Final Project

Pokemon Images Classifier

MSCA 31009 IP01 Machine Learning & Predictive Analytics

Zoey Chen

Github: https://github.com/wchen119/ML_FinalProject

Table of Contents

Problem Statement

There are many types of pokemons, and as a pokemon fan for 25 years. I want to build a classifier with **convolutional neural network** to predict whether a pokemon is a **fire-type** or a **water-type** based on it’s image

Assumptions & Hypotheses for data

Dataset:
https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types?resource=download

- **Pokemon.csv**
  - Pokemons from generation 1 to 7 with their corresponding types (primary and secondary)
- **Pokemon images**
  - Pokemon images

Assumptions & Hypo:

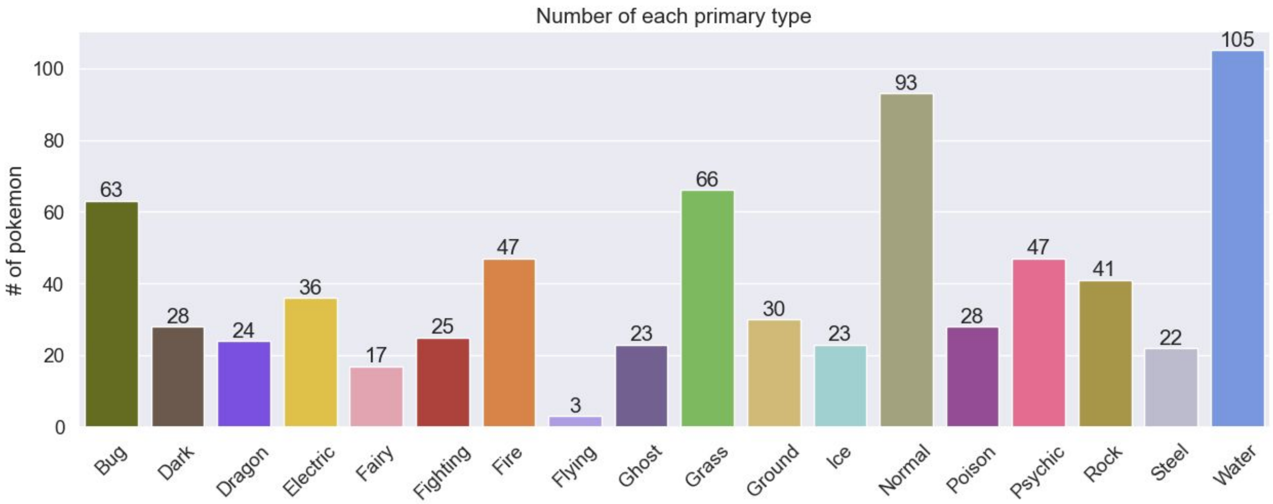
- I was originally debating if I should drop the pokemons that do not have type 2 values, as 50% of the pokemons have NA values in type 2 column. Since the full dataset is already small for training a convolutional neural network, I think decreasing the number of data points will decrease the accuracy of the classifier. So I just decided to take the Type1 value for my model.

	Name	Type1	Type2
0	bulbasaur	Grass	Poison
1	ivysaur	Grass	Poison
2	venusaur	Grass	Poison
3	charmander	Fire	NaN
4	charmeleon	Fire	NaN
...	...	...	...
804	stakataka	Rock	Steel
805	blacephalon	Fire	Ghost
806	zeraora	Electric	NaN
807	meltan	Steel	NaN
808	melmetal	Steel	NaN

Exploratory Data Analysis

To better show the visuals of the number of pokemons, I created a custom color palette based on the Type1 value for each pokemon & then plot them out

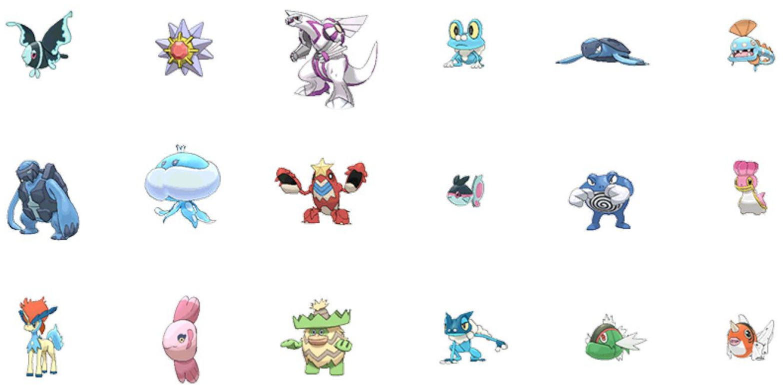
```
custom_colors = {
    'Bug': '#6d7815',
    'Dark': '#705848',
    'Dragon': '#7038f8',
    'Electric': '#f8d030',
    'Fairy': '#ee99ac',
    'Fighting': '#c03028',
    'Fire': '#f08030',
    'Flying': '#a890f0',
    'Ghost': '#705898',
    'Grass': '#78c850',
    'Ground': '#e0c068',
    'Ice': '#98d8d8',
    'Normal': '#a8a878',
    'Poison': '#a040a0',
    'Psychic': '#f85888',
    'Rock': '#b8a038',
    'Steel': '#b8b8d0',
    'Water': '#6890f0'
}
```



