MUSA650 Final Project Code

Name: Anne Evered

Date: May 2, 2020

Description:

This notebook contains code for analyses used in the final project for MUSA650. The goal of the prop given an image, can detect if the building in that image likely has flood damage. The data consists of Hurricane Harvey. The original source of the data can be found here https://ieee-dataport.org/open-ac hurricane-satellite-imagery-based-customized, but this notebook uses a cleaned and labeled version c https://www.kaggle.com/kmader/satellite-images-of-hurricane-damage

To run:

Load compressed data file using upload button in Google colab.

Note: because of the parameter sweep, this notebook can take several hours to run in entirey. Comme

Sources:

This notebook extends https://www.kaggle.com/yuempark/satellite-images-of-hurricane-damage anc Course Lectures and Example Assignments.

Load in Packages

```
from __future__ import print_function

!pip install tifffile

import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
import numpy as np
import pickle

from os import listdir
from os.path import isfile, join

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

```
from keras import backend as K

from PIL import Image

import tifffile as tiff

from skimage.color import rgb2gray

from keras.optimizers import RMSprop

!pip3 install contextily
!pip3 install geopandas
import geopandas as gpd
from matplotlib import pyplot as plt
from shapely.geometry import Point
%matplotlib inline
import contextily as ctx


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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/drive
```

Read in the Image File Paths

```
#UNCOMMENT TO UNZIP SATELLITE IMAGES
!unzip /content/satellite-images-of-hurricane-damage.zip


Dataset of All Files #####

mn names
 = ["label", "train_test_category", "image_name", "image_path"]

y dataframe
d.DataFrame(columns = column_names)

[]


/content/satellite-images-of-hurricane-damage/"

the folders to look for files in
```

```
y_folders = [dir_path + f for f in listdir(dir_path) if ".DS_Store" not in f]
ies = [f for f in listdir(dir_path) if ".DS_Store" not in f]

the files in subfolders and fill dataset wtih label, image name/path, and train/test 
, folder in zip(file_categories, file_category_folders):
tegories = [f for f in listdir(folder) if ".DS_Store" not in f]
l in label_categories:
s_in_category = [f for f in listdir(join(folder, label)) if (isfile(join(folder, labe
files.extend([join(folder, label, file) for file in files_in_category])
file in files_in_category:
image_df = image_df.append({'label' : label, 'train_test_category': category, 'image_

image_df
```

	label	train_test_category	image_name	
0	no_damage	validation_another	-95.626072_29.866551.jpeg	/conte
1	no_damage	validation_another	-95.62578_29.860985.jpeg	/conte
2	no_damage	validation_another	-95.17174399999999_29.514159000000003.jpeg	/conte
3	no_damage	validation_another	-95.062917_29.830639.jpeg	/conte
4	no_damage	validation_another	-95.631978_29.853457000000002.jpeg	/conte
...	...	...	...	
22995	damage	test_another	-95.175703_30.033993.jpeg	/conte
22996	damage	test_another	-95.616789_29.764003999999996.jpeg	/conte
22997	damage	test_another	-95.589701_29.75866.jpeg	/conte
22998	damage	test_another	-95.669167_29.820797999999996.jpeg	/conte
22999	damage	test_another	-96.872091_28.494535.jpeg	/conte

23000 rows × 4 columns

Exploratory Data Analysis and Data Preparation

Show Some Sample Images

```
#Open a sample image (with no damage label)
im_no_damage = Image.open(image_df.image_path[2])
im_no_damage.show()
```

```
#Open a sample image (with damage label)
im_damage = Image.open(image_df.image_path[22996])
im_damage.show()
```

Get Details of a Sample Image

```
#Load a sample image to better understand some details about that image
!pip install opencv-python
import cv2

# read it in unchanged, to make sure we aren't losing any information
sample_image = cv2.imread(image_df['image_path'][0], cv2.IMREAD_UNCHANGED)
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-pac
```

```
#Print out shape and pixel min/max details about sample image
print(np.shape(sample_image))
print(np.min(sample_image[:,:,:]))
print(np.max(sample_image[:,:,:]))
```

```
(128, 128, 3)
5
217
```

Add and Visualize Location Details

```
#Add in location (lat and long) to dataframe by parsing out file name
image_df['location'] = image_df['image_name'].apply(lambda x: x.replace('.jpeg',''))
image_df['lon'] = image_df['location'].apply(lambda x: float(x.split('_')[0]))
image_df['lat'] = image_df['location'].apply(lambda x: float(x.split('_')[-1]))
image_df.head()
```

	label	train_test_category	image_name	image_
0	no_damage	validation_another	-95.626072_29.866551.jpeg	/content/sa imag hurı dam
1	no_damage	validation_another	-95.62578_29.860985.jpeg	/content/sa imag hurı dam
2	no_damage	validation_another	-95.17174399999999_29.514159000000003.jpeg	/content/sa imag hurı dam
3	no_damage	validation_another	-95.062917_29.830639.jpeg	/content/sa imag hurı dam
4	no_damage	validation_another	-95.631978_29.853457000000002.jpeg	/content/sa imag hurı dam

```
image_df['Coordinates'] = list(zip(image_df['lon'], image_df['lat']))
image_df['Coordinates'] = image_df['Coordinates'].apply(Point)

#Convert to geopandas dataframe
image_gpd = gpd.GeoDataFrame(image_df, geometry="Coordinates", crs={"init": "epsg:385

image_gpd = image_gpd.to_crs(epsg=3857)
```

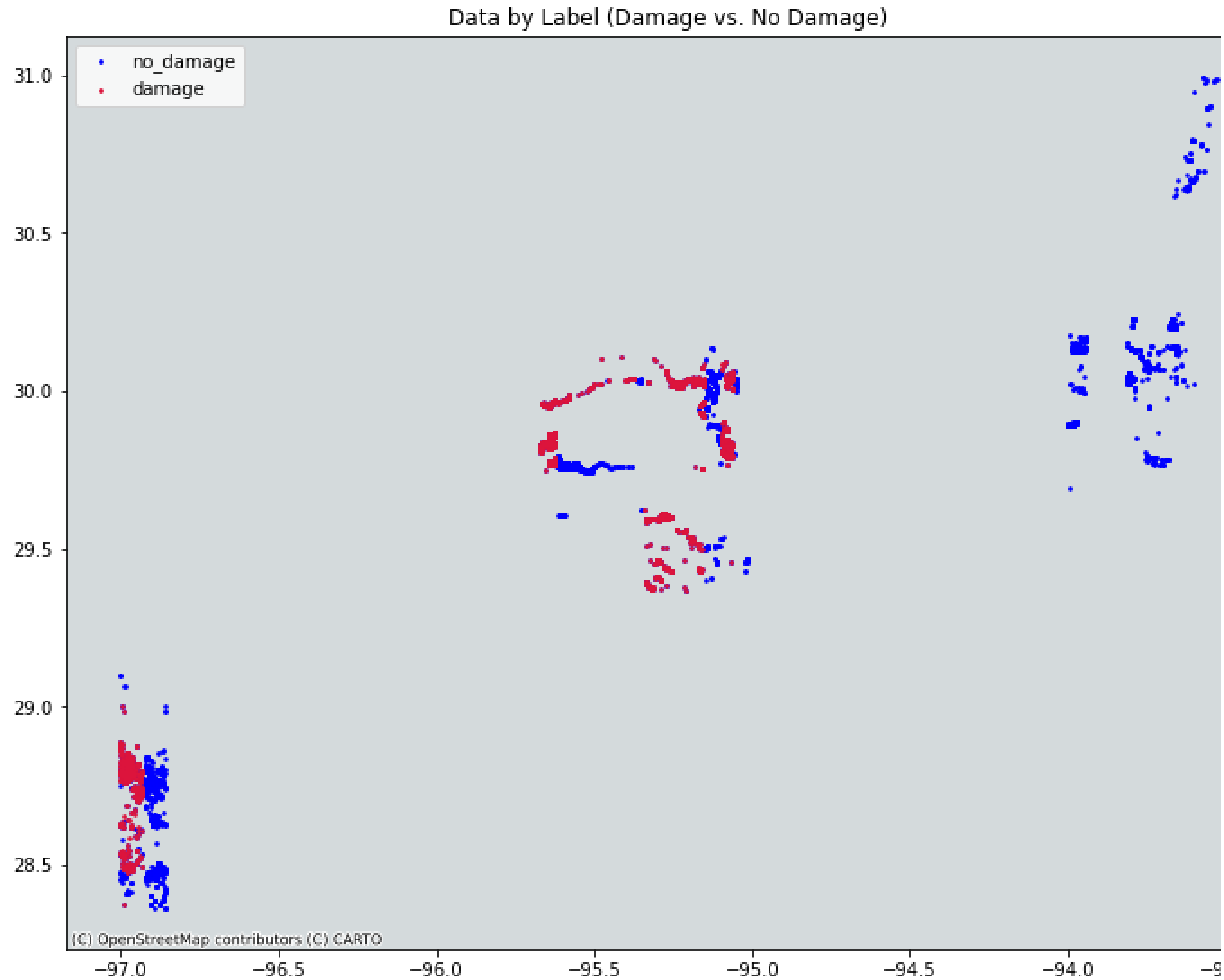
```
/usr/local/lib/python3.6/dist-packages/pyproj/crs/crs.py:53: FutureWarning: '+ini
  return _prepare_from_string(" ".join(pjargs))
```

```
image_gpd.crs
```

```
<Projected CRS: EPSG:3857>
Name: WGS 84 / Pseudo-Mercator
Axis Info [cartesian]:
- X[east]: Easting (metre)
- Y[north]: Northing (metre)
Area of Use:
- name: World - 85°S to 85°N
- bounds: (-180.0, -85.06, 180.0, 85.06)
Coordinate Operation:
- name: Popular Visualisation Pseudo-Mercator
- method: Popular Visualisation Pseudo Mercator
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

```
 create the axes
ig, ax = plt.subplots(figsize=(12, 12))

 plot a random sample of potholes
mage_gpd[image_gpd["label"]=="damage"].plot(ax=ax, markersize=10, marker='.', c="blue'
mage_gpd[image_gpd["label"]=="no_damage"].plot(ax=ax, markersize=10, marker='.', c="cr

x.legend()
x.set_title('Data by Label (Damage vs. No Damage)')

 NEW: plot the basemap underneath
tx.add_basemap(ax=ax, source=ctx.providers.CartoDB.Positron) #crs=image_gpd.crs,

 remove axis lines
ax.set_axis_off()
```

```
/usr/local/lib/python3.6/dist-packages/contextily/tile.py:629: UserWarning: The i
  warnings.warn(msg)
```



Get Unique Categories and Number of Images Per Category

```
image_gpd["train_test_category"].unique()
image_gpd["label"].unique()
```

```
array(['no_damage', 'damage'], dtype=object)
```

```
#Print the column names in image_df dataframe
image_df.columns
```

```
#Get some statistics about the number of images in different label categories
image_df.groupby('label').count()['image_path']
```

```
label
damage       15000
no_damage     8000
Name: image_path, dtype: int64
```

```
#Get some statistics about the number of images in different train/test categories
image_df.groupby('train_test_category').count()['image_path']
```

```
train_test_category
test                   2000
test_another           9000
train_another         10000
validation_another     2000
Name: image_path, dtype: int64
```

```
#Get some statistics about the number of images in different train/test categories
image_df.count()
```

```
label                  23000
train_test_category    23000
image_name             23000
image_path             23000
location               23000
lon                    23000
lat                    23000
Coordinates            23000
dtype: int64
```

```
#Get some statistics about the number of images in different train/test categories
image_df.groupby(['train_test_category', 'label']).count()['image_path']
```

```
train_test_category  label
test                 damage       1000
                     no_damage    1000
test_another         damage       8000
                     no_damage    1000
train_another        damage       5000
                     no_damage    5000
validation_another   damage       1000
                     no_damage    1000
Name: image_path, dtype: int64
```

