ResNet50

Residual learning framework ResNet is a powerful backbone model that is used very frequently in many computer vision tasks. On the ImageNet dataset residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity.

AlexNet, the winner of ImageNet 2012 and the model that apparently kick started the focus on deep learning had only 8 convolutional layers, the VGG network had 19 and Inception or GoogleNet had 22 layers and ResNet 152 had 152 layers. ResNet-50 that is a smaller version of ResNet 152 and frequently used for transfer learning. ResNet uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem

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

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

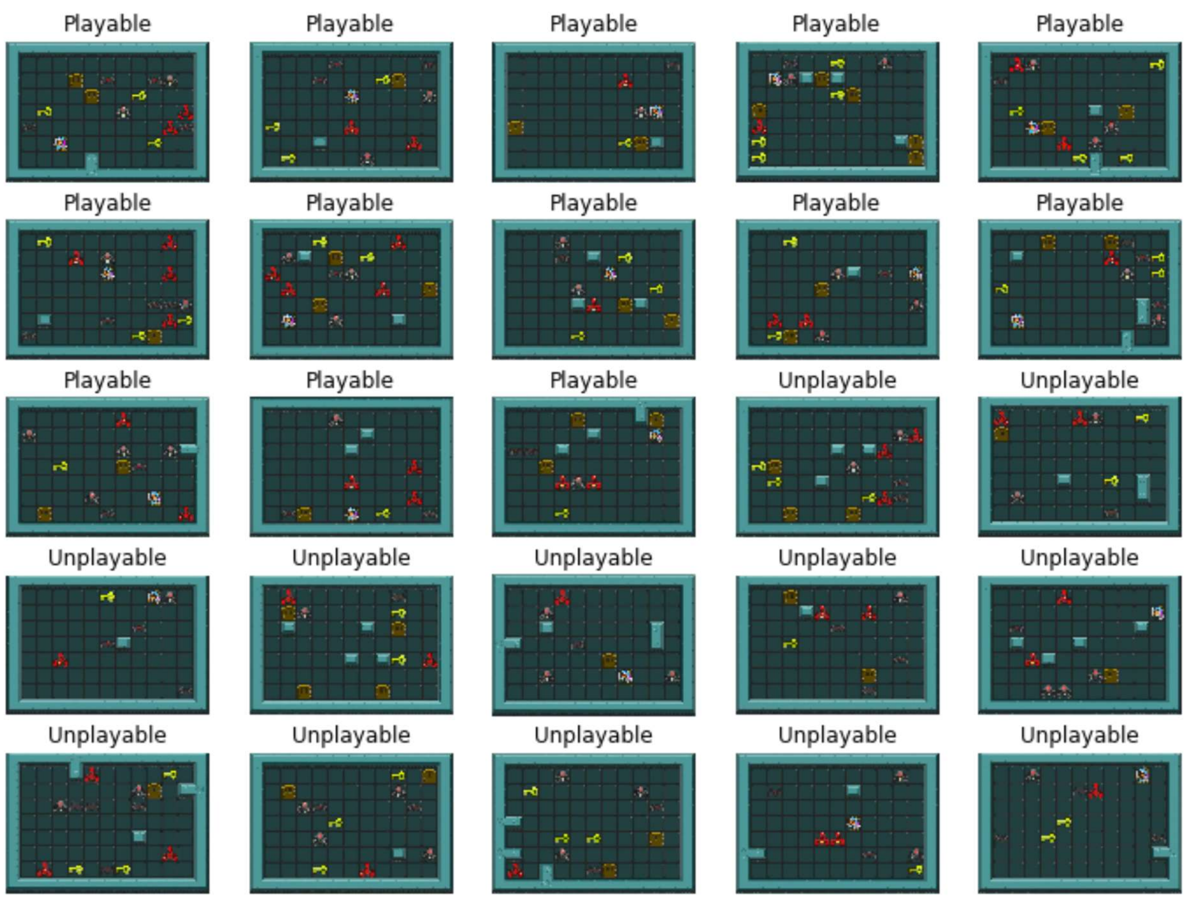
Out[2]:
```
''
```

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=0.3
0)

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


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)
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/ker
as_applications/resnet50.py:265: UserWarning: The output shape of `ResNet50(i
nclude_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')
```

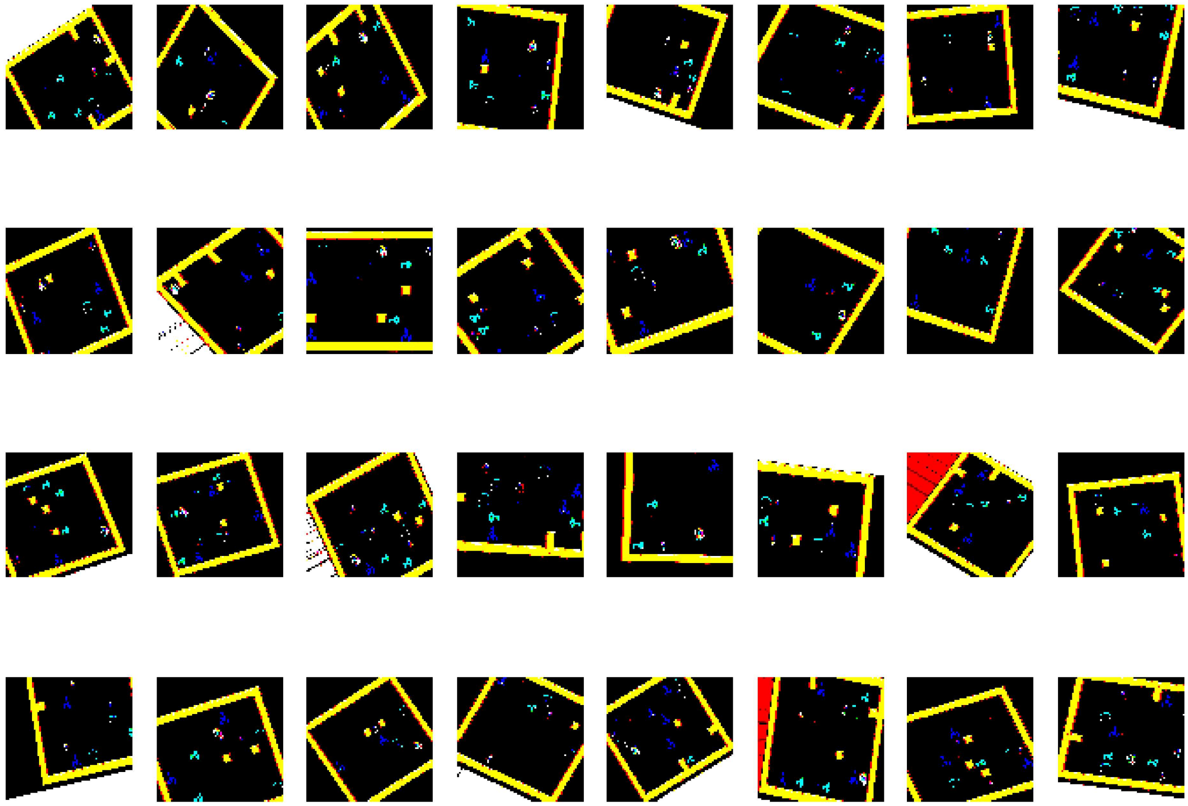
```
Found 2018 images belonging to 2 classes.
Found 30 images belonging to 2 classes.
```

Plot some images result of data augmentation.

In [7]:
```
x_batch, y_batch = next(train_generator)

plt.figure(figsize=(12, 9))
for k, (img, lbl) in enumerate(zip(x_batch, y_batch)):
    plt.subplot(4, 8, k+1)
    plt.imshow((img + 1) / 2)
    plt.axis('off')
```

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

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

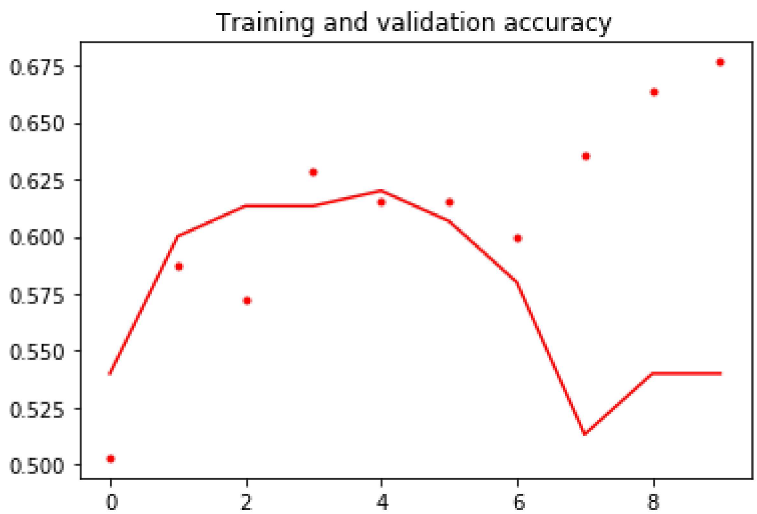


Transfer learning

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

MODEL_FILE = 'resnet50.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 [==============================] - 1103s 55s/step - loss: 0.9261 - accu
racy: 0.5031 - val_loss: 0.6825 - val_accuracy: 0.5400
Epoch 2/10
20/20 [==============================] - 963s 48s/step - loss: 0.7259 - accur
acy: 0.5875 - val_loss: 0.6715 - val_accuracy: 0.6000
Epoch 3/10
20/20 [==============================] - 921s 46s/step - loss: 0.7226 - accur
acy: 0.5721 - val_loss: 0.6362 - val_accuracy: 0.6133
Epoch 4/10
20/20 [==============================] - 961s 48s/step - loss: 0.6926 - accur
acy: 0.6281 - val_loss: 0.5873 - val_accuracy: 0.6133
Epoch 5/10
20/20 [==============================] - 959s 48s/step - loss: 0.7170 - accur
acy: 0.6156 - val_loss: 0.7146 - val_accuracy: 0.6200
Epoch 6/10
20/20 [==============================] - 958s 48s/step - loss: 0.6797 - accur
acy: 0.6156 - val_loss: 0.7219 - val_accuracy: 0.6067
Epoch 7/10
20/20 [==============================] - 925s 46s/step - loss: 0.7463 - accur
acy: 0.6000 - val_loss: 0.6586 - val_accuracy: 0.5800
Epoch 8/10
20/20 [==============================] - 960s 48s/step - loss: 0.6427 - accur
acy: 0.6359 - val_loss: 0.8472 - val_accuracy: 0.5133
Epoch 9/10
20/20 [==============================] - 924s 46s/step - loss: 0.6264 - accur
acy: 0.6639 - val_loss: 0.8034 - val_accuracy: 0.5400
Epoch 10/10
20/20 [==============================] - 958s 48s/step - loss: 0.6394 - accur
acy: 0.6766 - val_loss: 0.7292 - val_accuracy: 0.5400
```

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)
```



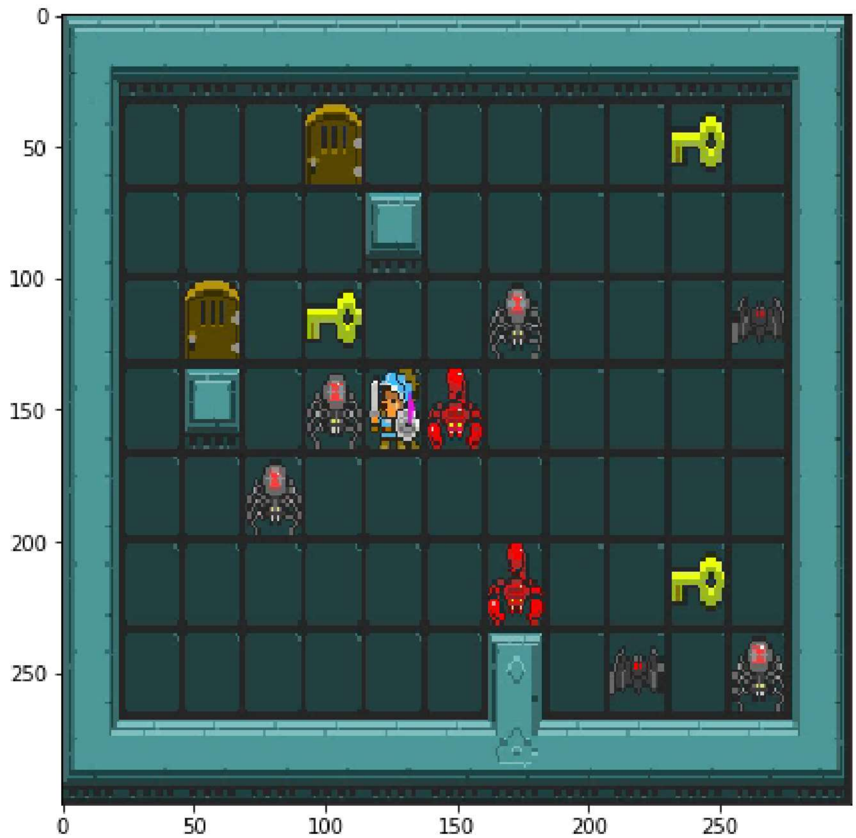
Prediction of the custom model

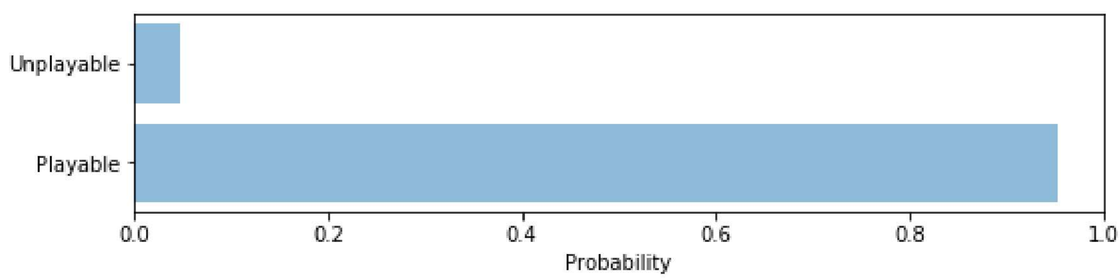
In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

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/l1.jpg', target_size=(HEIGHT, W
IDTH))
preds = predict(model, img)

plot_preds(np.asarray(img), preds)
preds
```

Out[12]:
```
array([0.9537766 , 0.04622348], dtype=float32)
```





In [13]:
```
TEST_DIR = 'test'
img_width = 300
img_height = 300
```

Predict classes

In [14]:
```
import numpy
test_generator = ImageDataGenerator()
test_data_generator = test_generator.flow_from_directory(
    TEST_DIR, # Put your path here
    target_size=(img_width, img_height),
    batch_size=32,
    shuffle=False)
test_steps_per_epoch = numpy.math.ceil(test_data_generator.samples / test_data
_generator.batch_size)

predictions = model.predict_generator(test_data_generator, steps=test_steps_pe
r_epoch)
# Get most likely class
predicted_classes = numpy.argmax(predictions, axis=1)
```

```
Found 30 images belonging to 2 classes.
```

Get ground-truth classes and class-labels

In [15]:
```
true_classes = test_data_generator.classes
class_labels = list(test_data_generator.class_indices.keys())
```

Use scikit-learn to get statistics

In [16]:
```
import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target
_names=class_labels)
print(report)
```

```
                      precision    recall  f1-score   support

  zeldaPlayablelevels       0.52      1.00      0.68        15
zeldaUnplayablelevels       1.00      0.07      0.12        15

             accuracy                           0.53        30
            macro avg       0.76      0.53      0.40        30
         weighted avg       0.76      0.53      0.40        30
```