Exploratory Data Analysis -2

For some pokemons, it’s not that obvious to see the connection between their appearance and their primary type - but for water and fire it’s usually blue and red!

- **<u>water</u>**



- **<u>fire</u>**

Feature Engineering and Transformation

- I start by limiting the dataset to fire & water type pokemon first and then generate training and validation sets of scaled images (rescale the rgb values to fit between 0 - 1) to feed into the model.

- I'll use training set to train the model and the validation set to evaluate the performance of the model later on.

- The splitting will be in 80:20 ratio and there are 122 Pokémons in our training set and 30 Pokémons in our validation set.

```
# limit data to Fire and Water types
df = df.query("Type1 == 'Fire' | Type1 == 'Water'")

print("Number of water-types:", len(df[df['Type1'] == 'Water']))
print("Number of fire-types:", len(df[df['Type1'] == 'Fire']))
```

```
Number of water-types: 105
Number of fire-types: 47
```

```
# shuffle the data
df = df.sample(frac=1).reset_index(drop=True)

train_gen = keras.preprocessing.image.ImageDataGenerator(
    validation_split=0.2,   # split the dataset into a training set and a validation set in an 80:20 ratio
    rescale=1./255          # rescale the rgb values to fit between 0 and 1
)


train_set = train_gen.flow_from_dataframe(
    df,
    x_col='Filepath',
    y_col='Type1',
    target_size=(120, 120),
    color_mode='rgba',
    class_mode='sparse',
    batch_size=32,
    seed=1,
    subset='training'
)

val_set = train_gen.flow_from_dataframe(
    df,
    x_col='Filepath',
    y_col='Type1',
    target_size=(120, 120),
    color_mode='rgba',
    class_mode='sparse',
    batch_size=32,
    seed=1,
    subset='validation'
)
```

```
Found 122 validated image filenames belonging to 2 classes.
Found 30 validated image filenames belonging to 2 classes.
```

Prevent the class imbalance problem

- **After limiting the data to water and fire types, there are more water than fire pokemons.**

- **To prevent the class imbalance problem - I decided to use AUC (Area under the ROC Curve) as a metric in addition to accuracy and loss.**

Model training

```python
img_input = layers.Input(shape=(120, 120, 4))

x = layers.Conv2D(filters=64, kernel_size=(8, 8), activation='relu')(img_input)
x = layers.MaxPool2D()(x)

x = layers.Conv2D(filters=128, kernel_size=(8, 8), activation='relu')(x)
x = layers.MaxPool2D()(x)

x = layers.Conv2D(filters=256, kernel_size=(8, 8), activation='relu')(x)
x = layers.MaxPool2D()(x)

x = layers.Flatten()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)

output = layers.Dense(units=1, activation='sigmoid')(x)

model = keras.Model(inputs=img_input, outputs=output)

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['acc', keras.metrics.AUC()]
)

# print model layers
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 120, 120, 4)]     0

 conv2d_3 (Conv2D)           (None, 113, 113, 64)      16448

 max_pooling2d_3 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 49, 49, 128)       524416

 max_pooling2d_4 (MaxPooling  (None, 24, 24, 128)      0
 2D)
```

```python
history = model.fit(
    train_set,
    validation_data=val_set,
    batch_size=32,
    epochs=100,
    callbacks=[
        keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        ),
        keras.callbacks.ReduceLROnPlateau()
    ]
)
```

```
Epoch 1/100
2023-05-22 15:27:35.862471: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
	 [[{{node Placeholder/_0}}]]
4/4 [==============================] - ETA: 0s - loss: 7.3664 - acc: 0.5164 - auc_1: 0.4285
2023-05-22 15:27:42.906261: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
	 [[{{node Placeholder/_0}}]]
4/4 [==============================] - 8s 2s/step - loss: 7.3664 - acc: 0.5164 - auc_1: 0.4285 - val_loss: 0.6879 - val_acc: 0.7333 - val_auc_1: 0.1676 - lr: 0.0010
Epoch 2/100
4/4 [==============================] - 7s 2s/step - loss: 0.6791 - acc: 0.6803 - auc_1: 0.3998 - val_loss: 0.5875 - val_acc: 0.7333 - val_auc_1: 0.5739 - lr: 0.0010
Epoch 3/100
4/4 [==============================] - 7s 2s/step - loss: 0.6234 - acc: 0.6803 - auc_1: 0.5976 - val_loss: 0.5929 - val_acc: 0.7333 - val_auc_1: 0.7614 - lr: 0.0010
Epoch 4/100
4/4 [==============================] - 7s 2s/step - loss: 0.6167 - acc: 0.6803 - auc_1: 0.6328 - val_loss: 0.5523 - val_acc: 0.7333 - val_auc_1: 0.7955 - lr: 0.0010
Epoch 5/100
4/4 [==============================] - 7s 2s/step - loss: 0.6006 - acc: 0.6803 - auc_1: 0.7172 - val_loss: 0.5211 - val_acc: 0.7333 - val_auc_1: 0.8608 - lr: 0.0010
Epoch 6/100
4/4 [==============================] - 8s 2s/step - loss: 0.5566 - acc: 0.6803 - auc_1: 0.8358 - val_loss: 0.4657 - val_acc: 0.7333 - val_auc_1: 0.9261 - lr: 0.0010
Epoch 7/100
```

Proposed Approaches with checks for over/underfitting

When developing machine learning models, it is essential to address the problems of overfitting and underfitting. I proposed 3 approaches here:

- **Data Splitting: Approach: Divide the dataset into three subsets: training set, validation set, and test set.**
  a. Checks: Ensure an adequate amount of data is allocated to each set. Verify that the training and validation sets have a similar distribution. Assess if the test set represents unseen data.

- **Regularization: Approach: Apply regularization techniques to prevent overfitting, such as L1 or L2 regularization, dropout, or early stopping.**
  a. Checks: Monitor the training and validation loss curves to observe if regularization is effectively preventing overfitting. Assess the impact of different regularization hyperparameters on model performance.

- **Cross-Validation: Approach: Implement k-fold cross-validation to obtain more reliable performance estimates.**
  a. Checks: Verify that the cross-validated performance metrics are consistent across different folds. Evaluate if the model's performance is stable across different iterations of cross-validation.

Proposed Solution with regularization, if needed

- I included a dropout layer in my code, which is a regularization technique used to reduce overfitting in neural networks.
- The dropout layer randomly sets a fraction of the input units to 0 at each update during training time. In this case, the dropout rate is set to 0.5, meaning that during training, half of the units in the dense layer will be randomly dropped or deactivated.
- This introduces noise and prevents the network from relying too heavily on any particular set of input features.
- Dropout acts as a form of regularization by effectively creating an ensemble of smaller subnetworks that work together to make predictions, providing better generalization to unseen data.

Model training

```python
img_input = layers.Input(shape=(120, 120, 4))

x = layers.Conv2D(filters=64, kernel_size=(8, 8), activation='relu')(img_input)
x = layers.MaxPool2D()(x)

x = layers.Conv2D(filters=128, kernel_size=(8, 8), activation='relu')(x)
x = layers.MaxPool2D()(x)

x = layers.Conv2D(filters=256, kernel_size=(8, 8), activation='relu')(x)
x = layers.MaxPool2D()(x)

x = layers.Flatten()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)

output = layers.Dense(units=1, activation='sigmoid')(x)

model = keras.Model(inputs=img_input, outputs=output)

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['acc', keras.metrics.AUC()]
)

# print model layers
model.summary()
```

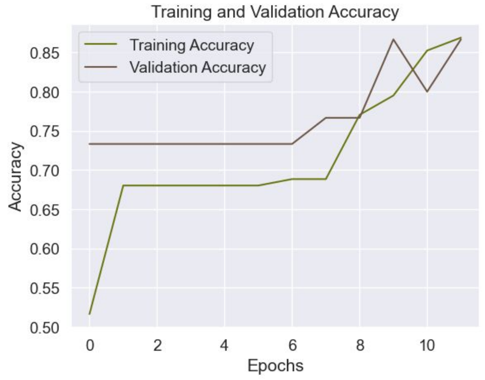
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 120, 120, 4)]     0

 conv2d (Conv2D)             (None, 113, 113, 64)      16448

 max_pooling2d (MaxPooling2D  (None, 56, 56, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 49, 49, 128)       524416

 max_pooling2d_1 (MaxPooling  (None, 24, 24, 128)      0
 2D)

