

# VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity.

```
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)

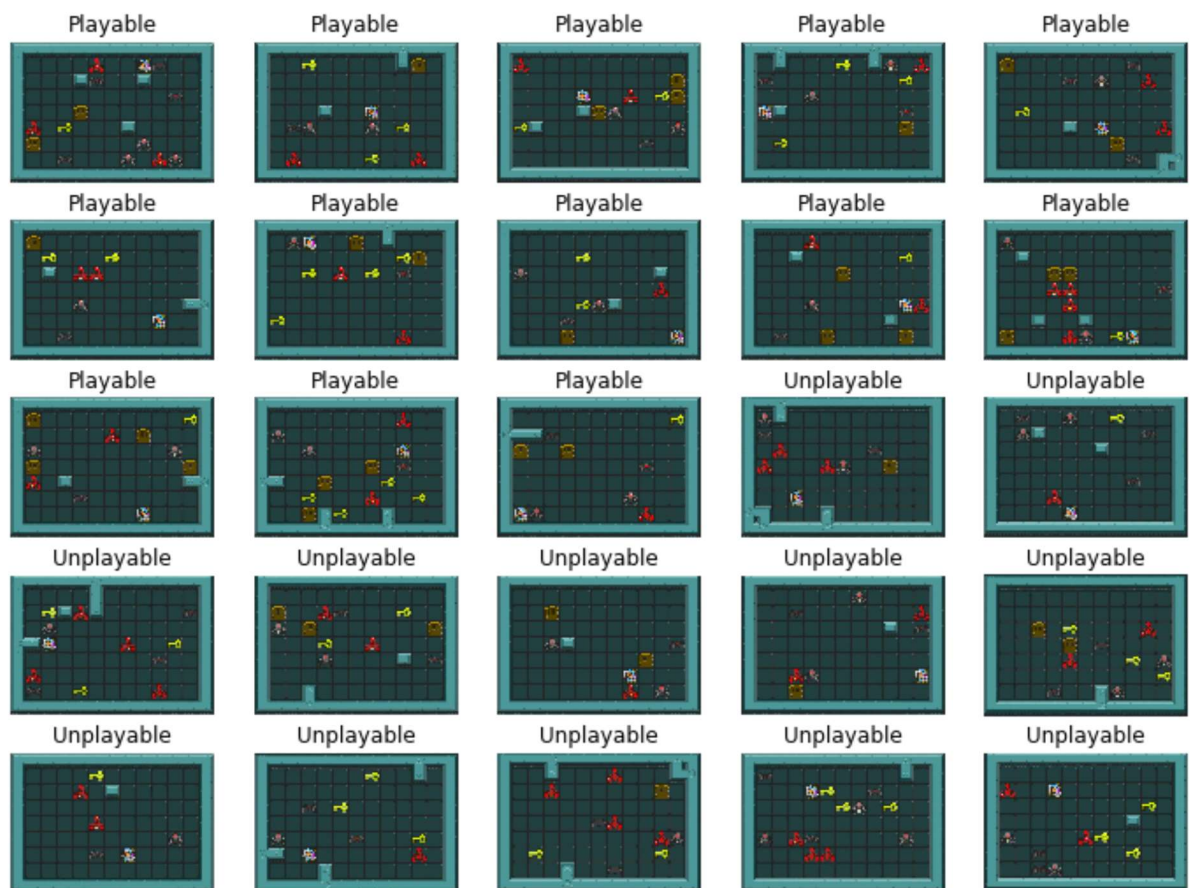
TRAIN_DIR = 'train'
TEST_DIR = 'test'
```

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 import VGG16

        CLASSES = 2
        img_width, img_height = 224, 224 # Default input size for VGG16
        # setup model
        base_model = VGG16(weights='imagenet',
                               include_top=False,
                               input_shape=(img_width, img_height, 3)) # 3 = number of channels in RGB pictures

        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.