Split by Damage/No Damage and Show Examples

```
#Split images into damage/no damage label sets
```

```
#split images into damage/no damage label sets
image_df_dmg = image_df[image_df["label"]=="damage"].copy().reset_index()
image_df_nodmg = image_df[image_df["label"]=="no_damage"].copy().reset_index()


#Show 20 examples from each label category
fig, ax = plt.subplots(nrows=4, ncols=10, sharex=True, sharey=True, figsize=(20,10))

ax = ax.flatten()

for i in range(20):
    img = cv2.imread(image_df_dmg['image_path'][i], cv2.IMREAD_UNCHANGED)
    ax[i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    ax[i].set_title('damage')

for i in range(20,40):
    img = cv2.imread(image_df_nodmg['image_path'][i], cv2.IMREAD_UNCHANGED)
    ax[i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    ax[i].set_title('no damage')

plt.show()
```

Image Loading and Pre-processing

```
# get the train-validation-test splits
image_df_train = image_df[image_df['train_test_category']=='train_another'].copy().res
image_df_val = image_df[image_df['train_test_category']=='validation_another'].copy()
image_df_test = image_df[image_df['train_test_category']=='test_another'].copy().reset
```

```
image_df_test_balanced = image_df[image_df['train_test_category']=='test'].copy().rese

#Get paths and labels per preset test/train/validatioin categories

# paths
train_path = image_df_train['image_path'].copy().values
val_path = image_df_val['image_path'].copy().values
test_path = image_df_test['image_path'].copy().values
test_balanced_path = image_df_test_balanced['image_path'].copy().values

# labels
train_labels = np.zeros(len(image_df_train), dtype=np.int8)
train_labels[image_df_train['label'].values=='damage'] = 1

val_labels = np.zeros(len(image_df_val), dtype=np.int8)
val_labels[image_df_val['label'].values=='damage'] = 1

test_labels = np.zeros(len(image_df_test), dtype=np.int8)
test_labels[image_df_test['label'].values=='damage'] = 1

test_balanced_labels = np.zeros(len(image_df_test_balanced), dtype=np.int8)
test_balanced_labels[image_df_test_balanced['label'].values=='damage'] = 1

labels_all = image_df_test['label'].values


!pip install imageio
from imageio import imread

#Read an initial image to get size of the feature vector
img = imread(image_df_train.image_path[0])
flat_image = img.flatten()

#Initialize data matrices for each of the
dataMat_train = np.zeros([image_df_train.shape[0], flat_image.shape[0]])
dataMat_val = np.zeros([image_df_val.shape[0], flat_image.shape[0]])
dataMat_test = np.zeros([image_df_test.shape[0], flat_image.shape[0]])
dataMat_test_balanced = np.zeros([image_df_test_balanced.shape[0], flat_image.shape[0
#dataMat_all = np.zeros([image_df.shape[0], flat_image.shape[0]])
```

```
Requirement already satisfied: imageio in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (
```

```
ill data matrix for each of the different datasets, also for an all data set (which w

ill the train data matrix with data for each of the images
```

```
int("Loading training data..")
r i, tmpRow in image_df_train.iterrows():
  img = imread(image_df_train.image_path[i])
  flat_image = img.flatten()
  dataMat_train[i,:] = flat_image

int("Loading validation data..")
ill the validation data matrix with data for each of the images
r i, tmpRow in image_df_val.iterrows():
  img = imread(image_df_val.image_path[i])
  flat_image = img.flatten()
  dataMat_val[i,:] = flat_image

int("Loading testing data..")
ill the test data matrix with data for each of the images
r i, tmpRow in image_df_test.iterrows():
  img = imread(image_df_test.image_path[i])
  flat_image = img.flatten()
  dataMat_test[i,:] = flat_image


int("Loading balanced testing data..")
ill the test balanced data matrix with data for each of the images
r i, tmpRow in image_df_test_balanced.iterrows():
  img = imread(image_df_test_balanced.image_path[i])
  flat_image = img.flatten()
  dataMat_test_balanced[i,:] = flat_image

rint("Loading all data..")
ill the data matrix (all images_) with data for each of the images
or i, tmpRow in image_df.iterrows():
   img = imread(image_df.image_path[i])
   flat_image = img.flatten()
   dataMat_all[i,:] = flat_image
```

```
Loading training data..
Loading validation data..
Loading testing data..
Loading balanced testing data..
```

```
dataMat_train
```

```
array([[ 69.,  82.,  65., ..., 160., 148., 132.],
       [104., 115.,  81., ...,  64.,  89.,  50.],
       [ 92.,  91.,  63., ...,  78.,  84.,  70.],
       ...,
       [ 99., 102.,  85., ..., 116., 115.,  97.],
       [142., 135., 106., ..., 209., 206., 197.],
       [151., 137., 110., ..., 131., 117.,  88.]])
```

```
dataMat_val
```

```
array([[ 65.,  67.,  53., ..., 138., 138., 110.],
       [ 38.,  56.,  32., ...,  56.,  58.,  44.],
       [ 96., 106.,  71., ...,  64.,  77.,  49.],
       ...,
       [ 47.,  62.,  43., ...,  69.,  97.,  57.],
       [ 63.,  95.,  56., ...,  42.,  51.,  32.],
       [ 80.,  79.,  59., ..., 108., 112.,  89.]])
```

```
dataMat_test
```

```
array([[ 95., 103.,  64., ...,  84., 108.,  76.],
       [158., 201., 156., ...,  91.,  94.,  63.],
       [175., 181., 171., ...,  93.,  99.,  65.],
       ...,
       [ 66.,  87.,  54., ..., 120., 123., 102.],
       [ 72.,  94.,  55., ...,  91., 102.,  62.],
       [ 57.,  74.,  38., ...,  57.,  83.,  46.]])
```

```
dataMat_test_balanced
```

```
array([[ 69.,  82.,  65., ..., 160., 148., 132.],
       [ 92.,  91.,  63., ...,  78.,  84.,  70.],
       [ 48.,  53.,  33., ...,  81.,  95.,  72.],
       ...,
       [ 47.,  62.,  43., ...,  69.,  97.,  57.],
       [122., 123.,  92., ...,  79.,  93.,  68.],
       [ 83.,  91.,  68., ..., 140., 137., 106.]])
```

```
#dataMat_all
```

Supervised Methods

Consulted Week 8 DL_Basics1_SimpleMLP

Additional Data Preparation

```
#Make copies of data matrices
X_train = np.copy(dataMat_train)
y_train = np.copy(train_labels)

X_test = np.copy(dataMat_test)
y_test = np.copy(test_labels)

X_val = np.copy(dataMat_val)
y_val = np.copy(val_labels)
```

```
y_val = np.copy(val_labels)

X_test_balanced = np.copy(dataMat_test_balanced)
y_test_balanced = np.copy(test_balanced_labels)

#Make alls X values floats
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
X_test_balanced = X_test_balanced.astype('float32')

#Normalize Data
X_train /= 255
X_test /= 255
X_val /= 255
X_test_balanced /= 255


#Reshape X data
X_train = X_train.reshape([-1, 128, 128, 3])

X_val = X_val.reshape([-1, 128, 128, 3])

X_test = X_test.reshape([-1, 128, 128, 3])

X_test_balanced = X_test_balanced.reshape([-1, 128, 128, 3])


# convert class vectors to binary class matrices
num_classes = 2

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
y_test_balanced = keras.utils.to_categorical(y_test_balanced, num_classes)
y_val = keras.utils.to_categorical(y_val, num_classes)
```

Function Definitions (used for multiple models)

```
#Define function for completing the fit and evaluate steps of different models

def fit_and_evaluate_model(model, X_train, X_test, X_val, y_train, y_test, y_val, batc
    ''' Fits and evaluates model based on train, validation and data.
        Returns model development history.
    '''
    #Fit
    history = model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          #validation split=0.2) #,
```

```
          validation_data=(X_val, y_val))

    #Score
    score = model.evaluate(X_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    return history



def model_history_plots(history):
    ''' Plots a accuracy and loss graph by epoch based on model development history.
    '''
    # list all data in history
    print(history.history.keys())
    # summarize history for 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', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    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()


# Model Evaluation metrics
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
from sklearn.metrics import classification_report
import sklearn.metrics as metrics

def confusion_matrix(model, y_test, X_test):
    ''' Calculates and prints confusion matrix and related metrics based
        on model and test data.
    '''
    y_pred_ohe = model.predict(X_test)  # shape=(n_samples, 12)
    y_pred_labels = np.argmax(y_pred_ohe, axis=1)
    y_test_labels = np.argmax(y_test, axis=1)
    confusion_matrix = metrics.confusion_matrix(y_true=y_test_labels, y_pred=y_pred_la

    print('Accuracy Score : ' + str(accuracy_score(y_test_labels, y_pred_labels)))
    print('Precision Score : ' + str(precision_score(y_test_labels,y_pred_labels)))
```

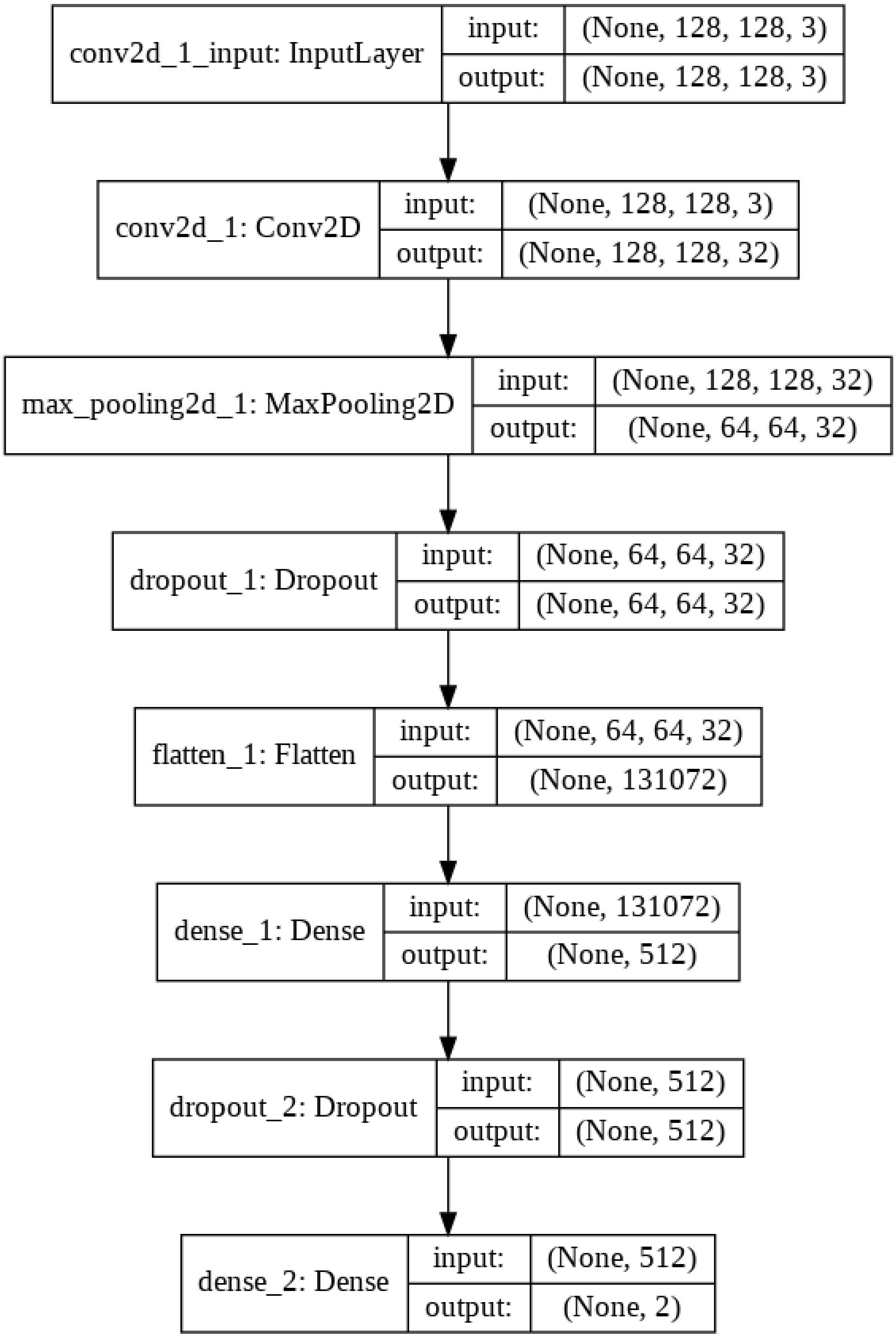
```
    print('Recall Score : ' + str(recall_score(y_test_labels,y_pred_labels)))
    print('F1 Score : ' + str(f1_score(y_test_labels,y_pred_labels)))

    #Dummy Classifier Confusion matrix
    from sklearn.metrics import confusion_matrix
    print('Confusion Matrix : \n' + str(confusion_matrix(y_test_labels,y_pred_labels))
```

