

```
In [1]: #!pip install keras==2.1.2
```

```
In [2]: import tensorflow as tf  
tf.test.gpu_device_name()
```

```
Out[2]: ''
```

Data preparation

```
In [3]: from glob import glob  
from sklearn.model_selection import train_test_split  
  
Playable = glob('train/zeldaPlayablelevels/*.jpg')  
Unplayable = glob('train/zeldaUnplayablelevels/*.jpg')  
  
Playable_train, Playable_test = train_test_split(Playable, test_size=0.30)  
Unplayable_train, Unplayable_test = train_test_split(Unplayable, test_size=0.30)  
  
TRAIN_DIR = 'train'  
TEST_DIR = 'test'
```

Plot some random images from the dataset.

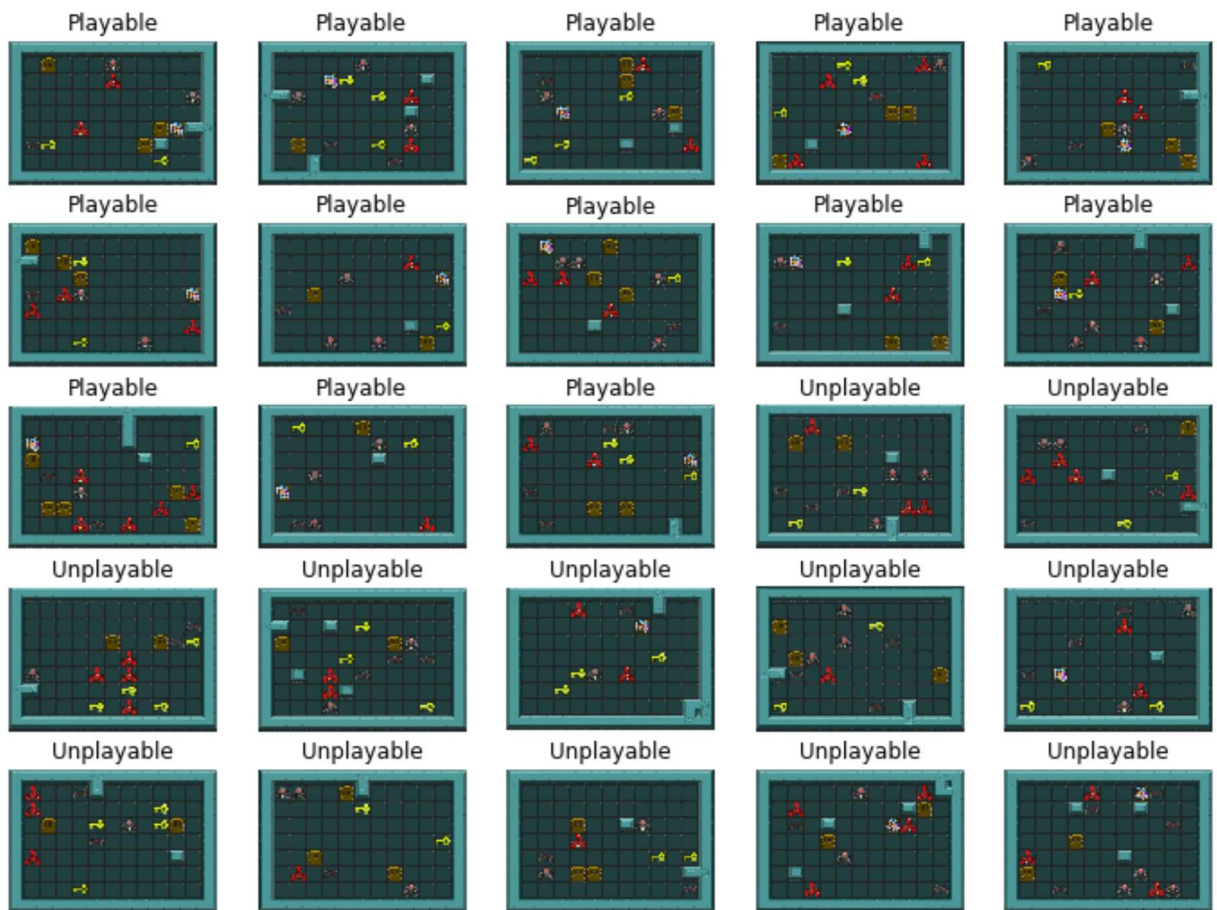
```

In [4]: import numpy as np
        from PIL import Image
        import matplotlib.pyplot as plt

        Playable = np.random.choice(Playable_train, 13)
        Unplayable = np.random.choice(Unplayable_train, 12)
        data = np.concatenate((Playable, Unplayable))
        labels = 13 * ['Playable'] + 12 * ['Unplayable']

        N, R, C = 25, 5, 5
        plt.figure(figsize=(12, 9))
        for k, (src, label) in enumerate(zip(data, labels)):
            im = Image.open(src).convert('RGB')
            plt.subplot(R, C, k+1)
            plt.title(label)
            plt.imshow(np.asarray(im))
            plt.axis('off')

