

Package imports

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Flatten, Dropout
from PIL import Image
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras import Sequential
from sklearn.model_selection import train_test_split
```

Explore directory to figure out what is where

```
In [2]: import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/pokemon-images-and-types/pokemon.csv
/kaggle/input/pokemon-images-and-types/images/images/horsea.png
/kaggle/input/pokemon-images-and-types/images/images/lotad.png
/kaggle/input/pokemon-images-and-types/images/images/klefki.png
/kaggle/input/pokemon-images-and-types/images/images/gumshoos.jpg
/kaggle/input/pokemon-images-and-types/images/images/oshawott.png
/kaggle/input/pokemon-images-and-types/images/images/furret.png
/kaggle/input/pokemon-images-and-types/images/images/sneasel.png
/kaggle/input/pokemon-images-and-types/images/images/nidoking.png
/kaggle/input/pokemon-images-and-types/images/images/ribombee.jpg
/kaggle/input/pokemon-images-and-types/images/images/charjabug.jpg
/kaggle/input/pokemon-images-and-types/images/images/unfezant.png
/kaggle/input/pokemon-images-and-types/images/images/timburr.png
/kaggle/input/pokemon-images-and-types/images/images/furfrou.png
/kaggle/input/pokemon-images-and-types/images/images/durant.png
/kaggle/input/pokemon-images-and-types/images/images/entei.png
/kaggle/input/pokemon-images-and-types/images/images/kabutops.png
/kaggle/input/pokemon-images-and-types/images/images/zigzafoon.png
/kaggle/input/pokemon-images-and-types/images/images/sandslash.png
```

Load the info data frame and look around

```
In [3]: manifest=pd.read_csv('/kaggle/input/pokemon-images-and-types/pokemon.csv')
```

```
In [4]: manifest.head()
```

Out[4]:

	Name	Type1	Type2
0	bulbasaur	Grass	Poison
1	ivysaur	Grass	Poison
2	venusaur	Grass	Poison
3	charmander	Fire	NaN
4	charmeleon	Fire	NaN

```
In [5]: manifest.describe()
```

Out[5]:

	Name	Type1	Type2
count	809	809	405
unique	809	18	18
top	fearow	Water	Flying
freq	1	114	95

Confirm that type 2 is only column with NaNs

```
In [6]: np.any(manifest.Name.isna()),np.any(manifest.Type1.isna()),np.any(manifest.Type2.isna())
```

Out[6]: (False, False, True)

Add a class for missing values

```
In [7]: manifest.Type2[manifest.Type2.isna()] = "No Second Type"
```

One-hot encode ys

```
In [8]: y1=pd.get_dummies(manifest.Type1)
```

```
In [9]: y2=pd.get_dummies(manifest.Type2)
```

```
In [10]: y1.head()
```

Out[10]:

	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

```
In [11]: y2
```

Out[11]:

	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	No Second Type	Normal	Poison	Psychic	Rock	Steel	Water
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
...	
804	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
805	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
806	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
807	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
808	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	

809 rows × 19 columns

I just rearranged the columns here, but turns out it might not have been necessary

```
In [12]: fixcols=y2.columns.tolist()[0:12]
for i in y2.columns.tolist()[13:19]:
    fixcols.append(i)
fixcols.append(y2.columns.tolist()[12])
y2=y2[fixcols]
```

Add a column for 'No second type' class to first class, to allow combining the two sets

```
In [13]: y1['No Second Type']=0
```

```
In [14]: y1.values.shape,y2.values.shape
y2+y1
```

Out[14]:

	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
...
804	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
805	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
806	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
807	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
808	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

809 rows × 19 columns

Concatenate the pokemon names with their one-hot encoded types

```
In [15]: manifest=pd.concat([manifest['Name'].reset_index(drop=True), y2+y1], axis=1)
```

In [16]: manifest

Out[16]:

	Name	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
0	bulbasaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
1	ivysaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
2	venusaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
3	charmander	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
4	charmeleon	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
...	
804	stakataka	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
805	blacephalon	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
806	zeraora	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
807	meltan	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
808	melmetal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

809 rows × 20 columns



Look at an example image and find out array dimensions

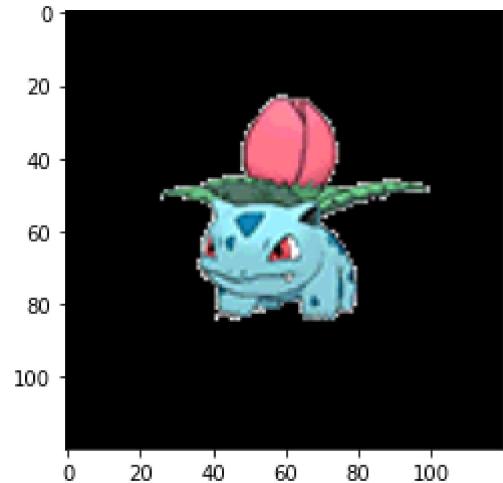
In [17]: # Load an example image