## Data augmentation

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

```
WIDTH = 224
HEIGHT = 224
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')
```

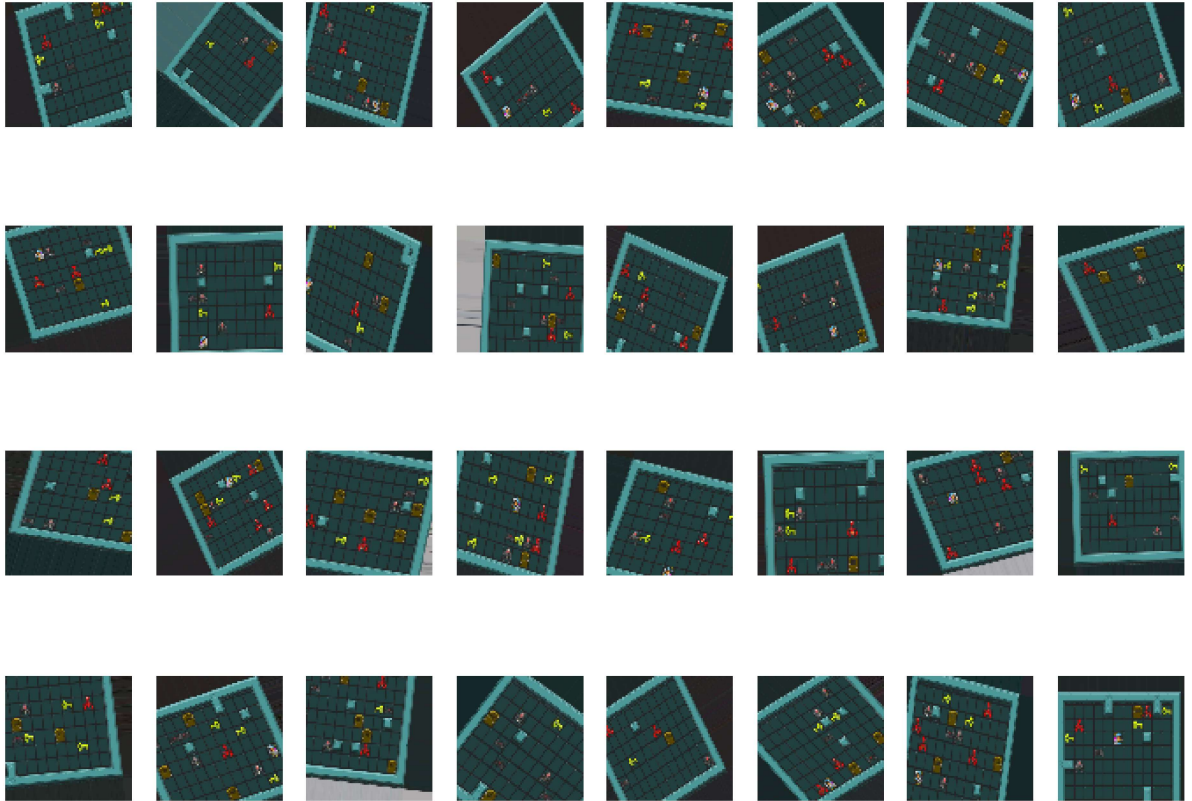
Found 2018 images belonging to 2 classes.

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')
```



## Transfer learning

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

        MODEL_FILE = 'vgg16.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)
```

