ImageClassificationReport

January 31, 2023

1 Import initial libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

2 Dataset

Name : Cifar-10

Type : Image (RGB)

Task : Multi-class(10) classification

description: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

[2]:
```
from tensorflow.keras.datasets import cifar10
```

[3]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

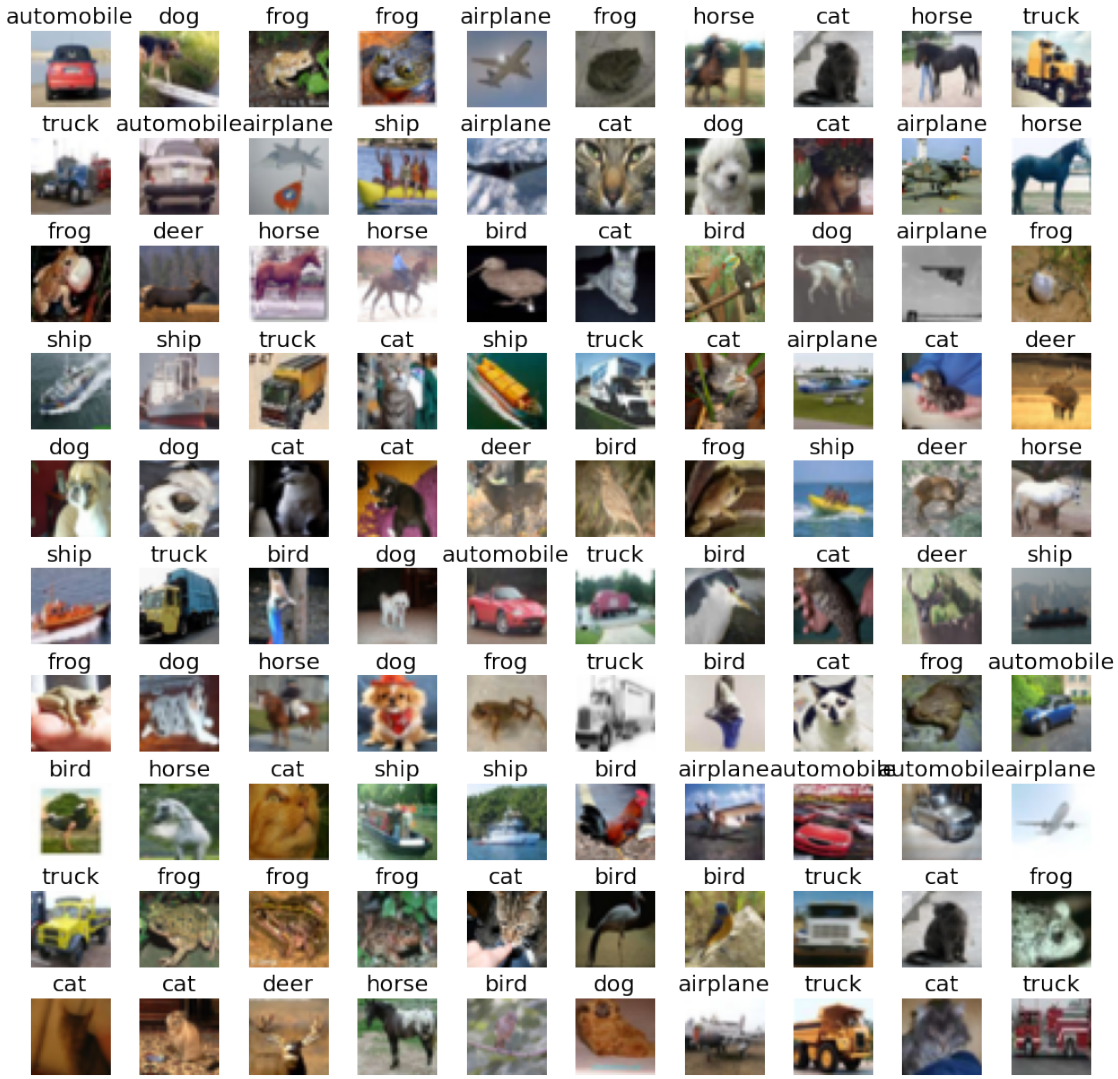
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 7s 0us/step
```

3 EDA

[4]:
```
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

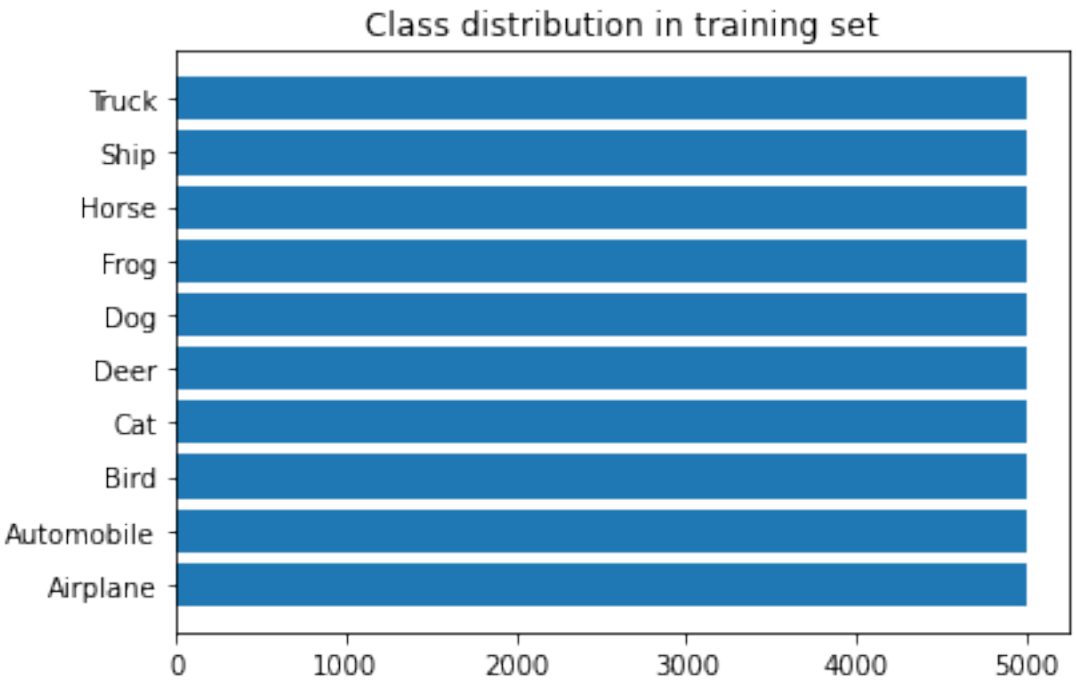
```
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

[5]:
```
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']
W_grid = 10
L_grid = 10


fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))

axes = axes.ravel()

n_train = len(X_train)

for i in np.arange(0, W_grid * L_grid):

    index = np.random.randint(0, n_train)
    axes[i].imshow(X_train[index,1:])
    label_index = int(y_train[index])
    axes[i].set_title(labels[label_index], fontsize = 20)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```



[6]:
```
classes_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',␣
 ↪'Horse', 'Ship', 'Truck']

classes, counts = np.unique(y_train, return_counts=True)
plt.barh(classes_name, counts)
plt.title('Class distribution in training set')
```

[6]:
```
Text(0.5, 1.0, 'Class distribution in training set')
```



4 Data Preprocessing

4.1 Scaling the matrices:

[7]:
```
X_train = X_train / 255
X_test = X_test / 255
```

4.2 OneHotEncoding dor labels:

[8]:
```
from tensorflow.keras.utils import to_categorical
```

[9]:
```
y_cat_train = to_categorical(y_train, 10)
y_cat_test = to_categorical(y_test, 10)
```

5 CNN Model

[10]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout,␣
 ↪BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# from sklearn.metrics import ConfusionMatrixDisplay
# from sklearn.metrics import classification_report, confusion_matrix
```

[11]:
```
def plot_history(history, title):

    plt.figure(figsize=(15,7))
    # Plot training & validation accuracy values
    plt.subplot(121)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(122)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

[12]:
```
cnn = Sequential()
cnn.add(Conv2D(64, (3,3), input_shape=(32,32,3), activation='relu'))
cnn.add(Dropout(0.2))

cnn.add(Conv2D(64, (3,3), activation='relu'))
cnn.add(Dropout(0.2))

cnn.add(Conv2D(64, (3,3), activation='relu'))
cnn.add(Dropout(0.2))

cnn.add(Flatten())
cnn.add(Dense(128, activation='relu'))
cnn.add(Dense(10, activation='softmax'))
```

[13]:
```
METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=METRICS)
```

[14]:
```
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 dropout (Dropout)           (None, 30, 30, 64)        0

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        36928

 dropout_1 (Dropout)         (None, 28, 28, 64)        0

 conv2d_2 (Conv2D)           (None, 26, 26, 64)        36928

 dropout_2 (Dropout)         (None, 26, 26, 64)        0

 flatten (Flatten)           (None, 43264)             0

 dense (Dense)               (None, 128)               5537920

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 5,614,858
Trainable params: 5,614,858
Non-trainable params: 0
_________________________________________________________________
```

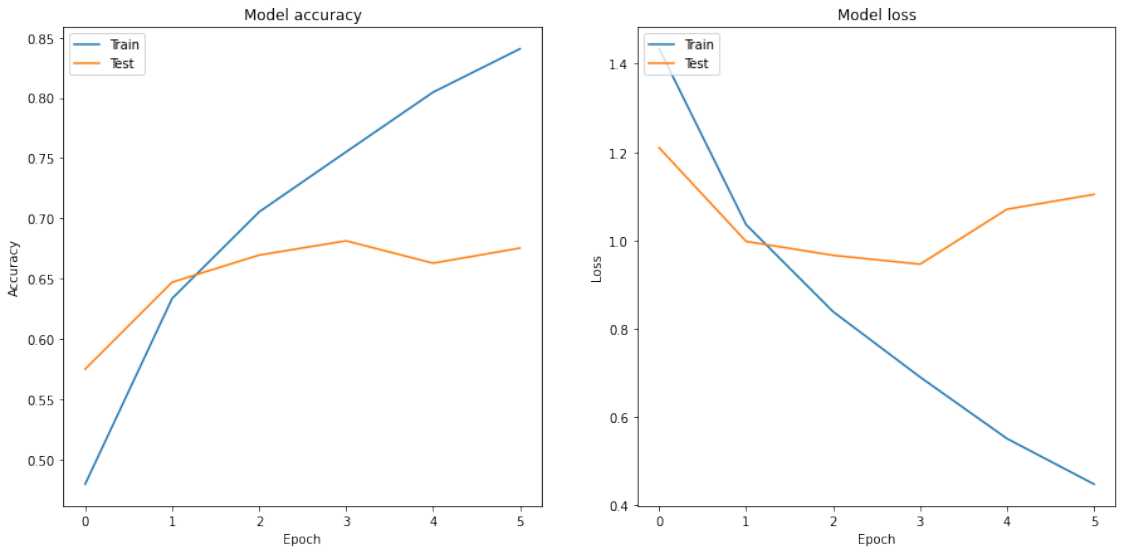
[15]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
```

[16]:
```
BATCH_SIZE = 32
EPOCHS = 10

model_history = cnn.fit(X_train, y_cat_train,
                        epochs = EPOCHS,
                        batch_size = BATCH_SIZE,
                        validation_data = (X_test, y_cat_test),
                        callbacks=[early_stop])
```

```
Epoch 1/10
1563/1563 [==============================] - 30s 13ms/step - loss: 1.4338 -
accuracy: 0.4796 - precision: 0.6921 - recall: 0.2757 - val_loss: 1.2097 -
val_accuracy: 0.5750 - val_precision: 0.7259 - val_recall: 0.4245
Epoch 2/10
1563/1563 [==============================] - 20s 13ms/step - loss: 1.0356 -
accuracy: 0.6336 - precision: 0.7624 - recall: 0.5034 - val_loss: 0.9977 -
val_accuracy: 0.6471 - val_precision: 0.7703 - val_recall: 0.5318
Epoch 3/10
```

```
1563/1563 [==============================] - 19s 12ms/step - loss: 0.8387 -
accuracy: 0.7054 - precision: 0.8025 - recall: 0.6118 - val_loss: 0.9663 -
val_accuracy: 0.6696 - val_precision: 0.7676 - val_recall: 0.5847
Epoch 4/10
1563/1563 [==============================] - 20s 13ms/step - loss: 0.6906 -
accuracy: 0.7552 - precision: 0.8265 - recall: 0.6850 - val_loss: 0.9462 -
val_accuracy: 0.6814 - val_precision: 0.7613 - val_recall: 0.6151
Epoch 5/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.5513 -
accuracy: 0.8047 - precision: 0.8567 - recall: 0.7567 - val_loss: 1.0706 -
val_accuracy: 0.6629 - val_precision: 0.7242 - val_recall: 0.6121
Epoch 6/10
1563/1563 [==============================] - 17s 11ms/step - loss: 0.4482 -
accuracy: 0.8407 - precision: 0.8765 - recall: 0.8052 - val_loss: 1.1041 -
val_accuracy: 0.6754 - val_precision: 0.7219 - val_recall: 0.6352
```

[17]:
```
plot_history(model_history, 'CNN model')
```



[ ]:
```
cnn.save("/content/model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.
```

[ ]:
```
!zip -r /content/model.zip /content/model
```

```
  adding: content/model/ (stored 0%)
  adding: content/model/assets/ (stored 0%)
  adding: content/model/variables/ (stored 0%)
```

```
  adding: content/model/variables/variables.index (deflated 67%)
  adding: content/model/variables/variables.data-00000-of-00001 (deflated 25%)
  adding: content/model/saved_model.pb (deflated 88%)
  adding: content/model/keras_metadata.pb (deflated 91%)
```

6 Extract miss classifications

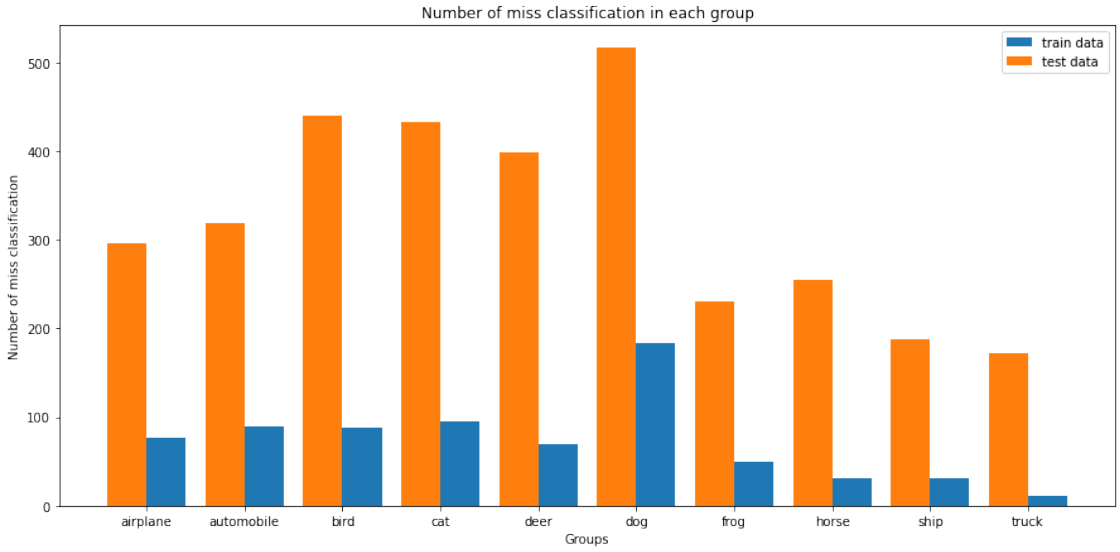
[48]:
```python
def evaluation_model(model, X_train, X_test, y_train, y_test):
  predictions = model.predict(X_test)
  predict_class = np.argmax(predictions, axis=1)
  predict_class = predict_class.tolist()
  predict_class = np.array(predict_class)
  y_test = y_test.reshape(1, -1)[0]

  miss_x_test = []
  miss_y_test = []
  for i in range(y_test.shape[0]):
    if y_test[i] != predict_class[i]:
      miss_x_test.append(X_test[i])
      miss_y_test.append((y_test[i], predict_class[i]))


  predictions = model.predict(X_train)
  predict_class = np.argmax(predictions, axis=1)
  predict_class = predict_class.tolist()
  predict_class = np.array(predict_class)
  y_train = y_train.reshape(1, -1)[0]

  miss_x_train = []
  miss_y_train = []
  for i in range(y_test.shape[0]):
    if y_train[i] != predict_class[i]:
      miss_x_train.append(X_train[i])
      miss_y_train.append((y_train[i], predict_class[i]))

  miss_count_per_class_train = {label:0 for label in labels}
  for miss in miss_y_train:
    miss_count_per_class_train[labels[miss[0]]] += 1

  print (miss_count_per_class_train)
  miss_count_per_class_test = {label:0 for label in labels}
  for miss in miss_y_test:
    miss_count_per_class_test[labels[miss[0]]] += 1

  return miss_count_per_class_train, miss_count_per_class_test

def miss_plot(miss1, miss2):
```

```
    plt.figure(figsize=(15,7))

    data = miss1
    names = list(data.keys())
    values = list(data.values())
    X_axis = np.arange(len(data))
    plt.bar(X_axis + 0.2, values,0.4, tick_label=names, label = "train data")

    data = miss2
    names = list(data.keys())
    values = list(data.values())

    X_axis = np.arange(len(data))
    plt.bar(X_axis - 0.2, values,0.4,  tick_label=names, label = "test data")

    plt.xticks(X_axis, names)
    plt.xlabel("Groups")
    plt.ylabel("Number of miss classification")
    plt.title("Number of miss classification in each group")
    plt.legend()

    plt.show()
```

[23]:
```
miss_count_train, miss_count_test = evaluation_model(cnn, X_train, X_test,␣
↪y_train, y_test)
```

```
313/313 [==============================] - 1s 3ms/step
1563/1563 [==============================] - 4s 3ms/step
{'airplane': 76, 'automobile': 90, 'bird': 88, 'cat': 95, 'deer': 69, 'dog':
183, 'frog': 50, 'horse': 31, 'ship': 31, 'truck': 11}
```

[49]:
```
miss_plot(miss_count_train, miss_count_test)
```



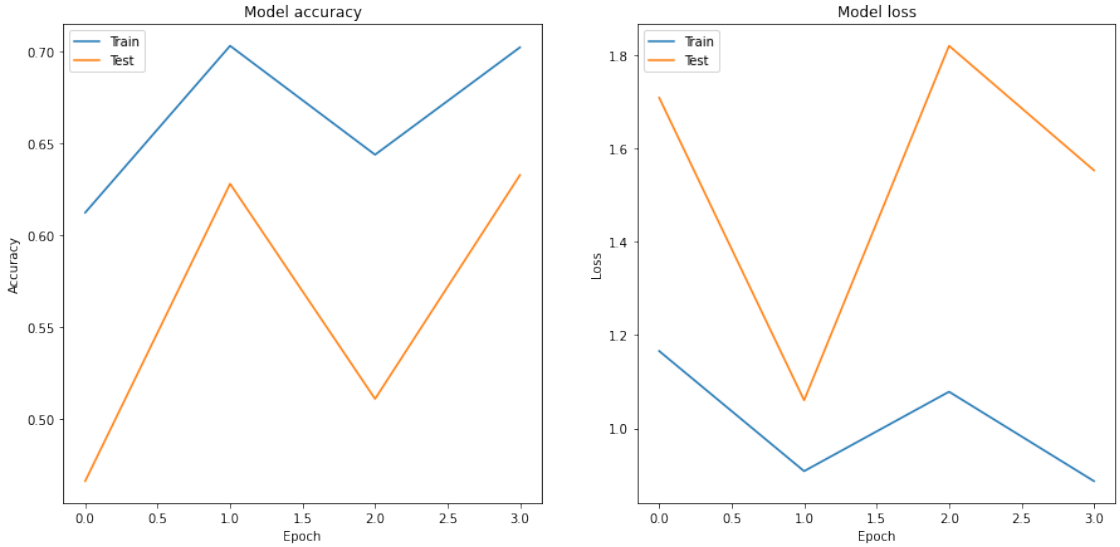
7 DenseNet model

[55]:
```
from keras.applications.densenet import DenseNet121
model = Sequential()
base_model = DenseNet121(input_shape=(32, 32, 3), include_top=False,␣
 ↪weights='imagenet', pooling='avg')
model.add(base_model)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

r = model.fit(X_train, y_cat_train,
              epochs=100,
              batch_size=32,
              validation_data=(X_test, y_cat_test),
                callbacks=[early_stop],
             )
```

```
Epoch 1/100
1563/1563 [==============================] - 100s 56ms/step - loss: 1.1657 -
accuracy: 0.6121 - val_loss: 1.7085 - val_accuracy: 0.4659
Epoch 2/100
1563/1563 [==============================] - 83s 53ms/step - loss: 0.9083 -
accuracy: 0.7029 - val_loss: 1.0602 - val_accuracy: 0.6278
Epoch 3/100
1563/1563 [==============================] - 83s 53ms/step - loss: 1.0785 -
accuracy: 0.6436 - val_loss: 1.8196 - val_accuracy: 0.5107
Epoch 4/100
```

```
1563/1563 [==============================] - 84s 54ms/step - loss: 0.8869 -
accuracy: 0.7021 - val_loss: 1.5527 - val_accuracy: 0.6326
```

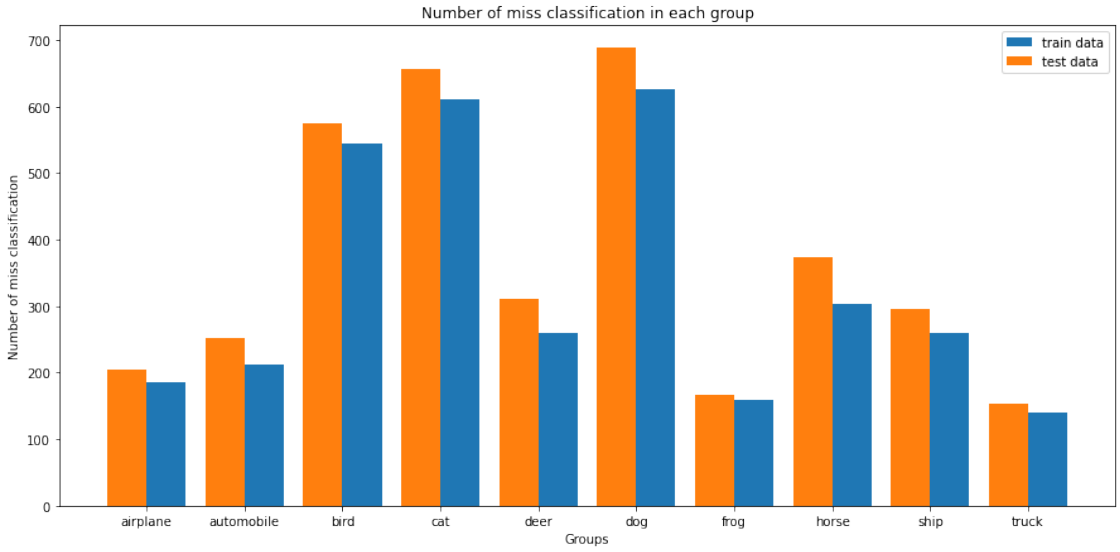
[58]:
```
plot_history(r, 'DenseNet model')
```



[56]:
```
miss_count_train, miss_count_test = evaluation_model(model, X_train, X_test,␣
↪y_train, y_test)
```

```
313/313 [==============================] - 7s 17ms/step
1563/1563 [==============================] - 23s 15ms/step
{'airplane': 186, 'automobile': 213, 'bird': 544, 'cat': 611, 'deer': 259,
'dog': 627, 'frog': 160, 'horse': 303, 'ship': 260, 'truck': 141}
```

[57]:
```
miss_plot(miss_count_train, miss_count_test)
```



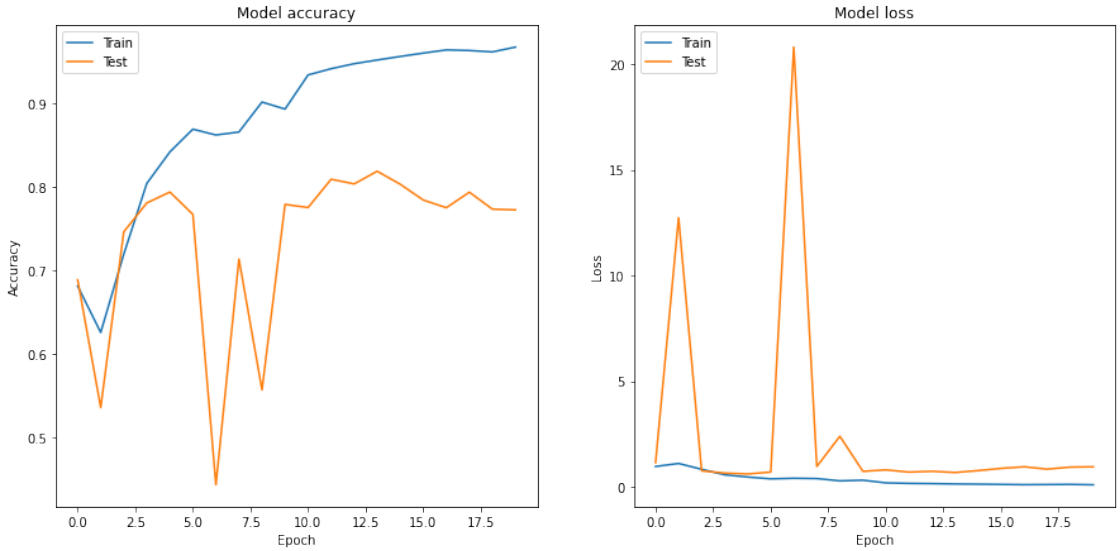
What if we train the model with out stopping it after some amount of epochs

[60]:
```
model = Sequential()
base_model = DenseNet121(input_shape=(32, 32, 3), include_top=False,␣
 ↪weights='imagenet', pooling='avg')
model.add(base_model)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

r = model.fit(X_train, y_cat_train,
              epochs=20,
              batch_size=100,
              validation_data=(X_test, y_cat_test),
             )
```

```
Epoch 1/20
500/500 [==============================] - 41s 62ms/step - loss: 0.9614 -
accuracy: 0.6815 - val_loss: 1.1440 - val_accuracy: 0.6888
Epoch 2/20
500/500 [==============================] - 29s 58ms/step - loss: 1.1043 -
accuracy: 0.6259 - val_loss: 12.7320 - val_accuracy: 0.5360
Epoch 3/20
500/500 [==============================] - 35s 71ms/step - loss: 0.8283 -
accuracy: 0.7195 - val_loss: 0.7547 - val_accuracy: 0.7463
Epoch 4/20
500/500 [==============================] - 30s 60ms/step - loss: 0.5678 -
accuracy: 0.8043 - val_loss: 0.6499 - val_accuracy: 0.7809
Epoch 5/20
```

```
500/500 [==============================] - 29s 58ms/step - loss: 0.4616 -
accuracy: 0.8419 - val_loss: 0.6081 - val_accuracy: 0.7940
Epoch 6/20
500/500 [==============================] - 31s 62ms/step - loss: 0.3761 -
accuracy: 0.8693 - val_loss: 0.6964 - val_accuracy: 0.7673
Epoch 7/20
500/500 [==============================] - 30s 60ms/step - loss: 0.4036 -
accuracy: 0.8623 - val_loss: 20.8043 - val_accuracy: 0.4435
Epoch 8/20
500/500 [==============================] - 30s 59ms/step - loss: 0.3893 -
accuracy: 0.8659 - val_loss: 0.9633 - val_accuracy: 0.7137
Epoch 9/20
500/500 [==============================] - 31s 61ms/step - loss: 0.2808 -
accuracy: 0.9017 - val_loss: 2.3894 - val_accuracy: 0.5571
Epoch 10/20
500/500 [==============================] - 29s 58ms/step - loss: 0.3108 -
accuracy: 0.8933 - val_loss: 0.7316 - val_accuracy: 0.7792
Epoch 11/20
500/500 [==============================] - 29s 58ms/step - loss: 0.1859 -
accuracy: 0.9342 - val_loss: 0.8005 - val_accuracy: 0.7754
Epoch 12/20
500/500 [==============================] - 29s 58ms/step - loss: 0.1613 -
accuracy: 0.9417 - val_loss: 0.6953 - val_accuracy: 0.8093
Epoch 13/20
500/500 [==============================] - 30s 59ms/step - loss: 0.1509 -
accuracy: 0.9476 - val_loss: 0.7335 - val_accuracy: 0.8037
Epoch 14/20
500/500 [==============================] - 29s 58ms/step - loss: 0.1343 -
accuracy: 0.9521 - val_loss: 0.6764 - val_accuracy: 0.8189
Epoch 15/20
500/500 [==============================] - 30s 60ms/step - loss: 0.1244 -
accuracy: 0.9563 - val_loss: 0.7663 - val_accuracy: 0.8033
Epoch 16/20
500/500 [==============================] - 30s 60ms/step - loss: 0.1124 -
accuracy: 0.9603 - val_loss: 0.8707 - val_accuracy: 0.7844
Epoch 17/20
500/500 [==============================] - 30s 60ms/step - loss: 0.1012 -
accuracy: 0.9641 - val_loss: 0.9460 - val_accuracy: 0.7752
Epoch 18/20
500/500 [==============================] - 29s 58ms/step - loss: 0.1069 -
accuracy: 0.9634 - val_loss: 0.8351 - val_accuracy: 0.7938
Epoch 19/20
500/500 [==============================] - 30s 60ms/step - loss: 0.1137 -
accuracy: 0.9617 - val_loss: 0.9317 - val_accuracy: 0.7735
Epoch 20/20
500/500 [==============================] - 29s 59ms/step - loss: 0.0954 -
accuracy: 0.9675 - val_loss: 0.9452 - val_accuracy: 0.7726
```

[61]:
```
plot_history(r, 'DenseNet model with 20 epochs')
```



[62]:
```
miss_count_train, miss_count_test = evaluation_model(model, X_train, X_test,␣
↪y_train, y_test)
```

```
313/313 [==============================] - 6s 14ms/step
1563/1563 [==============================] - 22s 14ms/step
{'airplane': 50, 'automobile': 46, 'bird': 74, 'cat': 205, 'deer': 126, 'dog':
104, 'frog': 41, 'horse': 95, 'ship': 82, 'truck': 152}
```

[63]:
```
miss_plot(miss_count_train, miss_count_test)
```