 conv2d_2 (Conv2D)           (None, 17, 17, 256)       2097408

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 256)        0
 2D)
```

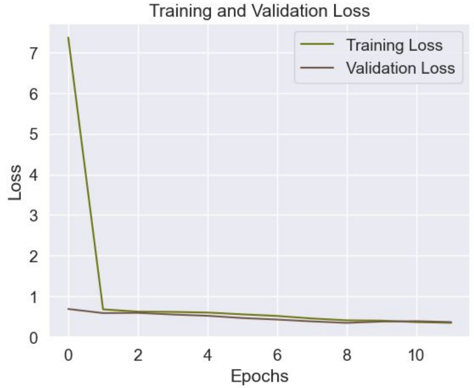
Result (Accuracy/ Loss)

- To check if there is overfitting in the model, I analyze the training and validation performance:

- From the accuracy plot, both training and validation accuracy jump at around 7 epochs.

- As for the loss plot, the training loss drops early on at around epoch 1, while validation loss starts off quite low.





Results

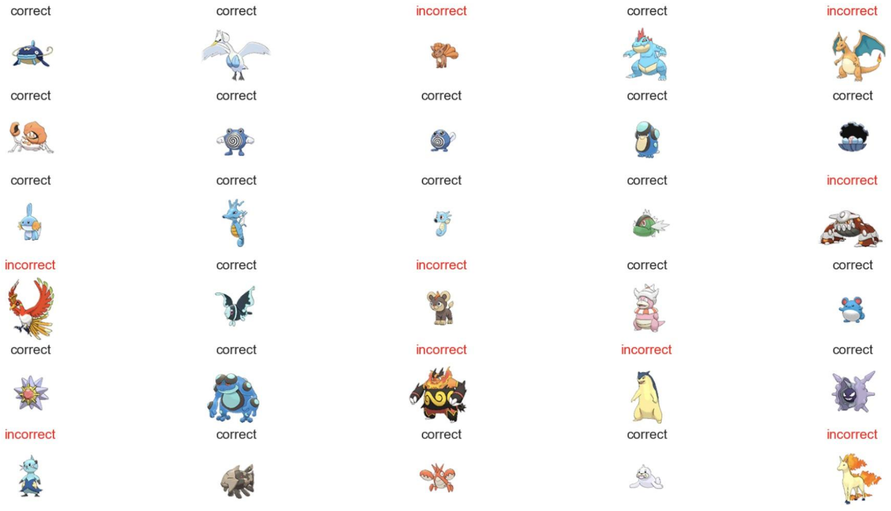
After running the model, there are 9 pokemons that are misclassified, and I output them as images for better visualizations.

I think the result turns out pretty good since most of them are still classified correctly.

There are definitely more improvements I can make in the future where I will elaborate more in the next slide.

```
Model predictions: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1]
Actual labels:     [1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0]
# of misclassified pokemon: 9
```

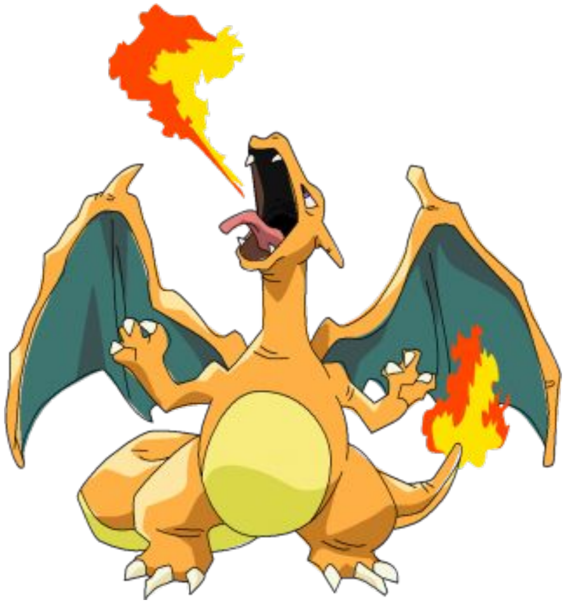
```
# obtain the images from the filepath at the determined indices
misclassified_imgs = []
for filepath in misclassified_filepaths:
  misclassified_imgs.append(mpimg.imread(filepath))

# plot results
f, axarr = plt.subplots(6,5, figsize=(20,10))
count = 0
for r in range(6):
  for c in range(5):
    axarr[r,c].imshow(misclassified_imgs[count])
    if correctness[count] == 'correct':
      axarr[r,c].set_title(correctness[count])
    else:
      axarr[r,c].set_title(correctness[count], color='red')
    axarr[r,c].set_axis_off()
    count += 1
plt.show()
```



Future Work

After successfully trained a classifier that can predict whether a pokemon is a fire or water type, I think there are definitely more things I can do in the future such as :

- **Try different combinations of pokemon to classify**
- **Expand the problem to classify all types of pokemons**
- **Add data augmentation**



Thank you