```



Model customization

```

In [5]: from keras.models import Model
        from keras.layers import Dense, GlobalAveragePooling2D, Dropout
        #from keras.applications.inception_v3 import InceptionV3, preprocess_input
        from keras.applications.resnet50 import ResNet50, preprocess_input
        from keras.regularizers import l2, l1

CLASSES = 2

HEIGHT = 300
WIDTH = 300

base_model = ResNet50(weights='imagenet',
                      include_top=False,
                      input_shape=(HEIGHT, WIDTH, 3))

x = base_model.output
x = GlobalAveragePooling2D(name='avg_pool')(x)
x = Dropout(0.4)(x)
x = Dense(2, activation='relu', name = 'Dense_1',
          kernel_regularizer=l2(0.01),
          activity_regularizer=l1(0.01))(x)

predictions = Dense(CLASSES, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# transfer learning
for layer in base_model.layers:
    layer.trainable = False

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

```

Using TensorFlow backend.

/Users/friends/anaconda3/envs/udacity-ehr-env/lib/python3.7/site-packages/keras_applications/resnet50.py:265: UserWarning: The output shape of `ResNet50(include_top=False)` has been changed since Keras 2.2.0.

warnings.warn('The output shape of `ResNet50(include_top=False)` ')

Data augmentation

```
In [6]: from keras.preprocessing.image import ImageDataGenerator

WIDTH = 300
HEIGHT = 300
BATCH_SIZE = 32

# data prep
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=(HEIGHT, WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical')

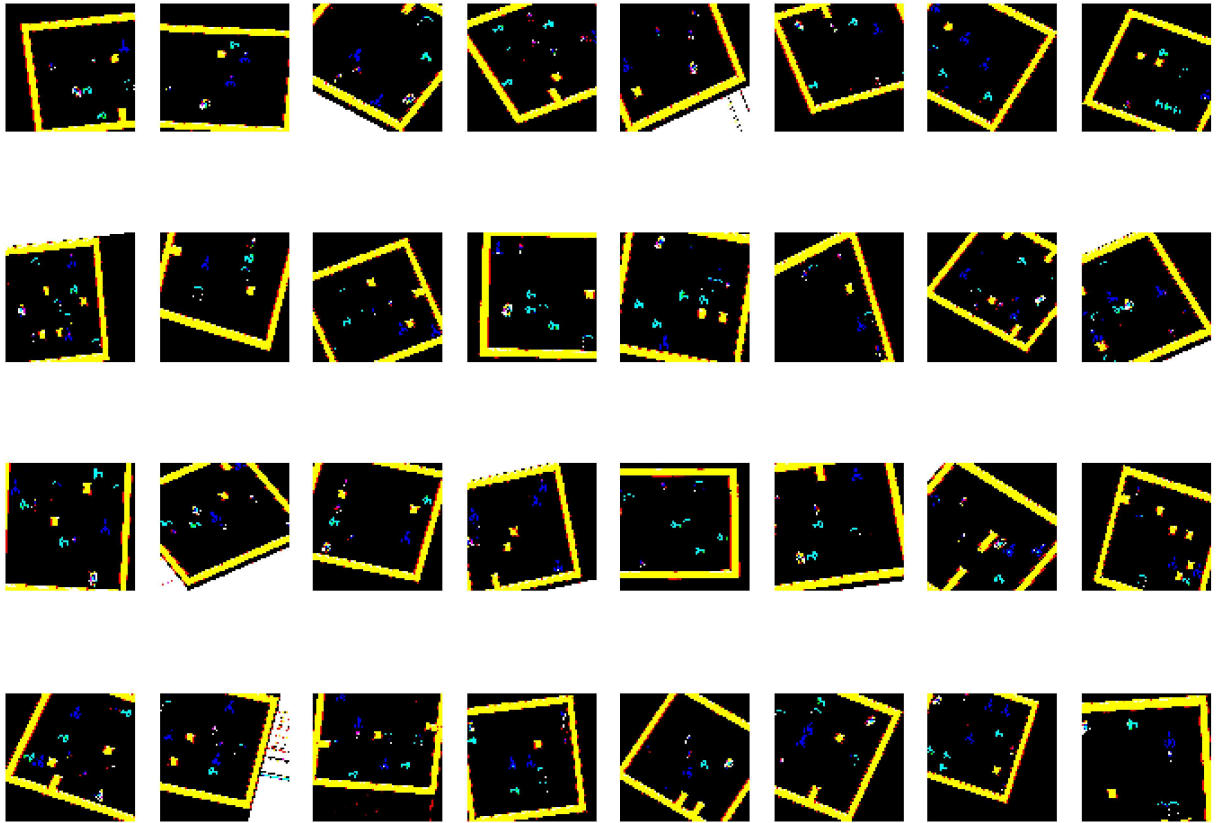
validation_generator = validation_datagen.flow_from_directory(
    TEST_DIR,
    target_size=(HEIGHT, WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical')
```

Found 2018 images belonging to 2 classes.

Found 30 images belonging to 2 classes.

Plot some images result of data augmentation.

ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Transfer learning

```

In [8]: EPOCHS = 10
        BATCH_SIZE = 32
        STEPS_PER_EPOCH = 20
        #STEPS_PER_EPOCH = 320
        VALIDATION_STEPS = 5

        MODEL_FILE = 'resnet50withregularization.model'

        history = model.fit_generator(
                                train_generator,
                                epochs=EPOCHS,
                                steps_per_epoch=STEPS_PER_EPOCH,
                                validation_data=validation_generator,
                                validation_steps=VALIDATION_STEPS)

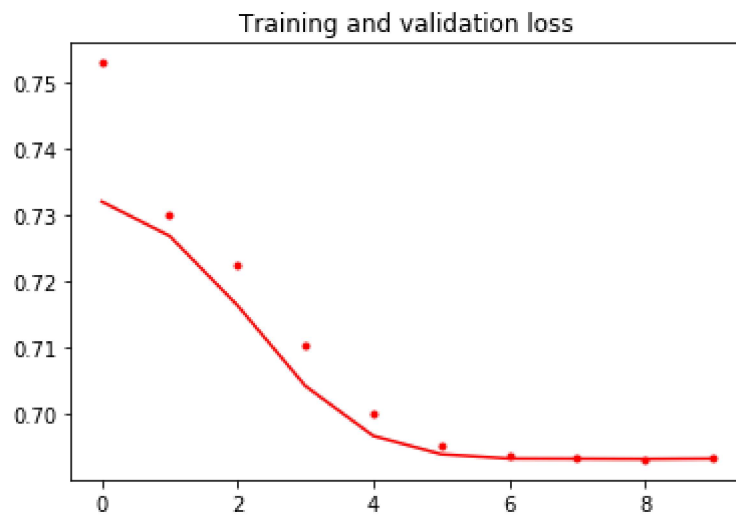
        model.save(MODEL_FILE)

Epoch 1/10
20/20 [=====] - 1048s 52s/step - loss: 0.7523 - accuracy: 0.4918 - val_loss: 0.7321 - val_accuracy: 0.5000
Epoch 2/10
20/20 [=====] - 1053s 53s/step - loss: 0.7299 - accuracy: 0.5031 - val_loss: 0.7269 - val_accuracy: 0.5000
Epoch 3/10
20/20 [=====] - 1046s 52s/step - loss: 0.7225 - accuracy: 0.4797 - val_loss: 0.7164 - val_accuracy: 0.5000
Epoch 4/10
20/20 [=====] - 1011s 51s/step - loss: 0.7104 - accuracy: 0.4951 - val_loss: 0.7042 - val_accuracy: 0.5000
Epoch 5/10
20/20 [=====] - 1052s 53s/step - loss: 0.7002 - accuracy: 0.5219 - val_loss: 0.6967 - val_accuracy: 0.5000
Epoch 6/10
20/20 [=====] - 1067s 53s/step - loss: 0.6952 - accuracy: 0.4875 - val_loss: 0.6939 - val_accuracy: 0.5000
Epoch 7/10
20/20 [=====] - 1075s 54s/step - loss: 0.6936 - accuracy: 0.4563 - val_loss: 0.6933 - val_accuracy: 0.5000
Epoch 8/10
20/20 [=====] - 1039s 52s/step - loss: 0.6933 - accuracy: 0.5063 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 9/10
20/20 [=====] - 1089s 54s/step - loss: 0.6931 - accuracy: 0.5164 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 10/10
20/20 [=====] - 1087s 54s/step - loss: 0.6933 - accuracy: 0.4934 - val_loss: 0.6932 - val_accuracy: 0.5000

```

```
In [9]: def plot_training(history):  
    acc = history.history['accuracy']  
    val_acc = history.history['val_accuracy']  
    loss = history.history['loss']  
    val_loss = history.history['val_loss']  
    epochs = range(len(acc))  
  
    plt.plot(epochs, acc, 'r.')  
    plt.plot(epochs, val_acc, 'r')  
    plt.title('Training and validation accuracy')  
  
    plt.figure()  
    plt.plot(epochs, loss, 'r.')  
    plt.plot(epochs, val_loss, 'r-')  
    plt.title('Training and validation loss')  
    plt.show()  
  
plot_training(history)
```