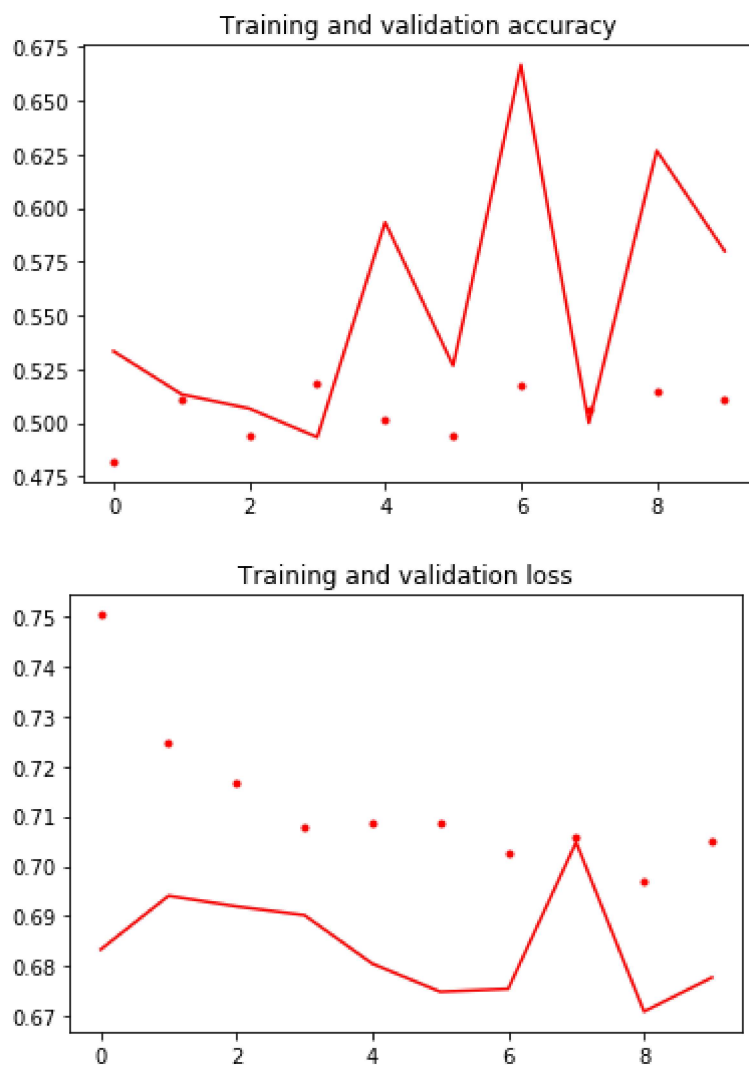
Epoch 1/10  
20/20 [=====] - 363s 18s/step - loss: 0.7435 - accuracy: 0.4820 - val\_loss: 0.6833 - val\_accuracy: 0.5333  
Epoch 2/10  
20/20 [=====] - 372s 19s/step - loss: 0.7248 - accuracy: 0.5109 - val\_loss: 0.6941 - val\_accuracy: 0.5133  
Epoch 3/10  
20/20 [=====] - 390s 20s/step - loss: 0.7168 - accuracy: 0.4938 - val\_loss: 0.6920 - val\_accuracy: 0.5067  
Epoch 4/10  
20/20 [=====] - 401s 20s/step - loss: 0.7080 - accuracy: 0.5188 - val\_loss: 0.6902 - val\_accuracy: 0.4933  
Epoch 5/10  
20/20 [=====] - 361s 18s/step - loss: 0.7019 - accuracy: 0.5016 - val\_loss: 0.6805 - val\_accuracy: 0.5933  
Epoch 6/10  
20/20 [=====] - 384s 19s/step - loss: 0.7088 - accuracy: 0.4938 - val\_loss: 0.6749 - val\_accuracy: 0.5267  
Epoch 7/10  
20/20 [=====] - 375s 19s/step - loss: 0.7025 - accuracy: 0.5172 - val\_loss: 0.6755 - val\_accuracy: 0.6667  
Epoch 8/10  
20/20 [=====] - 377s 19s/step - loss: 0.7057 - accuracy: 0.5063 - val\_loss: 0.7048 - val\_accuracy: 0.5000  
Epoch 9/10  
20/20 [=====] - 369s 18s/step - loss: 0.6859 - accuracy: 0.5148 - val\_loss: 0.6709 - val\_accuracy: 0.6267  
Epoch 10/10  
20/20 [=====] - 391s 20s/step - loss: 0.7049 - accuracy: 0.5109 - val\_loss: 0.6777 - val\_accuracy: 0.5800

```
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()
```

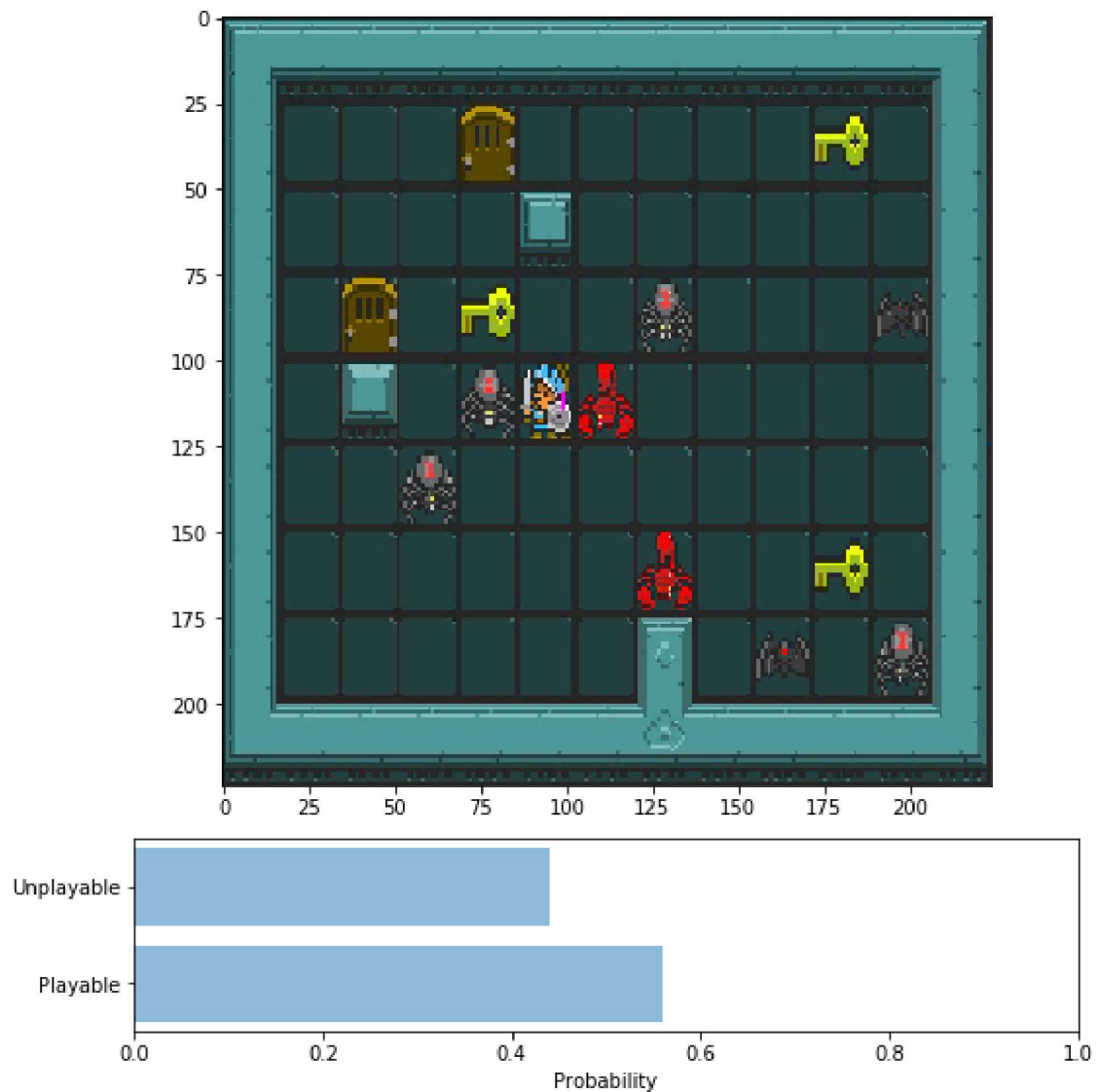
```
In [11]: model = load_model(MODEL_FILE)
```



```
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([0.5603117 , 0.43968827], dtype=float32)



```
In [13]: import numpy
TEST_DIR = 'test'
img_width = 224
img_height = 224
```

```

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_per_epoch)
# Get most likely class
predicted_classes = numpy.argmax(predictions, axis=1)

```

Found 30 images belonging to 2 classes.

```

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

```

```

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	0.50	1.00	0.67	15
zeldaUnplayablelevels	0.00	0.00	0.00	15
accuracy			0.50	30
macro avg	0.25	0.50	0.33	30
weighted avg	0.25	0.50	0.33	30

```

/Users/friends/anaconda3/envs/udacity-ehr-env/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))

```