```
#download data and prepare our environment! { display-mode: "form" }
print("Installing packages...")
!pip install hypopt tensorflowjs > /dev/null
print("Downloading files...")
!wget https://www.dropbox.com/s/fedqcdt4o0m2oxp/X.npy &> /dev/null
!wget https://www.dropbox.com/s/h7xh92w1w7px30a/X_g.npy &> /dev/null
!wget https://www.dropbox.com/s/grn9brfvzx74c8a/y.npy &> /dev/null
print("Importing stuff...")
import os
os.makedirs("static/js", exist_ok=True)
!wget -O static/js/skin_cancer_diagnosis_script.js 'https://storage.googleapis.com/inspirit-a
output = 'static/js/skin_cancer_diagnosis_script.js'

from google.colab.output import eval_js

import time
start_time = time.time()

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tqdm.notebook import tqdm

import keras
from keras import backend as K
from tensorflow.keras.layers import *
from keras.models import Sequential
from keras.layers import Dense, Conv2D
from keras.layers import Activation, MaxPooling2D, Dropout, Flatten, Reshape
from keras.wrappers.scikit_learn import KerasClassifier

import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import os
import random
from PIL import Image
import gdown

import argparse
import numpy as np
from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU, ZeroPadding2D, UpSampl
from keras.layers.merge import add, concatenate
from keras.models import Model
import struct
from google.colab.patches import cv2_imshow
from copy import deepcopy
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_auc_score
from sklearn.base import BaseEstimator

from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from keras.applications.mobilenet import MobileNet




from hypopt import GridSearch

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering

import cv2


import tensorflowjs as tfjs

from google.colab import files

import requests, io, zipfile
import os




import os.path
from os import path





print("Done!")
```

```
Installing packages...
Downloading files...
Importing stuff...
Done!
```

```
IMG_WIDTH = 100
IMG_HEIGHT = 75


#Let's load in our data from last time!
X = np.load("X.npy")
X_g = np.load("X_g.npy")
y = np.load("y.npy")


# Perform Data Augmentation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
X_g_train, X_g_test, y_train, y_test = train_test_split(X_g, y, test_size=0.4, random_state=1

X_augmented = []
X_g_augmented = []

y_augmented = []

for i in tqdm(range(len(X_train))):
  transform = random.randint(0,1)
  if (transform == 0):
    # Flip the image across the y-axis
    X_augmented.append(cv2.flip(X_train[i],1))
    X_g_augmented.append(cv2.flip(X_g_train[i],1))
    y_augmented.append(y_train[i])
  else:
    # Zoom 33% into the image
    zoom = 0.33

    centerX,centerY=int(IMG_HEIGHT/2),int(IMG_WIDTH/2)
    radiusX,radiusY= int((1-zoom)*IMG_HEIGHT*2),int((1-zoom)*IMG_WIDTH*2)

    minX,maxX=centerX-radiusX,centerX+radiusX
    minY,maxY=centerY-radiusY,centerY+radiusY

    cropped = (X_train[i])[minX:maxX, minY:maxY]
    new_img = cv2.resize(cropped, (IMG_WIDTH, IMG_HEIGHT))
    X_augmented.append(new_img)

    cropped = (X_g_train[i])[minX:maxX, minY:maxY]
    new_img = cv2.resize(cropped, (IMG_WIDTH, IMG_HEIGHT))
    X_g_augmented.append(new_img)

    y_augmented.append(y_train[i])
```

```
X_augmented = np.array(X_augmented)
X_g_augmented = np.array(X_g_augmented)