Prediction of the custom model

```
In [10]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from keras.preprocessing import image
from keras.models import load_model

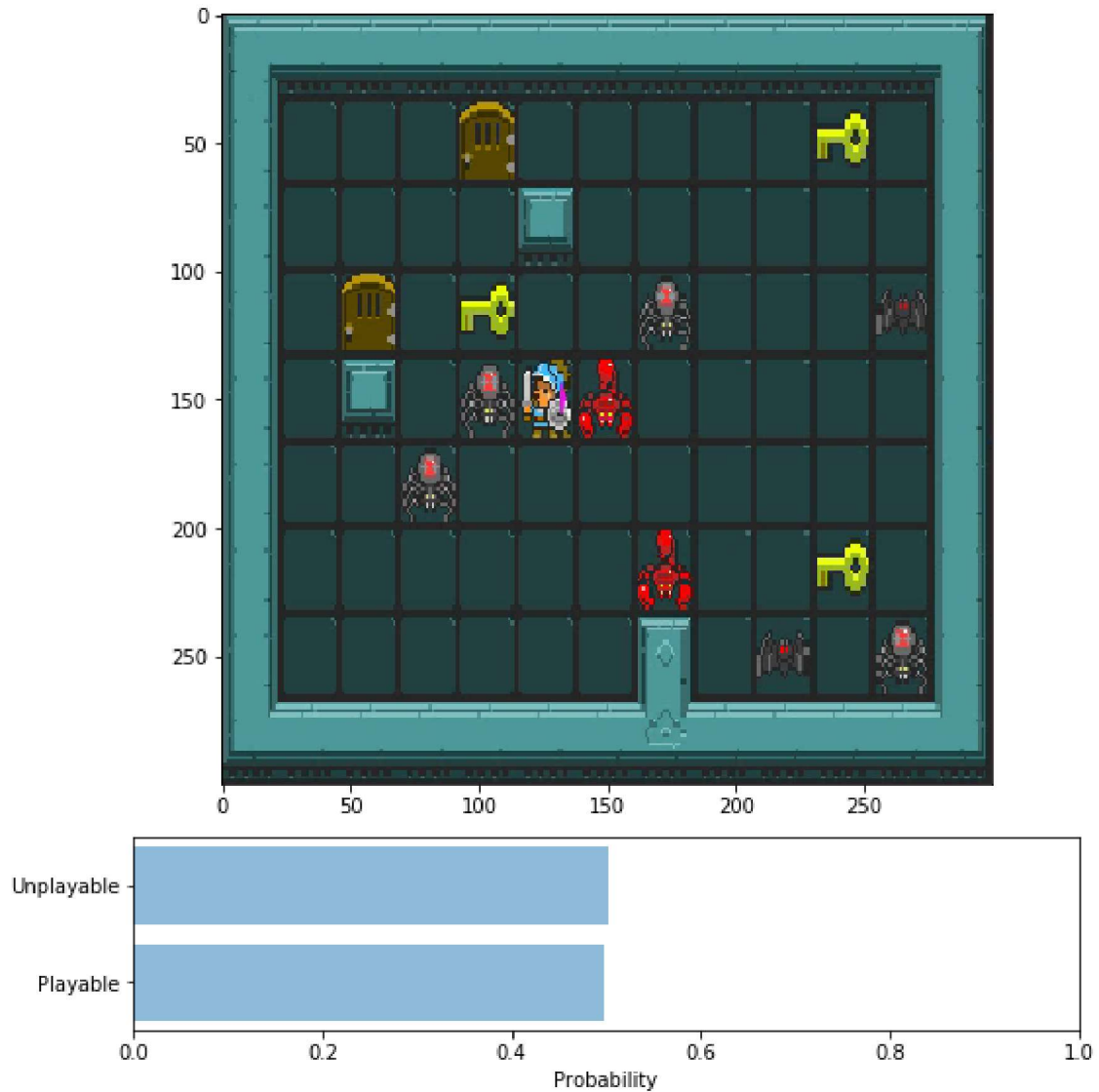
def predict(model, img):
    """Run model prediction on image
    Args:
        model: keras model
        img: PIL format image
    Returns:
        list of predicted labels and their probabilities
    """
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    preds = model.predict(x)
    return preds[0]

def plot_preds(img, preds):
    """Displays image and the top-n predicted probabilities in a bar graph
    Args:
        preds: list of predicted labels and their probabilities
    """
    labels = ("Playable", "Unplayable")
    gs = gridspec.GridSpec(2, 1, height_ratios=[4, 1])
    plt.figure(figsize=(8,8))
    plt.subplot(gs[0])
    plt.imshow(np.asarray(img))
    plt.subplot(gs[1])
    plt.barh([0, 1], preds, alpha=0.5)
    plt.yticks([0, 1], labels)
    plt.xlabel('Probability')
    plt.xlim(0, 1)
    plt.tight_layout()
```

```
In [11]: model = load_model(MODEL_FILE)
```

```
In [12]: img = image.load_img('test/zeldaPlayablelevels/11.jpg', target_size=(HEIGHT, WIDTH))  
         preds = predict(model, img)  
  
         plot_preds(np.asarray(img), preds)  
         preds
```

Out[12]: array([0.49712119, 0.5028788], dtype=float32)



In []: