acc[['accuracy','val_accuracy']]
df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac

# plotting the loss and accuracy
df_loss.plot(title='Model loss',figsize=(8,6)).set(xlabel='Epoch',ylabel='Los
df_acc.plot(title='Model Accuracy',figsize=(8,6)).set(xlabel='Epoch',ylabel='
```

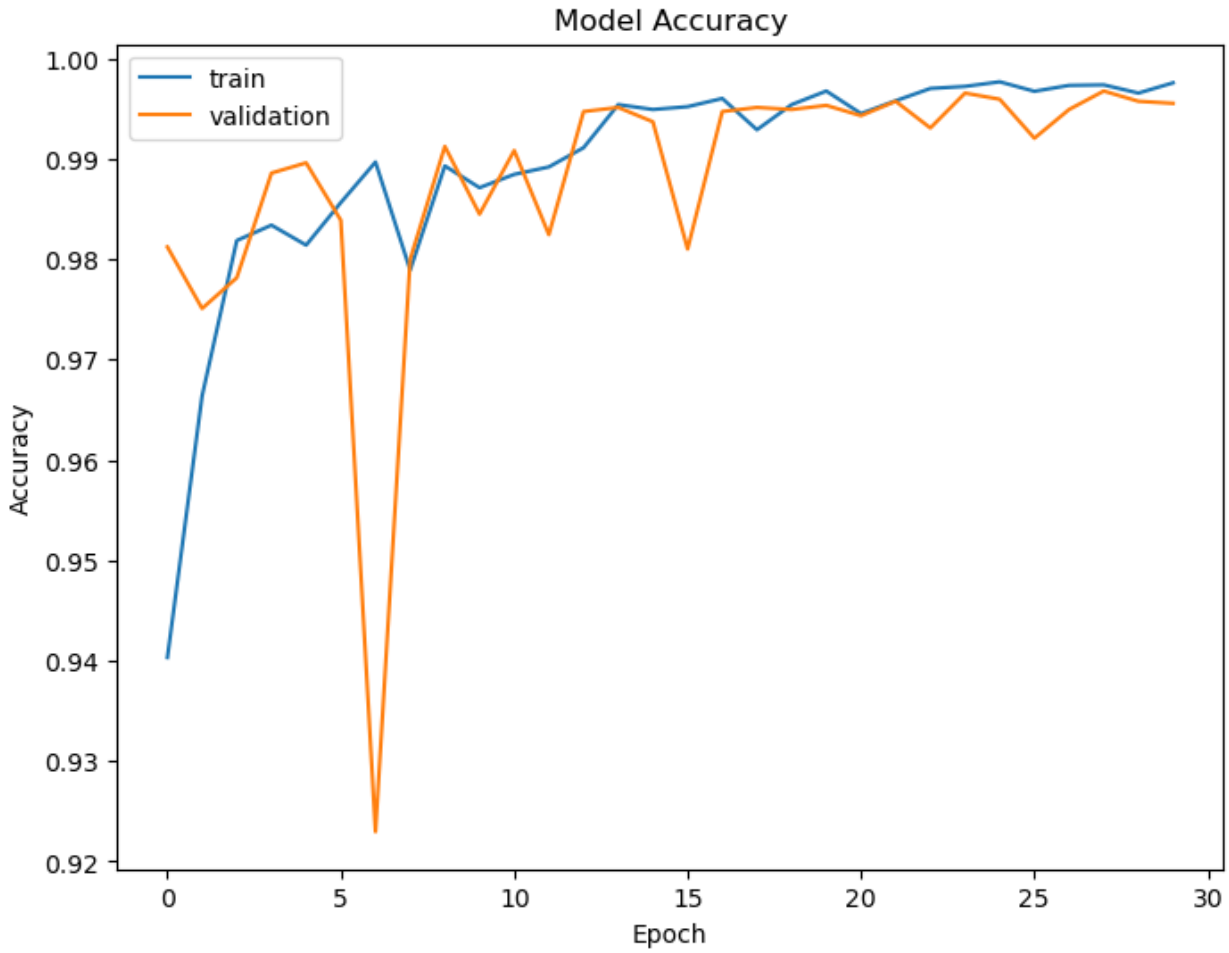
```
(30, 2)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/3758613577.py:1
0: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/3758613577.py:1
6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac
e=True)
```

Out[27]:

```
[Text(0.5, 0, 'Epoch'), Text(0, 0.5, 'Accuracy')]
```







The loss and accuracy plots looks similiar to the first model. Let's look at the classification report and the confusion matrix for this model.

In [ ]:

```
# plotting the loss and accuracy all in one frame
#pd.DataFrame(history2.history).plot(
#figsize=(8, 5), xlim=[0, 29], ylim=[0, 1], grid=True, xlabel="Epoch", style=
#plt.show()
```

In [28]:

```
# Calculating the predictions for the training dataset
#Get the linear value from the mode
logits_train2 = model2.predict(X_train)
pred_train2 = tf.nn.softmax(logits_train2) # this basically gives the probabi
ypred_train2 = np.argmax(pred_train2, axis = 1)
#print(y_pred_train.shape)
```

```python
#Calculating the predictions for the validation dataset
#Get the linear value from the model
logits_val2 = model2.predict(X_val)
pred_val2 = tf.nn.softmax(logits_val2) # this basically gives the probability
ypred_val2 = np.argmax(pred_val2, axis = 1)

# Calculating the predictions for the test dataset
#Get the linear value from the model
logits_test2 = model2.predict(X_test)
pred_test2 = tf.nn.softmax(logits_test2) # this basically gives the probabili
ypred_test2 = np.argmax(pred_test2, axis = 1)
```

```
1224/1224 [==============================] - 1s 424us/step
153/153 [==============================] - 0s 445us/step
153/153 [==============================] - 0s 467us/step
```

In [29]:

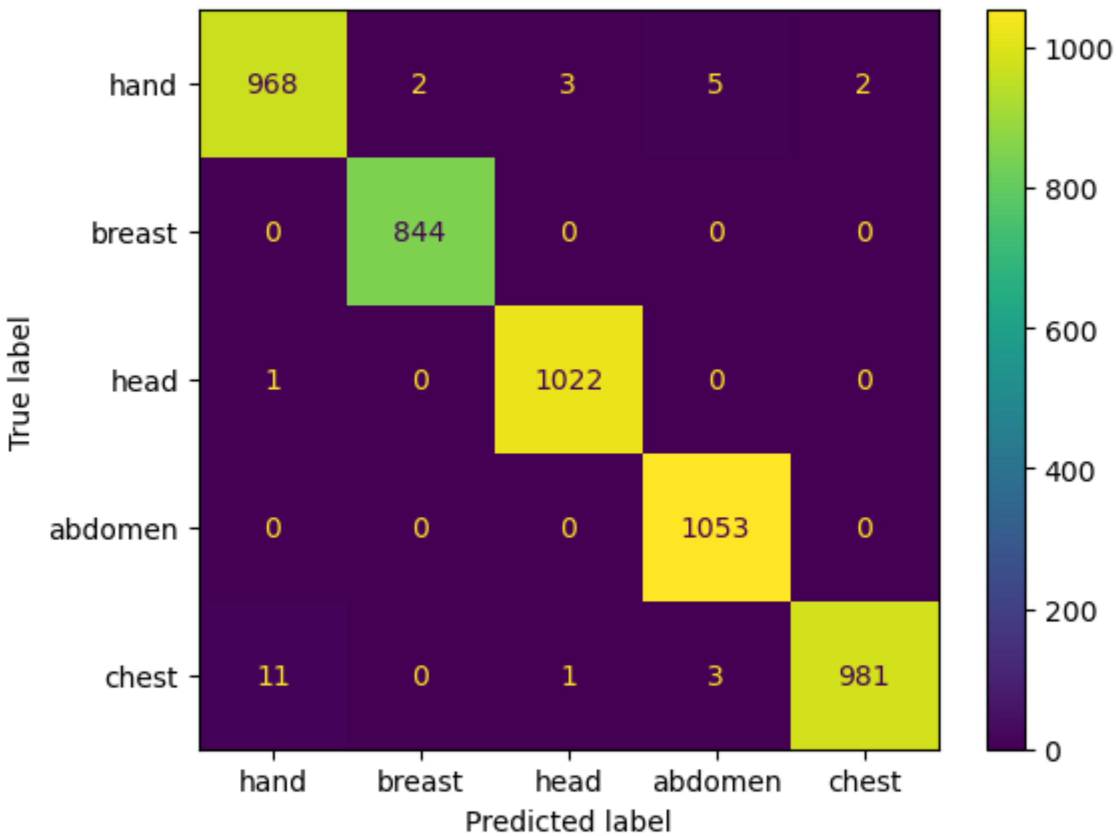
```python
# Let's look at the classification report
from sklearn.metrics import classification_report

print("Report: Train data")
print(classification_report(y_train, ypred_train2, target_names=['hand', 'bre

print("Report: Validation data")
print(classification_report(y_val, ypred_val2, target_names=['hand', 'breast'

print("Report: Test data")
print(classification_report(y_test, ypred_test2, target_names=['hand', 'breas
```

```
Report: Train data
              precision    recall  f1-score   support

        hand       1.00      1.00      1.00      7977
      breast       1.00      1.00      1.00      7217
        head       1.00      1.00      1.00      8016
     abdomen       1.00      1.00      1.00      7957
       chest       1.00      1.00      1.00      7996

    accuracy                           1.00     39163
   macro avg       1.00      1.00      1.00     39163
weighted avg       1.00      1.00      1.00     39163

Report: Validation data
              precision    recall  f1-score   support

        hand       0.99      0.99      0.99      1043
      breast       1.00      1.00      1.00       893
        head       0.99      1.00      0.99       961
     abdomen       1.00      1.00      1.00       990
       chest       1.00      0.99      0.99      1008

    accuracy                           1.00      4895
   macro avg       1.00      1.00      1.00      4895
weighted avg       1.00      1.00      1.00      4895

Report: Test data
              precision    recall  f1-score   support

        hand       0.99      0.99      0.99       980
      breast       1.00      1.00      1.00       844
        head       1.00      1.00      1.00      1023
     abdomen       0.99      1.00      1.00      1053
```

```
       chest       1.00      0.98      0.99       996

    accuracy                           0.99      4896
   macro avg       0.99      0.99      0.99      4896
weighted avg       0.99      0.99      0.99      4896
```

In [30]:

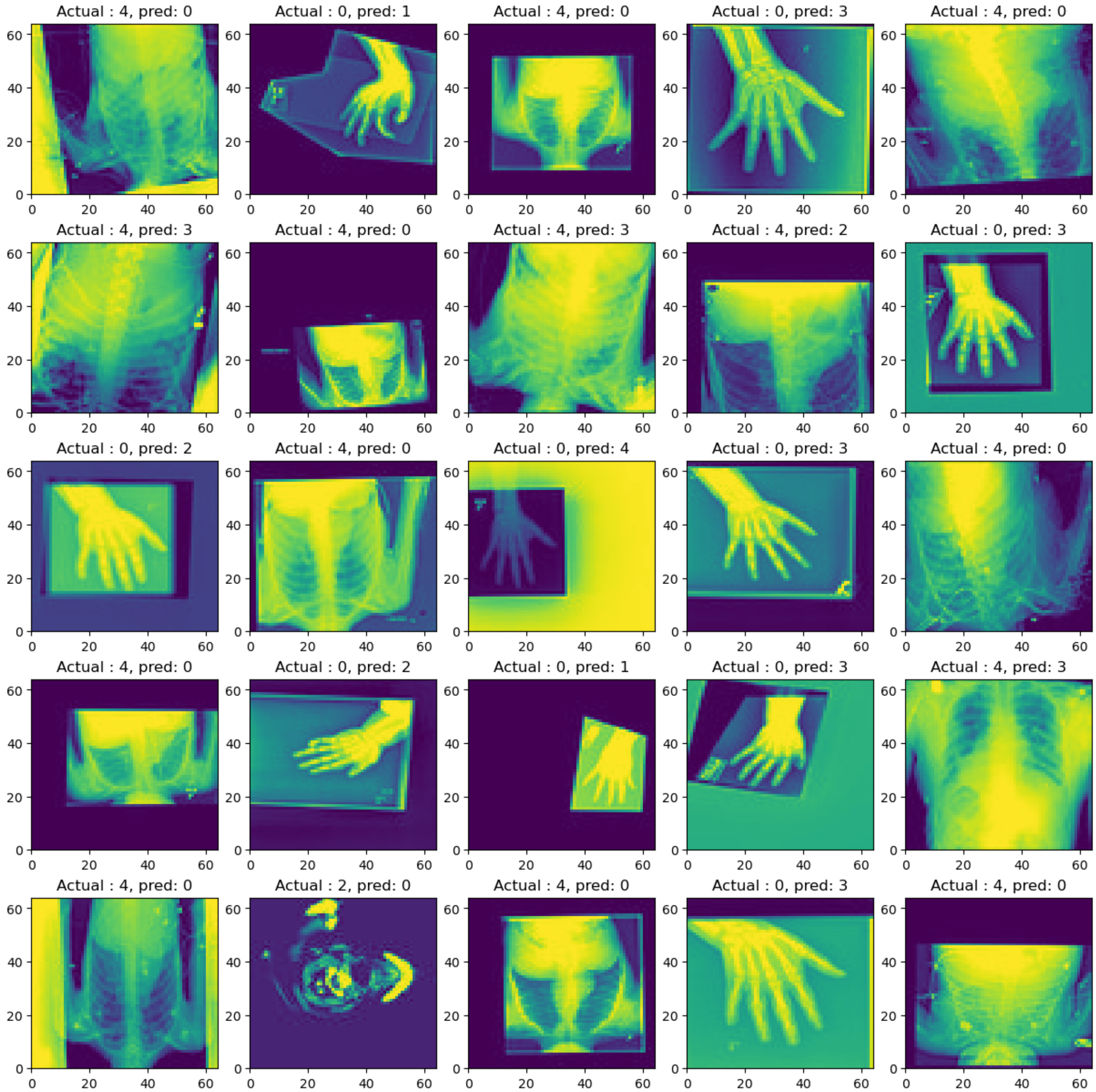
```python
# confusion matrix display
ConfusionMatrixDisplay.from_predictions(y_test, ypred_test2, display_labels =
plt.show()
```



The precision, recall and F1 score of the second model looks more or less similiar to the first model due to rounding upto 2 digits. But as you can see, there are 50 misclassified samples from the first model and 28 misclassified ones in from the second model. So definitely the second model is doing a better job in the classification problem. Let's take a look at all the misclassifed ones again.

In [31]:

```python
error_ind_test2 = np.where((y_test[:,0] != ypred_test2))[0] # error condidtio

# Plotting the first 25 of the misclassified images:
fig, axs = plt.subplots(5,5, constrained_layout=True, figsize = (12,12))
for i in range(5):
    for j in range(5):
        num = i*5+j
        ind = error_ind_test2[num]
        img = X_test[ind,:,:]
        axs[i,j].pcolormesh(img)
        axs[i,j].set_title(f"Actual : {y_test[ind,0]}, pred: {ypred_test2[ind
plt.show()
```



Both the models are finding it difficult to classify the pictures of hands coming in from different directions or stretched or rotated versions.

Convolutional Neural Networks (CNNs)

This dataset is a really good dataset and both the MLP models are doing a great job in the classification. However, a DNN model does not work well if the images are stretched, squeezed or rotatedd. Let's build a simple CNN model to see if we can classify these hands with better accuracy. CNN is the widely used techique in computer vision purposes which can build deep learning models efficiently with lesser number of shared parameters and it can identify different features with translational invariance.

Let's build the CNN model now.

In [32]:

```
# Let's change the shape of the data array for inputing to CNN
```

```
# the current array does not have a frequency axis. Let's add a new frequency
X_train_new = np.expand_dims(X_train, axis=3)
X_val_new = np.expand_dims(X_val, axis=3)
X_test_new = np.expand_dims(X_test, axis=3)
```

In [41]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Input

input_shape = X_train_new.shape[1:]
print(input_shape)

# Let's build a CNN model of this form
# Input >> ConV2D << Maxpool << Conv2D <<  Maxpool <<  ConV2D << Maxpool << C
cnn = Sequential([
    Input(shape=input_shape),
    Conv2D(filters=32, kernel_size=3, strides=(1,1), activation='relu', paddi
    MaxPool2D(pool_size=(2, 2), strides=(2,2), name = 'pool1'),
    Conv2D(filters=64, kernel_size=3, strides=(1,1), activation='relu', paddi
    MaxPool2D(pool_size=(2, 2), strides=(2,2), name = 'pool2'),
    Conv2D(filters=128, kernel_size=3, strides=(1,1), activation='relu', padd
    MaxPool2D(pool_size=(2, 2), strides=(2,2), name = 'pool3'),
    Conv2D(filters=256, kernel_size=3, strides=(1,1), activation='relu', padd
    MaxPool2D(pool_size=(2, 2), strides=(1,1), name = 'pool4'),
    Flatten(name = 'Flat'),
    Dense(512, activation='relu', name='Dense1'),
    Dense(128, activation='relu', name='Dense2'),
    Dense(32, activation='relu', name='Dense3'),
    Dense(5, activation='softmax', name='output' )])

cnn.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', met
cnn.summary()
```

```
(64, 64, 1)
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1 (Conv2D)              (None, 62, 62, 32)        320

 pool1 (MaxPooling2D)        (None, 31, 31, 32)        0

 conv2 (Conv2D)              (None, 29, 29, 64)        18496

 pool2 (MaxPooling2D)        (None, 14, 14, 64)        0

 conv3 (Conv2D)              (None, 12, 12, 128)       73856

 pool3 (MaxPooling2D)        (None, 6, 6, 128)         0

 conv4 (Conv2D)              (None, 4, 4, 256)         295168

 pool4 (MaxPooling2D)        (None, 3, 3, 256)         0

 Flat (Flatten)              (None, 2304)              0

 Dense1 (Dense)              (None, 512)               1180160

 Dense2 (Dense)              (None, 128)               65664

 Dense3 (Dense)              (None, 32)                4128
```

```
 output (Dense)              (None, 5)                 165

=================================================================
Total params: 1,637,957
Trainable params: 1,637,957
Non-trainable params: 0
_________________________________________________________________
```

This model has lesser number of parameters compared to the second DNN model. Now let's fit the model with the data.

In [42]:

```
history3 = cnn.fit(X_train_new, y_train, batch_size=64, epochs=10, validation
```

```
Epoch 1/10
612/612 [==============================] - 48s 78ms/step - loss: 0.2322 - accur
acy: 0.9717 - val_loss: 0.0116 - val_accuracy: 0.9967
Epoch 2/10
612/612 [==============================] - 47s 77ms/step - loss: 0.0225 - accur
acy: 0.9932 - val_loss: 0.0127 - val_accuracy: 0.9953
Epoch 3/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0097 - accur
acy: 0.9975 - val_loss: 0.0015 - val_accuracy: 0.9994
Epoch 4/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0118 - accur
acy: 0.9969 - val_loss: 0.0059 - val_accuracy: 0.9984
Epoch 5/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0035 - accur
acy: 0.9990 - val_loss: 0.0035 - val_accuracy: 0.9990
Epoch 6/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0160 - accur
acy: 0.9966 - val_loss: 0.0041 - val_accuracy: 0.9988
Epoch 7/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0031 - accur
acy: 0.9992 - val_loss: 0.0037 - val_accuracy: 0.9986
Epoch 8/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0055 - accur
acy: 0.9985 - val_loss: 0.0162 - val_accuracy: 0.9959
Epoch 9/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0214 - accur
acy: 0.9954 - val_loss: 0.0116 - val_accuracy: 0.9953
Epoch 10/10
612/612 [==============================] - 47s 76ms/step - loss: 0.0060 - accur
acy: 0.9989 - val_loss: 0.0070 - val_accuracy: 0.9982
```

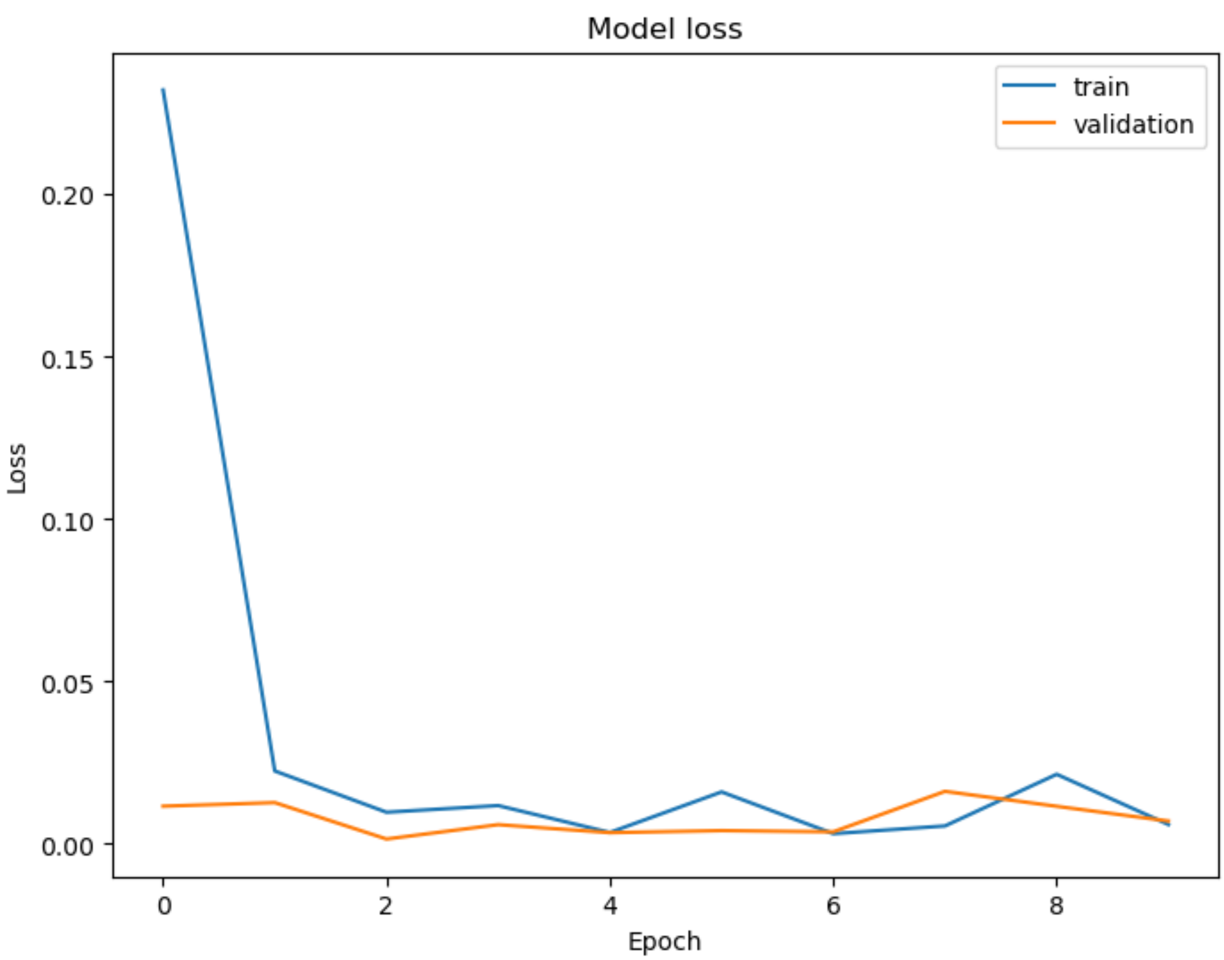
In [43]:

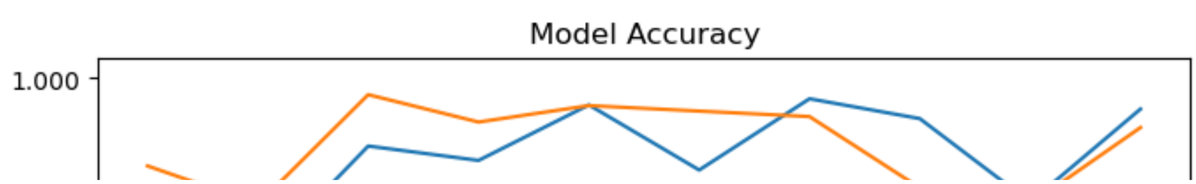
```
# Now let's plot the model loss and accuracy for both the training and valida
# of number of epochs.

# The history.history["loss"] entry is a dictionary with as many values as ep
# model was trained on.
df_loss_acc = pd.DataFrame(history3.history)

# losses data frame
df_loss= df_loss_acc[['loss','val_loss']]
df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)

print(df_loss.shape)

# accuracy data frame
```

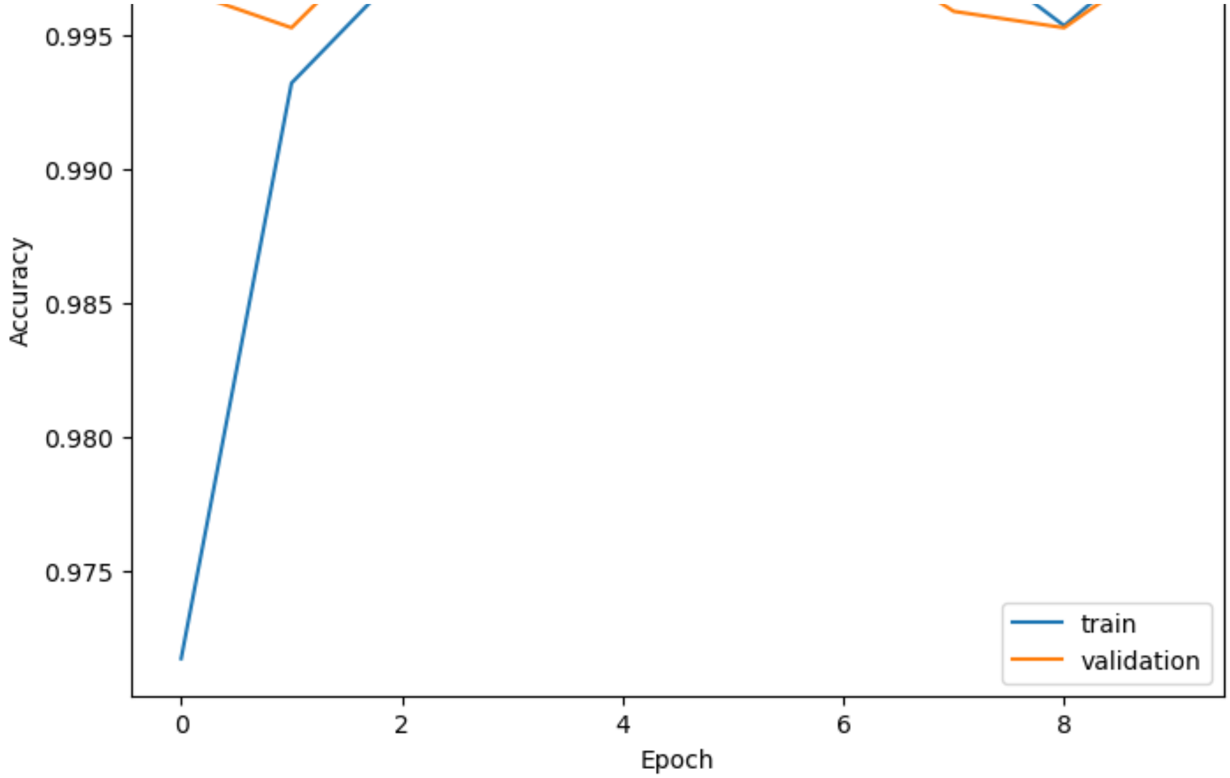
```
df_acc= df_loss_acc[['accuracy','val_accuracy']]
df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac

# plotting the loss and accuracy
df_loss.plot(title='Model loss',figsize=(8,6)).set(xlabel='Epoch',ylabel='Los
df_acc.plot(title='Model Accuracy',figsize=(8,6)).set(xlabel='Epoch',ylabel='
```

```
(10, 2)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/2268470446.py:1
0: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)
/var/folders/c8/g5hp4hlx7dv6gv7n9zdg74rc0000gn/T/ipykernel_2691/2268470446.py:1
6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplac
e=True)
```

Out[43]: [Text(0.5, 0, 'Epoch'), Text(0, 0.5, 'Accuracy')]







In [53]:

```
# Now calcualte the predictions
ypred_train3 = np.argmax(cnn.predict(X_train_new), axis=1)
ypred_val3 = np.argmax(cnn.predict(X_val_new), axis=1)
ypred_test3 = np.argmax(cnn.predict(X_test_new), axis=1)
```

```
1224/1224 [==============================] - 17s 14ms/step
153/153 [==============================] - 2s 14ms/step
153/153 [==============================] - 2s 14ms/step
```

In [54]:

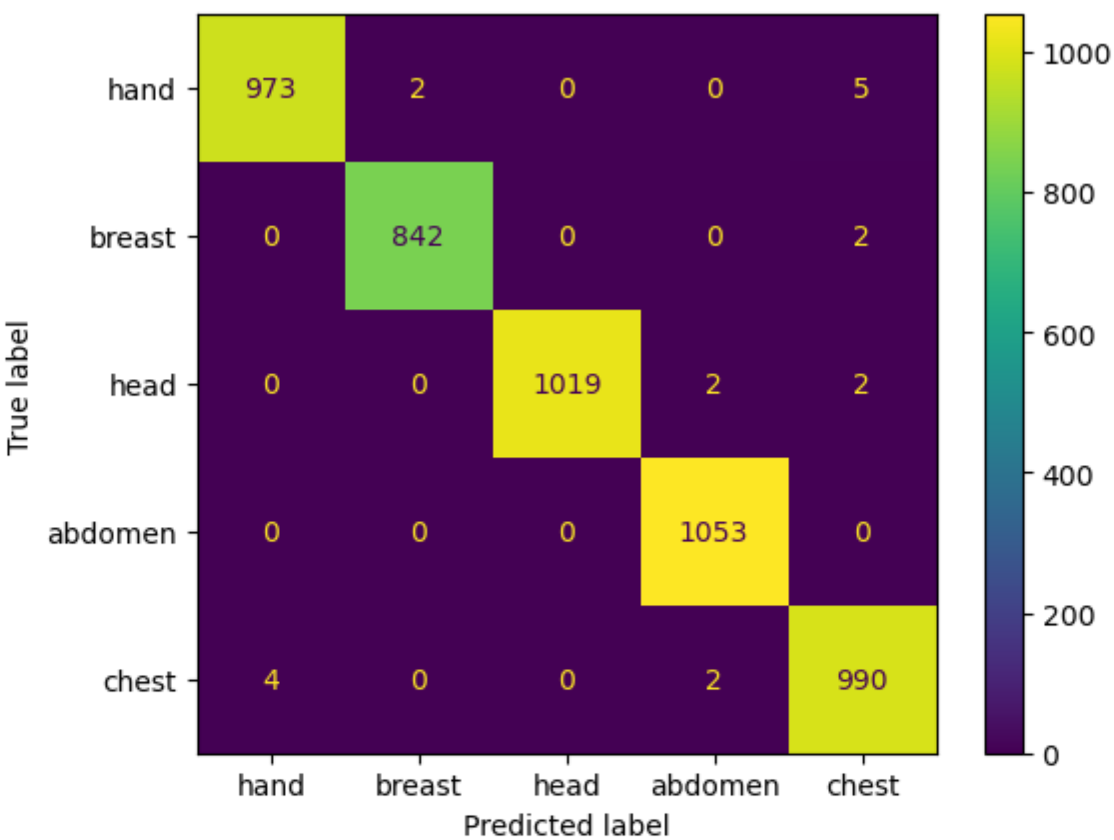
```
# Let's look at the classification report
from sklearn.metrics import classification_report

print("Report: Train data")
print(classification_report(y_train, ypred_train3, target_names=['hand', 'bre

print("Report: Validation data")
print(classification_report(y_val, ypred_val3, target_names=['hand', 'breast'

print("Report: Test data")
print(classification_report(y_test, ypred_test3, target_names=['hand', 'breas
```

```
Report: Train data
              precision    recall  f1-score   support

        hand       1.00      1.00      1.00      7977
      breast       1.00      1.00      1.00      7217
        head       1.00      1.00      1.00      8016
     abdomen       1.00      1.00      1.00      7957
       chest       1.00      1.00      1.00      7996

    accuracy                           1.00     39163
   macro avg       1.00      1.00      1.00     39163
weighted avg       1.00      1.00      1.00     39163
```

```
Report: Validation data
              precision    recall  f1-score   support

        hand       1.00      1.00      1.00      1043
      breast       1.00      1.00      1.00       893
        head       1.00      1.00      1.00       961
     abdomen       1.00      1.00      1.00       990
       chest       1.00      1.00      1.00      1008

    accuracy                           1.00      4895
   macro avg       1.00      1.00      1.00      4895
weighted avg       1.00      1.00      1.00      4895

Report: Test data
              precision    recall  f1-score   support

        hand       1.00      0.99      0.99       980
      breast       1.00      1.00      1.00       844
        head       1.00      1.00      1.00      1023
     abdomen       1.00      1.00      1.00      1053
       chest       0.99      0.99      0.99       996

    accuracy                           1.00      4896
   macro avg       1.00      1.00      1.00      4896
weighted avg       1.00      1.00      1.00      4896
```

In [55]:

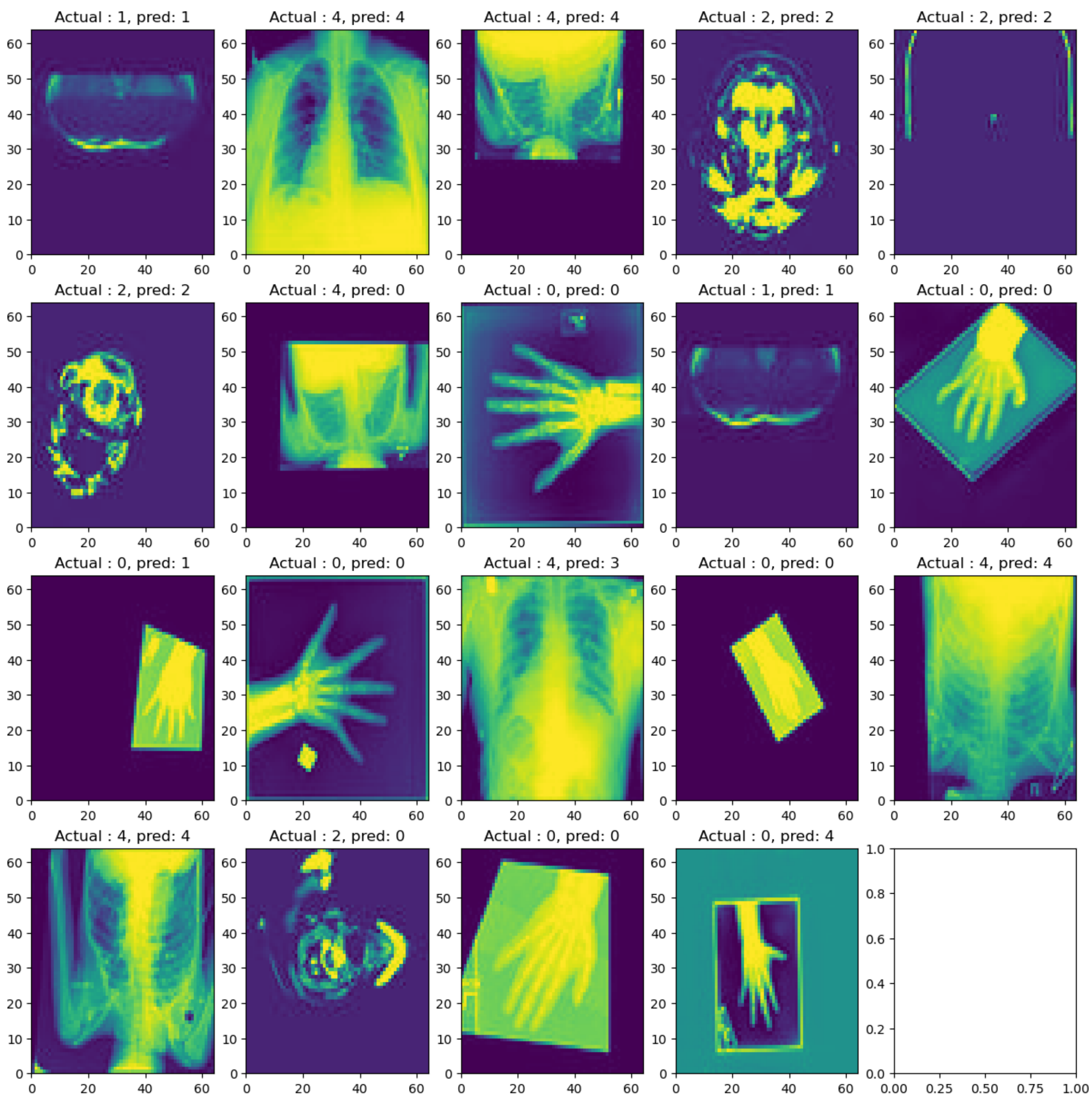
```python
# confusion matrix display
ConfusionMatrixDisplay.from_predictions(y_test, ypred_test3, display_labels =
plt.show()
```



The classification metrics of CNN are better than the DNN models. All the metric scores are above 99%. If we look here, the misclassification of the hand and the chest images have gone down and there are only 19 misclassified samples. Let's take a look at the misclassified examples.

In [57]:

```
error_ind_test3 = np.where((y_test[:,0] != ypred_test3))[0] # error condidtio
len_ind = len(error_ind_test3)

# Plotting the first 25 of the misclassified images:
fig, axs = plt.subplots(4,5, constrained_layout=True, figsize = (12,12))
for i in range(4):
    for j in range(5):
        num = i*5+j
        if num < len_ind:
            ind = error_ind_test3[num]
            img = X_test[ind,:,:]
            axs[i,j].pcolormesh(img)
            axs[i,j].set_title(f"Actual : {y_test[ind,0]}, pred: {ypred_test2
plt.show()
```



The number of misclassified images of hands and chest has gone down. Overall the CNN does a great job in the classification, but still there are some examples of hand and chest that are misclassified. An image augmentation of the existing hand images with rotation and training again might help to identify them. Since these numbers are pretty low, we are not going to do any image augmentation now.

Conclusion

For this medical image classification purpose, we utilized a curated and balanced medical MNIST dataset. Both the dense neural networks and CNNs are model are doing great job in classifying images with more than 99 % accuracy, precision, recall and F1-scores. Among