CNN Model 1

(simpler model, 2 drop out layers, on RGB images)

```
# build model (1)
def create_model():

    np.random.seed(42)

    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', input_shape=(128, 128, 3), activatio
    model.add(MaxPooling2D(pool_size=(2, 2))) #40x40
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(2, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer=RMSprop(),
                  metrics=['accuracy'])

    return model

M1 = create_model()
```

```
from keras.utils.vis_utils import plot_model
plot_model(M1, show_shapes=True)
```




```
history = fit_and_evaluate_model(M1, X_train, X_test, X_val, y_train, y_test, y_val, 
```

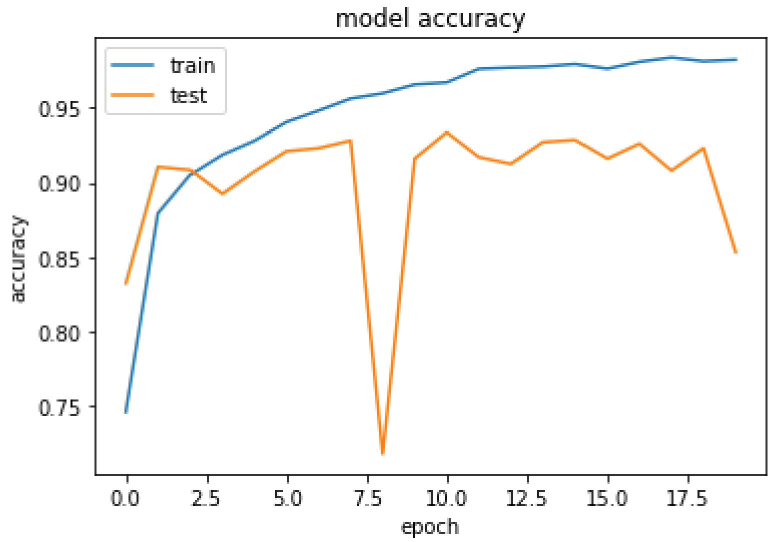
```
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 14s 1ms/step - loss: 1.5160 - accu
Epoch 2/20
10000/10000 [==============================] - 8s 792us/step - loss: 0.3147 - acc
Epoch 3/20
10000/10000 [==============================] - 8s 789us/step - loss: 0.2516 - acc
Epoch 4/20
10000/10000 [==============================] - 8s 799us/step - loss: 0.2308 - acc
Epoch 5/20
10000/10000 [==============================] - 8s 801us/step - loss: 0.2049 - acc
Epoch 6/20
10000/10000 [==============================] - 8s 789us/step - loss: 0.1814 - acc
Epoch 7/20
10000/10000 [==============================] - 8s 785us/step - loss: 0.1614 - acc
Epoch 8/20
10000/10000 [==============================] - 8s 786us/step - loss: 0.1427 - acc
Epoch 9/20
10000/10000 [==============================] - 8s 787us/step - loss: 0.1266 - acc
Epoch 10/20
10000/10000 [==============================] - 8s 787us/step - loss: 0.1164 - acc
Epoch 11/20
10000/10000 [==============================] - 8s 787us/step - loss: 0.1272 - acc
Epoch 12/20
10000/10000 [==============================] - 8s 789us/step - loss: 0.0918 - acc
Epoch 13/20
10000/10000 [==============================] - 8s 785us/step - loss: 0.0922 - acc
Epoch 14/20
10000/10000 [==============================] - 8s 787us/step - loss: 0.0757 - acc
Epoch 15/20
10000/10000 [==============================] - 8s 788us/step - loss: 0.0836 - acc
Epoch 16/20
10000/10000 [==============================] - 8s 789us/step - loss: 0.0977 - acc
Epoch 17/20
10000/10000 [==============================] - 8s 791us/step - loss: 0.0801 - acc
Epoch 18/20
10000/10000 [==============================] - 8s 809us/step - loss: 0.0822 - acc
Epoch 19/20
10000/10000 [==============================] - 8s 805us/step - loss: 0.0909 - acc
Epoch 20/20
10000/10000 [==============================] - 8s 800us/step - loss: 0.0826 - acc
Test loss: 0.350730651377583
Test accuracy: 0.9508888721466064
```

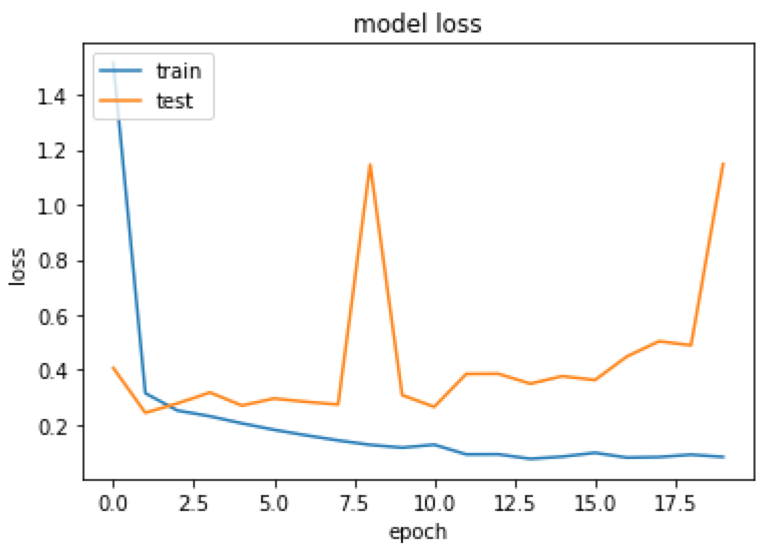
```
confusion_matrix(M1, y_test, X_test)
```

```
Accuracy Score : 0.9508888888888889
Precision Score : 0.9636809815950921
Recall Score : 0.98175
F1 Score : 0.9726315789473684
Confusion Matrix :
[[ 704  296]
 [ 146 7854]]
```

```
model_history_plots(history)
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





CNN Model 2

(single dense layer on gray-scale images)

```
from skimage.color import rgb2gray
```

```
#Convert data too grascale

# Initialize grayscale arrays
X_train_BW = np.zeros([X_train.shape[0],
                       X_train.shape[1],
                       X_train.shape[2]])

X_test_BW = np.zeros([X_test.shape[0],
                      X_test.shape[1],
                      X_test.shape[2]])

X_val_BW = np.zeros([X_val.shape[0],
                      X_val.shape[1],
                      X_val.shape[2]])

# convert RGB arrays to grayscale
for i in range(X_train.shape[0]):
  X_train_BW[i] = rgb2gray(X_train[i])

for i in range(X_test.shape[0]):
  X_test_BW[i] = rgb2gray(X_test[i])

for i in range(X_val.shape[0]):
  X_val_BW[i] = rgb2gray(X_val[i])

# flatten grayscale arrays
X_train_BW = X_train_BW.reshape(X_train_BW.shape[0],
                                X_train_BW.shape[1] * X_train_BW.shape[2])

X_test_BW = X_test_BW.reshape(X_test_BW.shape[0],
                              X_test_BW.shape[1] * X_test_BW.shape[2])

X_val_BW = X_val_BW.reshape(X_val_BW.shape[0],
                              X_val_BW.shape[1] * X_val_BW.shape[2])


print("X_train_BW shape:" + str(X_train_BW.shape))
print("X_test_BW shape:" + str(X_test_BW.shape))
print("X_val_BW shape:" + str(X_val_BW.shape))

print("y_train shape:" + str(y_train.shape))
print("y_test shape:" + str(y_test.shape))
print("y_val shape:" + str(y_train.shape))
```

```
M2 = Sequential()
M2.add(Dense(num_classes, activation='softmax', input_shape=(16384,)))

M2.summary()

M2.compile(loss='categorical_crossentropy',
                  optimizer=RMSprop(),
                  metrics=['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 2)                 32770
=================================================================
Total params: 32,770
Trainable params: 32,770
Non-trainable params: 0
_________________________________________________________________
```

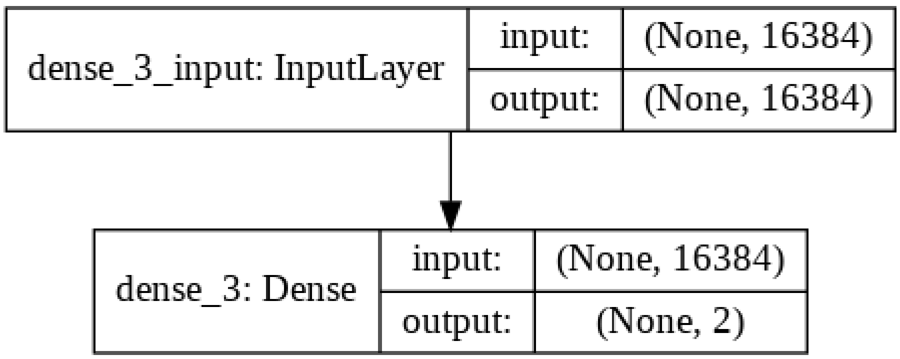
```
history = fit_and_evaluate_model(M2, X_train_BW, X_test_BW, X_val_BW, y_train, y_test,
```

```
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 1s 141us/step - loss: 2.8961 - acc
Epoch 2/20
10000/10000 [==============================] - 1s 137us/step - loss: 2.6505 - acc
Epoch 3/20
10000/10000 [==============================] - 1s 136us/step - loss: 2.5665 - acc
Epoch 4/20
10000/10000 [==============================] - 1s 138us/step - loss: 2.5626 - acc
Epoch 5/20
10000/10000 [==============================] - 1s 138us/step - loss: 2.4032 - acc
Epoch 6/20
10000/10000 [==============================] - 1s 138us/step - loss: 2.4450 - acc
Epoch 7/20
10000/10000 [==============================] - 1s 140us/step - loss: 2.4520 - acc
Epoch 8/20
10000/10000 [==============================] - 1s 137us/step - loss: 2.4478 - acc
Epoch 9/20
10000/10000 [==============================] - 1s 135us/step - loss: 2.4288 - acc
Epoch 10/20
10000/10000 [==============================] - 1s 132us/step - loss: 2.4046 - acc
Epoch 11/20
10000/10000 [==============================] - 1s 134us/step - loss: 2.4515 - acc
Epoch 12/20
10000/10000 [==============================] - 1s 133us/step - loss: 2.4664 - acc
Epoch 13/20
10000/10000 [==============================] - 1s 130us/step - loss: 2.3484 - acc
Epoch 14/20
10000/10000 [==============================] - 1s 131us/step - loss: 2.4479 - acc
Epoch 15/20
10000/10000 [==============================] - 1s 133us/step - loss: 2.4045 - acc
Epoch 16/20
10000/10000 [==============================] - 1s 131us/step - loss: 2.4271 - acc
Epoch 17/20
10000/10000 [==============================] - 1s 130us/step - loss: 2.4660 - acc
Epoch 18/20
10000/10000 [==============================] - 1s 131us/step - loss: 2.4205 - acc
Epoch 19/20
10000/10000 [==============================] - 1s 131us/step - loss: 2.3984 - acc
Epoch 20/20
10000/10000 [==============================] - 1s 132us/step - loss: 2.3971 - acc
Test loss: 0.5237596959405475
Test accuracy: 0.8375555276870728
```

```
M2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 2)                 32770     
=================================================================
Total params: 32,770
Trainable params: 32,770
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(M2, show_shapes=True)
```

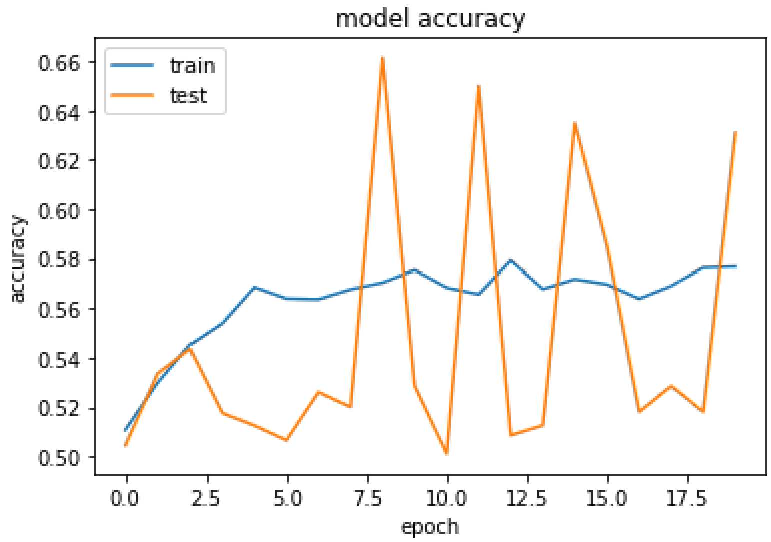


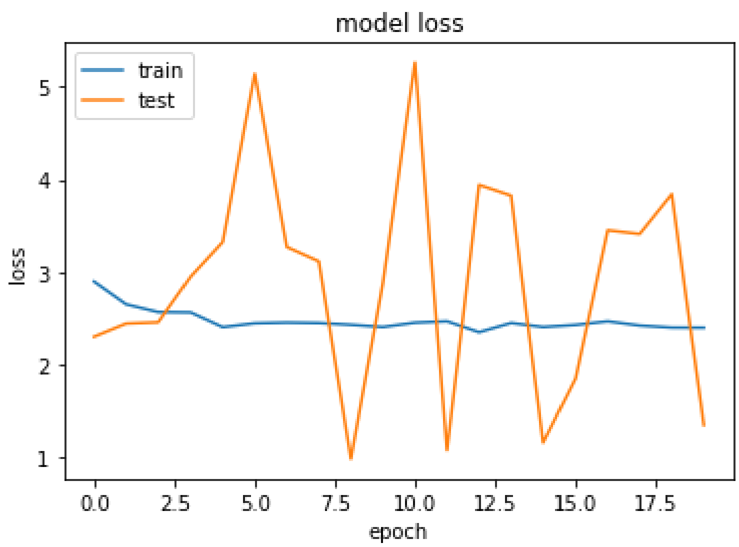

```
confusion_matrix(M2, y_test, X_test_BW)
```

```
Accuracy Score : 0.8375555555555556
Precision Score : 0.9186731557377049
Recall Score : 0.896625
F1 Score : 0.907515182186235
Confusion Matrix :
[[ 365  635]
 [ 827 7173]]
```

```
model_history_plots(history)
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





CNN Model 3

(more complex model, on RGB images)

```
# build model (3)

def create_model():

  np.random.seed(42)

  M3 = Sequential()

  M3.add(Conv2D(32, (3, 3), padding='same', input_shape=(128, 128, 3), activation='re
  M3.add(MaxPooling2D(pool_size=(2, 2))) #40x40
  M3.add(Dropout(0.25))
```

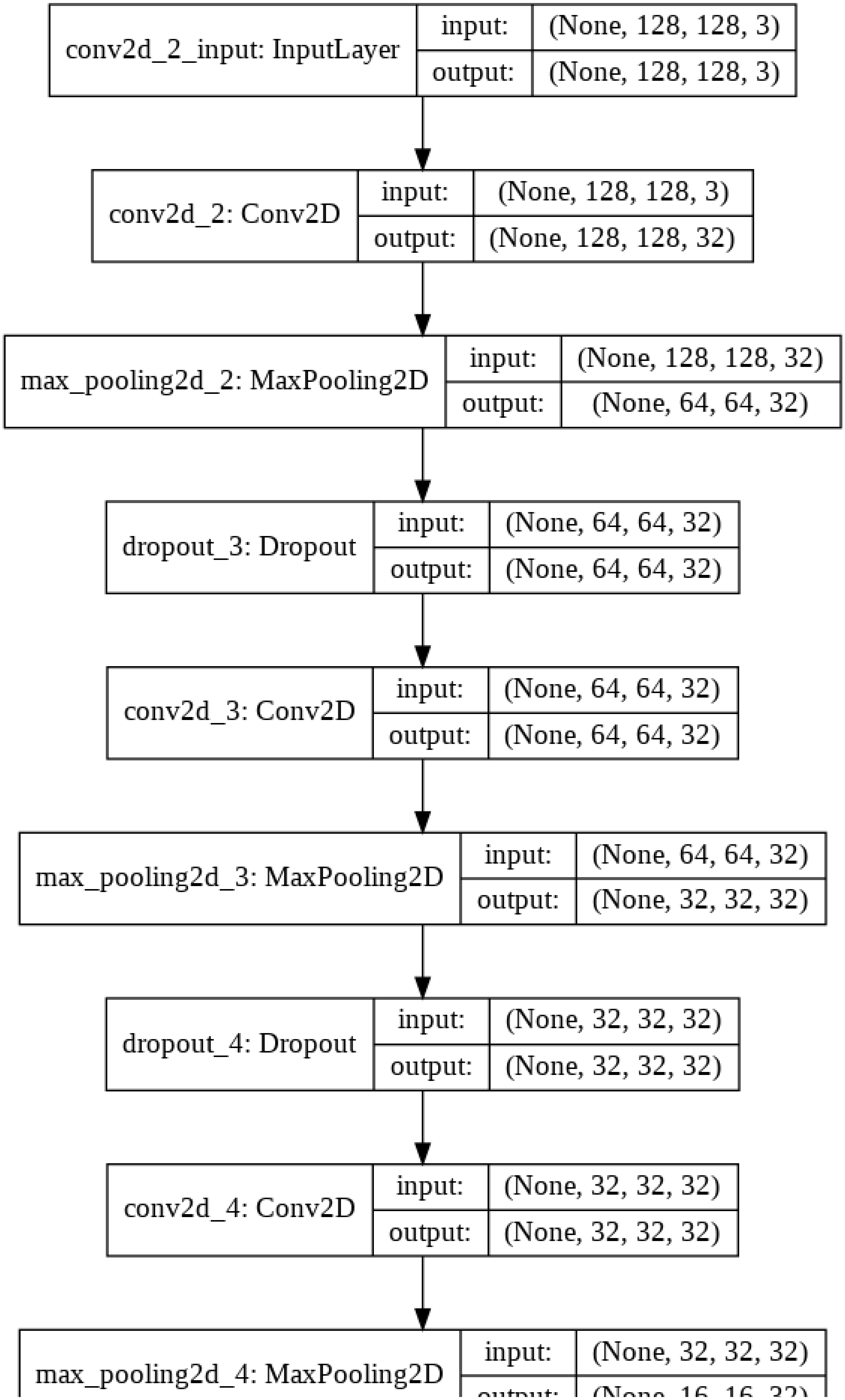
```
    M3.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    M3.add(MaxPooling2D(pool_size=(2, 2))) #20x20
    M3.add(Dropout(0.25))

    M3.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    M3.add(MaxPooling2D(pool_size=(2, 2))) #10x10
    M3.add(Dropout(0.25))

    M3.add(Conv2D(32, (10, 10), padding='same', activation='relu'))
    M3.add(MaxPooling2D(pool_size=(2, 2))) #5x5
    M3.add(Dropout(0.25))

    M3.add(Flatten())
    M3.add(Dense(512, activation='relu'))
    M3.add(Dropout(0.5))

    M3.add(Dense(2, activation='softmax'))

    M3.compile(loss='categorical_crossentropy',
                  optimizer=RMSprop(),
                  metrics=['accuracy'])

    return M3

M3 = create_model()

history = fit_and_evaluate_model(M3, X_train, X_test, X_val, y_train, y_test, y_val, b
```

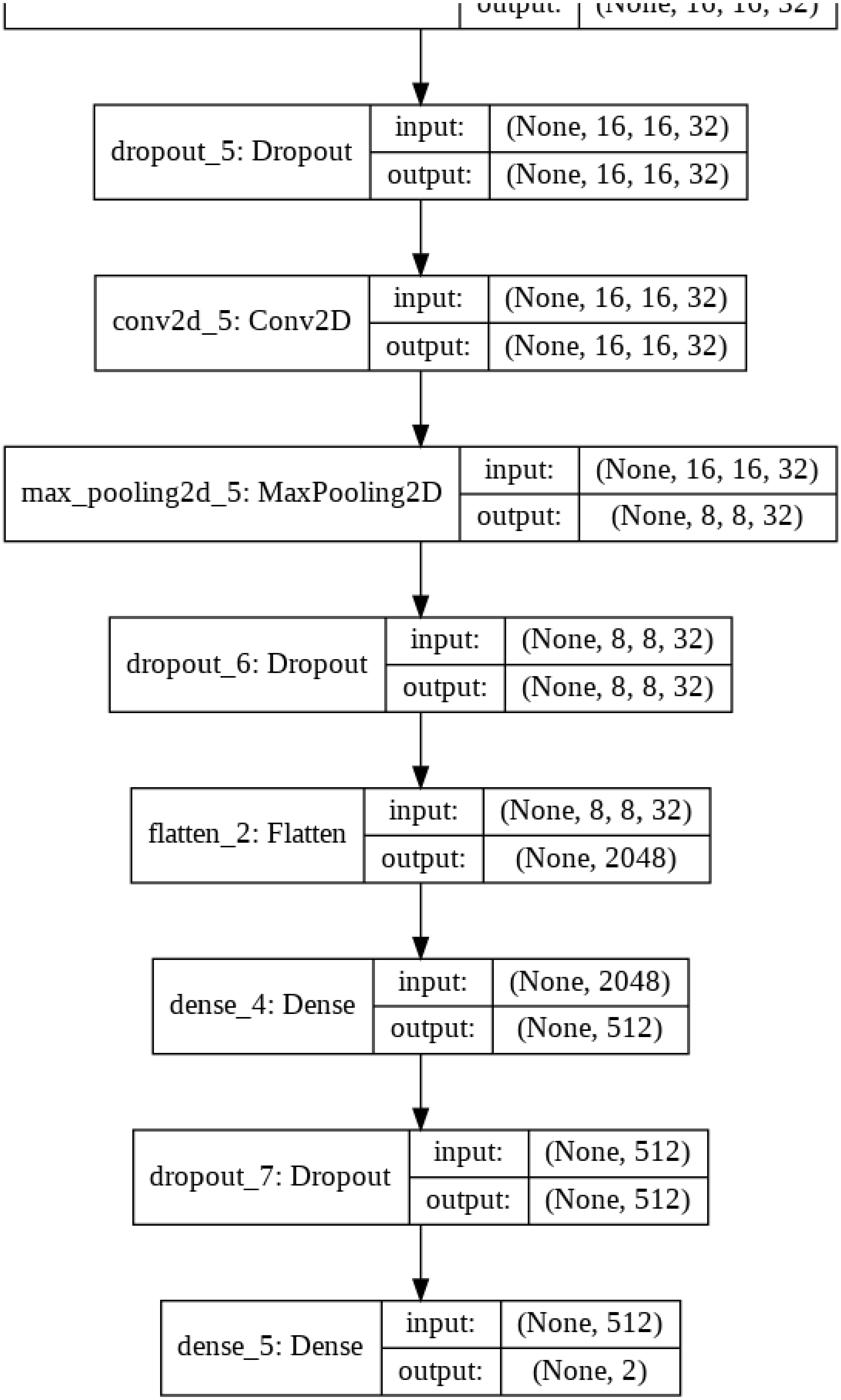
```
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 5s 458us/step - loss: 0.6109 - acc
Epoch 2/20
10000/10000 [==============================] - 4s 424us/step - loss: 0.3063 - acc
Epoch 3/20
10000/10000 [==============================] - 4s 428us/step - loss: 0.2315 - acc
Epoch 4/20
10000/10000 [==============================] - 4s 430us/step - loss: 0.2213 - acc
Epoch 5/20
10000/10000 [==============================] - 4s 430us/step - loss: 0.2010 - acc
Epoch 6/20
10000/10000 [==============================] - 4s 429us/step - loss: 0.1786 - acc
Epoch 7/20
10000/10000 [==============================] - 4s 426us/step - loss: 0.1787 - acc
Epoch 8/20
10000/10000 [==============================] - 4s 426us/step - loss: 0.1620 - acc
Epoch 9/20
10000/10000 [==============================] - 4s 427us/step - loss: 0.1714 - acc
Epoch 10/20
10000/10000 [==============================] - 4s 428us/step - loss: 0.1701 - acc
Epoch 11/20
10000/10000 [==============================] - 4s 431us/step - loss: 0.1542 - acc
Epoch 12/20
10000/10000 [==============================] - 4s 428us/step - loss: 0.1516 - acc
Epoch 13/20
10000/10000 [==============================] - 4s 427us/step - loss: 0.1687 - acc
Epoch 14/20
10000/10000 [==============================] - 4s 430us/step - loss: 0.1722 - acc
Epoch 15/20
10000/10000 [==============================] - 4s 431us/step - loss: 0.1636 - acc
Epoch 16/20
10000/10000 [==============================] - 4s 432us/step - loss: 0.1702 - acc
Epoch 17/20
10000/10000 [==============================] - 4s 429us/step - loss: 0.1525 - acc
Epoch 18/20
10000/10000 [==============================] - 4s 433us/step - loss: 0.1693 - acc
Epoch 19/20
10000/10000 [==============================] - 4s 430us/step - loss: 0.1604 - acc
Epoch 20/20
10000/10000 [==============================] - 4s 445us/step - loss: 0.1569 - acc
Test loss: 0.14463004183210432
Test accuracy: 0.9657777547836304
```

```
M3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 128, 128, 32)      896       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 32)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 64, 64, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 32)        102432    
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 32)          0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 32)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 512)               1049088   
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 1026      
=================================================================
Total params: 1,171,938
Trainable params: 1,171,938
Non-trainable params: 0
_________________________________________________________________
```

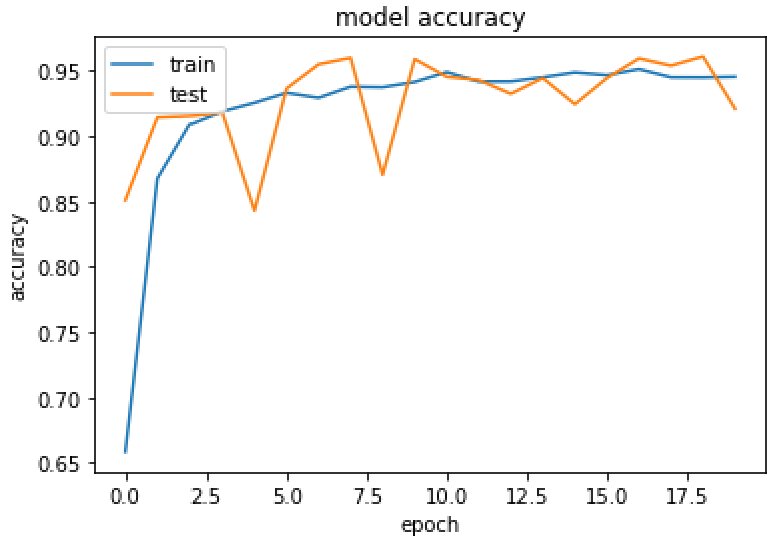
```
plot_model(M3, show_shapes=True)
```

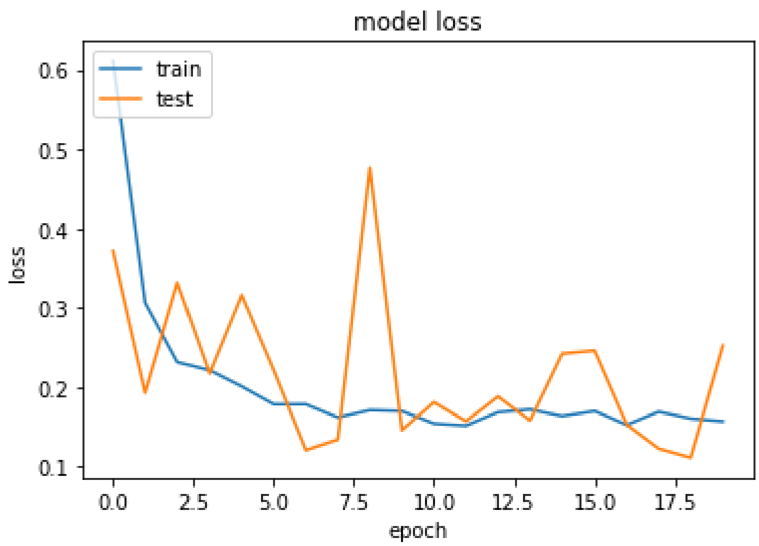

conv2d_2_input: InputLayer
input: (None, 128, 128, 3)
output: (None, 128, 128, 3)
conv2d_2: Conv2D
input: (None, 128, 128, 3)
output: (None, 128, 128, 32)
max_pooling2d_2: MaxPooling2D
input: (None, 128, 128, 32)
output: (None, 64, 64, 32)
dropout_3: Dropout
input: (None, 64, 64, 32)
output: (None, 64, 64, 32)
conv2d_3: Conv2D
input: (None, 64, 64, 32)
output: (None, 64, 64, 32)
max_pooling2d_3: MaxPooling2D
input: (None, 64, 64, 32)
output: (None, 32, 32, 32)
dropout_4: Dropout
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
conv2d_4: Conv2D
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
max_pooling2d_4: MaxPooling2D
input: (None, 32, 32, 32)


dropout_5: Dropout
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
conv2d_5: Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
max_pooling2d_5: MaxPooling2D
input: (None, 16, 16, 32)
output: (None, 8, 8, 32)
dropout_6: Dropout
input: (None, 8, 8, 32)
output: (None, 8, 8, 32)
flatten_2: Flatten
input: (None, 8, 8, 32)
output: (None, 2048)
dense_4: Dense
input: (None, 2048)
output: (None, 512)
dropout_7: Dropout
input: (None, 512)
output: (None, 512)
dense_5: Dense
input: (None, 512)
output: (None, 2)

```
model_history_plots(history)
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```






```
 (because could not get GridSearch to work)


 [10, 20, 50]
ns = [32, 64, 128]

ations = [(batch_size, epochs) for batch_size in batch_size_options for epochs in epoc

 = parameter_combinations[0]
_score = 0


 the combinations of parameters, train model and get history, and print
ccuracy. Store the best combination and scores.
in parameter_combinations:
```

```
ion)
del()
ombination[0]
nation[1]
it(X_train, y_train,
size=batch_size,
=epochs,
e=1,
ation_split=0.2) #,
tion_data=(X_val, y_val))


luate(X_test, y_test, verbose=0)
ss:', score[0])
curacy:', score[1])
score[1])
best_combination_score:
tion = combination
tion_score = score[1]


parameter combination is a batch size of " + str(best_combination[0]) + " and " + str(
```

```
(32, 10)
Train on 10000 samples, validate on 2000 samples
Epoch 1/10
10000/10000 [==============================] - 5s 465us/step - loss: 0.6833 - acc
Epoch 2/10
10000/10000 [==============================] - 4s 427us/step - loss: 0.3758 - acc
Epoch 3/10
10000/10000 [==============================] - 4s 429us/step - loss: 0.2630 - acc
Epoch 4/10
10000/10000 [==============================] - 4s 432us/step - loss: 0.2243 - acc
Epoch 5/10
10000/10000 [==============================] - 4s 425us/step - loss: 0.2129 - acc
Epoch 6/10
10000/10000 [==============================] - 4s 427us/step - loss: 0.2028 - acc
Epoch 7/10
10000/10000 [==============================] - 4s 428us/step - loss: 0.2041 - acc
Epoch 8/10
10000/10000 [==============================] - 4s 427us/step - loss: 0.1756 - acc
Epoch 9/10
10000/10000 [==============================] - 4s 426us/step - loss: 0.1916 - acc
Epoch 10/10
10000/10000 [==============================] - 4s 429us/step - loss: 0.1806 - acc
Test loss: 0.12815518315136432
Test accuracy: 0.9621111154556274
(32, 20)
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 5s 460us/step - loss: 0.6397 - acc
Epoch 2/20
10000/10000 [==============================] - 4s 431us/step - loss: 0.3637 - acc
Epoch 3/20
10000/10000 [==============================] - 4s 438us/step - loss: 0.2576 - acc
Epoch 4/20
10000/10000 [==============================] - 4s 435us/step - loss: 0.2232 - acc
Epoch 5/20
10000/10000 [==============================] - 4s 435us/step - loss: 0.1964 - acc
Epoch 6/20
10000/10000 [==============================] - 4s 438us/step - loss: 0.1795 - acc
Epoch 7/20
10000/10000 [==============================] - 4s 439us/step - loss: 0.1718 - acc
Epoch 8/20
10000/10000 [==============================] - 4s 439us/step - loss: 0.1622 - acc
Epoch 9/20
10000/10000 [==============================] - 4s 435us/step - loss: 0.1592 - acc
Epoch 10/20
10000/10000 [==============================] - 4s 439us/step - loss: 0.1616 - acc
Epoch 11/20
10000/10000 [==============================] - 4s 438us/step - loss: 0.1550 - acc
Epoch 12/20
10000/10000 [==============================] - 4s 436us/step - loss: 0.1588 - acc
Epoch 13/20
10000/10000 [==============================] - 4s 433us/step - loss: 0.1531 - acc
Epoch 14/20
10000/10000 [==============================] - 4s 435us/step - loss: 0.1503 - acc
Epoch 15/20
```

```
10000/10000 [==============================] - 4s 437us/step - loss: 0.1714 - acc
Epoch 16/20
10000/10000 [==============================] - 4s 438us/step - loss: 0.1526 - acc
Epoch 17/20
10000/10000 [==============================] - 4s 437us/step - loss: 0.1539 - acc
Epoch 18/20
10000/10000 [==============================] - 4s 436us/step - loss: 0.1542 - acc
Epoch 19/20
10000/10000 [==============================] - 4s 435us/step - loss: 0.1516 - acc
Epoch 20/20
10000/10000 [==============================] - 4s 438us/step - loss: 0.1554 - acc
Test loss: 0.16982826648569768
Test accuracy: 0.9496666789054871
(32, 50)
Train on 10000 samples, validate on 2000 samples
Epoch 1/50
10000/10000 [==============================] - 5s 460us/step - loss: 0.6369 - acc
Epoch 2/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.3397 - acc
Epoch 3/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.2508 - acc
Epoch 4/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.2238 - acc
Epoch 5/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.1969 - acc
Epoch 6/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.1790 - acc
Epoch 7/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1988 - acc
Epoch 8/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.1725 - acc
Epoch 9/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.1610 - acc
Epoch 10/50
10000/10000 [==============================] - 4s 431us/step - loss: 0.1626 - acc
Epoch 11/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1558 - acc
Epoch 12/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1472 - acc
Epoch 13/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.1408 - acc
Epoch 14/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.1500 - acc
Epoch 15/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.1474 - acc
Epoch 16/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1456 - acc
Epoch 17/50
10000/10000 [==============================] - 4s 425us/step - loss: 0.1642 - acc
Epoch 18/50
10000/10000 [==============================] - 4s 438us/step - loss: 0.1401 - acc
Epoch 19/50
10000/10000 [==============================] - 4s 432us/step - loss: 0.1430 - acc
Epoch 20/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1486 - acc
Epoch 21/50
```

```
Epoch 21/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.1761 - acc
Epoch 22/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.1514 - acc
Epoch 23/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.1629 - acc
Epoch 24/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1498 - acc
Epoch 25/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.1548 - acc
Epoch 26/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1633 - acc
Epoch 27/50
10000/10000 [==============================] - 4s 430us/step - loss: 0.1578 - acc
Epoch 28/50
10000/10000 [==============================] - 4s 431us/step - loss: 0.1699 - acc
Epoch 29/50
10000/10000 [==============================] - 4s 431us/step - loss: 0.1624 - acc
Epoch 30/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.2010 - acc
Epoch 31/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.1620 - acc
Epoch 32/50
10000/10000 [==============================] - 4s 426us/step - loss: 0.1683 - acc
Epoch 33/50
10000/10000 [==============================] - 4s 424us/step - loss: 0.1659 - acc
Epoch 34/50
10000/10000 [==============================] - 4s 425us/step - loss: 0.1746 - acc
Epoch 35/50
10000/10000 [==============================] - 4s 425us/step - loss: 0.1642 - acc
Epoch 36/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.1868 - acc
Epoch 37/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.1572 - acc
Epoch 38/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.2259 - acc
Epoch 39/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.1615 - acc
Epoch 40/50
10000/10000 [==============================] - 4s 441us/step - loss: 0.2297 - acc
Epoch 41/50
10000/10000 [==============================] - 4s 446us/step - loss: 0.1990 - acc
Epoch 42/50
10000/10000 [==============================] - 4s 439us/step - loss: 0.1884 - acc
Epoch 43/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.2000 - acc
Epoch 44/50
10000/10000 [==============================] - 4s 430us/step - loss: 0.2079 - acc
Epoch 45/50
10000/10000 [==============================] - 4s 446us/step - loss: 0.1924 - acc
Epoch 46/50
10000/10000 [==============================] - 4s 431us/step - loss: 0.2066 - acc
Epoch 47/50
10000/10000 [==============================] - 4s 431us/step - loss: 0.2580 - acc
Epoch 48/50
10000/10000 [==============================] - 4s 427us/step - loss: 0.2269 - acc
```

```
Epoch 49/50
10000/10000 [==============================] - 4s 428us/step - loss: 0.2136 - acc
Epoch 50/50
10000/10000 [==============================] - 4s 429us/step - loss: 0.2305 - acc
Test loss: 0.44999250287479825
Test accuracy: 0.8666666746139526
(64, 10)
Train on 10000 samples, validate on 2000 samples
Epoch 1/10
10000/10000 [==============================] - 4s 399us/step - loss: 0.7015 - acc
Epoch 2/10
10000/10000 [==============================] - 4s 360us/step - loss: 0.5639 - acc
Epoch 3/10
10000/10000 [==============================] - 4s 362us/step - loss: 0.3726 - acc
Epoch 4/10
10000/10000 [==============================] - 4s 359us/step - loss: 0.2707 - acc
Epoch 5/10
10000/10000 [==============================] - 4s 359us/step - loss: 0.2241 - acc
Epoch 6/10
10000/10000 [==============================] - 4s 361us/step - loss: 0.1916 - acc
Epoch 7/10
10000/10000 [==============================] - 4s 363us/step - loss: 0.1822 - acc
Epoch 8/10
10000/10000 [==============================] - 4s 359us/step - loss: 0.1551 - acc
Epoch 9/10
10000/10000 [==============================] - 4s 362us/step - loss: 0.1493 - acc
Epoch 10/10
10000/10000 [==============================] - 4s 361us/step - loss: 0.1530 - acc
Test loss: 0.11316825092687376
Test accuracy: 0.9545555710792542
(64, 20)
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 4s 390us/step - loss: 0.7140 - acc
Epoch 2/20
10000/10000 [==============================] - 4s 365us/step - loss: 0.5735 - acc
Epoch 3/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.3756 - acc
Epoch 4/20
10000/10000 [==============================] - 4s 367us/step - loss: 0.2704 - acc
Epoch 5/20
10000/10000 [==============================] - 4s 365us/step - loss: 0.2546 - acc
Epoch 6/20
10000/10000 [==============================] - 4s 367us/step - loss: 0.1911 - acc
Epoch 7/20
10000/10000 [==============================] - 4s 368us/step - loss: 0.1918 - acc
Epoch 8/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.1686 - acc
Epoch 9/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.1592 - acc
Epoch 10/20
10000/10000 [==============================] - 4s 365us/step - loss: 0.1560 - acc
Epoch 11/20
10000/10000 [==============================] - 4s 364us/step - loss: 0.1419 - acc
Epoch 12/20
10000/10000 [==============================] - 4s 368us/step - loss: 0.1347 - acc
```

```
10000/10000 [==============================] - 4s 368us/step - loss: 0.1347 - acc
Epoch 13/20
10000/10000 [==============================] - 4s 367us/step - loss: 0.1257 - acc
Epoch 14/20
10000/10000 [==============================] - 4s 367us/step - loss: 0.1265 - acc
Epoch 15/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.1077 - acc
Epoch 16/20
10000/10000 [==============================] - 4s 370us/step - loss: 0.1134 - acc
Epoch 17/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.1206 - acc
Epoch 18/20
10000/10000 [==============================] - 4s 367us/step - loss: 0.1000 - acc
Epoch 19/20
10000/10000 [==============================] - 4s 364us/step - loss: 0.0969 - acc
Epoch 20/20
10000/10000 [==============================] - 4s 366us/step - loss: 0.0836 - acc
Test loss: 0.29158892947600945
Test accuracy: 0.914222240447998
(64, 50)
Train on 10000 samples, validate on 2000 samples
Epoch 1/50
10000/10000 [==============================] - 4s 386us/step - loss: 0.7330 - acc
Epoch 2/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.5237 - acc
Epoch 3/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.3860 - acc
Epoch 4/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.2846 - acc
Epoch 5/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.2435 - acc
Epoch 6/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.2152 - acc
Epoch 7/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.1858 - acc
Epoch 8/50
10000/10000 [==============================] - 4s 368us/step - loss: 0.1730 - acc
Epoch 9/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.1678 - acc
Epoch 10/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.1409 - acc
Epoch 11/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.1369 - acc
Epoch 12/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.1373 - acc
Epoch 13/50
10000/10000 [==============================] - 4s 370us/step - loss: 0.1282 - acc
Epoch 14/50
10000/10000 [==============================] - 4s 371us/step - loss: 0.1241 - acc
Epoch 15/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.1130 - acc
Epoch 16/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.1129 - acc
Epoch 17/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.1273 - acc
Epoch 18/50
```

```
10000/10000 [==============================] - 4s 365us/step - loss: 0.1106 - acc
Epoch 19/50
10000/10000 [==============================] - 4s 363us/step - loss: 0.0992 - acc
Epoch 20/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0951 - acc
Epoch 21/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.1109 - acc
Epoch 22/50
10000/10000 [==============================] - 4s 363us/step - loss: 0.0889 - acc
Epoch 23/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.0852 - acc
Epoch 24/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0909 - acc
Epoch 25/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0965 - acc
Epoch 26/50
10000/10000 [==============================] - 4s 364us/step - loss: 0.0975 - acc
Epoch 27/50
10000/10000 [==============================] - 4s 363us/step - loss: 0.0840 - acc
Epoch 28/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0928 - acc
Epoch 29/50
10000/10000 [==============================] - 4s 371us/step - loss: 0.0855 - acc
Epoch 30/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0874 - acc
Epoch 31/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0738 - acc
Epoch 32/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0665 - acc
Epoch 33/50
10000/10000 [==============================] - 4s 369us/step - loss: 0.0817 - acc
Epoch 34/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0676 - acc
Epoch 35/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0722 - acc
Epoch 36/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0844 - acc
Epoch 37/50
10000/10000 [==============================] - 4s 368us/step - loss: 0.0847 - acc
Epoch 38/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0906 - acc
Epoch 39/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.0808 - acc
Epoch 40/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0811 - acc
Epoch 41/50
10000/10000 [==============================] - 4s 380us/step - loss: 0.0838 - acc
Epoch 42/50
10000/10000 [==============================] - 4s 378us/step - loss: 0.0727 - acc
Epoch 43/50
10000/10000 [==============================] - 4s 376us/step - loss: 0.0694 - acc
Epoch 44/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.0825 - acc
Epoch 45/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0720 - acc
Epoch 46/50
```

```
Epoch 46/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0862 - acc
Epoch 47/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0672 - acc
Epoch 48/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.0832 - acc
Epoch 49/50
10000/10000 [==============================] - 4s 365us/step - loss: 0.0640 - acc
Epoch 50/50
10000/10000 [==============================] - 4s 366us/step - loss: 0.0883 - acc
Test loss: 0.1290552060239845
Test accuracy: 0.9661111235618591
(128, 10)
Train on 10000 samples, validate on 2000 samples
Epoch 1/10
10000/10000 [==============================] - 4s 396us/step - loss: 0.7873 - acc
Epoch 2/10
10000/10000 [==============================] - 3s 341us/step - loss: 0.6836 - acc
Epoch 3/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.5773 - acc
Epoch 4/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.3939 - acc
Epoch 5/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.2993 - acc
Epoch 6/10
10000/10000 [==============================] - 3s 339us/step - loss: 0.2531 - acc
Epoch 7/10
10000/10000 [==============================] - 3s 339us/step - loss: 0.2142 - acc
Epoch 8/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.1973 - acc
Epoch 9/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.1799 - acc
Epoch 10/10
10000/10000 [==============================] - 3s 338us/step - loss: 0.1690 - acc
Test loss: 0.24936346492378247
Test accuracy: 0.913777768611908
(128, 20)
Train on 10000 samples, validate on 2000 samples
Epoch 1/20
10000/10000 [==============================] - 4s 365us/step - loss: 0.7039 - acc
Epoch 2/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.6142 - acc
Epoch 3/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.3925 - acc
Epoch 4/20
10000/10000 [==============================] - 3s 338us/step - loss: 0.2977 - acc
Epoch 5/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.2641 - acc
Epoch 6/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.2664 - acc
Epoch 7/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.2197 - acc
Epoch 8/20
10000/10000 [==============================] - 3s 337us/step - loss: 0.1875 - acc
Epoch 9/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.2516 - acc
```

```
Epoch 10/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.1649 - acc
Epoch 11/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.1532 - acc
Epoch 12/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.1525 - acc
Epoch 13/20
10000/10000 [==============================] - 3s 340us/step - loss: 0.1269 - acc
Epoch 14/20
10000/10000 [==============================] - 3s 340us/step - loss: 0.1428 - acc
Epoch 15/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.1568 - acc
Epoch 16/20
10000/10000 [==============================] - 3s 338us/step - loss: 0.1116 - acc
Epoch 17/20
10000/10000 [==============================] - 3s 338us/step - loss: 0.1182 - acc
Epoch 18/20
10000/10000 [==============================] - 3s 338us/step - loss: 0.0978 - acc
Epoch 19/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.1031 - acc
Epoch 20/20
10000/10000 [==============================] - 3s 339us/step - loss: 0.0967 - acc
Test loss: 0.1407008296889253
Test accuracy: 0.9626666903495789
(128, 50)
Train on 10000 samples, validate on 2000 samples
Epoch 1/50
10000/10000 [==============================] - 4s 367us/step - loss: 0.7483 - acc
Epoch 2/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.6882 - acc
Epoch 3/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.6133 - acc
Epoch 4/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.3848 - acc
Epoch 5/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.2975 - acc
Epoch 6/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.2868 - acc
Epoch 7/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.2189 - acc
Epoch 8/50
10000/10000 [==============================] - 3s 337us/step - loss: 0.2009 - acc
Epoch 9/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.1944 - acc
Epoch 10/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.1882 - acc
Epoch 11/50
10000/10000 [==============================] - 3s 341us/step - loss: 0.1529 - acc
Epoch 12/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.1657 - acc
Epoch 13/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.1464 - acc
Epoch 14/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.1599 - acc
Epoch 15/50
```

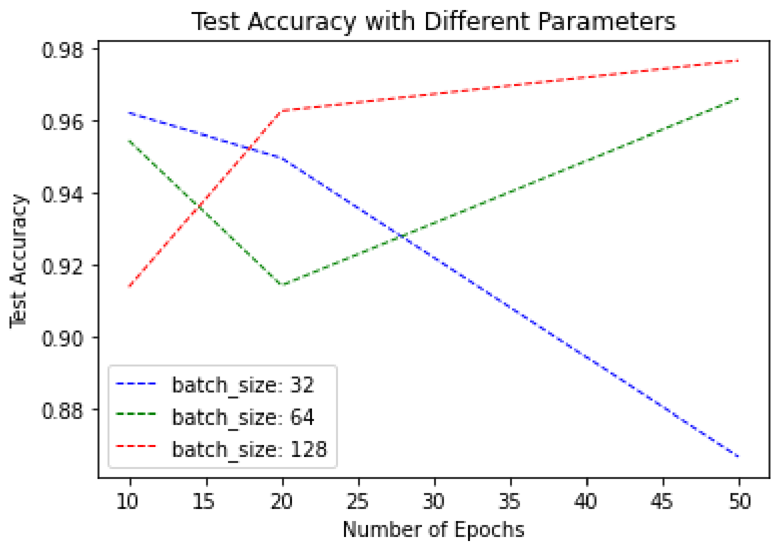
```
10000/10000 [==============================] - 3s 339us/step - loss: 0.1357 - acc
Epoch 16/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.1297 - acc
Epoch 17/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.1240 - acc
Epoch 18/50
10000/10000 [==============================] - 3s 343us/step - loss: 0.1130 - acc
Epoch 19/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.1147 - acc
Epoch 20/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.0984 - acc
Epoch 21/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0988 - acc
Epoch 22/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.0911 - acc
Epoch 23/50
10000/10000 [==============================] - 3s 337us/step - loss: 0.1230 - acc
Epoch 24/50
10000/10000 [==============================] - 3s 337us/step - loss: 0.0868 - acc
Epoch 25/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.0802 - acc
Epoch 26/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0718 - acc
Epoch 27/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.0749 - acc
Epoch 28/50
10000/10000 [==============================] - 3s 341us/step - loss: 0.0805 - acc
Epoch 29/50
10000/10000 [==============================] - 3s 341us/step - loss: 0.0706 - acc
Epoch 30/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0746 - acc
Epoch 31/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0639 - acc
Epoch 32/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0650 - acc
Epoch 33/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0581 - acc
Epoch 34/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0660 - acc
Epoch 35/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0671 - acc
Epoch 36/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0643 - acc
Epoch 37/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0535 - acc
Epoch 38/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0590 - acc
Epoch 39/50
10000/10000 [==============================] - 3s 342us/step - loss: 0.0634 - acc
Epoch 40/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0527 - acc
Epoch 41/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0452 - acc
Epoch 42/50
10000/10000 [==============================] - 3s 338us/step - loss: 0.0577 - acc
Epoch 43/50
```

```
10000/10000 [==============================] - 3s 338us/step - loss: 0.0558 - acc
Epoch 44/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0547 - acc
Epoch 45/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0339 - acc
Epoch 46/50
10000/10000 [==============================] - 3s 340us/step - loss: 0.0677 - acc
Epoch 47/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0589 - acc
Epoch 48/50
10000/10000 [==============================] - 3s 339us/step - loss: 0.0449 - acc
Epoch 49/50
10000/10000 [==============================] - 3s 344us/step - loss: 0.0487 - acc
Epoch 50/50
10000/10000 [==============================] - 3s 345us/step - loss: 0.0435 - acc
Test loss: 0.09219902786395202
Test accuracy: 0.9766666889190674
The best parameter combination is a batch size of 128 and 50 epochs, with a testi
```

```
import matplotlib.pyplot as plt

#Graph the results of the parameter sweep

#Create a dataset showing the parameter options and the accuracy score
d = {'combinations': parameter_combinations[0:len(scores)], 'accuracy_score': scores}
parameter_test_df = pd.DataFrame(data=d)
parameter_test_df["batch_size"] = parameter_test_df["combinations"].apply(lambda x: x
parameter_test_df["epochs"] = parameter_test_df["combinations"].apply(lambda x: x[1])
```

```
#Plot line for each batch_size showing accuracy by epoch
batch_df = parameter_test_df[parameter_test_df["batch_size"]==32]
plt.plot('epochs', 'accuracy_score', data=batch_df, marker='', color = 'blue', linewi

batch_df = parameter_test_df[parameter_test_df["batch_size"]==64]
plt.plot('epochs', 'accuracy_score', data=batch_df, marker='', color = 'green', linew

batch_df = parameter_test_df[parameter_test_df["batch_size"]==128]
plt.plot('epochs', 'accuracy_score', data=batch_df, marker='', color = 'red', linewid

plt.legend()

plt.ylabel('Test Accuracy')
plt.xlabel('Number of Epochs')
plt.title('Test Accuracy with Different Parameters')
```

```
Text(0.5, 1.0, 'Test Accuracy with Different Parameters')
```




```
parameter_test_df
```

	combinations	accuracy_score	batch_size	epochs
0	(32, 10)	0.962111	32	10
1	(32, 20)	0.949667	32	20
2	(32, 50)	0.866667	32	50
3	(64, 10)	0.954556	64	10
4	(64, 20)	0.914222	64	20
5	(64, 50)	0.966111	64	50
6	(128, 10)	0.913778	128	10
7	(128, 20)	0.962667	128	20
8	(128, 50)	0.976667	128	50

Train/Test Best Model annd Calculate Metrics

```
best_batch_size = best_combination[0]
best_epoch_number = best_combination[1]

#For testing:
#best_batch_size = 32
#best_epoch_number = 10


#train model 3 with the best parameter combination
M3 = create_model()

history = fit_and_evaluate_model(M3, X_train, X_test, X_val, y_train, y_test, y_val, 
```

