Politecnico di Torino

Bioinformatics

Delegate features extraction to a Convolutional Neural Networks

Author:
Gabriele Chiriatti
Alexander Grajales Quintero Duverley

September 25, 2018

Contents

Abstract

We present a method to classify images with a Pre-trained Model, that are not trained for that images. So we extract features from Pre-trained Model of Keras and after a fetures reduction, a supervised algorithm is used to classify the images.

1 Image Data Set

The images represents histological samples of subjects looking for lung lesions. These problems can be benign or malignant. In particular represent:

- Epithelioid Mesothelioma (ME) malignant tumor.
- Sarcomatoid Mesothelioma (MS) malignant tumor.
- Benign Fibrosis (F).

In the Data Set there are 3104 F images, 921 ME image and 3750 MS images.

2 The Project

In our work we used different Keras pre-trained model as features extractrion.
The model used are:

- VGG16
- VGG19
- DenseNet
- MobileNet
- ResNet50
- Xception
- NASNet

2.1 Feature Extraction

For our deep learning API we are using Keras which provides a high level abstraction to many of the lower level deep learning libraries like TensorFlow and Theano. Keras as mentioned comes with a bunch of pre-trained deep learning models. These models have been trained to recognise 1000 different categories from the ImageNet database. However we can also use them to extract a feature vector (a list of floating point values) of the models internal representation of a category. To get started with keras we first need to create an instance of the model we want to use.
The program used is called *extract.py*.
In this example we are using the function *data_imgnet* to select a specific model.

```python
def data_imgnet(name):
    if name == 'VGG16':
        return applications.vgg16.VGG16(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'VGG19':
        return applications.vgg19.VGG19(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'DenseNet':
        return applications.densenet.DenseNet201(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'MobileNet':
        return applications.mobilenet.MobileNet(weights='imagenet', include_top=False, pooling='avg', input_shape=(224, 224, 3))
    elif name == 'RenseNet50':
        return applications.resnet50.ResNet50(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'Xception':
        return applications.xception.Xception(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'InceptionResNetV2':
        return applications.inception_resnet_v2.InceptionResNetV2(weights='imagenet', include_top=False, pooling='avg')
    elif name == 'NASNet':
        return applications.nasnet.NASNetMobile(weights='imagenet', include_top=False, pooling='avg', input_shape=(224, 224, 3))
```

Here we are setting the weights to Imagenet which will automatically download the learn parameters from the ImageNet database. The next important arg here is *include_top=False* which removes the fully connected layer at the end/top of the network. This allows us to get the feature vector as opposed to a classification.

Once initialised the model we can then pass it an image and use it to predict what it might be. However since we don't want the prediction we instead will get a list of floating point values.

```python
def features_imgnet(data, model_data):

    data_aux = list(data)
    items = []

    for i in range(len(data_aux)):
        item = data_aux[i]
        img_file = item['image'].split("/")
        print("%s %s" % (item['id'], img_file[1]))

        img_path = './images/' + item['image']
        img = image.load_img(img_path, target_size=(224, 224))

        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)

        features = model_data.predict(x)[0]
        array_features = np.char.mod('%f', features)

        items.append({'id': item['id'], 'features': ','.join(array_features), 'cluster': item['cluster']})

    return items
```

We first load then convert our image into an array of RGB values. We then

pass it into the models predict method to extract our vector.

2.2 Feature Reduction

PCA and t-SNE are two tools available in scikit-learn library for features reduction. In Python, scikit-learn is a widely used library for implementing machine learning algorithms. (`http://scikit-learn.org/stable/index.html`)
The program used is called *tecniques.py*.
In the function *data_model* is possible to selct (by command line) 3 different method to reduce features. Is possible to use PCA, t-SNE or a combination of both method. In particular *fit_trasform (X[, y])* fit X into an embedded space and return that transformed output.

```
def data_model(name, n):
    if name == 'TSNE':
        return manifold.TSNE(random_state=0)
    elif name == 'TSNE-PCA':
        tsne = manifold.TSNE(random_state=0, perplexity=50, early_exaggeration=6.0)
        pca = decomposition.PCA(n_components=n)
        return pipeline.Pipeline([('reduce_dims', pca), ('tsne', tsne)])
    elif name == 'PCA':
        return decomposition.PCA(n_components=n)


def data_process(data, model):
    transformed = [d.split(',') for d in data['features']]

    x_data = numpy.asarray(transformed).astype('float64')
    x_data = x_data.reshape((x_data.shape[0], -1))

    vis_data = model.fit_transform(x_data)

    results = []
    for i in range(0, len(data)):
        results.append({'id': data['id'][i], 'x': vis_data[i][0], 'y': vis_data[i][1], 'cluster': data['cluster'][i]})
    return results
```

The output of the algorithm are saved in *results*:

```
try:
    model_data = data_model(model_name, num_components)
    data = pandas.read_csv(source_file, sep='\t')
    results = data_process(data, model_data)
```

2.3 Visualization of high-dimensional data in a low-dimensional space

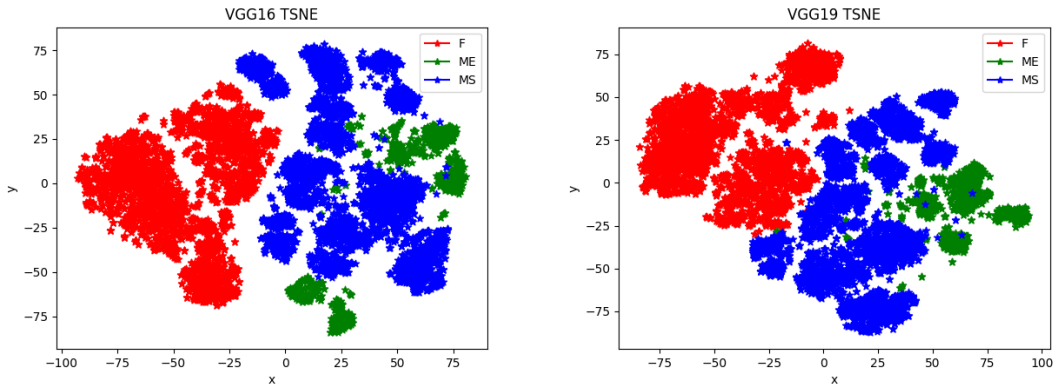
In order to visualize the data a reduction of our feature vector (dimensions) down to 2 is needed.
In our work we use three different method to do that:

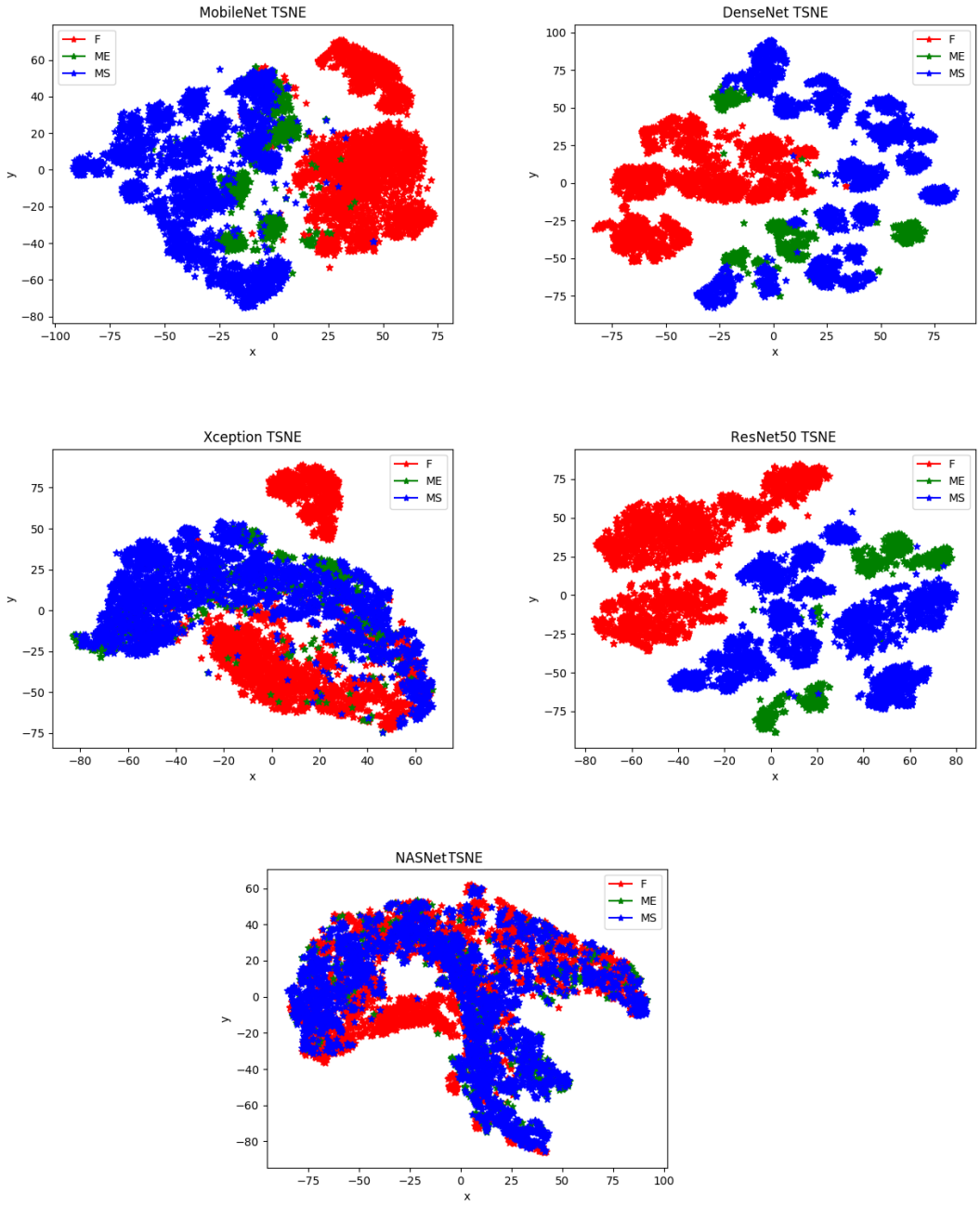
- Use t-SNE algorithm to reduce the features down to 2.
- Use PCA algorithm to reduce the features down to 2.
- Use PCA and t-SNE to reduce features down to 2.

In the following example the features are extracted from the last layer in the net.

2.3.1 T-distributed Stochastic Neighbor Embedding (t-SNE)

The image obtained for each model in this case are:

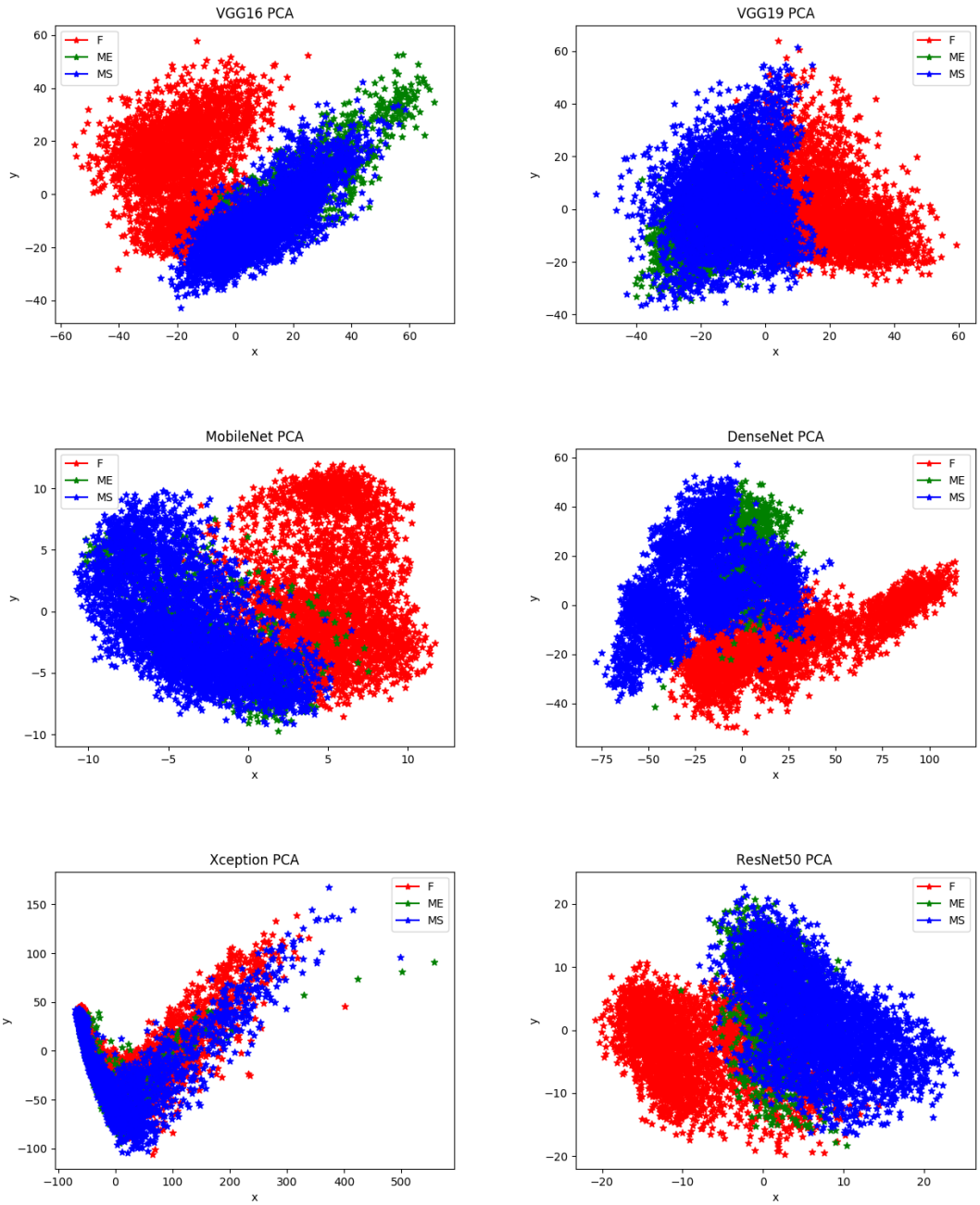


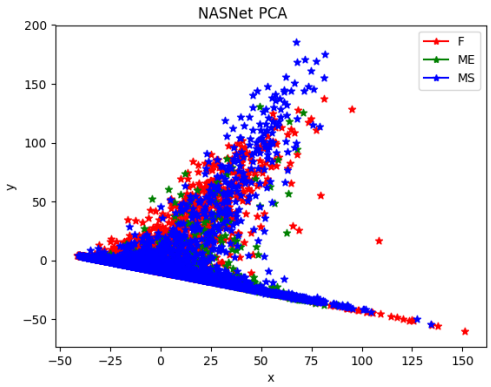


A drawback of this tecnique is that the t-SNE is not linear method so is not possible to save weights for a new image. Fine-tuning T-SNE equals tuning some heuristic-algorithm for data. This problem is not present if we use PCA.

2.3.2 Principal component analysis (PCA)

The PCA is now applicated for 2 dimensions extraction for each model





Is possible to see that this are the worst results. Infact is not raccomanded to use PCA to extract features down to 2.

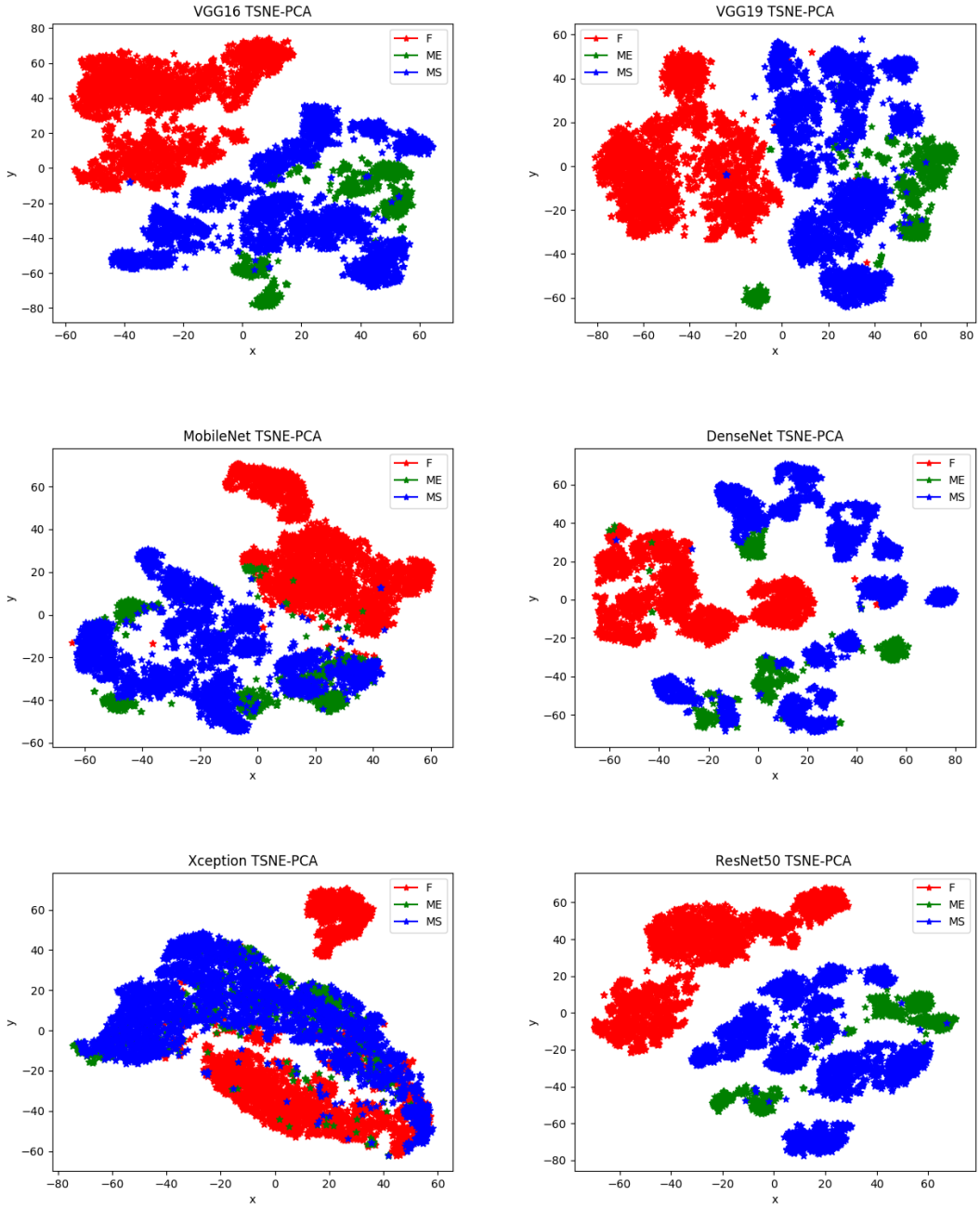
2.3.3 PCA and t-SNE

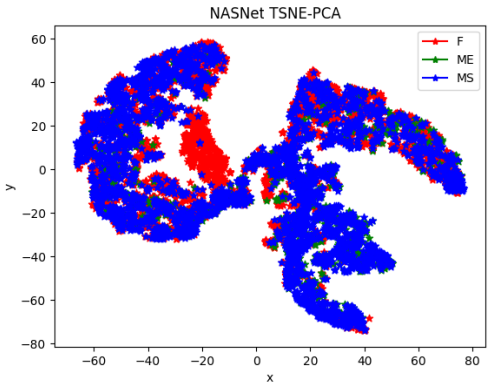
One other solution to visualize our data is to combine the two method: is possible to reduce features with PCA and use t-SNE to reduce features to two. The optimal number of component with PCA is calculated with a funtion *data_info*.

```
def data_info(data, model):
    transformed = [d.split(',') for d in data['features']]

    x_data = numpy.asarray(transformed).astype('float64')
    x_data = x_data.reshape((x_data.shape[0], -1))

    vis_data = model.fit_transform(x_data)

    # print(len(model.explained_variance_ratio_))

    cont_values = numpy.float32(0)
    cont_layer = 0
    for p in range(len(model.explained_variance_ratio_)):
        aux_info = numpy.float32(model.explained_variance_ratio_[p])

        if cont_values <= numpy.float32(0.995):
            cont_values = cont_values + aux_info
            cont_layer = cont_layer + 1

    global ncomp
    ncomp = cont_layer
```

The command *pca_obj.explained_variance_ratio_* is present in scikit-learn library and the ouput is a vector with percentage of variance explained by each of compo-

nents. Now, we can keep adding the variance percentages until we get the desired value (in our case, 0.995). The number of required components will be the value of the variable *comp*.
The images obtained in these case are:

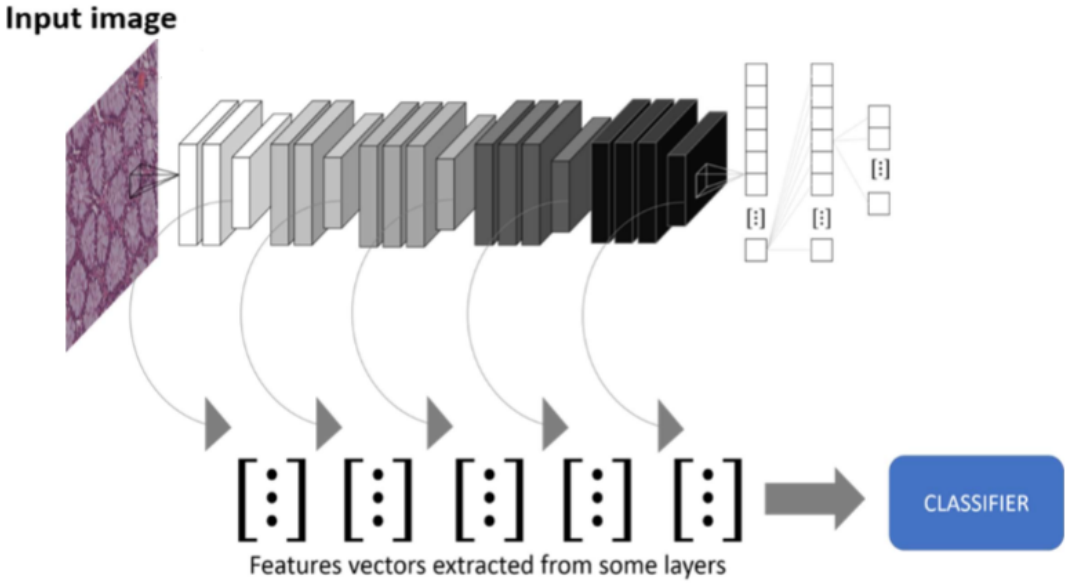




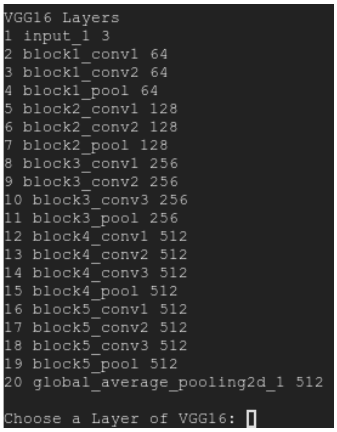
This is the best solution to visualize data infact is possible to see that the class are more defined than in the t-SNE case.

2.4 Extract features from an arbitrary intermediate layer

In the pre-trained model of Keras is also possible to extract features from an intermediate layer



The name of the layers are visualize in the prompt during program esegution as example are reported names of VGG16 layers:



The user can select different output layer by number.

```
def intm_layer(name, data):
    print("\nLayer of %s" % name)

    list_layers = []
    for i, layer in enumerate(data.layers):
        list_layers.append({'id': i + 1, 'layer': layer.name})
        print(i + 1, layer.name)

    input_var = input("\nEnter something: ")
    ivar = int(input_var)

    item = list_layers[ivar - 1]

    return Model(inputs=data.input, outputs=data.get_layer(item['layer']).output)
```

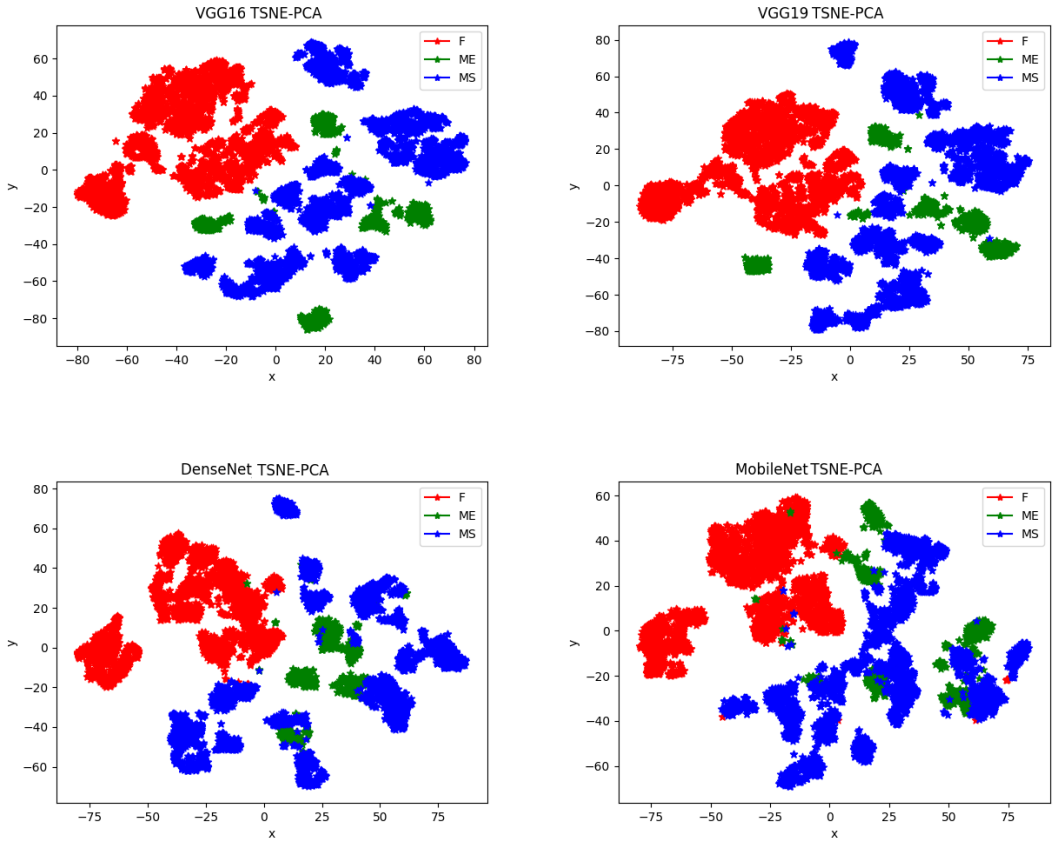
Where *layer* is the selected output layer.
In this paper the networks was analyzed at three levels of depth:

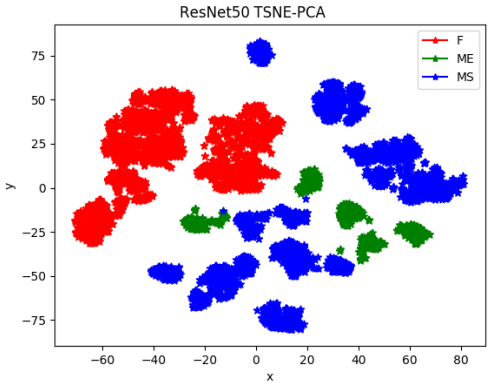
- Last layer (as in previous chapter).
- 75% of the net.
- 50% of the net.
- 25% of the net.

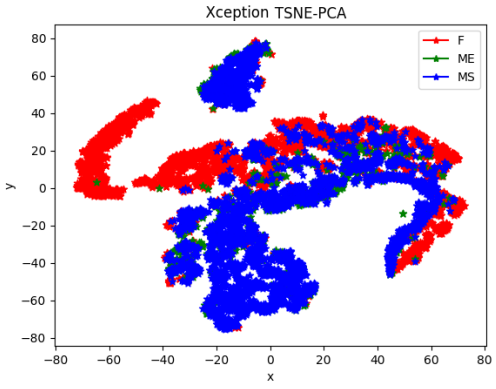
For each case the PCA and t-SNE algorithm are applicated in order to visualize the data in 2D.

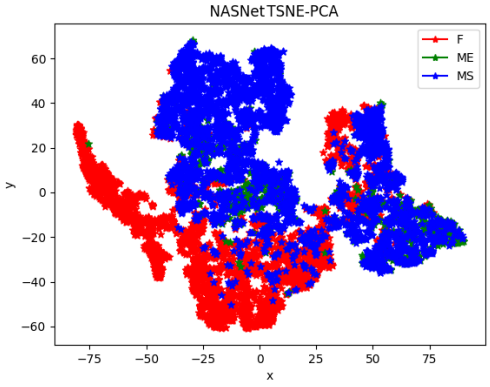
2.4.1 75% of the net

The images obtained with PCA and t-SNE for each model are:



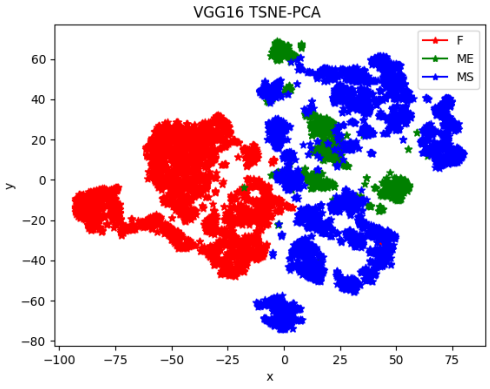


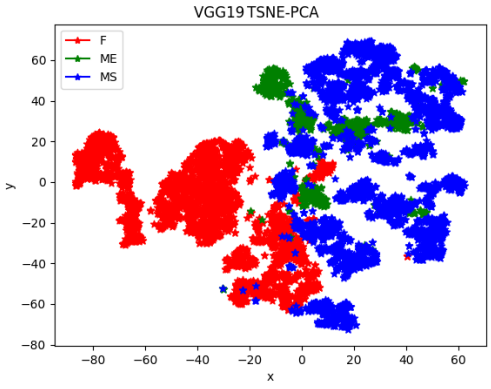


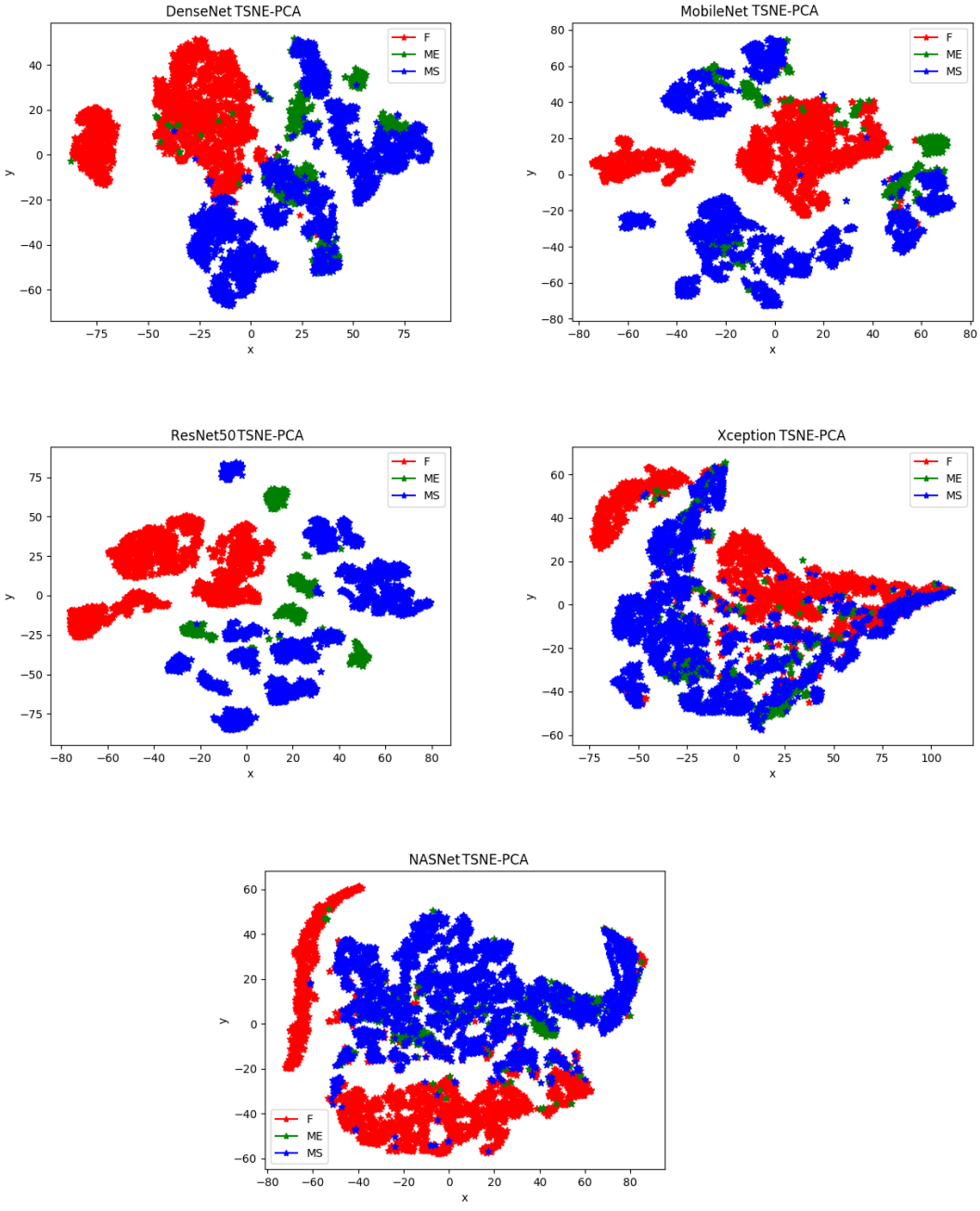


2.4.2 50% of the net

The images obtained with PCA and t-SNE for each model are:

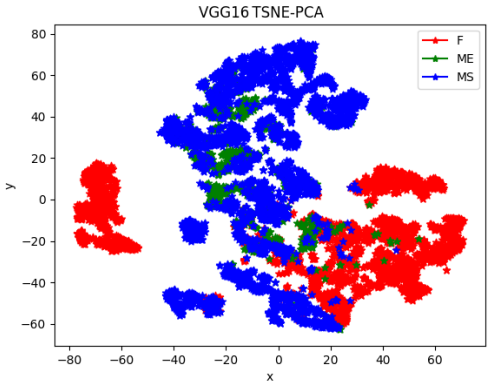


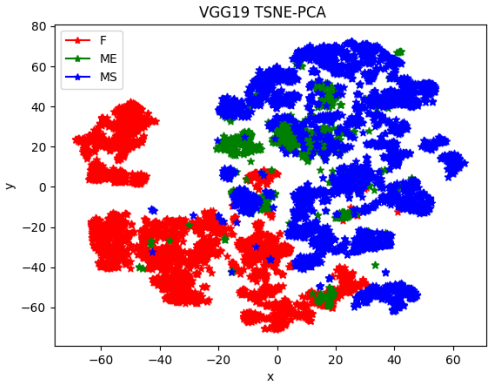


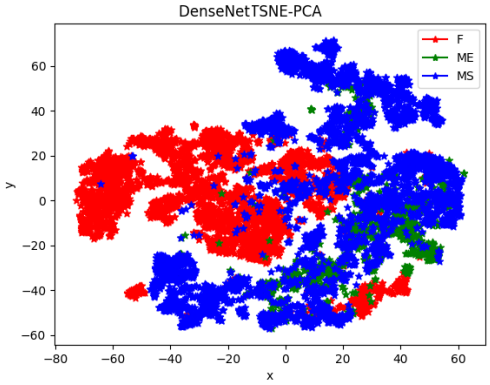


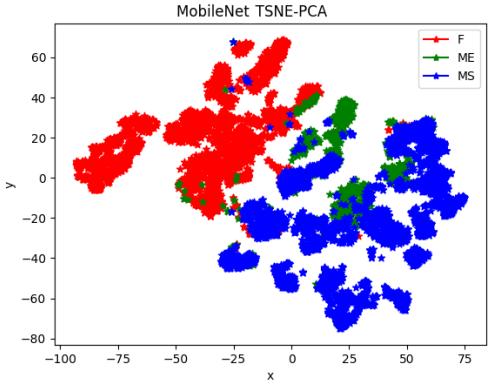
2.4.3 25% of the net

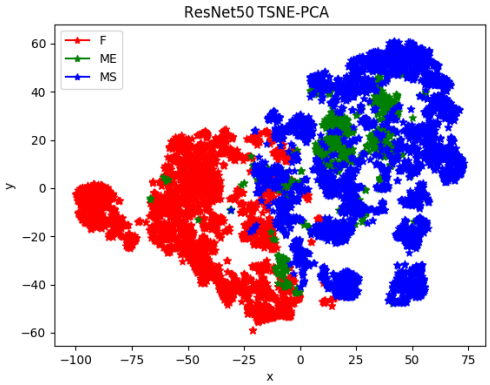
The images obtained with PCA and t-SNE for each model are:

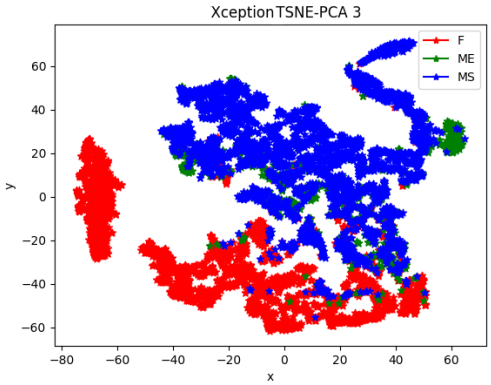

VGG16 TSNE-PCA
F
ME
MS
x
y

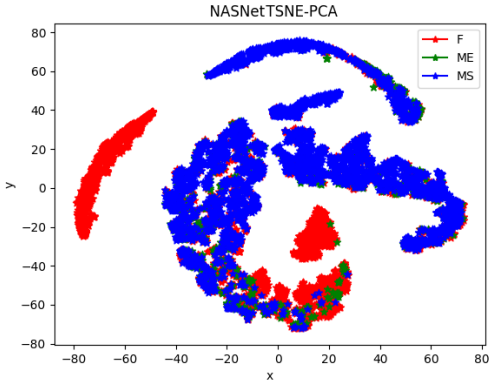

VGG19 TSNE-PCA
F
ME
MS
x
y


DenseNetTSNE-PCA
F
ME
MS
x
y


MobileNet TSNE-PCA
F
ME
MS
x
y


ResNet50 TSNE-PCA
F
ME
MS
x
y


XceptionTSNE-PCA 3
F
ME
MS
x
y


NASNetTSNE-PCA
F
ME
MS
80
60
40
20
0
−20
−40
−60
−80
y
−80
−60
−40
−20
0
20
40
60
80
x

2.5 Supervised Classifier

In order to classify our image a Supervised Classifier is needed.
In our work we used two different classifier:

- k-NN:k-nearest neighbors.
- SVM:Support Vector Machine.

2.5.1 k-NN

The Classifier is based on k-NN method and use the Euclidian distance from points.
The program used is called *KNN.py*.

```
def dist_euclidean(x1, x2, leng):
    dist = 0
    for x in range(leng):
        dist += pow((x1[x] - x2[x]), 2)
    return math.sqrt(dist)


def get_nbr(training, test, k):
    dist = []
    leng = len(test)-1
    for x in range(len(training)):
        dist0 = dist_euclidean(test, training[x], leng)
        dist.append((training[x], dist0))
    dist.sort(key=operator.itemgetter(1))
    nbr = []
    for x in range(k):
        nbr.append(dist[x][0])
    return nbr
```

In particular the function *dist_euclidian* computes the distance. The function *get_nbr* select the neighbour of the new test point. The number of neighbour is selected by the user.
The classifier is able to predict the label of the test point based to his neighbour.

The training set is chosen randomically. The training and test set are made with the function *load_dataset*.

```
def load_dataset(data, n, num_split, training=[], test=[]):

    data_aux = [[0 for x in range(n+1)] for y in range(len(data))]

    for i in range(len(data)):
        item = data[i]
        vals = item['features']
        ls_vals = vals.split(',')

        for j in range(len(ls_vals)):
            data_aux[i][j] = float(ls_vals[j])

        data_aux[i][n] = str(item['cluster'])
        if random.random() < num_split:
            training.append(data_aux[i])
        else:
            test.append(data_aux[i])

    return
```

In our code *num_split* is the percentage of elements used as training set and it can be modify by user. The used value is 0.7. (70% for training set) The other elements are used as test set and a confusion matrix is obtained in order to calculate the average accuracy of the model.

The choice of the number of neighbors (k) is made according these rules:

- k value should be odd.
- k value must not be multiples of the number of classes.
- Should not be too small or too large

An empiric formula to calculate it is to use $sqrt(n)$ where n is the size of the training data (reference `https://arxiv.org/ftp/arxiv/papers/1409/1409.0919.pdf`) but in our case the number is too large (25 neighbors).
For the computational cost of k-NN a small number of neighbour is needed.
We decide to use 5 neighbors because respect the rules and the computational cost is not so large.

2.5.2 SVM

SVM is available in scikit-learn library and follow the structure:Import library, object creation, fitting model and prediction. Is possible to use the library to call SVC method for multiclass classification. The program used is called *SVM.py*. The method analysed are:

- *"ovo":one-vs-one.*
- *"ovr":one-vs-rest.*

With the function *name_svm* is possible to select the method.

```
def name_svm(name):
    if name == 'SVC_ovo':
        return svm.SVC(decision_function_shape='ovo', kernel='linear', C=0.01)
    elif name == 'SVC_ovr':
        return svm.SVC(decision_function_shape='ovr', kernel='linear', C=0.01)
```

After that is possible to train the model and to test it. The test/training set is chosen, as in previous case, with the function *load_dataset*.

2.6 Results

The accuracy of both classifiers are also based on PCA reduction. The number of component of the features, after PCA reduction is chosen by the user. In our work we used the function*data_info*,as mentioned, to try to calculate the optimal number of component for PCA.

PCA optimal number component

Depth	25%	50%	75%	Last layer
VGG16	10	81	355	304
VGG19	24	29	342	301
MobileNet	17	51	157	539
DenseNet	19	22	16	249
ResNet50	49	56	628	940
Xception	18	32	31	262

2.6.1 k-NN

The result of k-NN classifier are reported through a confusion matrix for each pre trained model.
The output are calculted for k=5.

VGG16 k=5

Predicted

Know	F	ME	MS
F	0.999	0	0.001
ME	0	0.99	0.001
MS	0	0.005	0.995

Average Accuracy is 0.99
Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	1	0
MS	0	0.002	0.0.998

Average Accuracy is 0.99
75% of the net

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.993	0.007
MS	0	0.0004	0.996

Average Accuracy is 0.99
50% of the net

	Predicted		
Know	F	ME	MS
F	0.994	0	0.006
ME	0.01	0.85	0.14
MS	0	0.01	0.99

Average Accuracy is 0.94
25% of the net

VGG19 k=5

	Predicted		
Know	F	ME	MS
F	0.999	0	0.001
ME	0	0.99	0.001
MS	0	0.005	0.995

Average Accuracy is 0.99
Last layer

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	1	0
MS	0	0.002	0.0.998

Average Accuracy is 0.99
75% of the net

	Predicted		
Know	F	ME	MS
F	0.998	0	0.002
ME	0.01	0.96	0.03
MS	0.002	0.005	0.993

Average Accuracy is 0.98
50% of the net

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0.01	0.94	0.05
MS	0	0.01	0.99

Average Accuracy is 0.97
25% of the net

DenseNet k=5

	Predicted		
Know	F	ME	MS
F	0.989	0	0.001
ME	0.02	0.97	0.01
MS	0.001	0.03	0.996

Average Accuracy is 0.99
Last layer

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0.008	0.95	0.041
MS	0	0.004	0.0.996

Average Accuracy is 0.98
75% of the net

Predicted

Know	F	ME	MS
F	0.995	0	0.005
ME	0.01	0.92	0.06
MS	0.01	0.02	0.97

Average Accuracy is 0.96

50% of the net

Predicted

Know	F	ME	MS
F	0.98	00.005	0.015
ME	0.01	0.81	0.18
MS	0.01	0.004	0.95

Average Accuracy is 0.91

25% of the net

MobileNet k=5

Predicted

Know	F	ME	MS
F	0.998	0	0.002
ME	0.01	0.91	0.08
MS	0.002	0.003	0.995

Average Accuracy is 0.97

Last layer

Predicted

Know	F	ME	MS
F	0.998	0	0.002
ME	0.014	0.95	0.046
MS	0	0.004	0.0.996

Average Accuracy is 0.98

75% of the net

Predicted

Know	F	ME	MS
F	0.994	0.001	0.005
ME	0.01	0.94	0.05
MS	0.003	0.006	0.991

Average Accuracy is 0.97

50% of the net

Predicted

Know	F	ME	MS
F	0.995	00.002	0.003
ME	0.02	0.94	0.04
MS	0.001	0.001	0.98

Average Accuracy is 0.97

25% of the net

ResNet50 k=5

Predicted

Know	F	ME	MS
F	18	0	0
ME	0	1	0
MS	0	0.003	0.997

Average Accuracy is 0.99

Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	1	0
MS	0	0.002	0.0.998

Average Accuracy is 0.99

75% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.98	0.02
MS	0	0.006	0.994

Average Accuracy is 0.99

50% of the net

Predicted

Know	F	ME	MS
F	0.994	00.002	0.004
ME	0.01	0.92	0.07
MS	0	0.02	0.98

Average Accuracy is 0.95

25% of the net

Xception k=5

Predicted

Know	F	ME	MS
F	0.96	0	0.04
ME	0.1	0.59	0.31
MS	0.03	0.04	0.93

Average Accuracy is 0.83

Last layer

Predicted

Know	F	ME	MS
F	0.94	0.02	0.04
ME	0.08	0.66	0.26
MS	0.02	0.04	0.0.94

Average Accuracy is 0.84

75% of the net

Predicted

Know	F	ME	MS
F	0.95	0.01	0.04
ME	0.06	0.81	0.13
MS	0.01	0.02	0.97

Average Accuracy is 0.91

50% of the net

Predicted

Know	F	ME	MS
F	0.98	0.002	0.01
ME	0.02	0.83	0.15
MS	0.01	0.01	0.98

Average Accuracy is 0.93

25% of the net

NASNet k=5

Predicted

Know	F	ME	MS
F	0.73	0	0.27
ME	0.1	0.15	0.75
MS	0.13	0.01	0.86

Average Accuracy is 0.58

Last layer

Predicted

Know	F	ME	MS
F	0.94	0.006	0.054
ME	0.04	0.61	0.35
MS	0.03	0.05	0.0.92

Average Accuracy is 0.82

75% of the net

	Predicted		
Know	F	ME	MS
F	0.95	0.005	0.04
ME	0.04	0.75	0.22
MS	0.003	0.057	0.94

Average Accuracy is 0.88

50% of the net

	Predicted		
Know	F	ME	MS
F	0.94	0.02	0.04
ME	0.03	0.73	0.24
MS	0.01	0.04	0.95

Average Accuracy is 0.87

25% of the net

2.6.2 SVM

The result of SVM classifier are reported through a confusion matrix for each pre trained model.
The output are calculted for "ovo" method and "ovr" method.

VGG16 "ovo" method

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.91	0.08
MS	0	0.004	0.995

Average Accuracy is 0.96
Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	1	0
MS	0	0.018	0.998

Average Accuracy is 0.99
75% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.98	0.015
MS	0	0.004	0.996

Average Accuracy is 0.99
50% of the net

Predicted

Know	F	ME	MS
F	9.98e-01	0.02	1e-03
ME	1.075e-02	7.27e-01	2.61e-01
MS	08.96e-04	3.76e-02	9.61e-01

Average Accuracy is 0.89
25% of the net

VGG16 "ovr" method

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.96	0.04
MS	9.22e-4	0.006	0.93

Average Accuracy is 0.98
Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.985	0.015
MS	0	0.009	0.0.991

Average Accuracy is 0.99
75% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.98	0.02
MS	0	0.004	0.996

Average Accuracy is 0.99

50% of the net

Predicted

Know	F	ME	MS
F	9.98e-01	0	2e-03
ME	8e-03	6.91e-01	3e-01
MS	8.79e-04	2.99e-02	9.69e-01

Average Accuracy is 0.88

25% of the net

VGG19 "ovo" method

Predicted

Know	F	ME	MS
F	1	0	0
ME	3.73e-03	0.95	0.047
MS	9.22e-4	0.006	0.93

Average Accuracy is 0.98

Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.997	0.003
MS	0	0.004	0.0.996

Average Accuracy is 0.99

75% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.96	0.04
MS	8.76e-4	4.38e-3	0.994

Average Accuracy is 0.98

50% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.89	0.11
MS	0	0.03	0.97

Average Accuracy is 0.95

25% of the net

VGG19 "ovr" method

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.95	0.05
MS	9.25e-4	0.009	0.89

Average Accuracy is 0.98

Last layer

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.996	0.004
MS	0	0.002	0.0.997

Average Accuracy is 0.99

75% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.94	0.06
MS	0	5e-3	0.995

Average Accuracy is 0.98

50% of the net

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.89	0.11
MS	0	0.02	0.98

Average Accuracy is 0.95

25% of the net

DenseNet "ovo" method

Predicted

Know	F	ME	MS
F	0.97	0.003	1
ME	0	0.96	0.04
MS	0.009	0.02	0.981

Average Accuracy is 0.98

Last layer

Predicted

Know	F	ME	MS
F	0.99	0.002	0.004
ME	0.003	0.73	0.267
MS	0.001	0.04	0.96

Average Accuracy is 0.89

75% of the net

Predicted

Know	F	ME	MS
F	0.76	0	0.24
ME	0.03	0	0.97
MS	0.007	0	0.993

Average Accuracy is 0.58

50% of the net

Predicted

Know	F	ME	MS
F	0.20	0	0.80
ME	0	0	1
MS	0.05	0	0.995

Average Accuracy is 0.40

25% of the net

DenseNet "ovr" method

Predicted

Know	F	ME	MS
F	1	0	0
ME	0	0.93	0.07
MS	0	0.02	0.98

Average Accuracy is 0.97

Last layer

Predicted

Know	F	ME	MS
F	0.995	0	0.004
ME	0.01	0.78	0.21
MS	0.001	0.04	0.96

Average Accuracy is 0.91

75% of the net

Predicted

Know	F	ME	MS
F	0.73	0	0.27
ME	0.03	0	0.97
MS	0.0099	0	0.993

Average Accuracy is 0.58

50% of the net

Predicted

Know	F	ME	MS
F	0.02	0	0.88
ME	0	0	1
MS	0.008	0	0.992

Average Accuracy is 0.37

25% of the net

MobileNet "ovo" method

Predicted

Know	F	ME	MS
F	0.995	0.001	0.003
ME	0.02	0.84	0.14
MS	0.002	0.018	0.98

Average Accuracy is 0.94

Last layer

Predicted

Know	F	ME	MS
F	0.996	0.002	0.002
ME	0.04	0.39	0.56
MS	0.003	0.001	0.996

Average Accuracy is 0.80

75% of the net

Predicted

Know	F	ME	MS
F	0.995	0.001	0.004
ME	0.03	0.75	0.22
MS	0.017	0.005	0.99

Average Accuracy is 0.91

50% of the net

Predicted

Know	F	ME	MS
F	0.988	0.002	0.01
ME	0.05	0.36	0.59
MS	0.011	0.013	0.976

Average Accuracy is 0.77

25% of the net

MobileNet "ovr" method

Predicted

Know	F	ME	MS
F	0.997	0.001	0.002
ME	0.01	0.86	0.12
MS	0.002	0.015	0.983

Average Accuracy is 0.95

Last layer

Predicted

Know	F	ME	MS
F	0.995	0.003	0.002
ME	0.04	0.38	0.57
MS	0.003	0	0.997

Average Accuracy is 0.79

75% of the net

	Predicted		
Know	F	ME	MS
F	0.998	0.001	0.001
ME	0.03	0.77	0.2
MS	0	0.013	0.987

Average Accuracy is 0.91
50% of the net

	Predicted		
Know	F	ME	MS
F	0.985	0.002	0.013
ME	0.04	0.31	0.65
MS	0.008	0.009	0.982

Average Accuracy is 0.76
25% of the net

ResNet50 "ovo" method

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.97	0.03
MS	0	0.002	0.998

Average Accuracy is 0.99
Last layer

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.97	0.03
MS	0	0	1

Average Accuracy is 0.99
75% of the net

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.93	0.07
MS	0	0.008	0.992

Average Accuracy is 0.97
50% of the net

	Predicted		
Know	F	ME	MS
F	0.976	0	0.023
ME	0.09	0	0.91
MS	0.008	0	0.992

Average Accuracy is 0.65
25% of the net

ResNet50 "ovr" method

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.98	0.02
MS	0	0.004	0.996

Average Accuracy is 0.99
Last layer

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.96	0.04
MS	0	0.003	0.997

Average Accuracy is 0.98
75% of the net

	Predicted		
Know	F	ME	MS
F	1	0	0
ME	0	0.94	0.06
MS	0	0.002	0.998

Average Accuracy is 0.98

50% of the net

	Predicted		
Know	F	ME	MS
F	0.978	0	0.022
ME	0.099	0	0.90
MS	0.004	0	0.996

Average Accuracy is 0.66

25% of the net

Xception "ovo" method

	Predicted		
Know	F	ME	MS
F	0.97	0.005	0.025
ME	0.03	0.70	0.27
MS	0.03	0.04	0.927

Average Accuracy is 0.87

Last layer

	Predicted		
Know	F	ME	MS
F	0.96	0.005	0.037
ME	0.067	0.51	0.423
MS	0.03	0.05	0.92

Average Accuracy is 0.80

75% of the net

	Predicted		
Know	F	ME	MS
F	0.97	0.005	0.02
ME	0.03	0.60	0.37
MS	0.02	0.05	0.93

Average Accuracy is 0.84

50% of the net

	Predicted		
Know	F	ME	MS
F	0.98	0.01	0.01
ME	0.02	0.80	0.18
MS	0.006	0.30	0.97

Average Accuracy is 0.92

25% of the net

Xception "ovr" method

	Predicted		
Know	F	ME	MS
F	0.96	0.007	0.023
ME	0.05	0.70	0.25
MS	0.02	0.04	0.944

Average Accuracy is 0.87

Last layer

	Predicted		
Know	F	ME	MS
F	0.95	0.01	0.04
ME	0.05	0.55	0.4
MS	0.03	0.06	0.91

Average Accuracy is 0.80

75% of the net

Predicted

Know	F	ME	MS
F	0.96	0.003	0.037
ME	0.038	0.61	0.348
MS	0.022	0.043	0.935

Average Accuracy is 0.84

50% of the net

Predicted

Know	F	ME	MS
F	0.992	0.003	0.005
ME	0.025	0.785	0.19
MS	0.01	0.03	0.96

Average Accuracy is 0.91

25% of the net

NASNet "ovo" method

Predicted

Know	F	ME	MS
F	0.83	0	0.017
ME	0.156	0.152	0.692
MS	0.1	0.02	0.88

Average Accuracy is 0.62

Last layer

Predicted

Know	F	ME	MS
F	0.98	0.001	0.01
ME	0.012	0.37	0.61
MS	0.004	0.03	0.966

Average Accuracy is 0.77

75% of the net

Predicted

Know	F	ME	MS
F	0.93	0	0.067
ME	0.06	0	0.94
MS	0.04	0	0.96

Average Accuracy is 0.63

50% of the net

Predicted

Know	F	ME	MS
F	0.89	0	0.11
ME	0.07	0	0.93
MS	0.08	0	0.92

Average Accuracy is 0.60

25% of the net

NASNet "ovr" method

Predicted

Know	F	ME	MS
F	0.84	0	0.016
ME	0.12	0.13	0.75
MS	0.01	0.01	0.89

Average Accuracy is 0.62

Last layer

Predicted

Know	F	ME	MS
F	0.988	0.002	0.01
ME	0.032	0.38	0.58
MS	0.01	0.04	0.95

Average Accuracy is 0.77

75% of the net

	Predicted		
Know	F	ME	MS
F	0.94	0	0.06
ME	0.03	0	0.97
MS	0.04	0	0.96

Average Accuracy is 0.63

50% of the net

	Predicted		
Know	F	ME	MS
F	0.87	0	0.13
ME	0.08	0	0.92
MS	0.09	0	0.91

Average Accuracy is 0.59

25% of the net

3 Visual Categorization with Bags of Keypoints

The bag of keypoints method was created by Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, Cédric Bray in 2004.
The main steps of the method are:

- Detection and description of image patches
- Assigning patch descriptors to a set of predetermined clusters (a vocabulary) with a vector quantization algorithm
- Constructing a bag of keypoints, which counts the number of patches assigned to each cluster
- Applying a multi-class classifier, treating the bag of keypoints as the feature vector, and thus determine which category or categories to assign to the image.

MatLab 2017 provide a tool to use this method.
Two different cases are analyzed:

- The 30% of images from each set are used for the training data and the remainder, 70%, for the validation data.
- The 70% of images from each set are used for the training data and the remainder, 30%, for the validation data.

Is possible to compute the confusion matrix in both case:

First case is:

	Predicted		
Know	F	ME	MS
F	0.95	0.02	0.03
ME	0.01	0.83	0.16
MS	0.04	0.20	0.76

Average Accuracy is 0.84

Second case is:

	Predicted		
Know	F	ME	MS
F	0.96	0.01	0.03
ME	0.03	0.87	0.11
MS	0.05	0.19	0.75

Average Accuracy is 0.86

Is now possible to compare our result with the result of MatLab tool.

4 Conclusion