Parameter tunning for CNN

In [4]:
```
# this part will prevent tensorflow to allocate all the avaliable GPU Memory

import tensorflow as tf
from keras import backend as k

# Don't pre-allocate memory; allocate as-needed
config = tf.ConfigProto()
config.gpu_options.allow_growth = True

# Create a session with the above options specified.
k.tensorflow_backend.set_session(tf.Session(config=config))
```

```
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x magic: more info.
```

```
Using TensorFlow backend.
```

In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import tarfile
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
import time
import math
import cv2
from tqdm import tqdm_notebook as tqdm
from sklearn.metrics import log_loss,confusion_matrix
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D,MaxPooling2D,Conv2D,Dropout,BatchNormalization
from keras import backend as k
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
from keras.utils import np_utils
from keras.preprocessing import image
from sklearn.datasets import load_files
import cv2
import pickle
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,precision_score,recall_score
from statistics import mode
from sklearn.utils.multiclass import unique_labels
```

In [0]:
```
classes = {'0': 'letter',
 '1': 'form',
 '10': 'budget',
 '11': 'invoice',
 '12': 'presentation',
 '13': 'questionnaire',
 '14': 'resume',
 '15': 'memo',
 '2': 'email',
 '3': 'handwritten',
 '4': 'advertisement',
 '5': 'scientific report',
 '6': 'scientific publication',
 '7': 'specification',
 '8': 'file folder',
 '9': 'news article'}
```

In [0]:
```
from keras.callbacks import ReduceLROnPlateau
# function to plot loss
def plot_loss(x,y,x_label,y_label,title):
    sns.barplot(x,y)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.show()

# function to tune optimizers
def tune_optimizers(num_epochs,optimizers,baseline_model):
  loss=[]
  epochs_=[i for i in range(num_epochs)]
  print('Searching for optimizers..')
  for optimizer in optimizers:
    model=baseline_model
    print('Optimizer:',optimizer)
    print('-'*50)
    model.compile(loss = "categorical_crossentropy", optimizer = optimizer, metrics=["accuracy"])
    reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.2, patience=3, min_lr=0.0001,mode='auto')
    callbacks=[reduce_lr]
    print('Training Started')
    history=model.fit_generator(
    train_generator,
    steps_per_epoch =2500//batch_size,
    epochs = num_epochs,
    validation_data = val_generator,
    validation_steps =2500//batch_size,
    verbose=0,callbacks=callbacks)
    print('Training completed.')
    #print('Trining loss:')
    #plot_loss(epochs_,history.history['loss'],'epochs','train loss',optimizer)
    #print('Trining Accuracy:')
```

```
    #plot_loss(epochs_,history.history['acc'],'epochs','train accuracy',optimizer)
    #print('Validation loss:')
    #plot_loss(epochs_,history.history['val_loss'],'epochs','val loss',optimizer)
    #print('Validation Accuracy:')
    #plot_loss(epochs_,history.history['val_acc'],'epochs','val acc',optimizer)
    index=np.argmin(history.history['val_loss'])
    print('Best val loss:',history.history['val_loss'][index])
    loss.append(history.history['val_loss'][index])
  plot_loss(optimizers,loss,'optimizers','val loss','optimizers vs best val_loss')
  print('Best optimizer with current setup:',optimizers[np.argmin(loss)])
  return loss

# function to tune batch size
def tune_batchsize(num_epochs,batch_sizes,baseline_model):
  loss=[]
  epochs_=[i for i in range(num_epochs)]
  print('Searching for batchsize..')
  for batch_size in batch_sizes:
    model=baseline_model
    reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.2, patience=3, min_lr=0.0001,mode='auto')
    callbacks=[reduce_lr]
    print('batch size:',batch_size)
    print('-'*50)
    #model.compile(loss = "categorical_crossentropy", optimizer = 'adam', metrics=["accuracy"])
    print('Training Started..')
    history=model.fit_generator(
    train_generator,
    steps_per_epoch =2500//batch_size,
    epochs = num_epochs,
    validation_data = val_generator,
    validation_steps =2500//(batch_size*2),
    verbose=0,callbacks=callbacks)
    print('Training completed.')
    index=np.argmin(history.history['val_loss'])
    print('Best val loss:',history.history['val_loss'][index])
    loss.append(history.history['val_loss'][index])
  plot_loss(batch_sizes,loss,'batch_sizes','val loss','batch_size vs best val_loss')
  print('Best batch_size with current setup:',batch_sizes[np.argmin(loss)])
  return loss

# function to tune learning rate
def tune_learning_rate(num_epochs,learning_rates,baseline_model):
  loss=[]
  epochs_=[i for i in range(num_epochs)]
  print('Searching for best learning rate..')
  for lr_rate in learning_rates:
    model=baseline_model
    reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.2, patience=3, min_lr=0.0001,mode='auto')
    callbacks=[reduce_lr]
    print('learning rate:',lr_rate)
    print('-'*50)
    adam=keras.optimizers.Adam(lr=lr_rate, beta_1=0.9, beta_2=0.999, amsgrad=False)
    model.compile(loss = "categorical_crossentropy", optimizer = adam, metrics=["accuracy"])
    print('Training Started..')
    history=model.fit_generator(
    train_generator,
    steps_per_epoch =2500//batch_size,
    epochs = num_epochs,
    validation_data = val_generator,
    validation_steps =2500//(batch_size*2),
    verbose=0,callbacks=callbacks)
    print('Training completed.')
    index=np.argmin(history.history['val_loss'])
    print('Best val loss:',history.history['val_loss'][index])
    loss.append(history.history['val_loss'][index])
  plot_loss(learning_rates,loss,'learning_rates','val loss','lerning_rates vs best val_loss')
  print('Best learning rate with current setup and Adam optimizer:',learning_rates[np.argmin(loss)])
  return loss
```

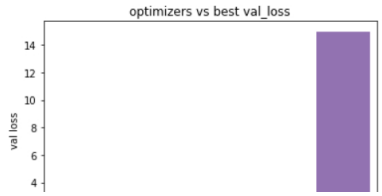
Sample parameter tunning

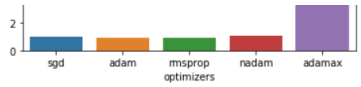
In [0]:
```
# load baseline model at that point
from keras.model import load_model
baseline_model=load_model('vgg16.h5')
```

In [0]:
```
# confihurating hyperparameter ranges
batch_sizes=[32,64,128]
optimizers=['sgd','adam','rmsprop','nadam','adamax']
lr_rates=[0.0001,0.001,0.01,0.08,0.1]
baseline_model=model
num_epochs=10
```

In [126]:
```
optimizer_loss=tune_optimizers(10,optimizers,baseline_model)
```

```
Searching for optimizers..
Optimizer: sgd
Training Started
Training completed.
Best val loss: 0.9950649035282624
Optimizer: adam
Training Started
Training completed.
Best val loss: 0.9617227529868101
Optimizer: rmsprop
Training Started
Training completed.
Best val loss: 0.9239901426510934
Optimizer: nadam
Training Started
Training completed.
Best val loss: 1.0510053038597107
Optimizer: adamax
Training Started
Training completed.
Best val loss: 14.981562858972794
```

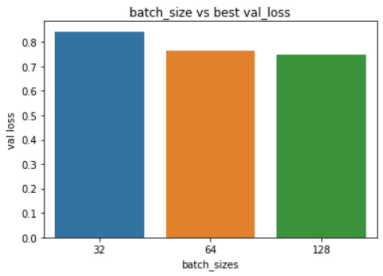





```
Best optimizer with current setup: rmsprop
```

In [148]:
```
batch_loss=tune_batchsize(num_epochs,batch_sizes,model)
```

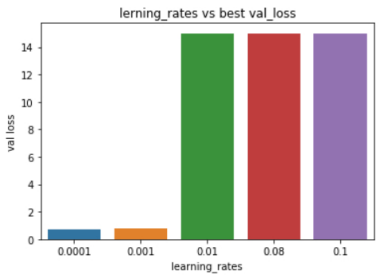
```
Searching for batchsize..
batch size: 32
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 0.8426857269727267
batch size: 64
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 0.7654928502283598
batch size: 128
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 0.7468728555573357
```




```
Best batch_size with current setup: 128
```

In [150]:
```
learlingrate_loss=tune_learning_rate(num_epochs,lr_rates,model)
```

```
Searching for best learning rate..
learning rate: 0.0001
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 0.7527321028081995
learning rate: 0.001
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 0.7593578068833602
learning rate: 0.01
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 15.004674158598247
learning rate: 0.08
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 15.004674158598247
learning rate: 0.1
--------------------------------------------------
Training Started..
Training completed.
Best val loss: 15.004674158598247
```




```
Best learning rate with current setup and Adam optimizer: 0.0001
```