Package imports

In [9]:
```
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 [10]:
```
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/luxio.png
/kaggle/input/pokemon-images-and-types/images/images/oshawott.png
/kaggle/input/pokemon-images-and-types/images/images/parasect.png
/kaggle/input/pokemon-images-and-types/images/images/taillow.png
/kaggle/input/pokemon-images-and-types/images/images/porygon.png
/kaggle/input/pokemon-images-and-types/images/images/feraligatr.png
/kaggle/input/pokemon-images-and-types/images/images/eelektross.png
/kaggle/input/pokemon-images-and-types/images/images/scyther.png
/kaggle/input/pokemon-images-and-types/images/images/magmar.png
/kaggle/input/pokemon-images-and-types/images/images/swalot.png
/kaggle/input/pokemon-images-and-types/images/images/magneton.png
/kaggle/input/pokemon-images-and-types/images/images/skitty.png
/kaggle/input/pokemon-images-and-types/images/images/stunfisk.png
/kaggle/input/pokemon-images-and-types/images/images/clawitzer.png
/kaggle/input/pokemon-images-and-types/images/images/eelektrik.png
/kaggle/input/pokemon-images-and-types/images/images/poliwhirl.png
/kaggle/input/pokemon-images-and-types/images/images/whimsicott.png
/kaggle/input/pokemon-images-and-types/images/images/passimian.jpg
```

Load the info data frame and look around

In [11]:
```
manifest=pd.read_csv('/kaggle/input/pokemon-images-and-types/pokemon.csv')
```

In [12]:
```
manifest.head()
```

Out[12]:

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

In [13]:
```
manifest.describe()
```

Out[13]:

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

Confirm that type 2 is only column with NaNs

In [14]:
```
np.any(manifest.Name.isna()),np.any(manifest.Type1.isna()),np.any(manifest.Type2.isna())
```

Out[14]:
```
(False, False, True)
```

Add a class for missing values

In [15]:
```
manifest.Type2[manifest.Type2.isna()] = "No Second Type"
```

One-hot encode *y*s

In [16]:
```
y1=pd.get_dummies(manifest.Type1)
```

In [17]:
```
y2=pd.get_dummies(manifest.Type2)
```

In [18]:
```
y1.head()
```

Out[18]:

	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 [19]:
```
y2
```

Out[19]:

	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	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
804	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
805	0	0	0	0	0	0	0	0	1	0	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	0
807	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
808	0	0	0	0	0	0	0	0	0	0	0	0	1	0	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 [20]:
```
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 [21]:
```
y1['No Second Type']=0
```

In [22]:
```
y1.values.shape,y2.values.shape
y2+y1
```

Out[22]:

	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 [23]:
```
manifest=pd.concat([manifest['Name'].reset_index(drop=True), y2+y1], axis=1)
```

In [24]:
```
manifest
```

Out[24]:

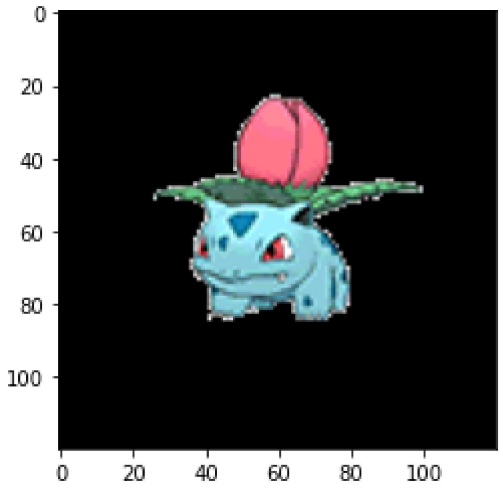
	Name	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
0	bulbasaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
1	ivysaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
2	venusaur	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
3	charmander	0	0	0	0	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 [25]:
```
# 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 [26]:
```
np.max(img_array)
```

Out[26]:
```
255.0
```

Obtain and normalise x values between (0,1)

In [27]:
```
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 shoul
d be converted to RGBA images
  ' expressed in bytes should be converted ' +
```

In [28]:
```
names=pd.DataFrame(names)
```

In [29]:
```
names.head()
```

Out[29]:

	Name	ix
0	luxio	0
1	oshawott	1
2	parasect	2
3	taillow	3
4	porygon	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 [30]:
```
len(names.Name.unique()),len(manifest.Name.unique()),len(names.ix.unique())
```

Out[30]:
```
(809, 809, 809)
```

In [31]:
```
manifest=manifest.merge(names,how='left')
```

In [32]:
```
np.where(manifest.ix.isna())
```

Out[32]:
```
(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 [33]:
```
len(os.listdir('/kaggle/input/pokemon-images-and-types/images/images/'))
```

