Creating a prediction model by implementation using Convolutional Neural Network (CNN)

```
# Imports
import numpy as np
import cv2
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
import keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
import tensorflow as tf
```

```
# Link Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive
```

Load the data from the folder of .npy files

Load the numpy array images onto the memory

```
# Load the data
cat = np.load('/content/drive/MyDrive/1 2564/CP461 Computer Vision/Lab/Lab14/data_cat.npy')
dog = np.load('/content/drive/MyDrive/1 2564/CP461 Computer Vision/Lab/Lab14/data_dog.npy')
sheep = np.load('/content/drive/MyDrive/1 2564/CP461 Computer Vision/Lab/Lab14/data_sheep.npy')

print(cat.shape)
print(dog.shape)
print(sheep.shape)
```

```
(123202, 784)
(152159, 784)
(126121, 784)
```

```
# Add a column with labels, 0=cat, 1=dog, 2=sheep
cat = np.c_[cat, np.zeros(len(cat))]
dog = np.c_[dog, np.ones(len(dog))]
sheep = np.c_[sheep, 2*np.ones(len(sheep))]

print(cat.shape)
print(dog.shape)
print(sheep.shape)
```

```
(123202, 785)
(152159, 785)
(126121, 785)
```

Show the example data

```
# Function to plot 28x28 pixel drawings that are stored in a numpy array.
# Specify how many rows and cols of pictures to display (default 4x5).
# If the array contains less images than subplots selected, surplus subplots remain empty.
import random

def plot_samples(input_array, rows=1, cols=10, title=''):
    fig, ax = plt.subplots(figsize=(cols,rows))
    ax.axis('off')
    plt.title(title)

    for i in list(range(0, min(len(input_array),(rows*cols)) )):
        a = fig.add_subplot(rows,cols,i+1)
        rand_i = random.randint(0,input_array.shape[0])
        imgplot = plt.imshow(input_array[rand_i,:784].reshape((28,28)), cmap='gray_r', interpolation='nearest')
        plt.xticks([])
        plt.yticks([])


plot_samples(cat, title='Sample cat drawings\n')
plot_samples(dog, title='Sample dog drawings\n')
plot_samples(sheep, title='Sample sheep drawings\n')
```

Create the 9,000 training and 9,000 testing sample data

```
# merge the cat and sheep arrays,
# and split the features (X) and labels (y).
# Convert to float32 to save some memory.
X = np.concatenate((cat[:6000,:-1], dog[:6000,:-1], sheep[:6000,:-1]), axis=0).astype('float32') # all columns b
y = np.concatenate((cat[:6000,-1], dog[:6000,-1], sheep[:6000,-1]), axis=0).astype('float32') # the last column

# train/test split (divide by 255 to obtain normalized values between 0 and 1)
# I will use a 50:50 split
X_train, X_test, y_train, y_test = train_test_split(X/255., y, test_size=0.5, random_state=0)

print(X_train.shape," ",X_test.shape)
print(y_train.shape," ",y_test)
```

```
(9000, 784)   (9000, 784)
(9000,)   [0. 1. 0. ... 2. 1. 2.]
```

Convolutional Neural Network (CNN)

Let's try out a Convolutional Neural Network (CNN) with Keras.

I will use a model from this tutorial by Jason Brownlee.

It has the following these layers:

```
# one hot encode outputs
y_train_cnn = np_utils.to_categorical(y_train)
y_test_cnn = np_utils.to_categorical(y_test)
num_classes = y_test_cnn.shape[1]

# reshape to be [samples][pixels][width][height]
# X_train_cnn = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
# X_test_cnn = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')

#"tensorflow"
X_train_cnn = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test_cnn = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# s = X_train_cnn.shape
# print s, num_classes
print(X_train_cnn.shape,", there are ",num_classes, ' classes')
```

```
(9000, 28, 28, 1) , there are  3  classes
```

1. Convolutional layer with 30 feature maps of size 5×5.
2. Pooling layer taking the max over 2*2 patches.
3. Convolutional layer with 15 feature maps of size 3×3.

4. Pooling layer taking the max over 2*2 patches.
5. Dropout layer with a probability of 20%.
6. Flatten layer.
7. Fully connected layer with 128 neurons and rectifier activation.
8. Fully connected layer with 50 neurons and rectifier activation.
9. Output layer.

```
# define the CNN model
def cnn_model():
    # create model
    model = Sequential()

    # input_shape=(3,224,224) #"theano"
    # input_shape=(224,224,3). #"tensorflow"

    # 1. Convolutional layer with 30 feature maps of size 5×5.
    model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    # 2. Pooling layer taking the max over 2*2 patches.
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # 3. Convolutional layer with 15 feature maps of size 3×3.
    model.add(Conv2D(15, (3, 3), activation='relu'))
    # 4. Pooling layer taking the max over 2*2 patches.
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # 5. Dropout layer with a probability of 20%.
    model.add(Dropout(0.2))

    # 6. Flatten layer.
    model.add(Flatten())

    # 7. Fully connected layer with 128 neurons and rectifier activation.
    model.add(Dense(128, activation='relu'))

    # 8. Fully connected layer with 50 neurons and rectifier activation.
    model.add(Dense(50, activation='relu'))

    # 9. Output layer.
    model.add(Dense(num_classes, activation='softmax'))

    # Compile model (Normal LR)
    # model.compile(loss='categorical_crossentropy',
    #               optimizer='Adam',
    #               metrics=['accuracy'])

    # Compile model (Define LR)
    opt = tf.keras.optimizers.Adam(learning_rate=0.001) # Set optimize function
    model.compile(loss='categorical_crossentropy'
                 , optimizer=opt
                 , metrics=['accuracy']
                 )
    return model
```

```
  # specificity, sensitivity
```

```
# build the model
model = cnn_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 24, 24, 30)        780

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 30)       0
 2D)

 conv2d_3 (Conv2D)           (None, 10, 10, 15)        4065

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 15)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 15)          0

 flatten_1 (Flatten)         (None, 375)               0

 dense_3 (Dense)             (None, 128)               48128

 dense_4 (Dense)             (None, 50)                6450

 dense_5 (Dense)             (None, 3)                 153

=================================================================
Total params: 59,576
Trainable params: 59,576
Non-trainable params: 0
_________________________________________________________________
```

Train a model

By using epochs=10

```
%%time

# Fit the model
history = model.fit(X_train_cnn, y_train_cnn,
          validation_data=(X_test_cnn, y_test_cnn),
          epochs=30,
          batch_size=32)
```

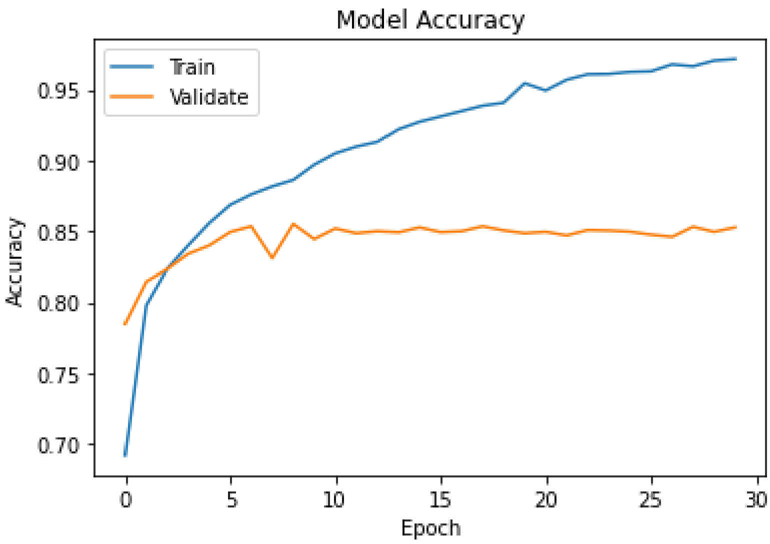
```
Epoch 1/30
282/282 [==============================] - 11s 37ms/step - loss: 0.6993 - accuracy
Epoch 2/30
282/282 [==============================] - 8s 28ms/step - loss: 0.4870 - accuracy:
```

```
Epoch 3/30
282/282 [==============================] - 8s 28ms/step - loss: 0.4262 - accuracy:
Epoch 4/30
282/282 [==============================] - 8s 28ms/step - loss: 0.3923 - accuracy:
Epoch 5/30
282/282 [==============================] - 8s 28ms/step - loss: 0.3596 - accuracy:
Epoch 6/30
282/282 [==============================] - 8s 28ms/step - loss: 0.3324 - accuracy:
Epoch 7/30
282/282 [==============================] - 8s 28ms/step - loss: 0.3139 - accuracy:
Epoch 8/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2980 - accuracy:
Epoch 9/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2816 - accuracy:
Epoch 10/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2585 - accuracy:
Epoch 11/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2439 - accuracy:
Epoch 12/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2333 - accuracy:
Epoch 13/30
282/282 [==============================] - 8s 28ms/step - loss: 0.2143 - accuracy:
Epoch 14/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1947 - accuracy:
Epoch 15/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1794 - accuracy:
Epoch 16/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1768 - accuracy:
Epoch 17/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1657 - accuracy:
Epoch 18/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1512 - accuracy:
Epoch 19/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1480 - accuracy:
Epoch 20/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1223 - accuracy:
Epoch 21/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1277 - accuracy:
Epoch 22/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1158 - accuracy:
Epoch 23/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1035 - accuracy:
Epoch 24/30
282/282 [==============================] - 8s 28ms/step - loss: 0.1015 - accuracy:
Epoch 25/30
282/282 [==============================] - 8s 28ms/step - loss: 0.0963 - accuracy:
Epoch 26/30
282/282 [==============================] - 8s 28ms/step - loss: 0.0955 - accuracy:
Epoch 27/30
282/282 [==============================] - 8s 28ms/step - loss: 0.0835 - accuracy:
Epoch 28/30
282/282 [==============================] - 8s 28ms/step - loss: 0.0858 - accuracy:
Epoch 29/30
```

Showing the result

```
# Plotting the history of accuracy
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc = 'upper left')
plt.show()
```




```
# Summarizing the history of loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validate'], loc = 'upper left')
plt.show()
```

Evaluate by testing the trained model with the test data

```
# Final evaluation of the model
scores = model.evaluate(X_test_cnn, y_test_cnn, verbose=0)
print('Final CNN accuracy: ', scores[1]*100, "%")
```

```
Final CNN accuracy:  85.26666760444641 %
```

Show the confusion matrix of testing result

```
from sklearn.metrics import confusion_matrix


predict_x = model.predict(X_test_cnn)
classes_x = np.argmax(predict_x,axis=1)
# print(classes_x.shape)

classes_y = np.argmax(y_test_cnn, axis=1)
# print(classes_y.shape)

cm = confusion_matrix(classes_y, classes_x)
print(cm)
```

```
[[2445  436   56]
 [ 338 2500  194]
 [  56  246 2729]]
```

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

labels = ["Cat", "Dog", "Sheep"]

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cm, classes=labels,
                      title='Confusion matrix, without normalization')
plt.show()

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cm, classes=labels, normalize='True',
                      title='Confusion matrix, with normalization')
plt.show()
```

```
Confusion matrix, without normalization
```

Sensitivity and Specification

```
[ 56 246 2729]]
```

```
def perf_measure(true_classes, prediction):
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    for i in range(len(prediction)):
        if true_classes[i]==prediction[i]==1:
           TP += 1
        if prediction[i]==1 and true_classes[i]!=prediction[i]:
           FP += 1
        if true_classes[i]==prediction[i]==0:
           TN += 1
        if prediction[i]==0 and true_classes[i]!=prediction[i]:
           FN += 1

    return(TP, FP, TN, FN)

TP, FP, TN, FN = perf_measure(classes_y, classes_x)
print(perf_measure(classes_y, classes_x))
print('Precision = ', TP/(TP+FP))
print('Recall or Sensitivity = ', TP/(TP+FN))
print('Specitivity = ', TN/(TN+FP))
```

```
(2500, 682, 2445, 394)
Precision =  0.7856693903205532
Recall or Sensitivity =  0.8638562543192813
Specitivity =  0.7818995842660698
```

Classification report

```
from sklearn.metrics import classification_report
# target_names = ["Class {}".format(i) for i in range(num_classes)]
target_names = ["Cat", "Dog", "Sheep"]
print(classification_report(classes_y, classes_x, target_names=target_names))
```

```
              precision    recall  f1-score   support

         Cat       0.86      0.83      0.85      2937
         Dog       0.79      0.82      0.80      3032
       Sheep       0.92      0.90      0.91      3031

    accuracy                           0.85      9000
   macro avg       0.85      0.85      0.85      9000
weighted avg       0.85      0.85      0.85      9000
```

Save the trained model

```
# Save weights
model.save_weights('/content/drive/MyDrive/model_cnn/save_NN_weight.h5')
model.save('/content/drive/MyDrive/model_cnn/save_NN_model.model')
print("Model is saved")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/model_cnn/save_NN_model.model/assets
Model is saved
```