```
)000/10000 [==============================] - 3s 338us/step - loss: 0.6730 - accur
poch 3/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.5186 - accur
poch 4/50
)000/10000 [==============================] - 3s 335us/step - loss: 0.4324 - accur
poch 5/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.3281 - accur
poch 6/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.3109 - accur
poch 7/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.2702 - accur
poch 8/50
)000/10000 [==============================] - 3s 339us/step - loss: 0.2368 - accur
poch 9/50
)000/10000 [==============================] - 3s 342us/step - loss: 0.2358 - accur
poch 10/50
)000/10000 [==============================] - 3s 341us/step - loss: 0.1929 - accur
poch 11/50
)000/10000 [==============================] - 3s 340us/step - loss: 0.1812 - accur
poch 12/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.1541 - accur
poch 13/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.1423 - accur
poch 14/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1361 - accur
poch 15/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1317 - accur
poch 16/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1260 - accur
poch 17/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1257 - accur
poch 18/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1122 - accur
poch 19/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1124 - accur
poch 20/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.1120 - accur
poch 21/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.0984 - accur
poch 22/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.1190 - accur
poch 23/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.0884 - accur
poch 24/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.0863 - accur
poch 25/50
)000/10000 [==============================] - 3s 336us/step - loss: 0.0942 - accur
poch 26/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.0908 - accur
poch 27/50
)000/10000 [==============================] - 3s 337us/step - loss: 0.0811 - accur
poch 28/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.0763 - accur
poch 29/50
)000/10000 [==============================] - 3s 338us/step - loss: 0.0881 - accur
```

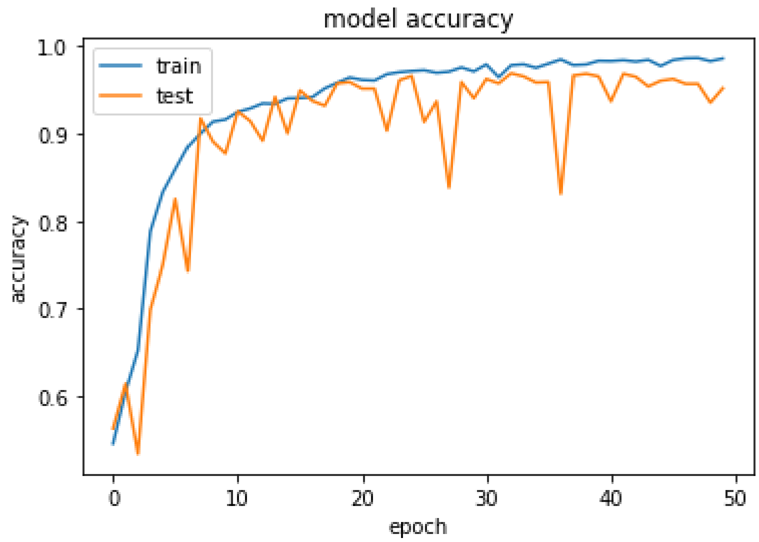
```
poch 30/50
0000/10000 [==============================] - 3s 338us/step - loss: 0.0738 - accur
poch 31/50
0000/10000 [==============================] - 3s 339us/step - loss: 0.0668 - accur
poch 32/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0706 - accur
poch 33/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0715 - accur
poch 34/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0688 - accur
poch 35/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0623 - accur
poch 36/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0676 - accur
poch 37/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0765 - accur
poch 38/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0538 - accur
poch 39/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0718 - accur
poch 40/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0701 - accur
poch 41/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0545 - accur
poch 42/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0596 - accur
poch 43/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0578 - accur
poch 44/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0686 - accur
poch 45/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0493 - accur
poch 46/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0744 - accur
poch 47/50
0000/10000 [==============================] - 3s 338us/step - loss: 0.0530 - accur
poch 48/50
0000/10000 [==============================] - 3s 338us/step - loss: 0.0551 - accur
poch 49/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0913 - accur
poch 50/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0657 - accur
est loss: 0.10566971201340979
est accuracy: 0.9712222218513489
rain on 10000 samples, validate on 2000 samples
poch 1/50
0000/10000 [==============================] - 4s 362us/step - loss: 0.7569 - accur
poch 2/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.6645 - accur
poch 3/50
0000/10000 [==============================] - 3s 340us/step - loss: 0.6694 - accur
poch 4/50
0000/10000 [==============================] - 3s 340us/step - loss: 0.5024 - accur
poch 5/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.4309 - accur
poch 6/50
```

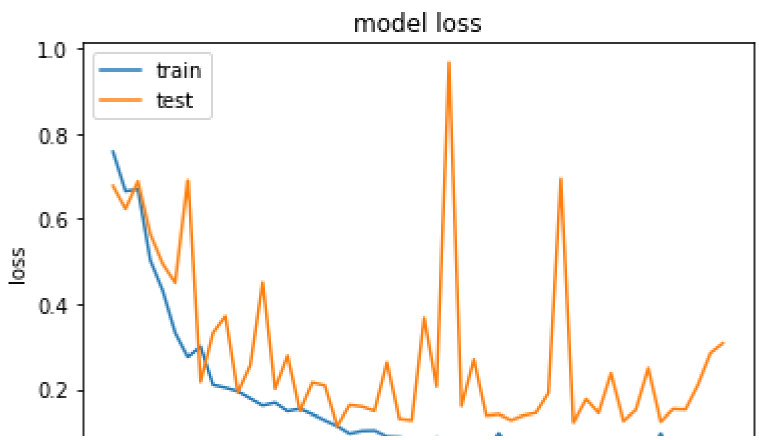
```
poch 6/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.3324 - accur
poch 7/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.2758 - accur
poch 8/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.3001 - accur
poch 9/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.2104 - accur
poch 10/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.2037 - accur
poch 11/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.1953 - accur
poch 12/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.1786 - accur
poch 13/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.1621 - accur
poch 14/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.1688 - accur
poch 15/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.1490 - accur
poch 16/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.1545 - accur
poch 17/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.1418 - accur
poch 18/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.1279 - accur
poch 19/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.1145 - accur
poch 20/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0967 - accur
poch 21/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.1031 - accur
poch 22/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.1039 - accur
poch 23/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0901 - accur
poch 24/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0886 - accur
poch 25/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0832 - accur
poch 26/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0797 - accur
poch 27/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0879 - accur
poch 28/50
0000/10000 [==============================] - 3s 337us/step - loss: 0.0832 - accur
poch 29/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0742 - accur
poch 30/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0844 - accur
poch 31/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0592 - accur
poch 32/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0973 - accur
poch 33/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0618 - accur
```

```
poch 34/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0735 - accur
poch 35/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0728 - accur
poch 36/50
0000/10000 [==============================] - 3s 333us/step - loss: 0.0548 - accur
poch 37/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0504 - accur
poch 38/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0683 - accur
poch 39/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0641 - accur
poch 40/50
0000/10000 [==============================] - 3s 333us/step - loss: 0.0558 - accur
poch 41/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0546 - accur
poch 42/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0454 - accur
poch 43/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0545 - accur
poch 44/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0450 - accur
poch 45/50
0000/10000 [==============================] - 3s 336us/step - loss: 0.0970 - accur
poch 46/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0464 - accur
poch 47/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0391 - accur
poch 48/50
0000/10000 [==============================] - 3s 335us/step - loss: 0.0424 - accur
poch 49/50
0000/10000 [==============================] - 3s 334us/step - loss: 0.0570 - accur
poch 50/50
0000/10000 [==============================] - 3s 338us/step - loss: 0.0470 - accur
est loss: 0.1492071437138899
est accuracy: 0.9687777757644653
```

```
#make model history plots
model_history_plots(history)
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```






```
#print confusion matrix and other metriics
confusion_matrix(M3, y_test, X_test)
```

```
Accuracy Score : 0.9687777777777777
Precision Score : 0.9927230945997702
Recall Score : 0.972
F1 Score : 0.9822522579422726
Confusion Matrix :
[[ 943   57]
 [ 224 7776]]
```

Checking Images Misclassified (M3)

```
best_model = M3
```

```
#Referenced classmate HWs and MUSA650 class lectures for functions here

import collections, numpy

y_test_true_tmp = np.argmax(y_test, axis = 1)
```

```
print(y_test_true_tmp)
unique, counts = numpy.unique(test_labels, return_counts=True)
label_count = [8000, 1000] #HARD CODED THIS - MIGHT FIND BETTER WAY
label_Dict = dict(zip([0, 1], ["no damage","damage"]))

def eval_model_by_class(model, test_set):
  ''' Determine correct number and percentage of predictions per label class.
  '''
  y_test_pred_tmp = model.predict_classes(test_set)
  y_test_true = [label_Dict[x] for x in y_test_true_tmp]
  y_test_pred = [label_Dict[x] for x in y_test_pred_tmp]

  pred_df = pd.DataFrame({'y_true': y_test_true, 'y_pred': y_test_pred})
  pred_df['accurate_preds'] = pred_df.y_true == pred_df.y_pred
  pred_df = pred_df.groupby(['y_true']).sum().reset_index()
  pred_df['label_count'] = label_count
  pred_df['class_acc'] = pred_df.accurate_preds / pred_df.label_count
  pred_df = pred_df.sort_values(by = 'class_acc').reset_index()
  pred_df['overall_acc'] = sum(pred_df.accurate_preds) / sum(pred_df.label_count)
  pred_df = pred_df.sort_values('y_true').reset_index(drop = True)

  return(pred_df)

def find_wrong_preds(model, test_set):
  ''' Find wrong predictions.
  '''
  y_test_pred_tmp = model.predict_classes(test_set)

  y_test_true = [label_Dict[x] for x in y_test_true_tmp]
  y_test_pred = [label_Dict[x] for x in y_test_pred_tmp]

  pred_df = pd.DataFrame({'y_true': y_test_true, 'y_pred': y_test_pred})
  pred_df['accurate_preds'] = pred_df.y_true == pred_df.y_pred
  pred_df = pred_df.sort_values('y_true')

  return(pred_df)
```

```
[0 0 0 ... 1 1 1]
```

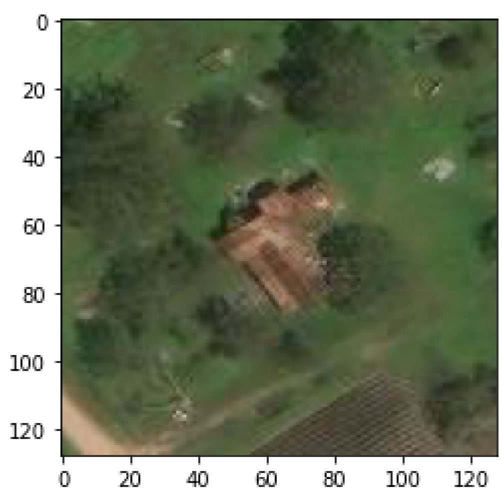
```
wrong_preds = find_wrong_preds(best_model, X_test)
```

```
wrong_preds
```

	y_true	y_pred	accurate_preds
4499	damage	damage	True
6008	damage	damage	True
6007	damage	damage	True
6006	damage	damage	True
6005	damage	damage	True
...	...	...	...
660	no damage	no damage	True
659	no damage	no damage	True

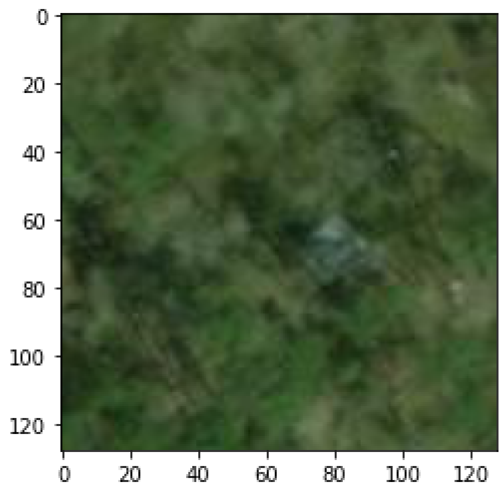
```
#Find images of from each label class that were labeled incorrectly.

wrong_preds_actually_damage = wrong_preds[(wrong_preds.accurate_preds == False) &
                                          (wrong_preds.y_true == "damage")]

wrong_preds_actually_no_damage = wrong_preds[(wrong_preds.accurate_preds == False) &
                                             (wrong_preds.y_true == "no damage")]


# Example 1
i = wrong_preds_actually_damage.index[0]

print("This is an image labeled as having " + str(label_Dict[y_test_true_tmp[i]]))
print("It was mislabeled as having " + str(wrong_preds_actually_damage.y_pred[i]))
plt.imshow(X_test[i]);
```

```
This is an image labeled as having damage
It was mislabeled as having no damage
```



```
# Example 2
i = wrong_preds_actually_no_damage.index[0]

print("This is an image labeled as having " + str(label_Dict[y_test_true_tmp[i]]))
print("It was mislabeled as having " + str(wrong_preds_actually_no_damage.y_pred[i]))
plt.imshow(X_test[i]);
```

```
This is an image labeled as having no damage
It was mislabeled as having damage
```



```
#Show more misclassification examples
fig, ax = plt.subplots(nrows=2, ncols=5, sharex=True, sharey=True, figsize=(25,10))

ax = ax.flatten()

for j in range(5):
    i = wrong_preds_actually_damage.index[j]
    ax[j].imshow(X_test[i])
    ax[j].set_title('True: damage; Pred: no damage')

for j in range(5, 10):
    i = wrong_preds_actually_no_damage.index[j]
    ax[j].imshow(X_test[i])
    ax[j].set_title('True: no damage; Pred: damage')

plt.show()
```


```
model_pred_df = eval_model_by_class(best_model, X_test)
model_pred_df
```

	index	y_true	accurate_preds	label_count	class_acc	overall_acc
0	0	damage	7776.0	8000	0.972	0.968778
1	1	no damage	943.0	1000	0.943	0.968778

ROC Curve

```
from sklearn.metrics import roc_curve
y_pred = M3.predict_proba(X_test)[:,1]
#y_pred_ohe = M3.predict_proba(X_test)[:,1]
#y_pred_labels = np.argmax(y_pred_ohe, axis=1)
y_test_labels = np.argmax(y_test, axis=1)
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test_labels, y_pred)
```

```
from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)
auc_keras
```

```
0.9907734375000001
```

```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```