Out[33]:
```
809
```

In [34]:
```
missing=[]
for i in manifest[manifest.ix.isna()].Name.values:
    missing.append(i in names.Name)
np.any(missing)
```

Out[34]:
```
False
```

In [35]:
```
dropme=np.where(manifest.ix.isna())
```

In [36]:
```
dropme
```

Out[36]:
```
(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 [37]:
```
manifest.dropna(inplace=True)
```

In [38]:
```
manifest = manifest.set_index('ix',drop=True).sort_index()
```

In [39]:
```
manifest
```

Out[39]:

ix	Name	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
0.0	luxio	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1.0	oshawott	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
2.0	parasect	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3.0	taillow	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
4.0	porygon	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
804.0	slaking	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
805.0	psyduck	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
806.0	galvantula	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
807.0	roggenrola	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
808.0	binacle	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

721 rows × 20 columns

In [40]:
```
np.delete(x,dropme,axis=0).shape
```

Out[40]:
```
(721, 120, 120, 3)
```

In [41]:
```
x= np.delete(x,dropme,axis=0)
```

In [42]:
```
manifest.drop('Name',axis=1).values.shape
```

Out[42]:
```
(721, 19)
```

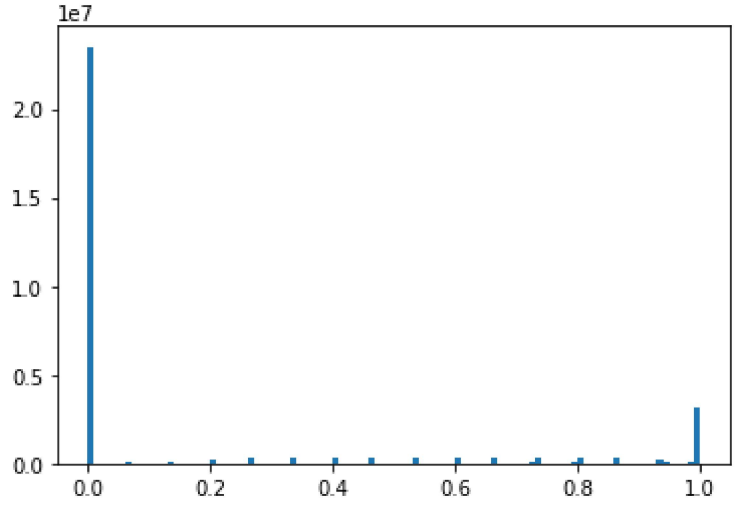
In [43]:
```
y=manifest.drop('Name',axis=1).values
```

In [44]:
```
type(y),type(x)
```

Out[44]:
```
(numpy.ndarray, numpy.ndarray)
```

In [45]:
```
x = np.float32(x)
```

In [46]:
```
plt.hist(x.ravel(),bins=100);
```



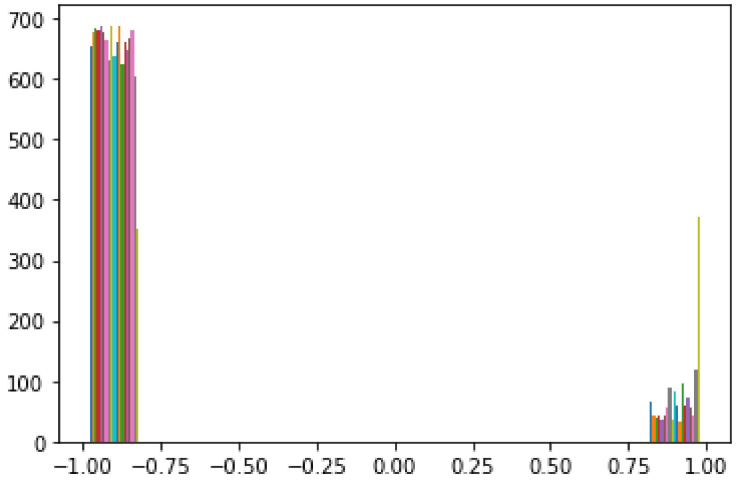
Note: my first goal was to classify both the first and second class simultaneously, later in the notebook I focussed only on classifying the first class

First model

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 [47]:
```
new_y = np.where(y==0, -1, y)
```

In [48]:
```
plt.hist(new_y);
```



In [49]:
```
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 [50]:
```
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 1/2000
5/5 [==============================] - 1s 195ms/step - loss: 1.0884 - mean_absolute_error: 1.0080 - val_loss: 1.0137 - val_mea
n_absolute_error: 1.0021
```

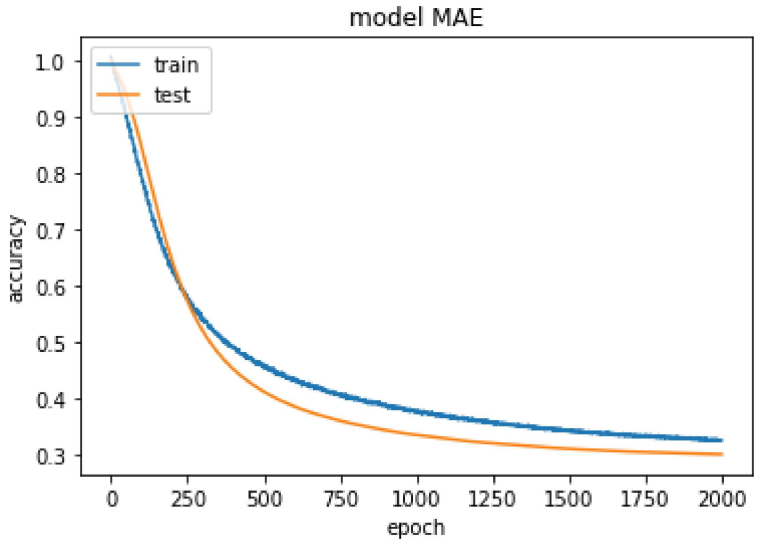
```
Epoch 2/2000
5/5 [==============================] - 0s 67ms/step - loss: 1.0845 - mean_absolute_error: 1.0062 - val_loss: 1.0109 - val_mean
_absolute_error: 1.0008
Epoch 3/2000
5/5 [==============================] - 0s 66ms/step - loss: 1.0747 - mean_absolute_error: 1.0020 - val_loss: 1.0083 - val_mean
_absolute_error: 0.9995
Epoch 4/2000
5/5 [==============================] - 0s 64ms/step - loss: 1.0701 - mean_absolute_error: 0.9996 - val_loss: 1.0056 - val_mean
_absolute_error: 0.9983
Epoch 5/2000
5/5 [==============================] - 0s 65ms/step - loss: 1.0660 - mean_absolute_error: 0.9984 - val_loss: 1.0030 - val_mean
_absolute_error: 0.9970
Epoch 6/2000
5/5 [==============================] - 0s 83ms/step - loss: 1.0633 - mean_absolute_error: 0.9964 - val_loss: 1.0004 - val_mean
_absolute_error: 0.9958
Epoch 7/2000
```

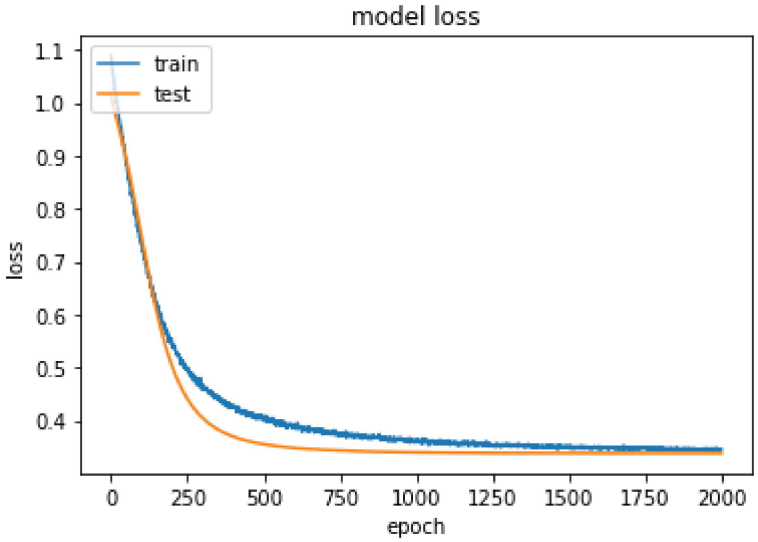
Last training curve plots

In [53]:
```
print(history.history.keys())
```

```
dict_keys(['loss', 'mean_absolute_error', 'val_loss', 'val_mean_absolute_error'])
```

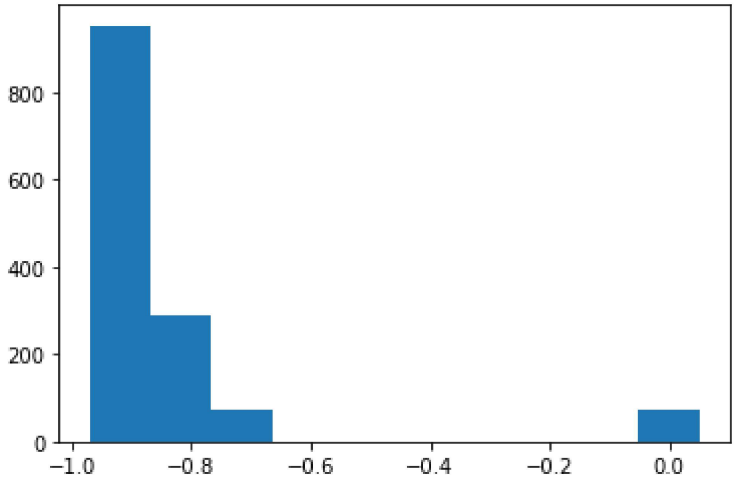
In [54]:
```
plt.plot(history.history['mean_absolute_error'])
plt.plot(history.history['val_mean_absolute_error'])
plt.title('model MAE')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
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 [55]:
```
preds=model.predict(X_test)
```

In [65]:
```
plt.hist(preds.ravel());
```



In [77]:
```
correct=0
for i, j in zip(preds.argsort(axis=1)[:,:2],y_test.argsort(axis=1)[:,:2]):
    for k in i:
        if k in j:
            correct+=1
print("top 2 test accuracy, since we are predicting two class labels",correct/(len(preds)*2))
```

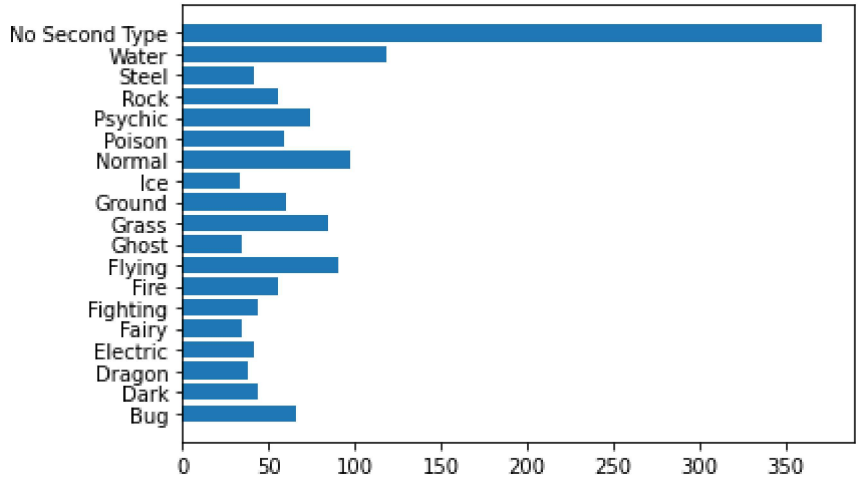
```
top 2 test accuracy, since we are predicting two class labels 0.4452054794520548
```

In [80]:
```
manifest.head()
```

Out[80]:

ix	Name	Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost	Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water	No Second Type
0.0	luxio	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1.0	oshawott	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
2.0	parasect	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3.0	taillow	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
4.0	porygon	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

In [106]:
```
plt.barh(y=range(19),width=np.array(manifest.sum(axis=0))[1:],tick_label=manifest.columns[1:],orientation='horizontal');
```



Since the 'No Second type class' completely predominates, we shift the problem over to only predicting the primary type...

Let's start over and predict only the first Type:

In [185]:
```
manifest=pd.read_csv('/kaggle/input/pokemon-images-and-types/pokemon.csv')
y=pd.get_dummies(manifest.Type1)
manifest=pd.concat([manifest['Name'].reset_index(drop=True), y], axis=1)
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)
names=pd.DataFrame(names)
manifest=manifest.merge(names,how='left')
dropme=np.where(manifest.ix.isna())
manifest.dropna(inplace=True)
manifest = manifest.set_index('ix',drop=True).sort_index()
x= np.delete(x,dropme,axis=0)
y=manifest.drop('Name',axis=1).values
x = np.float32(x)
X_train, X_test, y_train, y_test = train_test_split(
     x, y, test_size=0.15, random_state=42)
```

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

In [187]:
```
X_test.shape,y_test.shape
```

Out[187]:
```
((109, 120, 120, 3), (109, 18))
```

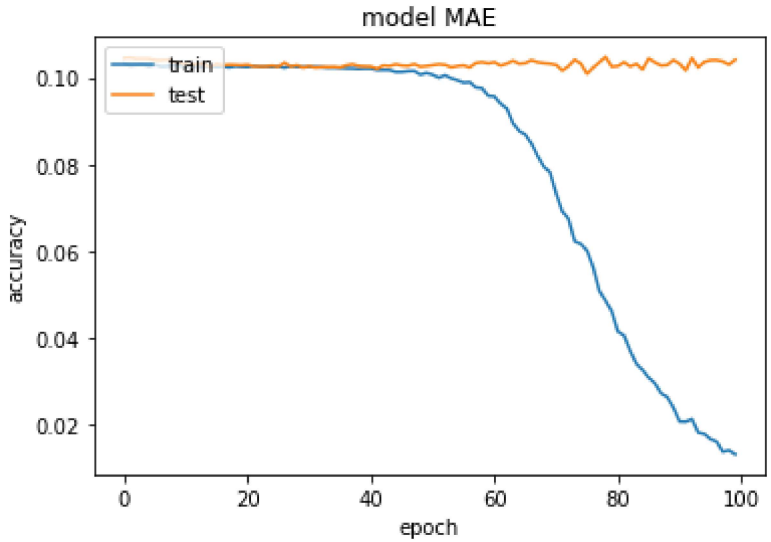
In [209]:

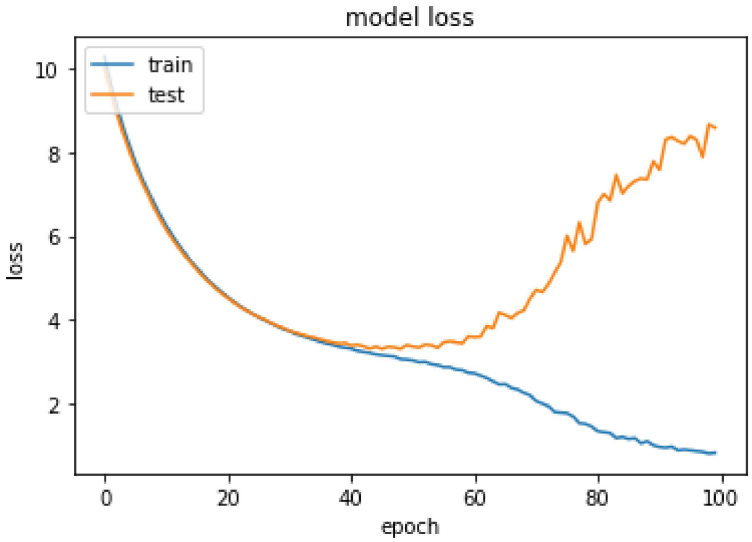
```
model = Sequential()
model.add(tf.keras.layers.GaussianNoise(stddev=0.2,input_shape=(x.shape[1],x.shape[2],x.shape[3])))
model.add(Conv2D(filters=256,kernel_size=(10,10),activation='relu',padding='same',kernel_regularizer=tf.keras.regularizers.l1_l2(l
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=128,kernel_size=(8,8),activation='relu',padding='same',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(6,6),activation='relu',padding='same',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(4,4),activation='relu',padding='same',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(MaxPooling2D())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',padding='same',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(4096,activation='relu',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(Dropout(rate=0.25))
model.add(Dense(4096,activation='relu',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(Dropout(rate=0.25))
model.add(Dense(4096,activation='relu',kernel_regularizer=tf.keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),
    bias_regularizer=tf.keras.regularizers.l2(1e-6),
    activity_regularizer=tf.keras.regularizers.l2(1e-6)))
model.add(Dropout(rate=0.25))
model.add(Dense(y.shape[1],activation='softmax'))

adam=tf.keras.optimizers.Adam(
    learning_rate=0.0001,
```

```
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=True)

model.compile(optimizer=adam,
              loss="categorical_crossentropy",
              metrics=[tf.keras.metrics.MeanAbsoluteError()])
history = model.fit(X_train,y_train,batch_size=8,epochs=100,validation_split=0.15,shuffle=True)
```

```
Epoch 1/100
65/65 [==============================] - 5s 79ms/step - loss: 10.2752 - mean_absolute_error: 0.1033 - val_loss: 10.0128 - val_
mean_absolute_error: 0.1045
Epoch 2/100
65/65 [==============================] - 5s 71ms/step - loss: 9.6347 - mean_absolute_error: 0.1027 - val_loss: 9.4567 - val_me
an_absolute_error: 0.1046
Epoch 3/100
65/65 [==============================] - 5s 74ms/step - loss: 9.0978 - mean_absolute_error: 0.1029 - val_loss: 8.9064 - val_me
an_absolute_error: 0.1044
Epoch 4/100
65/65 [==============================] - 5s 72ms/step - loss: 8.5980 - mean_absolute_error: 0.1031 - val_loss: 8.4444 - val_me
an_absolute_error: 0.1044
Epoch 5/100
65/65 [==============================] - 5s 72ms/step - loss: 8.1700 - mean_absolute_error: 0.1026 - val_loss: 8.0353 - val_me
an_absolute_error: 0.1044
Epoch 6/100
65/65 [==============================] - 5s 71ms/step - loss: 7.7678 - mean_absolute_error: 0.1030 - val_loss: 7.6395 - val_me
an_absolute_error: 0.1041
Epoch 7/100
```

In [210]:
```
plt.plot(history.history['mean_absolute_error'])
plt.plot(history.history['val_mean_absolute_error'])
plt.title('model MAE')
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 [220]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.layers import Input
model = VGG16(include_top=False)
new_input = Input(shape=(x.shape[1],x.shape[2],x.shape[3]))
model = VGG16(weights=None, input_tensor=new_input, classes=y.shape[1])
```

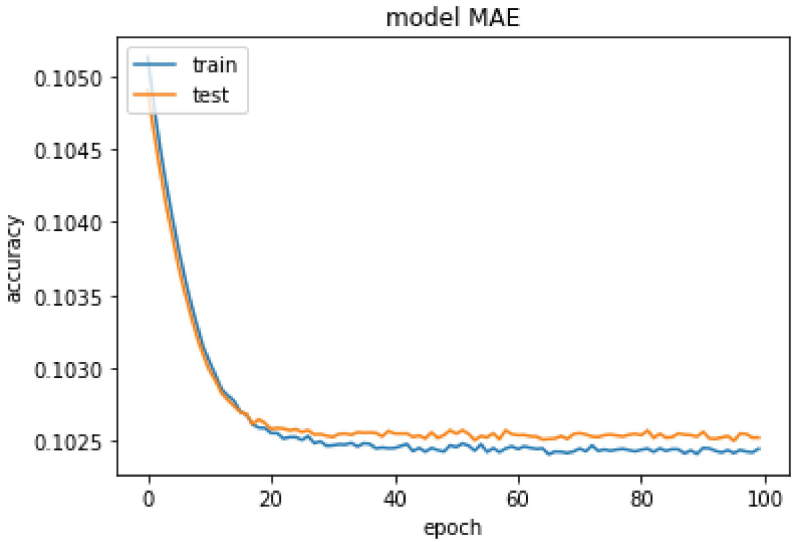
In [221]:
```
x = preprocess_input(x)
```

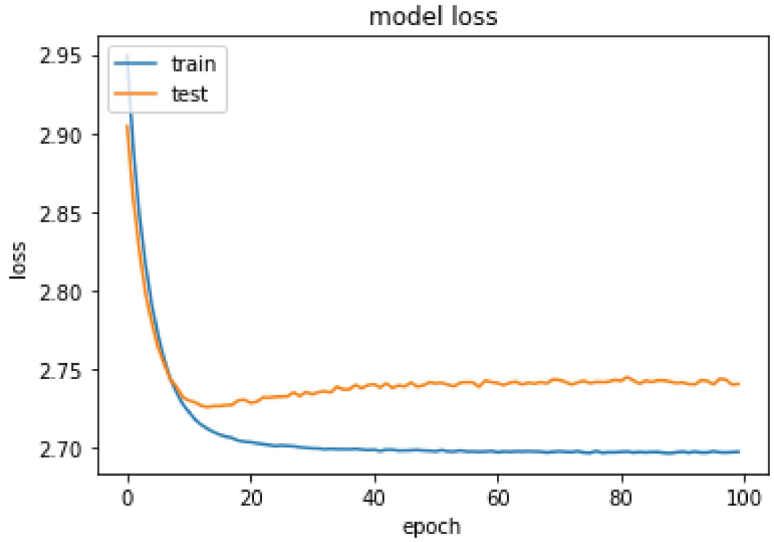
In [222]:
```
X_train, X_test, y_train, y_test = train_test_split(
     x, y, test_size=0.15, random_state=42)
```

In [223]:
```
adam=tf.keras.optimizers.Adam(
    learning_rate=0.0000001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=True)
model.compile(optimizer=adam,
              loss="categorical_crossentropy",
              metrics=[tf.keras.metrics.MeanAbsoluteError()])