```
img = load_img('/kaggle/input/pokemon-images-and-types/images/images/ivysaur.png')
print(type(img))
# convert to numpy array
img_array = img_to_array(img)
print(img_array.dtype)
print(img_array.shape)
#show image
plt.imshow(img);
```

<class 'PIL.Image.Image'>

float32

(120, 120, 3)



Check pixel value range

In [18]: `np.max(img_array)`

Out[18]: 255.0

Obtain and normalise x values between (0,1)

In [19]: `names = []
index = 0

for dirname, _, filenames in os.walk('/kaggle/input/pokemon-images-and-types/images/images/'):
 for filename in filenames:
 file_path_i = os.path.join(dirname, filename)
 name_i = filename.replace('.png', '')
 names.append({'Name':name_i, 'ix':index})
 if index==0:
 x=load_img(file_path_i)
 x=img_to_array(x)
 x.shape=(1,120,120,3)
 x=x/255.
 else:
 xi=load_img(file_path_i)
 xi=img_to_array(xi)
 xi.shape=(1,120,120,3)
 xi=xi/255.
 x = np.concatenate((x,xi),axis=0)
 index+=1
#print(index)`

/opt/conda/lib/python3.7/site-packages/PIL/Image.py:968: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
expressed in bytes should be converted '+

In [20]: `names=pd.DataFrame(names)`

```
In [21]: names.head()
```

```
Out[21]:
```

	Name	ix
0	horsea	0
1	lotad	1
2	klefki	2
3	gumshoos.jpg	3
4	oshawott	4

Here I was just figuring out why merging on Name resulted in dropped values, turns out there are some images for which there is no data in the manifest, so ended up dropping those

```
In [22]: len(names.Name.unique()),len(manifest.Name.unique()),len(names.ix.unique())
```

```
Out[22]: (809, 809, 809)
```

```
In [23]: manifest=manifest.merge(names,how='left')
```

```
In [24]: np.where(manifest.ix.isna())
```

```
Out[24]: (array([721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733,
    734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746,
    747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759,
    760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772,
    773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785,
    786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798,
    799, 800, 801, 802, 803, 804, 805, 806, 807, 808]),)
```

```
In [25]: len(os.listdir('/kaggle/input/pokemon-images-and-types/images/images/'))
```

```
Out[25]: 809
```

```
In [26]: missing=[]
for i in manifest[manifest.ix.isna()].Name.values:
    missing.append(i in names.Name)
np.any(missing)
```

```
Out[26]: False
```

```
In [27]: dropme=np.where(manifest.ix.isna())
```

```
In [28]: dropme
```

```
Out[28]: (array([721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733,
       734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746,
       747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759,
       760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772,
       773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785,
       786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798,
       799, 800, 801, 802, 803, 804, 805, 806, 807, 808]),)
```

```
In [29]: manifest.dropna(inplace=True)
```

```
In [30]: manifest = manifest.set_index('ix',drop=True).sort_index()
```

```
In [31]: manifest
```

```
Out[31]:
```

	Name	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
ix																				
0.0	horsea	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1.0	lotad	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
2.0	klefki	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4.0	oshawott	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
5.0	furret	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
...	
801.0	swoobat	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
802.0	bastiodon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
803.0	mantine	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
806.0	volcarona	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
808.0	azumarill	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0

721 rows × 20 columns

```
In [32]: np.delete(x,dropme,axis=0).shape
```

```
Out[32]: (721, 120, 120, 3)
```

```
In [33]: x= np.delete(x,dropme,axis=0)
```

```
In [34]: manifest.drop('Name',axis=1).values.shape
```

```
Out[34]: (721, 19)
```

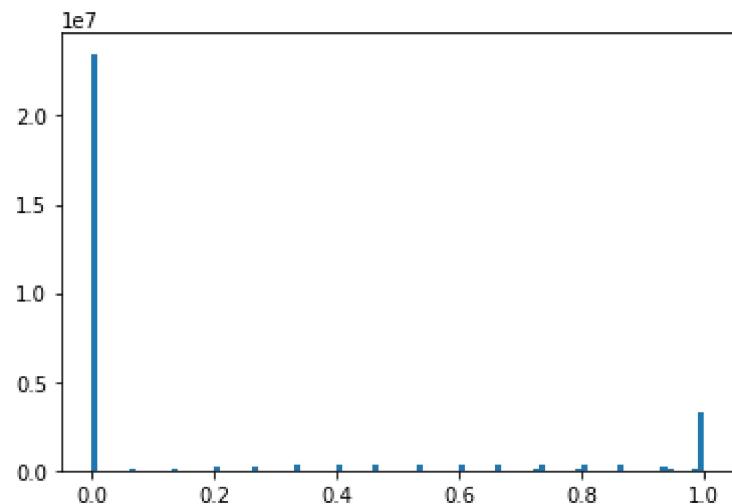
```
In [35]: y=manifest.drop('Name',axis=1).values
```

```
In [36]: type(y),type(x)
```

```
Out[36]: (numpy.ndarray, numpy.ndarray)
```

```
In [37]: x = np.float32(x)
```

```
In [38]: plt.hist(x.ravel(),bins=100);
```



First model

```
In [39]: model = Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),activation='tanh',input_shape=(x.shape[1],x.shape[2],x.shape[3])))
model.add(MaxPooling2D())
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(y.shape[1],activation='sigmoid'))
```

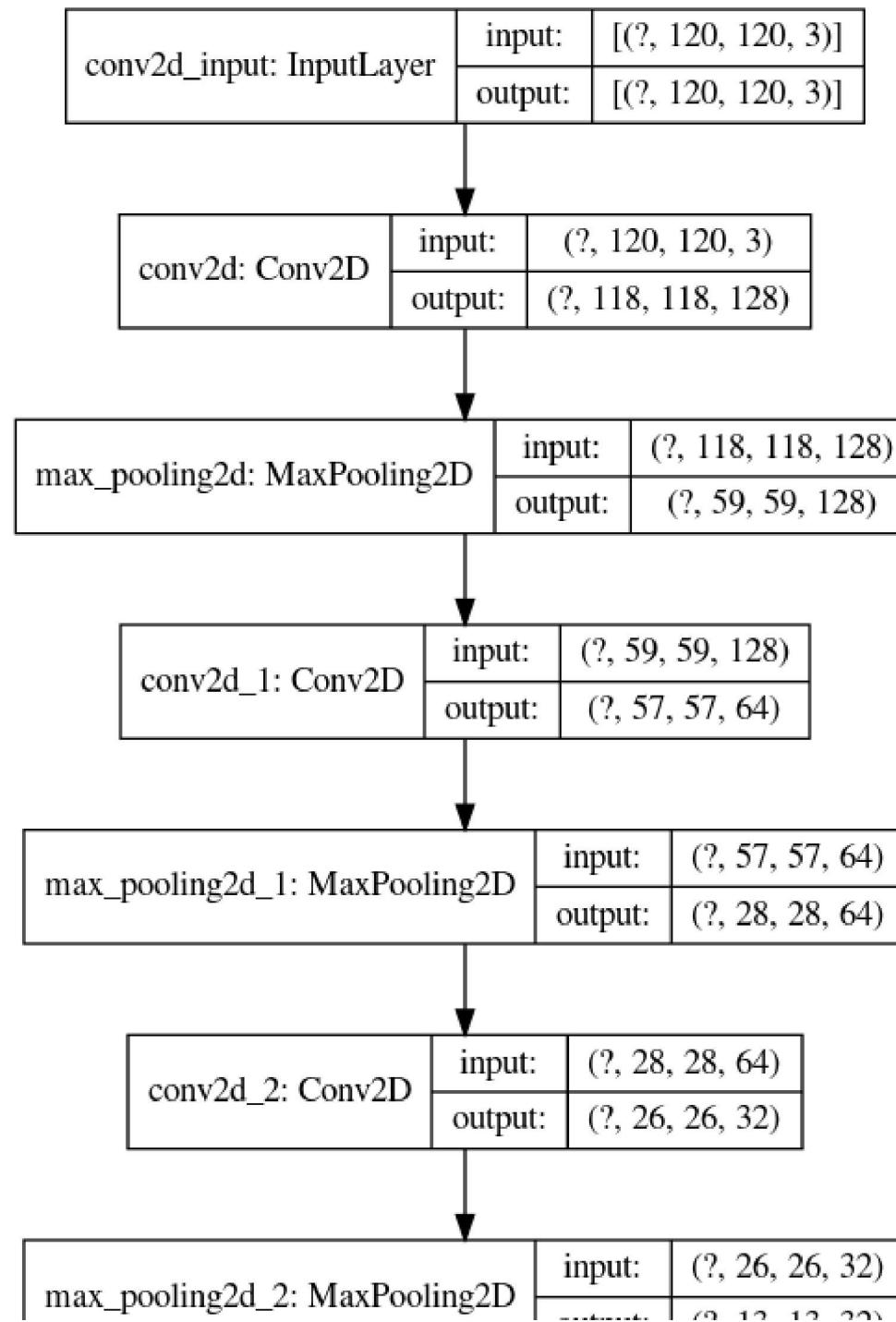
```
In [40]: model.summary()
```

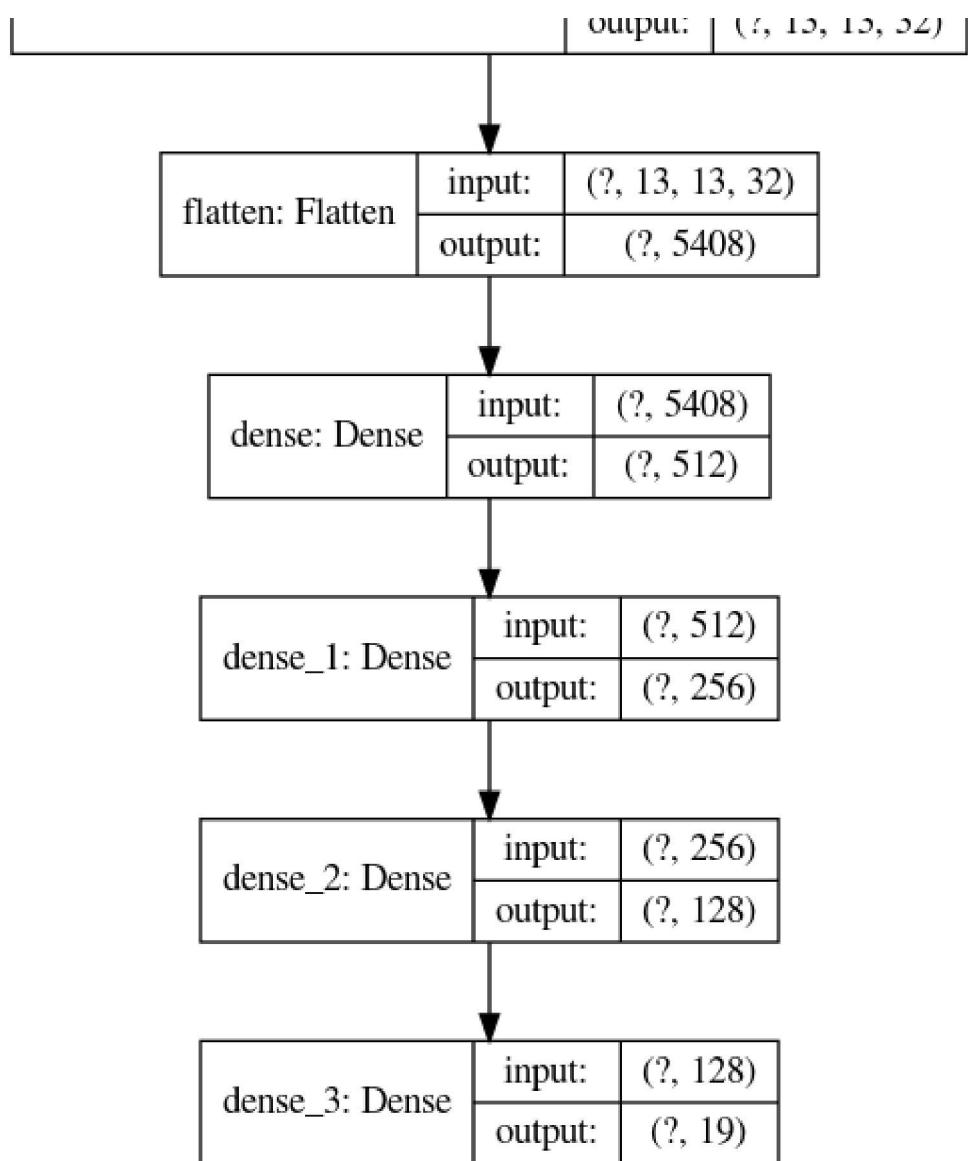
Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 118, 118, 128)	3584
<hr/>		
max_pooling2d (MaxPooling2D)	(None, 59, 59, 128)	0
<hr/>		
conv2d_1 (Conv2D)	(None, 57, 57, 64)	73792
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 28, 28, 64)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 26, 26, 32)	18464
<hr/>		
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
<hr/>		
flatten (Flatten)	(None, 5408)	0
<hr/>		
dense (Dense)	(None, 512)	2769408
<hr/>		
dense_1 (Dense)	(None, 256)	131328
<hr/>		
dense_2 (Dense)	(None, 128)	32896
<hr/>		
dense_3 (Dense)	(None, 19)	2451
<hr/>		
Total params: 3,031,923		
Trainable params: 3,031,923		
Non-trainable params: 0		

```
In [41]: tf.keras.utils.plot_model(model, show_shapes=True)
```

Out[41]:





```
In [46]: adam=tf.keras.optimizers.Adam(  
    learning_rate=0.000001,  
    beta_1=0.9,  
    beta_2=0.999,  
    epsilon=1e-07,  
    amsgrad=False)
```

```
In [47]: model.compile(optimizer=adam,  
                      loss="binary_crossentropy",  
                      metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=2)])
```

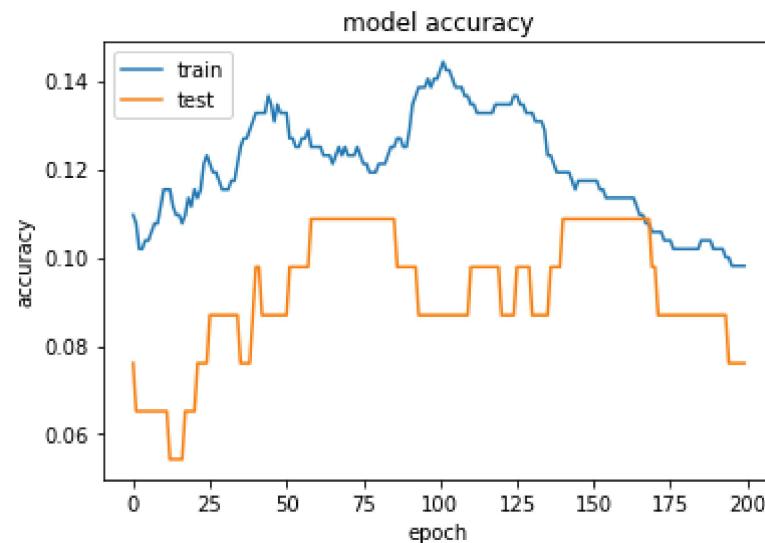
```
In [48]: X_train, X_test, y_train, y_test = train_test_split(  
          x, y, test_size=0.15, random_state=42)
```

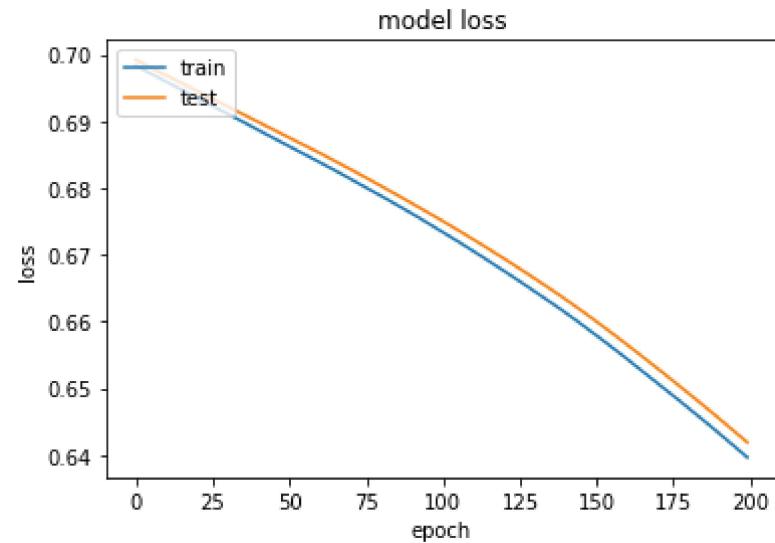
```
In [49]: history = model.fit(X_train,y_train,batch_size=32,epochs=200,validation_split=0.15,shuffle=True)
```

...

```
In [51]: print(history.history.keys())  
  
dict_keys(['loss', 'top_k_categorical_accuracy', 'val_loss', 'val_top_k_categorical_accuracy'])
```

```
In [53]: plt.plot(history.history['top_k_categorical_accuracy'])
plt.plot(history.history['val_top_k_categorical_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()
```





Second model

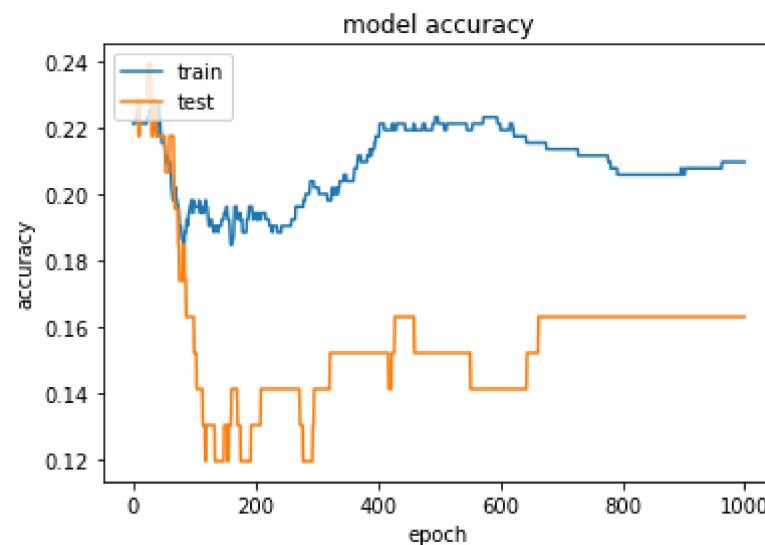
```
In [54]: model = Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),activation='tanh',input_shape=(x.shape[1],x.shape[2],x.shape[3])))
model.add(MaxPooling2D())
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(y.shape[1],activation='sigmoid'))

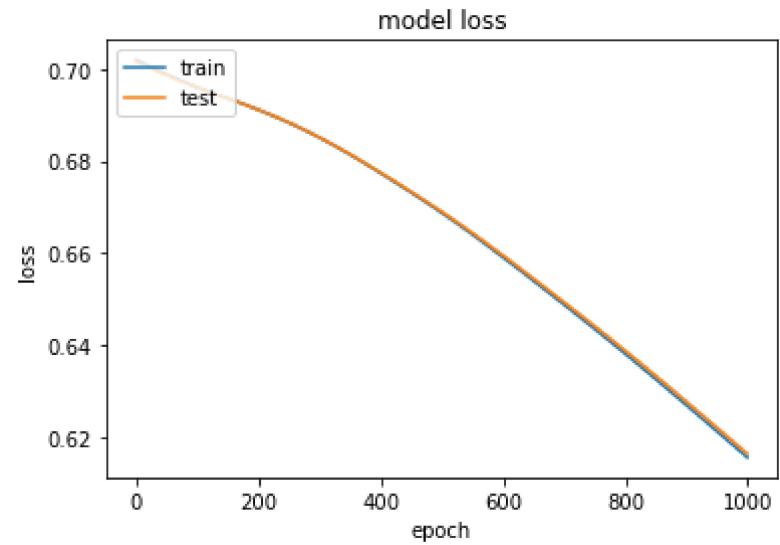
adam=tf.keras.optimizers.Adam(
    learning_rate=0.0000001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False)

model.compile(optimizer=adam,
              loss="binary_crossentropy",
              metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=5)])
history = model.fit(X_train,y_train,batch_size=32,epochs=1000,validation_split=0.15,shuffle=True)
```

...

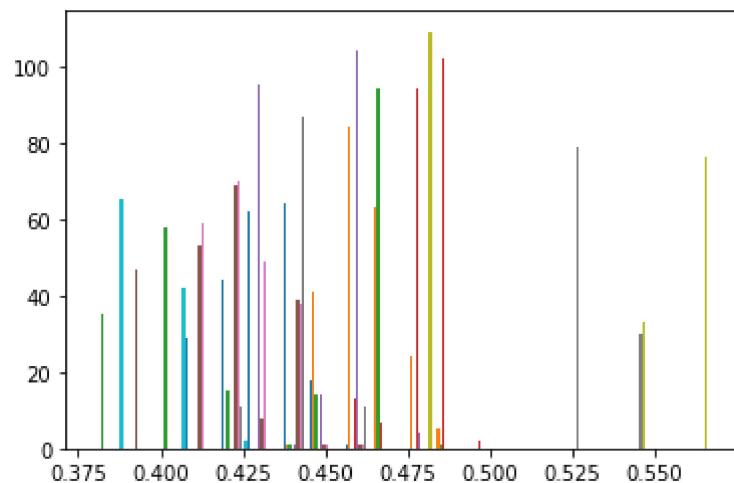
```
In [55]: plt.plot(history.history['top_k_categorical_accuracy'])
plt.plot(history.history['val_top_k_categorical_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
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()
```





Investigating ouput layer predictions, as I expected, it's hovering close to 0.5

```
In [59]: plt.hist(model.predict(X_test));
```



Another model

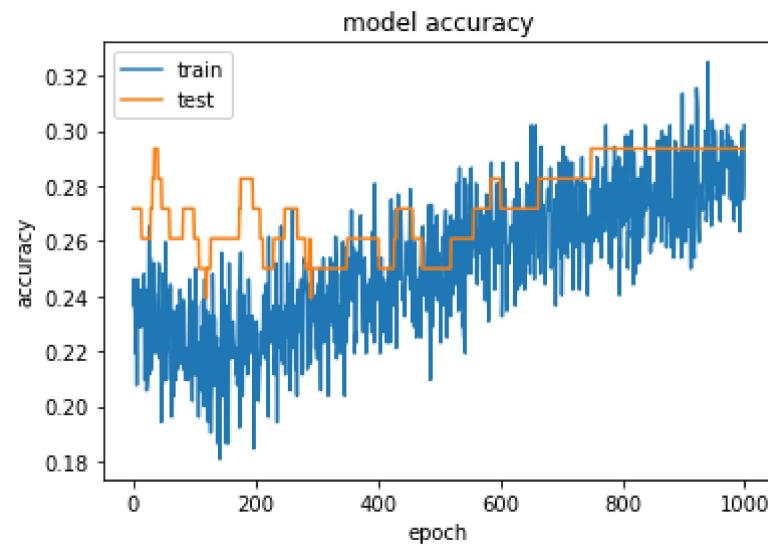
```
In [60]: model = Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),activation='tanh',input_shape=(x.shape[1],x.shape[2],x.shape[3])))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(512,activation='tanh'))
model.add(Dense(256,activation='tanh'))
model.add(Dense(128,activation='tanh'))
model.add(Dense(y.shape[1],activation='softmax'))

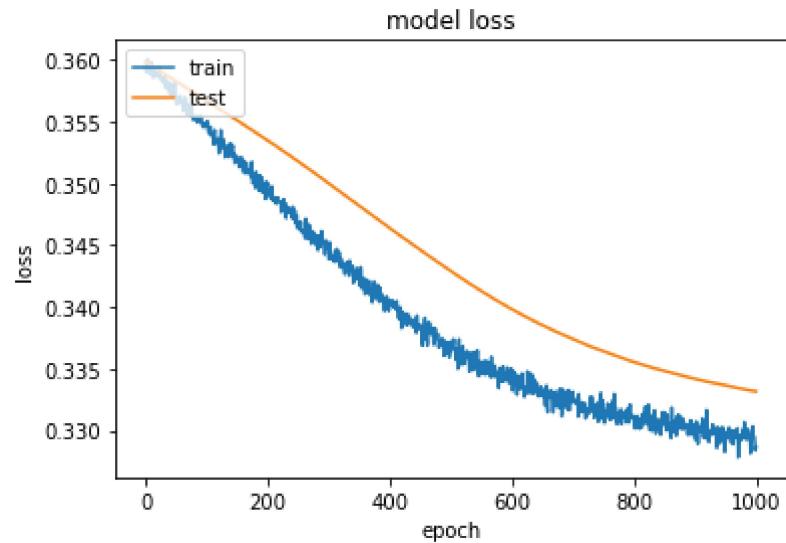
adam=tf.keras.optimizers.Adam(
    learning_rate=0.0000001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False)

model.compile(optimizer=adam,
              loss="binary_crossentropy",
              metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=5)])
history = model.fit(X_train,y_train,batch_size=32,epochs=1000,validation_split=0.15,shuffle=True)
```

...

```
In [61]: plt.plot(history.history['top_k_categorical_accuracy'])
plt.plot(history.history['val_top_k_categorical_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()
```

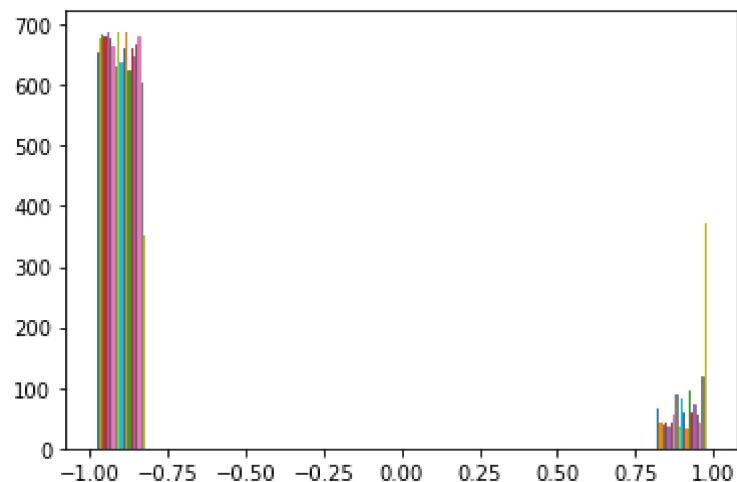




Here, I decided to give more range to the final layer's output by scaling y to (-1,1) and using a Tanh output activation function instead

In [89]: `new_y = np.where(y==0, -1, y)`

```
In [91]: plt.hist(new_y);
```



```
In [92]: X_train, X_test, y_train, y_test = train_test_split(  
    x, new_y, test_size=0.1, random_state=42)
```

Build and compile final model with MSE loss and tanh activation in final layer, very deep model (with perhaps too much?) dropout

In []:

```
model = Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),activation='tanh',input_shape=(x.shape[1],x.shape[2],x.shape[3])))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='tanh'))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(1024,activation='tanh'))
model.add(Dropout(rate=0.25))
model.add(Dense(512,activation='tanh'))
model.add(Dropout(rate=0.25))
model.add(Dense(256,activation='tanh'))
model.add(Dropout(rate=0.25))
model.add(Dense(128,activation='tanh'))
model.add(Dropout(rate=0.25))
model.add(Dense(64,activation='tanh'))
model.add(Dense(y.shape[1],activation='tanh'))

adam=tf.keras.optimizers.Adam(
    learning_rate=0.000001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False)

model.compile(optimizer=adam,
              loss="mse",
              metrics=[tf.keras.metrics.MeanAbsoluteError()])
history = model.fit(X_train,y_train,batch_size=128,epochs=2000,validation_split=0.1,shuffle=True)
```

Epoch 942/2000

5/5 [=====] - 0s 64ms/step - loss: 0.3585 - mean_absolute_error: 0.3439 - val_loss: 0.3361 - val_mean_absolute_error: 0.3045

Epoch 943/2000

```
5/5 [=====] - 0s 66ms/step - loss: 0.3579 - mean_absolute_error: 0.3448 - val_loss: 0.3361 - val_mean_absolute_error: 0.3044
Epoch 944/2000
5/5 [=====] - 0s 65ms/step - loss: 0.3581 - mean_absolute_error: 0.3436 - val_loss: 0.3361 - val_mean_absolute_error: 0.3044
Epoch 945/2000
5/5 [=====] - 0s 65ms/step - loss: 0.3591 - mean_absolute_error: 0.3432 - val_loss: 0.3361 - val_mean_absolute_error: 0.3043
Epoch 946/2000
5/5 [=====] - 0s 66ms/step - loss: 0.3612 - mean_absolute_error: 0.3461 - val_loss: 0.3361 - val_mean_absolute_error: 0.3042
Epoch 947/2000
5/5 [=====] - 0s 66ms/step - loss: 0.3595 - mean_absolute_error: 0.3432 - val_loss: 0.3361 - val_mean_absolute_error: 0.3042
Epoch 948/2000
```

Last training curve plots

```
In [ ]: plt.plot(history.history['top_k_categorical_accuracy'])
plt.plot(history.history['val_top_k_categorical_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()
```

```
In [ ]: preds=model.predict(X_test)
```

```
In [ ]: two_max_args = []
for i in range(preds.shape[0]):
```