y_augmented = np.array(y_augmented)

X_train = np.vstack((X_train,X_augmented))
X_g_train = np.vstack((X_g_train,X_g_augmented))

y_train = np.append(y_train,y_augmented)
```

```
    100%                                          580/580 [00:06<00:00, 95.70it/s]
```

```
#VIEWING SHAPE OF VARIABLES AFTER DATA AUGMENTATION
print(X_train.shape)
print(X_g_train.shape)
print(y_train.shape)
```

```
     (1160, 75, 100, 3)
     (1160, 75, 100)
     (1160,)
```

```
def CNNClassifier(epochs=1, batch_size=10, layers=5, dropout=0.5, activation='relu'):
  def set_params():
    i = 1
  def create_model():
    model = Sequential()
    model.add(Reshape((IMG_WIDTH, IMG_HEIGHT, 3)))

    for i in range(layers):
      model.add(Conv2D(64, (3, 3), padding='same'))
      model.add(Activation(activation))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation(activation))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout / 2.0))

    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation(activation))
    model.add(Conv2D(128, (3, 3)))
    model.add(Activation(activation))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(dropout / 2.0))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation(activation))
    model.add(Dropout(dropout))
    model.add(Dense(7))
    model.add(Activation('softmax'))

    # initiate RMSprop optimizer
    opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

    # Let's train the model using RMSprop
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=[tf.keras.metrics.AUC()])
    return model
  return KerasClassifier(build_fn=create_model, epochs=epochs, batch_size=batch_size, verbose
```

```
#transform our y labels into one hot encoded labels for training.

y_train_onehot = np.zeros((y_train.size, y_train.max().astype(int)+1))
y_train_onehot[np.arange(y_train.size),y_train.astype(int)] = 1

y_test_onehot = np.zeros((y_test.size, y_test.max().astype(int)+1))
y_test_onehot[np.arange(y_test.size),y_test.astype(int)] = 1
```

```
#initialize and train our CNN

cnn = CNNClassifier()

cnn.fit(X_train.astype(np.float32), y_train_onehot.astype(np.float32),
        validation_data=(X_test.astype(np.float32),y_test_onehot.astype(np.float32))
        ,verbose=1)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v
  "The `lr` argument is deprecated, use `learning_rate` instead.")
116/116 [==============================] - 61s 59ms/step - loss: 6.1329 - auc: 0.5273 -
<keras.callbacks.History at 0x7fef96753ed0>
```

```
#Let's save and download our trained model, so that we can use it in a web app later on.

tfjs.converters.save_keras_model(cnn.model, 'cnn_model')
```

```
# Click here to define model_stats()
def model_stats(name, y_test, y_pred, y_pred_proba):
  cm = confusion_matrix(y_test, y_pred)

  print(name)

  accuracy = accuracy_score(y_test,y_pred)
  print ("The accuracy of the model is " + str(round(accuracy,5)))

  y_test_onehot = np.zeros((y_test.size, y_test.max().astype(int)+1))
  y_test_onehot[np.arange(y_test.size),y_test.astype(int)] = 1
  roc_score = roc_auc_score(y_test_onehot, y_pred_proba)

  print ("The ROC AUC Score of the model is " + str(round(roc_score,5)))

  return cm


# use this above function to evaluate model
y_pred = cnn.predict(X_test)
y_pred_proba = cnn.predict_proba(X_test)

cnn_cm = model_stats("CNN",y_test,y_pred,y_pred_proba)
```

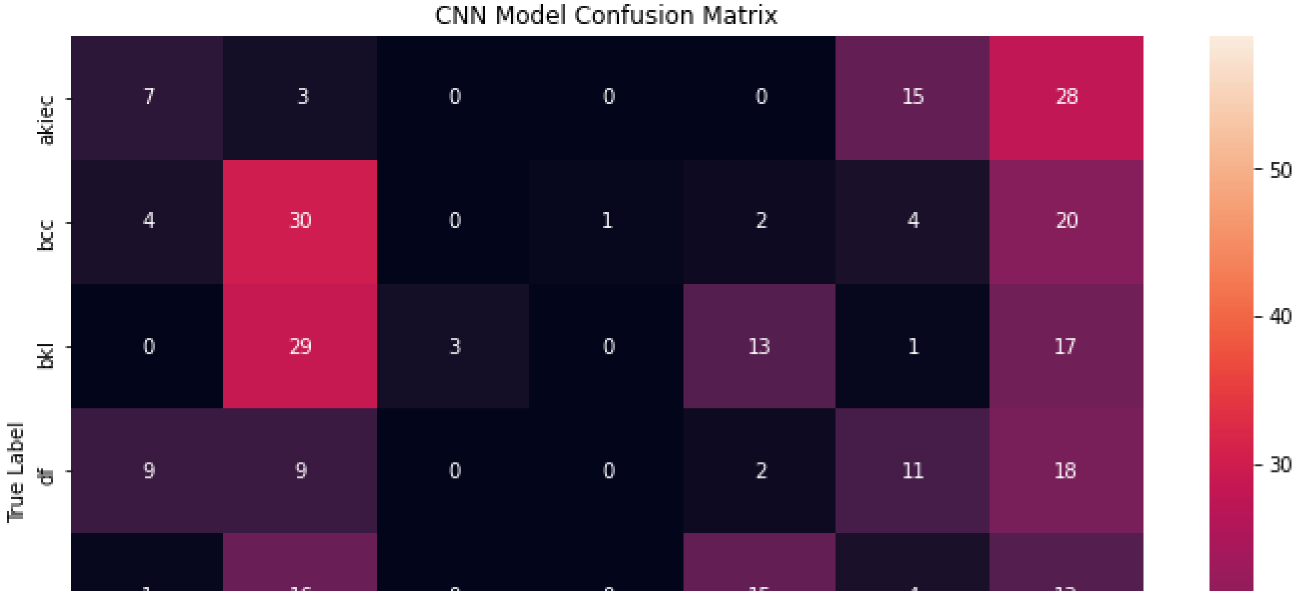
```
/usr/local/lib/python3.7/dist-packages/keras/engine/sequential.py:450: UserWarning: `moc
  warnings.warn('`model.predict_classes()` is deprecated and '
39/39 [==============================] - 1s 11ms/step
39/39 [==============================] - 0s 9ms/step
CNN
The accuracy of the model is 0.3385
The ROC AUC Score of the model is 0.73265
```

```
# redefine the plot_cm() function from  first notebook.
def plot_cm(name, cm):
  classes = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

  df_cm = pd.DataFrame(cm, index = [i for i in classes], columns = [i for i in classes])
  df_cm = df_cm.round(5)

  plt.figure(figsize = (12,8))
  sns.heatmap(df_cm, annot=True, fmt='g')
  plt.title(name + " Model Confusion Matrix")
  plt.xlabel("Predicted Label")
  plt.ylabel("True Label")
  plt.show()


#plot confusion matrix
plot_cm("CNN", cnn_cm)
```



It looks like our custom CNN's performance is better than the Logistic Regression, KNN, and Decision Tree models. More training epochs or a bigger dataset would probably help with the performance.

```
#FIRST, DEFINE CNN MODEL HERE:

def CNNClassifier_Modified(epochs=10, batch_size=10, layers=5, dropout=0.5, activation='relu'
  def set_params():
    i = 1
  def create_model():
    model = Sequential()
    model.add(Reshape((IMG_WIDTH, IMG_HEIGHT, 3)))

    # Your Code Here
    model.add(Flatten())
    model.add(Dense(7))
    model.add(Activation('softmax'))

    # initiate RMSprop optimizer
    opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

    # Let's train the model using RMSprop
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=[tf.keras.metrics.AUC()])
    return model
  return KerasClassifier(build_fn=create_model, epochs=epochs, batch_size=batch_size, verbose


#TRAIN MODEL

cnn = CNNClassifier_Modified()

cnn.fit(X_train.astype(np.float32), y_train_onehot.astype(np.float32),
cnn.fit(X_train.astype(np.float32), y_train_onehot.astype(np.float32),
        validation_data=(X_test.astype(np.float32),y_test_onehot.astype(np.float32))
        ,verbose=1, epochs = 5)
```

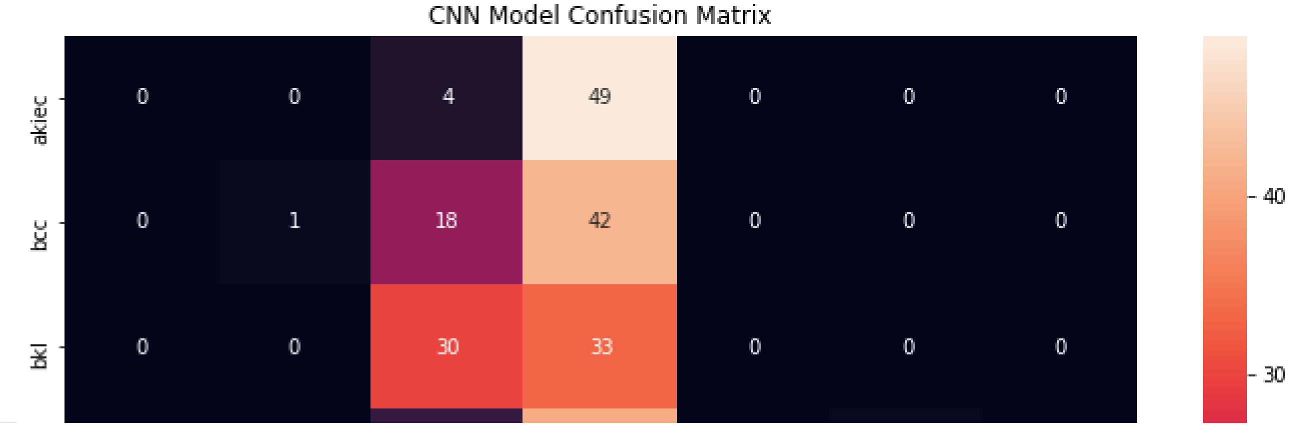
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2
  "The `lr` argument is deprecated, use `learning_rate` instead.")
Epoch 1/5
116/116 [==============================] - 2s 9ms/step - loss: 486.2185 - auc_1: 0.5285
Epoch 2/5
116/116 [==============================] - 1s 5ms/step - loss: 329.3897 - auc_1: 0.5480
Epoch 3/5
116/116 [==============================] - 1s 5ms/step - loss: 312.2527 - auc_1: 0.5566
Epoch 4/5
116/116 [==============================] - 1s 5ms/step - loss: 285.5937 - auc_1: 0.5553
Epoch 5/5
116/116 [==============================] - 1s 5ms/step - loss: 294.9322 - auc_1: 0.5580
<keras.callbacks.History at 0x7fef95d25950>
```

```
#EVALUATE MODEL PERFORMANCE

y_pred = cnn.predict(X_test)
y_pred_proba = cnn.predict_proba(X_test)
cnn_cm = model_stats("CNN",y_test,y_pred,y_pred_proba)
```

```
39/39 [==============================] - 0s 1ms/step
/usr/local/lib/python3.7/dist-packages/keras/engine/sequential.py:450: UserWarning: `moc
  warnings.warn('`model.predict_classes()` is deprecated and '
39/39 [==============================] - 0s 1ms/step
CNN
The accuracy of the model is 0.25323
The ROC AUC Score of the model is 0.58126
```

```
#FINALLY, PLOT CONFUSION MATRIX BELOW
plot_cm("CNN", cnn_cm)
```


CNN Model Confusion Matrix

```
#USE GRID SEARCH WITH CNN model
#In the variable param_grid we can specify which parameters in our CNN we want to modify.

param_grid = {
              'epochs' :              [10, 20, 30],
              'batch_size' :          [32, 64,128],
              'layers' :              [1, 3, 5],
              'dropout' :             [0.2, 0.3, 0.5],
              'activation' :          ['relu', 'elu']
             }
```

```
# redefine parameter grid to have ONLY 4 possible combos for simplicity
param_grid = {
              'epochs' :              [1, 2],
              'dropout' :             [0.2, 0.3],
             }


# create a validation slice in our dataset.

X_test_small, X_val, y_test_small, y_val = train_test_split(X_test, y_test, test_size=0.5, ra


# Define our Grid Search CNN Class
class gridSearchCNN():

    keras_model = None
    model = Sequential()
    #epochs=10
    epochs=1
    batch_size=10
    layers=5
    dropout=0.5
    activation='relu'

    def __init__(self, **params):
      pass

    def fit(self, X, y, sample_weight = None):
        print("fitting")
        self.keras_model.fit(X,y)
        print("fitted")
        return self.keras_model
    def predict(self, X):
        return self.keras_model.predict(X)
    def predict_proba(self, X):
        return self.keras_model.predict_proba(X)
    def score(self, X, y, sample_weight = None):
        print("scoring")
        #return self.keras_model.score(X,y)
        y_pred_proba = self.keras_model.predict_proba(X)
        roc_auc_score_val = roc_auc_score(y, y_pred_proba)
        print("scored")
        return roc_auc_score_val



    def createKerasCNN(self,):

      def create_model():
        self.model = Sequential()
        self.model.add(Reshape((IMG_WIDTH, IMG_HEIGHT, 3)))

        for i in range(self.layers):
          self.model.add(Conv2D(64, (3, 3), padding='same'))
          self.model.add(Activation(self.activation))

        self.model.add(Conv2D(64, (3, 3)))
        self.model.add(Activation(self.activation))
        self.model.add(MaxPooling2D(pool_size=(2, 2)))
        self.model.add(Dropout(self.dropout / 2.0))

        self.model.add(Conv2D(128, (3, 3), padding='same'))
        self.model.add(Activation(self.activation))
        self.model.add(Conv2D(128, (3, 3)))
        self.model.add(Activation(self.activation))
        self.model.add(MaxPooling2D(pool_size=(2, 2)))
        self.model.add(Dropout(self.dropout / 2.0))

        self.model.add(Flatten())
        self.model.add(Dense(512))
        self.model.add(Activation(self.activation))
        self.model.add(Dropout(self.dropout))
        self.model.add(Dense(7))
        self.model.add(Activation('softmax'))

        # initiate RMSprop optimizer
        opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

        # Let's train the model using RMSprop
        self.model.compile(loss='categorical_crossentropy',
                      optimizer=opt,
                      metrics=[tf.keras.metrics.AUC()])
        return self.model

      return KerasClassifier(build_fn=create_model, epochs=self.epochs,
                            batch_size=self.batch_size, verbose=2)

    def get_params(self, deep = True):
        return {
            'epochs': self.epochs,
            'batch_size': self.batch_size,
            'layers': self.layers,
            'dropout': self.dropout,
            'activation': self.activation
            }

    def set_params(self, **params):
      if 'epochs' in params.keys():
        self.epochs = params['epochs']
      if 'batch_size' in params.keys():
        self.batch_size = params['batch_size']
      if 'layers' in params.keys():
        self.layers = params['layers']
      if 'dropout' in params.keys():
        self.dropout = params['dropout']
      if 'activation' in params.keys():
        self.activation = params['activation']

      self.keras_model = self.createKerasCNN()
      return self


#TRANSFER LEARNING
#machine learning
#take an existing pre-trained network and modify the weights for the top level neurons by tra
#
#
def transfer_learning_model():
  mobilenet_model = MobileNet(input_shape=(IMG_HEIGHT,IMG_WIDTH,3), include_top=False, poolin

  transfer_model = Sequential()
  transfer_model.add(mobilenet_model)
  transfer_model.add(Dropout(0.1))
  transfer_model.add(BatchNormalization())
  transfer_model.add(Dense(256, activation="relu"))
  transfer_model.add(Dropout(0.1))
  transfer_model.add(BatchNormalization())
  transfer_model.add(Dense(7, activation="softmax"))

  # initiate RMSprop optimizer
  opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

  # Let's train the model using RMSprop
  transfer_model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=[tf.keras.metrics.AUC()])

  return transfer_model
```

```
print(X_train.shape)
print(y_train_onehot.shape)

print(X_test.shape)
print(y_test_onehot.shape)
```

```
(1160, 75, 100, 3)
(1160, 7)
(387, 75, 100, 3)
(387, 7)
```

```
transfer_model.fit(X_train.astype(np.float32), y_train_onehot.astype(np.float32),
        validation_data=(X_test.astype(np.float32),y_test_onehot.astype(np.float32))
        ,verbose=1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-e7c8f31d1b2b> in <module>()
----> 1 transfer_model.fit(X_train.astype(np.float32), y_train_onehot.astype(np.float32
      2         validation_data=(X_test.astype(np.float32),y_test_onehot.astype(np.floa
      3         ,verbose=1)

AttributeError: 'function' object has no attribute 'fit'
```

SEARCH STACK OVERFLOW

```
y_pred = transfer_model.predict(X_test)
y_pred_proba = transfer_model.predict_proba(X_test)
transfer_cm = model_stats("Transfer CNN",y_test,y_pred,y_pred_proba)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-4fd0625dfb64> in <module>()
----> 1 y_pred = transfer_model.predict(X_test)
      2 y_pred_proba = transfer_model.predict_proba(X_test)
      3 transfer_cm = model_stats("Transfer CNN",y_test,y_pred,y_pred_proba)

NameError: name 'transfer_model' is not defined
```

SEARCH STACK OVERFLOW

```
plot_cm("Transfer Learning CNN", transfer_cm)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-84d1a1ad3360> in <module>()
----> 1 plot_cm("Transfer Learning CNN", transfer_cm)

NameError: name 'transfer_cm' is not defined
```

SEARCH STACK OVERFLOW

```
tfjs.converters.save_keras_model(transfer_model.model, 'transfer_model')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-32-2fa4ed9b480b> in <module>()
----> 1 tfjs.converters.save_keras_model(transfer_model.model, 'transfer_model')

AttributeError: 'function' object has no attribute 'model'
```

SEARCH STACK OVERFLOW

Great work! We've just developed various ML models to perform classification on our skin lesion dataset! Now, our next step is to package this model into a mobile application. Run the code cell below.

```
!zip -r ./cnn_model.zip ./cnn_model/
```

```
      adding: cnn_model/ (stored 0%)
      adding: cnn_model/group1-shard14of25.bin (deflated 8%)
      adding: cnn_model/model.json (deflated 85%)
      adding: cnn_model/group1-shard11of25.bin (deflated 8%)
      adding: cnn_model/group1-shard17of25.bin (deflated 8%)
      adding: cnn_model/group1-shard23of25.bin (deflated 8%)
      adding: cnn_model/group1-shard3of25.bin (deflated 8%)
      adding: cnn_model/group1-shard8of25.bin (deflated 8%)
      adding: cnn_model/group1-shard22of25.bin (deflated 8%)
      adding: cnn_model/group1-shard16of25.bin (deflated 8%)
      adding: cnn_model/group1-shard6of25.bin (deflated 8%)
      adding: cnn_model/group1-shard15of25.bin (deflated 8%)
      adding: cnn_model/group1-shard9of25.bin (deflated 8%)
      adding: cnn_model/group1-shard2of25.bin (deflated 8%)
      adding: cnn_model/group1-shard19of25.bin (deflated 8%)
      adding: cnn_model/group1-shard7of25.bin (deflated 8%)
      adding: cnn_model/group1-shard24of25.bin (deflated 8%)
      adding: cnn_model/group1-shard5of25.bin (deflated 8%)
      adding: cnn_model/group1-shard25of25.bin (deflated 8%)
      adding: cnn_model/group1-shard1of25.bin (deflated 8%)
      adding: cnn_model/group1-shard20of25.bin (deflated 8%)
      adding: cnn_model/group1-shard10of25.bin (deflated 8%)
      adding: cnn_model/group1-shard21of25.bin (deflated 8%)
      adding: cnn_model/group1-shard12of25.bin (deflated 8%)
      adding: cnn_model/group1-shard4of25.bin (deflated 8%)
```