history = model.fit(X_train,y_train,batch_size=8,epochs=100,validation_split=0.15,shuffle=True)
```

```
Epoch 1/100
65/65 [==============================] - 3s 41ms/step - loss: 2.9491 - mean_absolute_error: 0.1051 - val_loss: 2.9043 - val_me
an_absolute_error: 0.1049
Epoch 2/100
65/65 [==============================] - 3s 39ms/step - loss: 2.8900 - mean_absolute_error: 0.1048 - val_loss: 2.8581 - val_me
an_absolute_error: 0.1046
Epoch 3/100
65/65 [==============================] - 2s 38ms/step - loss: 2.8475 - mean_absolute_error: 0.1045 - val_loss: 2.8249 - val_me
an_absolute_error: 0.1044
Epoch 4/100
65/65 [==============================] - 2s 38ms/step - loss: 2.8162 - mean_absolute_error: 0.1043 - val_loss: 2.7977 - val_me
an_absolute_error: 0.1041
Epoch 5/100
65/65 [==============================] - 2s 38ms/step - loss: 2.7909 - mean_absolute_error: 0.1040 - val_loss: 2.7796 - val_me
an_absolute_error: 0.1039
Epoch 6/100
65/65 [==============================] - 3s 39ms/step - loss: 2.7722 - mean_absolute_error: 0.1038 - val_loss: 2.7638 - val_me
an_absolute_error: 0.1037
Epoch 7/100
```

In [224]:
```
plt.plot(history.history['mean_absolute_error'])
plt.plot(history.history['val_mean_absolute_error'])
plt.title('model MAE')
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 [231]:
```
preds=model.predict(X_test)
```

In [232]:
```
preds.shape,y_test.shape
```

Out[232]:
```
((109, 18), (109, 18))
```

In [236]:
```
len(np.where(np.argmax(preds,axis=1)==np.argmax(y_test,axis=1)))/y_test.shape[0]
```

Out[236]:
```
0.009174311926605505
```

In [237]:
```
preds=model.predict(X_train)
```

In [238]:
```
len(np.where(np.argmax(preds,axis=1)==np.argmax(y_train,axis=1)))/y_train.shape[0]
```

Out[238]:
```
0.0016339869281045752
```

In [239]:
```
x.shape
```

Out[239]:
```
(721, 120, 120, 3)
```

Conclusion

There are simply not enough training observations in this dataset to be able to predict Pokemon classes with, most deep learning set-ups require thousands, if not millions of observations to train on

In [ ]: