*

Squeezing Ten Pounds of Data in a Five Pound Sack
- or -
The Compression Problem

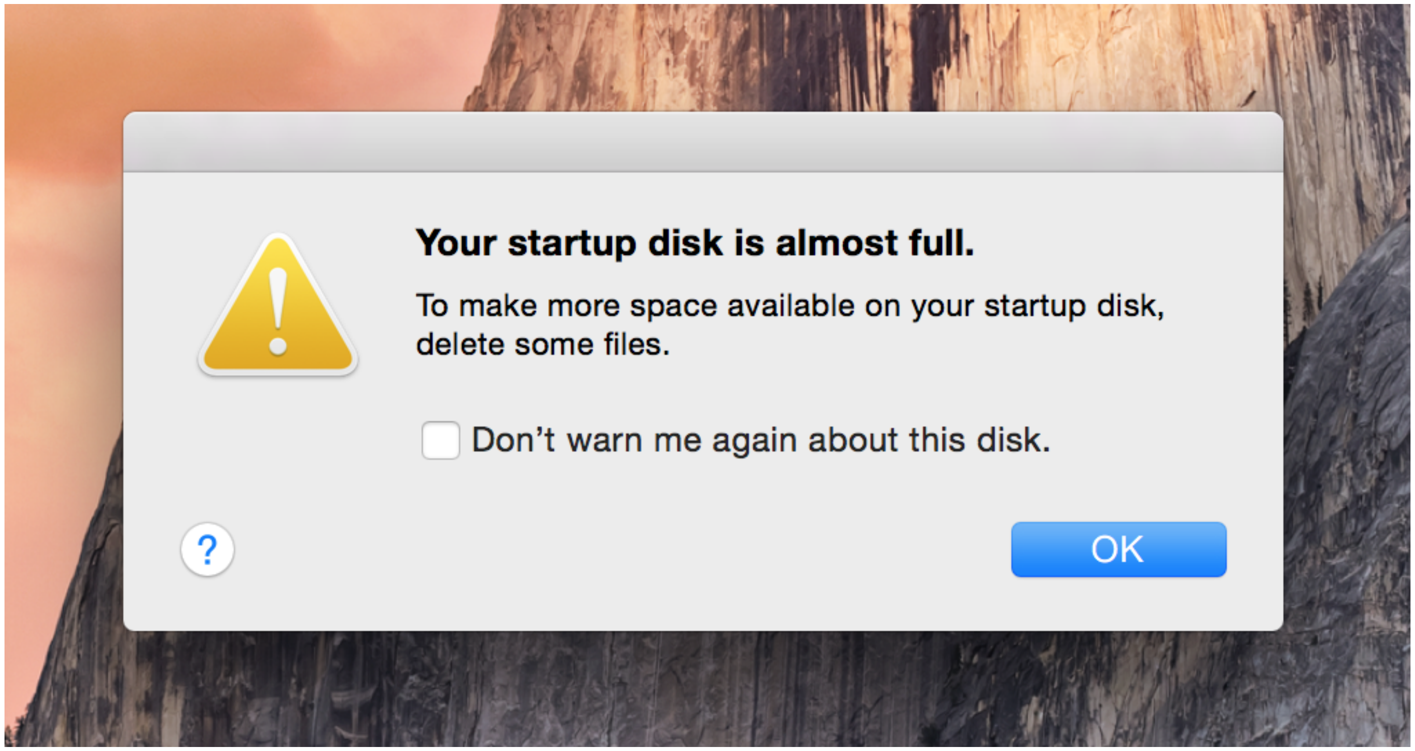
ISC1057

Janet Peterson and John Burkardt

Computational Thinking

Fall Semester 2016

When you're packing for a trip, you want to minimize the number of suitcases and bags you take, but maximize the things you can transport. Sometimes it takes several repackings, rolling and squashing and rearranging, before you are satisfied.



Your startup disk is almost full.

To make more space available on your startup disk, delete some files.

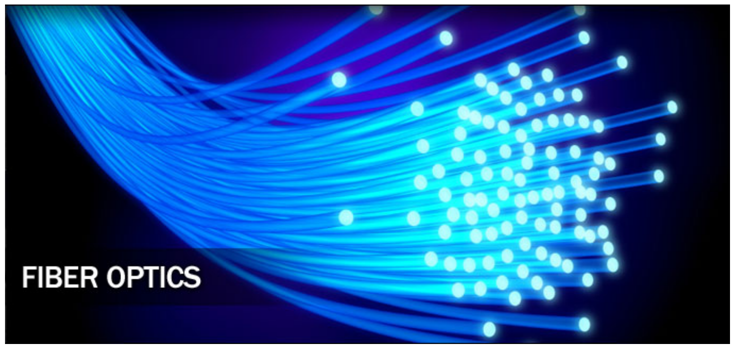
☐ Don't warn me again about this disk.

The world of computing has similar problems.

Data, such as files, pictures, movies, songs, web pages, must be stored, but there is only a limited amount of storage, and an ever-increasing stream of data.

Not only must data be stored, it very often must be transferred, and sometimes over a very slow network. In North America, 70% of Internet traffic involves services that are streaming data, such as movies or music. When the local network is overloaded, a movie becomes unwatchable, music becomes noise, users are dissatisfied.

Dropbox



One solution to these problems is to increase capacity:

Buy more suitcases or bigger ones.

Buy another hard drive, or write stuff to a memory stick or DropBox.

Add more network servers, more cables, convert to fiber optics.

These temporary solutions all cost money and only help a few users.

Imagine if we could keep our suitcase, but shrink our clothes.

If the shrinking method is fast, safe, and cheap, then we have made a great improvement for everyone.

This is the idea behind data compression: don't buy bigger "buckets" and "pipes", get smaller data instead!

BALLAST—ORDERS TO GO IN BALLAST.

CODEX.	INTERPRETATION.
Stallion	PROCEED in ballast to Key West.
Stalwart	Proceed in ballast to.
Stammel	Proceed and report for orders at.
Stampede	Proceed to a cotton port.
Stannate	Proceed to New Orleans.
Stanzas	Proceed to Mobile.
Starling	Proceed to Savannah.
Starosty	Proceed to Charleston.
Starvation	Proceed to New Orleans or Mobile.
Statesman	Proceed to Charleston or Savannah.
Stateswoman	Proceed to cotton port in Gulf of Mexico.
Statics	Go direct in ballast with all possible dispatch to.
Stationery	If you cannot get fair freight go in ballast.
Statistology	If you can get fair rate direct.
Statocracy	Call off and if freights are not favorable.
Statuette	If you can make the voyage quicker, touch off Tybee for advice.
Staylaced	Better not change ports for freight.
Stayless	Call off only and proceed elsewhere if freights are reported higher.
Steadfast	Make all possible dispatch.
Stearine	Hurry up and go to place as soon as possible.
Steel	*See Date Table, page* 150.
Stegnotic	IN BALLAST.
Stellary	Go in ballast.
Stelography	To go out in ballast.
Stencil	Vessel is going in ballast.
Stentograph	Come (or go) home in ballast.
Stentorian	Vessel is bound in ballast to.
Stercorate	To go in ballast 2/6 extra.
Stereobate	Coals for ballast.
Stereograph	Keep in cotton ballast.
Sterometer	Buy cotton ballast.
Stereopticon	Buy a good ballasting of.
Stereoscope	Merchants will not take vessel unless she goes in ballast.
Stereotomy	Owners don't want vessel to go in ballast.
Stereotype	Charterers supplying ballasting of coals at.
Sterility	Charterers object to supply coals for ballasting at place referred to.
Sternutative	Sufficient coals for ballast to be kept aboard till vessel gets to.
Stertorous	Has gone in ballast.
Stewpan	Go for guano in ballast.
Stibial	Proceed to Callao; vessel is chartered for guano.
Stitching	Go for rice in ballast.

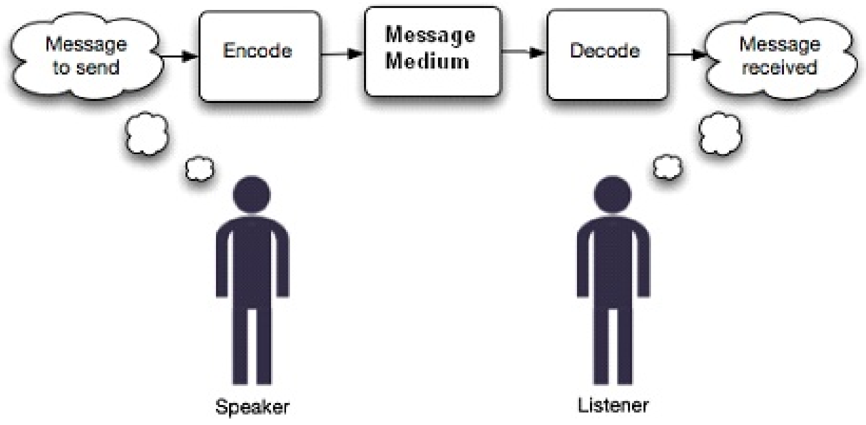
Compression Example 1:

Before the internet, radio, and telephone, people relied on the telegraph for communication. Sending a telegram could be time consuming and expensive. The user wrote the message on a standard form, went to a telegraph office, where an operator collected a fee based on the number of words, and then transmitted the message using short and long taps (dots and dashes) on a single key.

To save money, users shortened their messages. The message "Your mother is sick. Come home as soon as you can." would be written "MOM SICK. COME IMMEDIATELY."

Commercial users relied on sending hundreds of telegrams a day; the messages they sent were of so simple that it was possible to write most of them down on a menu. They realized that instead of sending the message, they could just send the index of the message. To avoid mistakes, it was common to use words as the index of the messages.

The message *If you cannot get fair freight, go in ballast* has an index word of **STATIONERY**; the recipient needs the code book in order to turn the one word telegram into the message.



The message goes through three steps, which we can think of as **ENCODE**, **TRANSMIT** and **DECODE**.

The shipping code also provides secrecy; someone without the code book can't understand what the message **STATIONERY** means.

Abbreviations A to L	
2moro	Tomorrow
2nte	Tonight
AEAP	As Early as Possible
ALAP	As Late as Possible
ASAP	As Soon as Possible
ASL	Age / Sex / Location?
B3	Blah, Blah, Blah
B4YKI	Before You Know it
BFF	Best Friends, Forever
BM&Y	Between Me and You
BRB	Be right Back
BRT	Be right There
BTAM	Be that as it May
C-P	Sleepy
CTN	Cannot talk now
CUS	See You Soon
CWOT	Complete Waste of Time
CYT	See You Tomorrow
E123	Easy as 1, 2, 3
EM?	Excuse Me?
EOD	End of Day
F2F	Face to Face
FC	Fingers Crossed
FOAF	Friend of a Friend
GR8	Great
HAK	Hugs and Kisses
IDC	I Don't Care
IDK	I Don't Know
ILU / ILY	I Love You
IMU	I Miss You
IRL	In Real Life
J/K	Just Kidding
JC	Just Checking
JTLYK	Just to Let You Know
KFY	Kiss for You
KMN	Kill Me Now
KPC	Keeping Parents Clueless
L8R	Later

Abbreviations M to Z	
MoF	Male or Female
MTFBWY	May the Force be with You
MYOB	Mind Your Own Business
N-A-Y-L	In a While
NAZ	Name, Addess, ZIP
NC	No Comment
NIMBY	Not in my Backyard
NM	Never Mind / Nothing Much
NP	No Problem
NSFW	Not Safe for Work
NTIM	Not that it Matters
NVM	Never Mind
OATUS	On a totally Unrelated Subject
OIC	Oh, I See
OMW	On My Way
OTL	Out to Lunch
OTP	On the Phone
P911	Parent Alert
PAL	Parents are Listening
PAW	Parents are Watching
PIR	Parent in Room
POS	Parent over Shoulder
PROP(S)	Proper Respect / Proper Recognition
QT	Cutie
RN	Right Now
RU	Are You
SEP	Someone else's Problem
SITD	Still in the Dark
SLAP	Sounds like a Plan
SMIM	Send Me an Instant Message
SO	Significant Other
TMI	Too Much Information
UR	Your / You are
W8	Wait
WB	Welcome Back
WYCM	Will You Call Me?
WYWH	Wish You Were Here
XOXOXOX	Hugs, Kisses, ...

Compression Example 2:

There are many shortcuts and abbreviations invented for texting on cellphones.

Reasons for this include:

- you're often texting while doing something else: sitting in class, or at lunch, or walking, and you don't have much time to get your message in.
- a cellphone doesn't have room to display a length message, so short messages are preferred.
- it's often awkward to type on the keypad of a cellphone, and so shortening the message reduces the work.

Compression is a favorite tool of computers, servers and networks:

Messages sent over the Internet are compressed, transmitted, and then decompressed.

Audio signals (telephone, CD, streaming music) is compressed; (moments of silence can be squeezed out, for one thing.)

When you take a picture on your phone, and want to send it to someone, you are offered the choice of low, medium or high resolution versions of the picture to send.

Almost all software packages are downloaded in a compressed form and have to be decompressed before use.

The **ZIP** and **GZIP** formats are popular ways of compressing computer files to reduce the amount of storage required.

When compressing information, there is an important choice to be aware of. Data compression can be:

- lossless, uncompressing the compressed file recovers all the original information;
- lossy, uncompressing the compressed file does not return all information; some is lost.

Our shipping code example is lossless, because when the compressed word **STATIONERY** is uncompressed, we get back exactly the original message that was sent.

We will start by considering lossless compression, but after that, we will come back to lossy compression and see that there may be times when it is the right solution to a compression problem.

The idea of lossless compression is simple.

We start with a long message, and notice patterns that make it easy to describe the message briefly, but accurately.

For example, if someone needs to see you for an hour during the work week, you might respond by listing every possible hour and whether you're free or not: *Monday at 8: busy, Monday at 9: busy, Monday at 10: busy...* and listing your status for each of the 40 possible hours.

But you might be able to correctly summarize the situation by saying *"Monday and Tuesday are full, and I'm booked from 1 to 3 on Thursday and Friday, but otherwise I'm free."*

The person receiving this message can perfectly reconstruct your calendar, but you didn't have to list each hour explicitly.

A simple lossless compression is called run length encoding, or **RLE**.

Suppose the messages we want to send are strings of letters, and consider how we might send the following message:

AAAAAAAAAAAAAAAAAAAAAAAABCBCBCBCBCBCBCBCBCBCAAAAAADEFDEFDEF

Over the phone, you might describe this message as:

21 A's, then 10 BC's, then 6 A's, then 3 DEF's

and if you wrote the message, you might write:

21A,10BC,6A,3DEF.

Notice that a message that was 56 letters long has been compressed to 16 letters, numbers and commas.

The compression ratio is the size of the original message divided by the size of the compressed message. In this case, our compression ratio is 56 / 16 = 3.5.

Run length encoding is a useful procedure in cases where messages often consist of letters or sets of letters that are immediately repeated several times.

Messages written in English don't have this property, but computer data is full os such situations.

A simple case is the fax machine, which accepts a document, somehow takes a picture of it, and transmits that to another site.

In fact, the fax machine "picture" is simply a black or white dot at regularly spaced locations on the page. This means that, whether the original document is text or a photograph, the fax message will consist of 1's for black, and 0's for white.

```
@@.....@@.....@@@@@@@@@.....@@............@@@@@@@@.
@@.....@@.....@@............@@............@@.....@@
@@@@@@@@@.....@@@@@@@@@.....@@............@@@@@@@@.
@@.....@@.....@@............@@............@@.......
@@.....@@.....@@@@@@@@@.....@@@@@@@@@.....@@.......
```

This picture suggests how a scanner might “see” the word **HELP** on a document, using 255 characters.

RLE, applied to each row, reduces this to 147 characters:

```
2@,5.,2@,5.,9@,5.,2@,12.,8@,1.
2@,5.,2@,5.,2@,12.,2@,12.,2@,5.,2@
9@,5.,9@,5.,2@,12.,8@,1.
2@,5.,2@,5.,2@,12.,2@,12.2@,7.
2@,5.,2@,5.,9@,5.,9@,5.,2@,7.
```

for a compression ratio of $255/147 = 1.73$.

Run length encoding is a simple idea, and it misses many opportunities for compression because it only works if the repetitions are adjacent.

For instance, RLE can compress $\mathbf{ABABAB}$ but it can do nothing with the repeated $\mathbf{AB}$'s in the string $\mathbf{ABXABYABZ}$.

Because compression can be so useful, many clever ideas have been suggested for handling more complicated situations by lossless compression.

Consider you were given the following message to repeat over the telephone:

VJGDNQMYLHKWVJGDNQMYLHADXSGF0VJQGNQMYLHADXSGFVJGNMQMYLHEWADXSGF

Notice that two strings of letters occur several times:

VJGDNQMYLHKWVJGDNQMYLHADXSGF0VJGDNQMYLHADXSGFVJGDNQMYLHEWADXSGF

So while you are giving the message, it would save a lot of time to be able to say: *"this part is the same as something I told you earlier."*

To be precise, you would have to point to the start and length of the first occurrence of this string.

VJGDNQMYLHKWVJGDNQMYLHADXSGF0VJGDNQMYLHADXSGFVJGDNQMYLHEWADXSGF

The first 12 characters don't show any repetition so we just have to read them out, *"V, J, G, D, N, Q, M, Y, L, H, K, W"*.

The next 10 characters are the same as earlier ones, so you could say, *"back 12, copy 10"*.

The next 7 characters are new, so we say *"A, D, X, S, G, F, 0"*.

The next 16 characters are a repeat, so we say *"back 17, copy 16"*.

Another repeat follows, so we say *"back 16, copy 10"*.

Two new characters have to be listed, *"E, W"*.

We finish saying *"back 18, copy 6"*.

Our original 63 character string was:

VJGDNQMYLHKWVJGDNQMYLHADXSGF0VJGDNQMYLHADXSGFVJGDNQMYLHEWADXSGF

Using the abbreviations **b** for "back" and **c** for "copy", our revised string could be written as:

VJGDNQMYLH-KW-b12c10-ADXSGF-0-b17c16-b16c10-EW-b18c6

Ignoring the dashes, which we inserted for clarity, our message has been shortened from 63 characters to 44, for a compression ratio of $63/44{=}1.43$.

It's easy to come up with a compression scheme if you know beforehand what kind of strings are going to be repeated. For a message in which "The Supreme Court of the United States" will appear many times, you can write **SCOTUS** with the understanding that your recipient will expand this abbreviation back to its original form.

But this method, called **The Same As Earlier Trick**, allows you to compress a message containing any kind of repeated string.

Here's one extra feature of The Same As Earlier Trick.

Suppose your message was the 16 characters:

FGFGFGFGFGFGFGFG

Then your "compressed" version might be

FG-b2c2-b2c2-b2c2-b2c2-b2c2-b2c2-b2c2

which is longer than the original. But in fact, you could write

FG-b2c14

achieving a compression ratio of 2.28

In order to understand the next approach to compression, it is necessary to be a little more realistic about how information is stored in the computer.

When the user enters letters like $\mathbf{a}$, $\mathbf{b}$ or $\mathbf{c}$, the computer doesn't actually store these letters. The computer can only store numbers, and so it uses a table that can "translate" letters and other symbols to a numeric code, or convert the numeric code back to a letter.

For example, the letter $\mathbf{a}$ might be represented by the numeric code $\mathbf{27}$, $\mathbf{b}$ by $\mathbf{28}$, $\mathbf{c}$ by $\mathbf{29}$ and so on.

When the user enters a character such as $\mathbf{a}$, we say it is encoded into the numeric symbol $\mathbf{27}$, and when the computer sends the numeric symbol $\mathbf{27}$ back to the user, it is decoded as $\mathbf{a}$.

$$a \rightarrow \text{(encode)} \rightarrow 27 \rightarrow \text{(decode)} \rightarrow a$$

This means that a string which the user enters, such as **cab**, will be stored in the computer as a string of numbers, namely **29,27,28**.

You probably know that computers do not use the decimal system, but a base two system called *binary arithmetic*.

We'll explain things using base 10, but the points will still apply to the actual binary arithmetic used in the computer.

Now we look more closely at how a computer handles characters. The following example table *(not the actual table used in computers!)* contains numeric codes for 100 symbols. These codes run from 0 to 99, that is, they are each a one- or two-digit decimal number.

The list includes the alphabetic characters in lower and uppercase, punctuation marks, other symbols, and some accented letters that occur in foreign languages.

space	00	T	20	n	40	(	60	á	80
A	01	U	21	o	41	)	61	à	81
B	02	V	22	p	42	*	62	é	82
C	03	W	23	q	43	+	63	è	83
D	04	X	24	r	44	,	64	í	84
E	05	Y	25	s	45	-	65	ì	85
F	06	Z	26	t	46	.	66	ó	86
G	07	a	27	u	47	/	67	ò	87
H	08	b	28	v	48	:	68	ú	88
I	09	c	29	w	49	;	69	ù	89
J	10	d	30	x	50	<	70	Á	90
K	11	e	31	y	51	=	71	À	91
L	12	f	32	z	52	>	72	É	92
M	13	g	33	!	53	?	73	È	93
N	14	h	34	”	54	{	74	Í	94
O	15	i	35	#	55	|	75	Ì	95
P	16	j	36	$	56	}	76	Ó	96
Q	17	k	37	%	57	_	77	Ò	97
R	18	l	38	&	58	Ø	78	Ú	98
S	19	m	39	’	59	ø	79	Ù	99

Let's look at how an example sentence is encoded by the table.

The 23 character English sentence “**Meet your fiancé there.**” would become the following list of numeric codes:

```
M  e  e  t     y  o  u  r     f  i  a  n  c  e     t  h  e  r  e  .
13 31 31 46 00 51 41 47 44 00 32 35 27 40 29 82 00 46 34 31 44 31 66
```

Since we're concerned about compression, it's important to realize that, when measuring the length of the list of numeric codes, we should not include any spaces or commas. As far as the computer is concerned, the message has become the following string of 46 characters:

```
1331314600514147440032352740298200463431443166
```

Since each numeric code is exactly two digits, we can easily break this string into its 23 separate codes if we need to.

All the numeric codes used two digits, so $\mathbf{A}$ is coded as $\mathbf{01}$, not $\mathbf{1}$.

Because of this, when we see a coded string like

`1331314600514147440032352740298200463431443166`

we only break it up into digit pairs:

`13 31 31 46 00 51 ...`

and we would not break it up into, say:

`1 33 13 14 60 0 5 1...`

The translation between characters and numeric codes must always have a single interpretation, even when the numeric codes are written packed together, with no spaces or commas between them.

This rule will become an important issue shortly!

Now we are ready to discuss the ideas behind the next compression procedure, known as **The Shorter Symbol Trick**. Actually, this trick is based on something we do all the time in everyday communication. The idea is that if you use a phrase often enough, it's worthwhile to come up with a shorter version of it.

Everyone knows that **USA** is short for *The United States of America*; we save a lot of time or typing by using the 3 letter abbreviation for this 24 letter phrase.

The sky is blue in color is another 24 letter phrase, but no one has bothered to come up with an abbreviation for it.

What is the difference? One phrase is rarely used; the other occurs often, so inventing and using an abbreviation for it is worth the effort.

Let's see if we can apply the Shorter Symbol Trick to compress the message we considered earlier, *Meet your fiancé there.* We know we should focus on the most commonly occurring items.

The letters e and t are the most common in English, but in the original table we used two digits for each of them. How about cutting them down to one-digit codes? Suppose $\mathbf{e}$ is coded as $\mathbf{8}$ and $\mathbf{t}$ as $\mathbf{9}$.

Now we cut down the encoded message from 46 to 40 decimal digits:

```
M  e e t    y  o  u  r     f  i  a  n  c  e     t h  e r  e  .
13 8 8 9 00 51 41 47 44 00 32 35 27 40 29 82 00 9 34 8 44 8 66
```

or, written all together:

```
1388900514147440032352740298200934844866
```

The person sending the message is happy, because now the encoded message has been compressed...but we have created a serious problem for the person receiving the message, who has to do the decoding.

Since we switched to using both 1-digit and 2-digit numeric codes, it's no longer clear how to chop up the message into the individual codes. Let's concentrate on the first five digits of the message.

```
13889005141474400323527402982009348448 66
```

could be intepreted as $\mathbf{13\ \ 8\ \ 8\ \ 9}$ or $\mathbf{13\ \ 88\ \ 9}$ or $\mathbf{13\ \ 8\ \ 89}$ which would decode to **Meet** or **Mút** or **Meù**.

There is no way to tell which of these three messages is the intended one.

Our scheme has made shorter compressed messages, but now we can't decode them!

Luckily, we can salvage the Shorter Symbol Trick if we are willing to make some symbols longer.

One way to solve our problem is to put a $\mathbf{7}$ in front of every one of the ambiguous 2-digit codes.

This will allow us to have short (1 digit) symbols for $\mathbf{e}$ and $\mathbf{t}$, medium (2 digit) symbols for other alphabetic characters, and long (3 digit) symbols for rarely used characters.

Our new coding table includes all the changes:

space	00	T	20	n	40	(	60	á	780
A	01	U	21	o	41	)	61	à	781
B	02	V	22	p	42	*	62	é	782
C	03	W	23	q	43	+	63	è	783
D	04	X	24	r	44	,	64	í	784
E	05	Y	25	s	45	-	65	ì	785
F	06	Z	26	t	9	.	66	ó	786
G	07	a	27	u	47	/	67	ò	787
H	08	b	28	v	48	:	68	ú	788
I	09	c	29	w	49	;	69	ù	789
J	10	d	30	x	50	<	770	Á	790
K	11	e	8	y	51	=	771	À	791
L	12	f	32	z	52	>	772	É	792
M	13	g	33	!	53	?	773	È	793
N	14	h	34	”	54	{	774	Í	794
O	15	i	35	#	55	|	775	Ì	795
P	16	j	36	$	56	}	776	Ó	796
Q	17	k	37	%	57	_	777	Ò	797
R	18	l	38	&	58	Ø	778	Ú	798
S	19	m	39	’	59	ø	779	Ù	799

Using the new table, our message becomes 41 characters long:

```
13889005141474400323527402978200934844866
```

but now there is only one way to decode the message:

```
13 8 8 9 00 51 41 47 44 00 32 35 27 40 29 782 00 9 34 8 44 8 66
 M e e t     y  o  u  r     f  i  a  n  c  e’     t  h e  r e  .
```

The original message used 46 characters and now we’re down to 41. In practice, the compression ratios are much better.

The text book, for instance, requires about half a million characters of storage. However, using just the two compression tricks we have described, the compressed version is about 160,000 characters, for a compression factor greater than 3.

One example of how the tricks we have discussed are used in real life is when ZIP files are created:

1. the original file is transformed using the Same As Earlier trick;
2. the transformed file is analyzed to see which symbols occur most frequently;
3. using the Shorter Symbol Trick, a coding table is created in which frequent symbols get shorter codes;
4. the transformed file is encoded with the new coding table.

To expand the ZIP file, these steps are undone in the reverse order using the UNZIP program.

The file **dictionary.txt** is 3.15 million characters in size:

```
ls dictionary.txt
-rwxrwxrwx 1 jburkardt staff 3151520 May 29 2013 dictionary.txt
```

```
zip dictionary.zip dictionary.txt
ls dictionary.zip
-rw-r--r-- 1 jburkardt staff 896590 Apr 26 10:44 dictionary.zip
```

ZIP compresses the file to less than 1 million characters.

```
rm dictionary.txt
unzip book.zip
ls dictionary.txt
-rwxrwxrwx 1 jburkardt staff 3151520 May 29 2013 dictionary.txt
```

UNZIP recovers the exact original text.

We have been considering lossless compression, that is, methods for squeezing a message into a smaller version, which can later be expanded to the exact same information.

Another option is available, called lossy compression. The name is chosen to indicate that, if you compress a file, then when you uncompress it later, *some of the original information has been permanently lost.*

Lossy compression is almost <u>never</u> used in text. Even if we can make out most of the meaning of the following "compressed" text, it is not possible to be exactly sure what the original was.

Howard Beale's monolog in "Network", compressed by removing **a, e, i, o, u**:

```
dnt hv t tll y thngs r bd.
vrybdy knws thngs r bd
ts dprssn.
vrybdys t f wrk r scrd f lsng thr jb.
Th dllr bys nckls wrth; bnks r gng bst;
shpkprs kp gn ndr the cntr;
pnks r rnnng wld n th strt,
nd thrs nbdy nywhr wh sms t knw wht t d,
nd thrs n nd t t.
```

Howard Beale's monolog in "Network":

```
I don't have to tell you things are bad.
Everybody knows things are bad.
It's a depression.
Everybody's out of work or scared of losing their job.
The dollar buys a nickel's worth; banks are going bust;
shopkeepers keep a gun under the counter;
punks are running wild in the street,
and there's nobody anywhere who seems to know what to do,
and there's no end to it.
```

Although text is not appropriate for lossy compression, it turns out that many other kinds of data files are ideal candidates for lossy compression.

The reason is that a typical computer data file may contain a mixture of useful and useless information. For example, a simple way to record sound involves taking 44,100 samples per second.

A typical 640MB CD can hold about 1 hour of uncompressed music, or 2 hours of losslessly compressed music, or 7 hours of music compressed using the lossy MP3 method.

The MP3 recording is played (decoded, actually), the resulting music will not be identical to the original samples. However, it is generally close enough that the listener would have great difficulty detecting the differences.

The techniques used to compress audio recordings are based in part on recognizing the limitations of the human ear:

- you can't hear very low or high frequency tones;
- you can't hear very soft tones;
- you can't hear soft tones just after a loud one;
- if low and high frequency tones are played with the same loudness, you hear the low frequency (drumbeat over flute);
- if music is to be played in a noisy environment, only loud or low frequency information will be heard at all.

Using ideas like this, a compression program can remove as much as 80 or 85% of the original information, producing a stripped-down recording that will still sound very close to the original.

Photos, drawings, and movies can also be compressed, and again, compression is aided by understanding limitations of the human eye, including:

- just as the human ear can't hear high frequency tones, the human eye is not good at detecting high frequency variations in an image, for example, a pattern of alternating black and white lines, if thin enough, will simply register as solid gray.
- the eye essentially averages the information in a small neighborhood around any spot in a picture. Thus, we can blur the image somewhat, without noticeable effect.
- although a computer can display millions of different colors, the human eye is not very sensitive to small color differences;

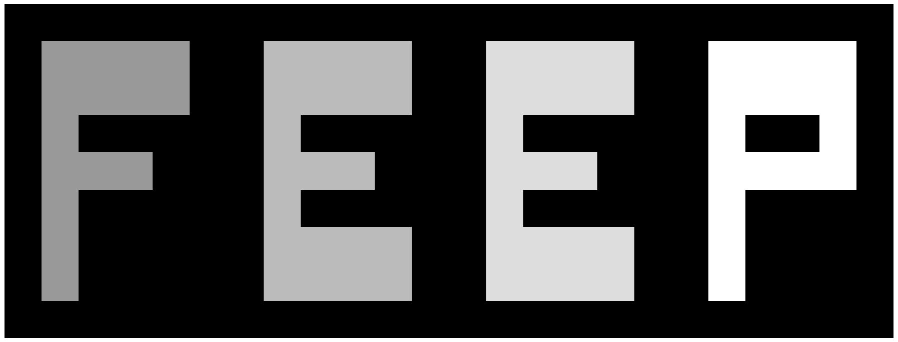
While programs like JPEG use very sophisticated techniques to compress image files, we can easily demonstrate some of the simplest ones. For example, we have **The Leave It Out Trick**.

We will look at this method in terms of a black and white image. Note that most such images are actually black and white *and hundreds of shades of gray*, and so are sometimes called grayscale images.

An uncompressed image is stored as a rectangular table, like a piece of graph paper. Each entry in the table, or box in the graph paper, is called a pixel (an abbreviation of *picture element*, and contains a number representing the shade at that point in the picture.

Here is a simple uncompressed grayscale image file, (called **FEEP**), using gray shades between 0 (black) and 15 (white):

```
0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
0 9  9  9  9  0  0 11 11 11 11  0  0 13 13 13 13 0  0 15 15 15 15  0
0 9  9  9  9  0  0 11 11 11 11  0  0 13 13 13 13 0  0 15 15 15 15  0
0 9  0  0  0  0  0 11  0  0  0  0  0 13  0  0  0 0  0 15  0  0 15  0
0 9  9  9  0  0  0 11 11 11  0  0  0 13 13 13  0 0  0 15 15 15 15  0
0 9  0  0  0  0  0 11  0  0  0  0  0 13  0  0  0 0  0 15  0  0  0  0
0 9  0  0  0  0  0 11 11 11 11  0  0 13 13 13 13 0  0 15  0  0  0  0
0 9  0  0  0  0  0 11 11 11 11  0  0 13 13 13 13 0  0 15  0  0  0  0
0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 0  0  0  0  0  0  0
```



The FEEP file used a table of 9 rows by 24 columns, containing grayscale values between 0 and 15.

When we display the FEEP picture, we see the regions of darkness and light corresponding to the numeric values.

However, the image does not seem to be made of squares of solid colors.

This is because, in order to be viewed, we must decode the image file. Most programs that decode or display an image file will automatically try to smooth out the rectangular boundaries between various pixels of different colors, to make the picture look better.

Thus, it is actually a little difficult to force our blocky picture to show up as a bunch of blocks!

A more typical grayscale graphics file might use 256 shades of gray, going from 0 for full black to 255 for full white.

Computer generated pictures often have height to width ratios of 4 to 3, (portrait) or 3 to 4 (landscape)

A modern camera or cellphone may store images as a pixel table of 4,000 rows by 3,000 columns, for a total of 12,000,000 pixels.

Many television screens have an array of 1920 pixels wide and 1080 pixels high, or 2,073,000 total pixels.

Many images on computers and the Internet are somewhat smaller, with a typical size being 480 x 360 = 165,600 pixels,

The total number of pixels is known as the resolution of the picture.

Now 12 million gray dots is a lot of information, and your eye might be perfectly satisfied with a much simplified version.

In fact, if I want to send a picture from my phone to someone else, the phone suggests sending a medium or low resolution version, because the picture will still be good enough to view, and I can reduce my data transmission charge by sending a smaller image.

How is a lower resolution version created?

Suppose that we started with a 460 by 360 pixel image file, and simply removed every other row and column. Our example would drop from 165,600 pixels to 230 * 180 = 41,400 pixels, that is, it would be reduced to $1/4$ the number of pixels.

This compression method is easy...and definitely lossy.

But will it result in a usable picture?

Let's experiment by starting with a standard size image, and cutting out the even rows and even columns.

This creates a file that is 1/4 the size of the original, by permanently losing 3/4 of the information.

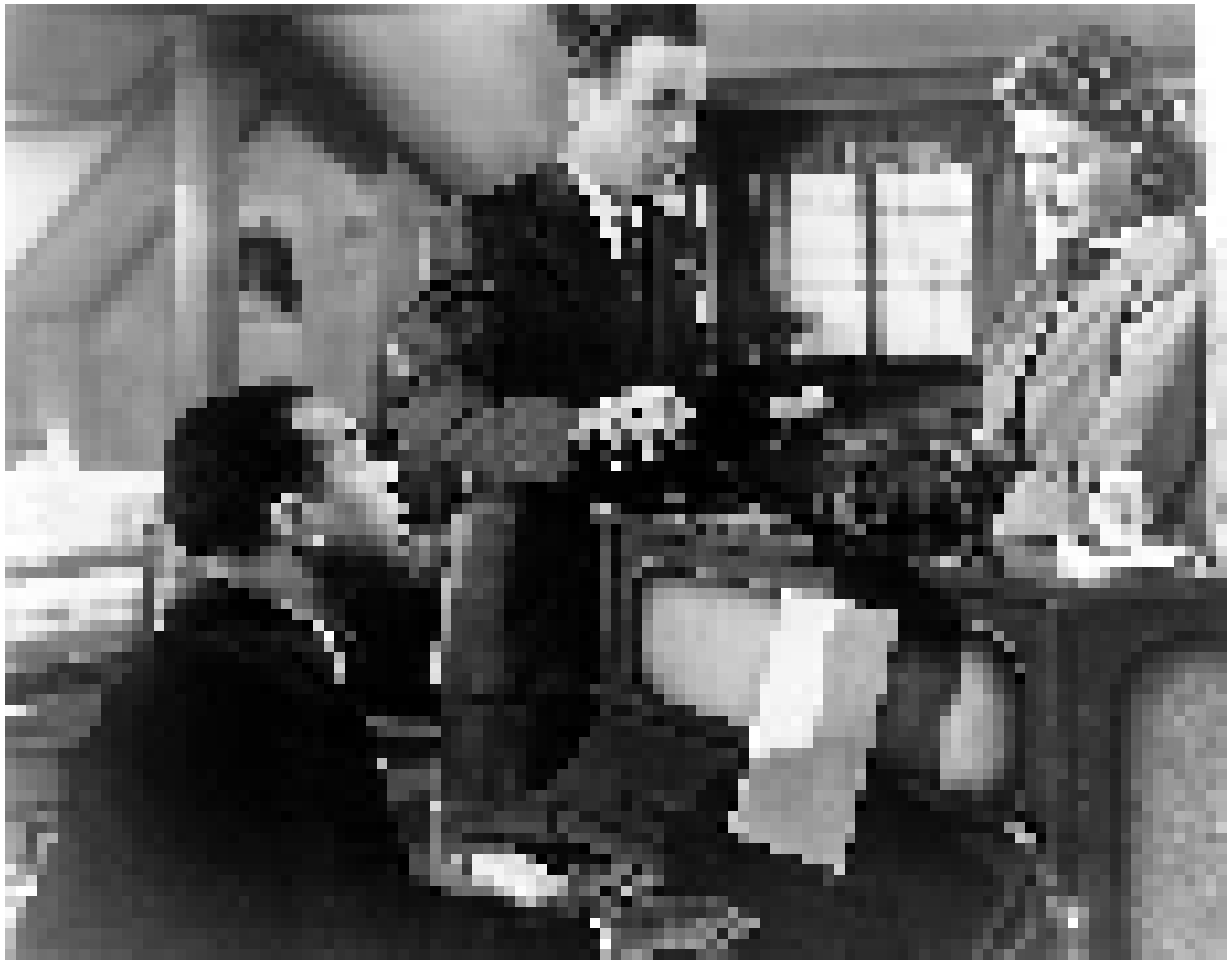
Our only hope is that, because the missing information is right next to information we are keeping, the eye will not notice the small disruptions.

And if we are satisfied with the compressed image, we can try a second compression....

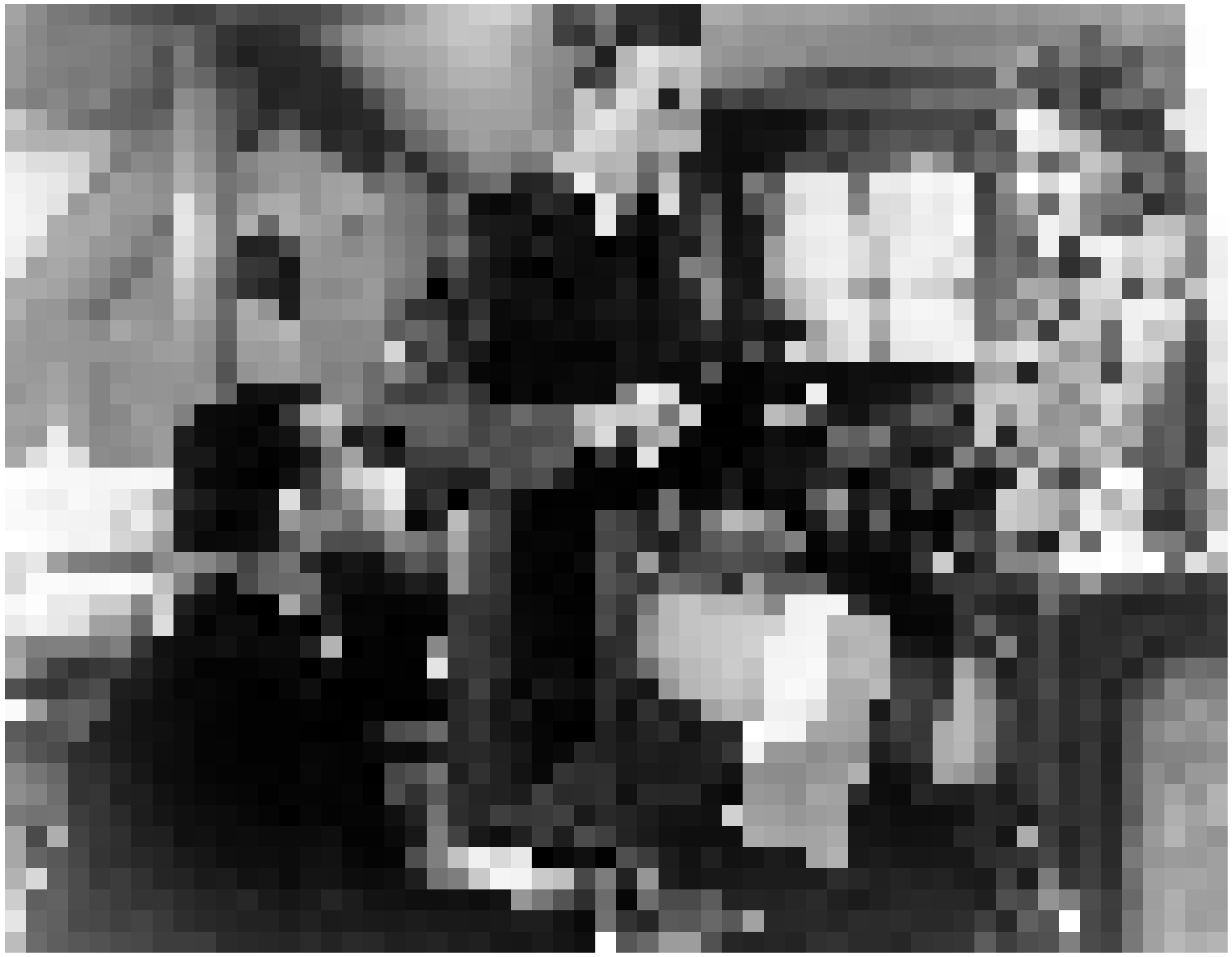
460 * 360 = 165,600 pixels

230 * 180 = 41,400 pixels

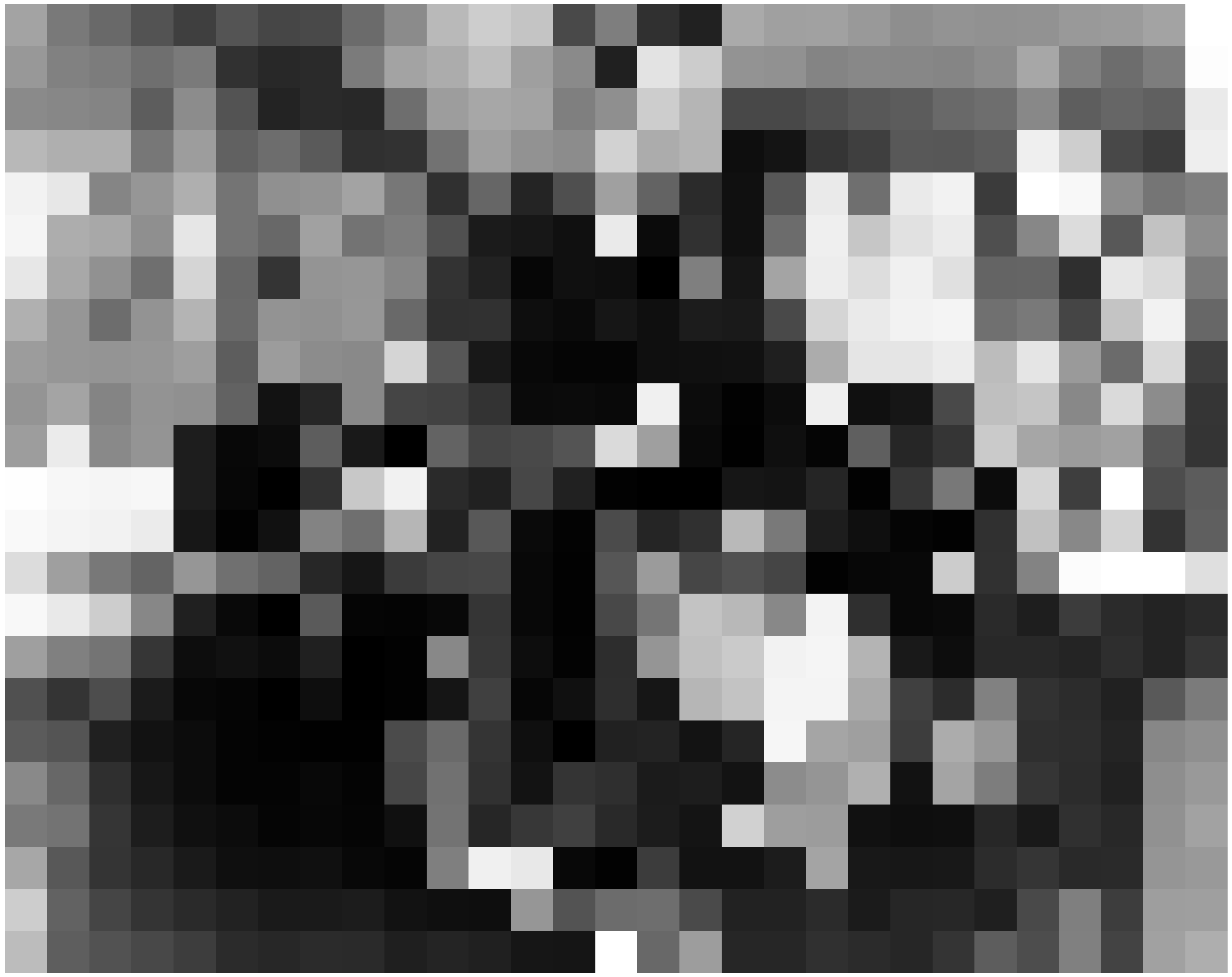
115 * 90 = 10,350 pixels



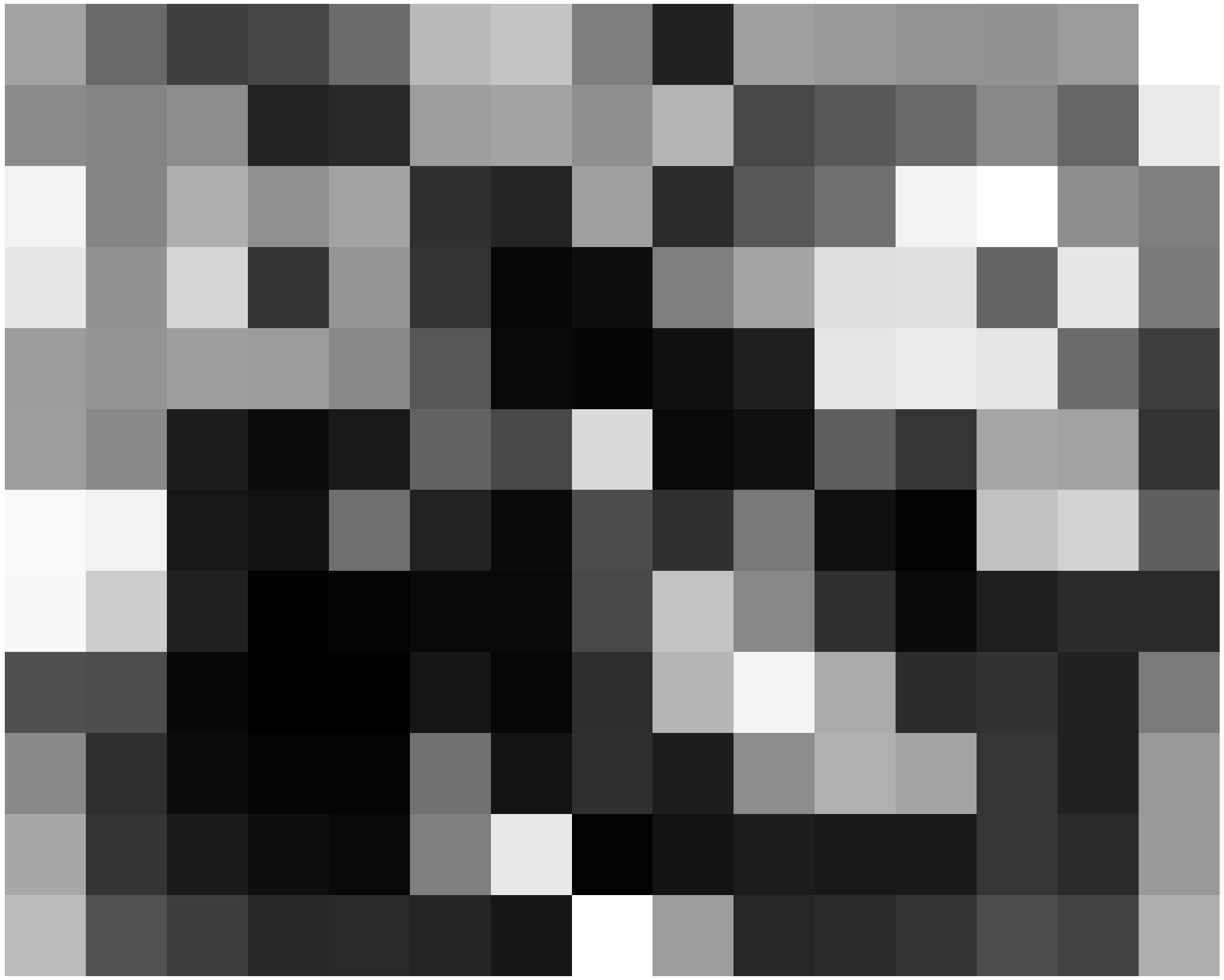
58 * 45 = 2,610 pixels



29 * 23 = 667 pixels



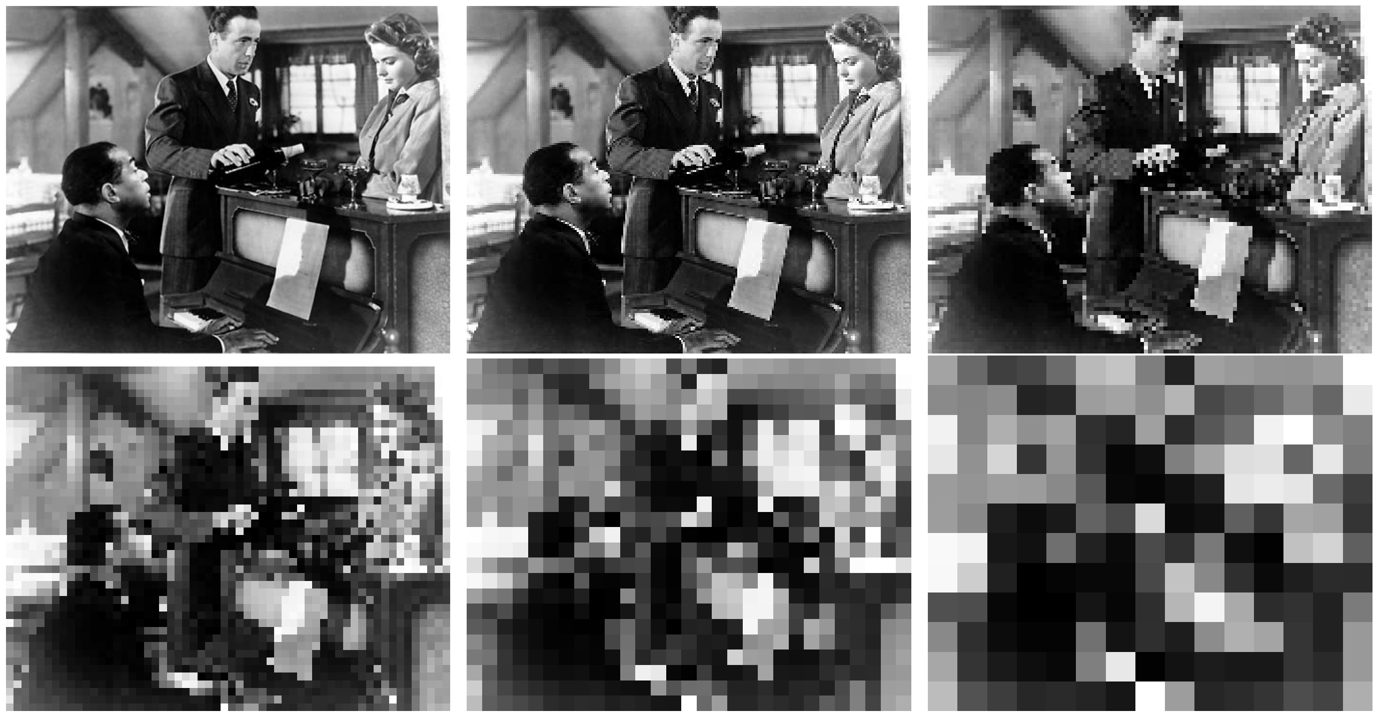
15 * 12 = 180 pixels



If we look at all six versions of our image together, even the second compressed version may not look too bad...because we've reduced its size to show it together with the others.

But if we go back to the larger version, our eye may be able to spot some unnatural looking jagged boundaries that correspond to the rectangular pixels that make up the computer image.

But even after one or two more compressions, we can still see the main items in the picture. Even with only 667 pixels, we may not be able to tell what is going on, but we can guess that there are 3 or 4 people in the picture. Of course, with 180 pixels, there is just not enough information for our eyes to make any sense of things.



We have applied the Leave It Out Trick in a very simple manner, by dropping rows and columns of the image file without worrying about the information they contain.

Because images are used so frequently, and the raw images created by cameras and medicals devices are so large, it has become important to develop much better lossy compression techniques for squeezing out as much information as possible, while retaining the information necessary for the eye to “read” the picture correctly.

The JPEG format is a widely used image compression technique that uses the Leave It Out Trick in a more intelligent and careful way.

JPEG files are more complicated than the simple image file format we discussed earlier; it is enough to know that the file size is measured in characters. If we convert our 480x360 pixel image to the standard JPEG format, it is stored using 60,231 characters, rather than 165,600 pixels.

Although 60,231 is smaller than 165,600, we have not actually asked JPEG to compress the image information at all.

The number of characters is smaller, only because JPEG knows how to store several pixels into a single character.

To compress the image, JPEG divides up the pixels into small squares of 8x8 pixels called blocks, and looks very carefully at them.

If all 64 pixels in a block are the same color, than the computer can simply “leave out” 63 numbers, remember simply that the block is all the same shade.

If the 64 pixels in a block are close to some color, then the computer can consider averaging the colors and using that average for the entire block.

If the colors in the block change smoothly from the left to the right, then the computer can store the left and righthand colors, and remember that the pixels in between get in between shades.

There are other patterns of variation that are simpler to describe than simply recording the individual shades of all 64 pixels.

We may allow JPEG to replace the exact data by a pattern even if the pattern is only an approximate match.

The quality of a JPEG image is 100% if we never allow any such approximation.

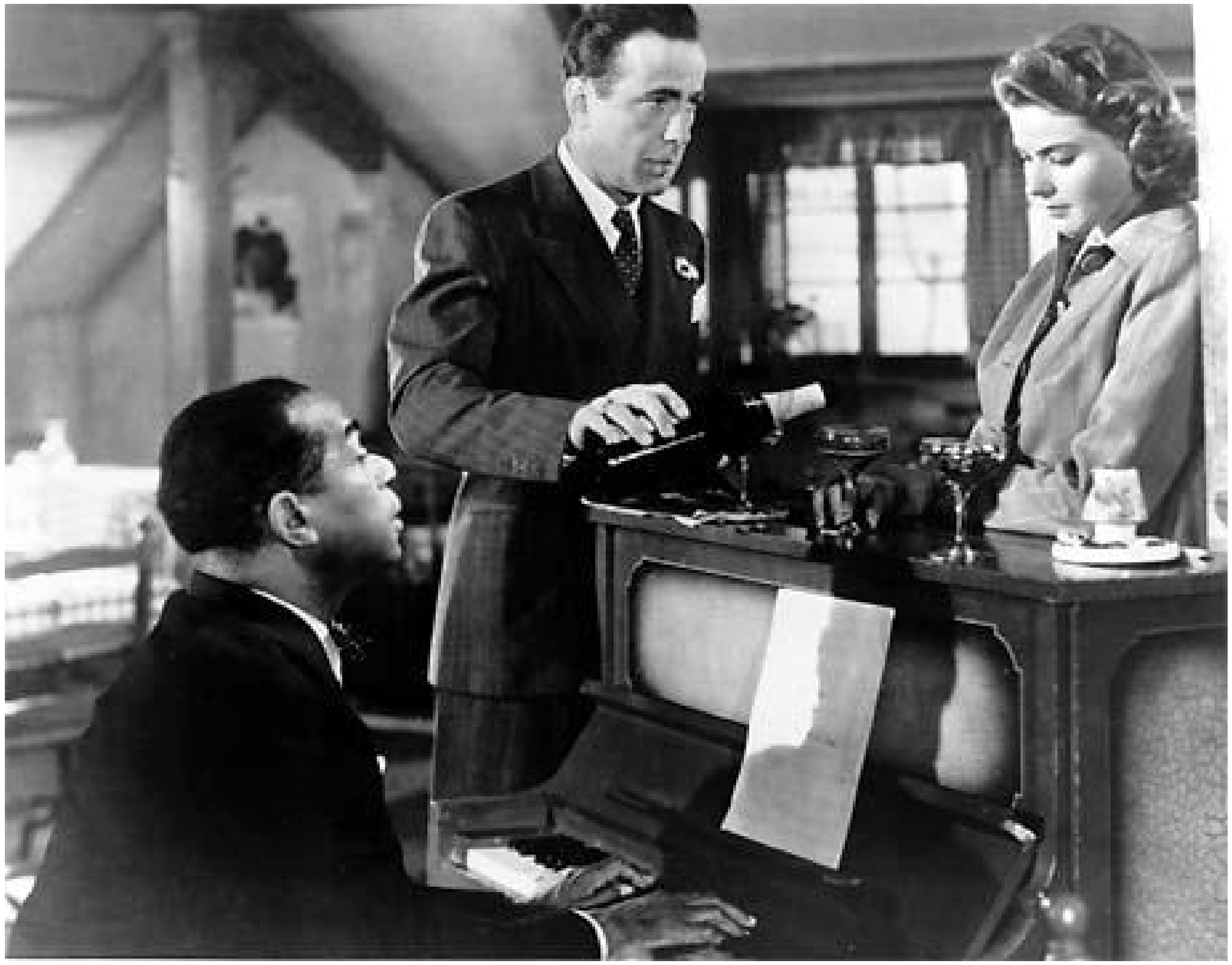
As we reduce the quality, we are allowing JPEG more leeway in replacing exact picture data with approximate patterns. As we reduce the quality, we improve the chances for compression.

We will see that JPEG can sometimes reduce the size of an image file by a compression ratio of 20 without the eye noticing much difference.

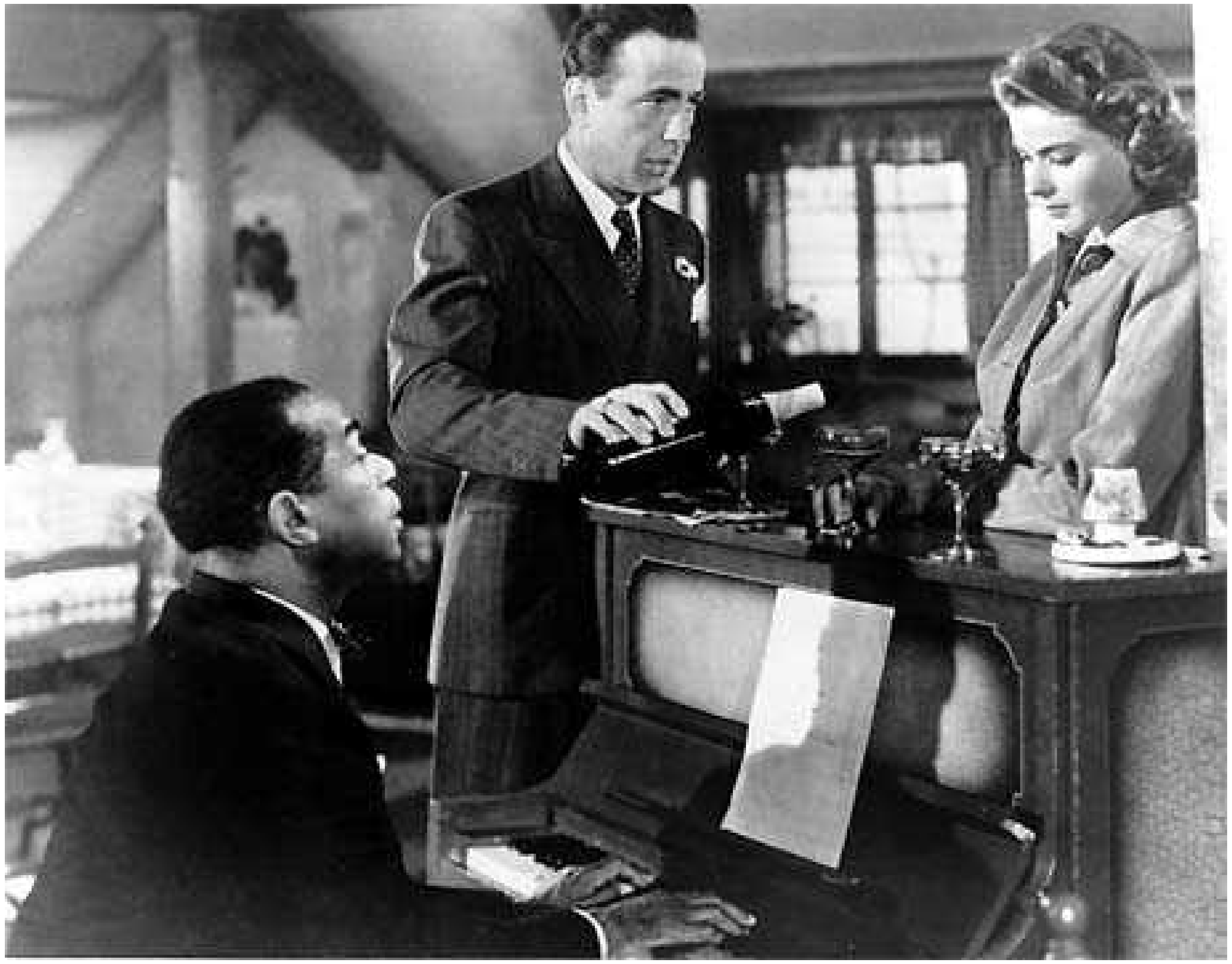
Here, we will repeat our exercise with the black and white image, decreasing the requested JPEG quality from 100% down to 1%.

JPEG 100% Quality = 60,231 characters

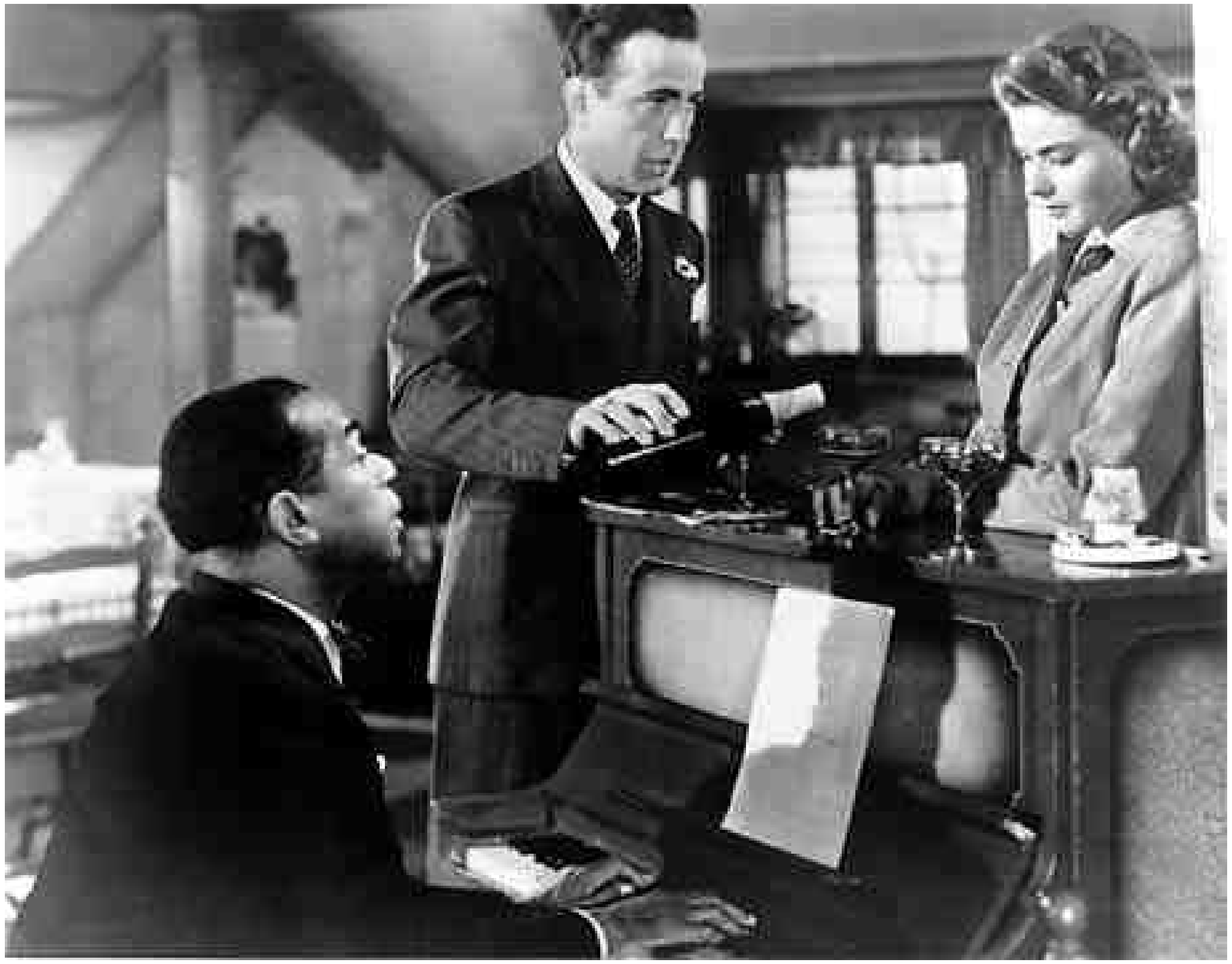
JPEG 75% Quality = 26,878 characters



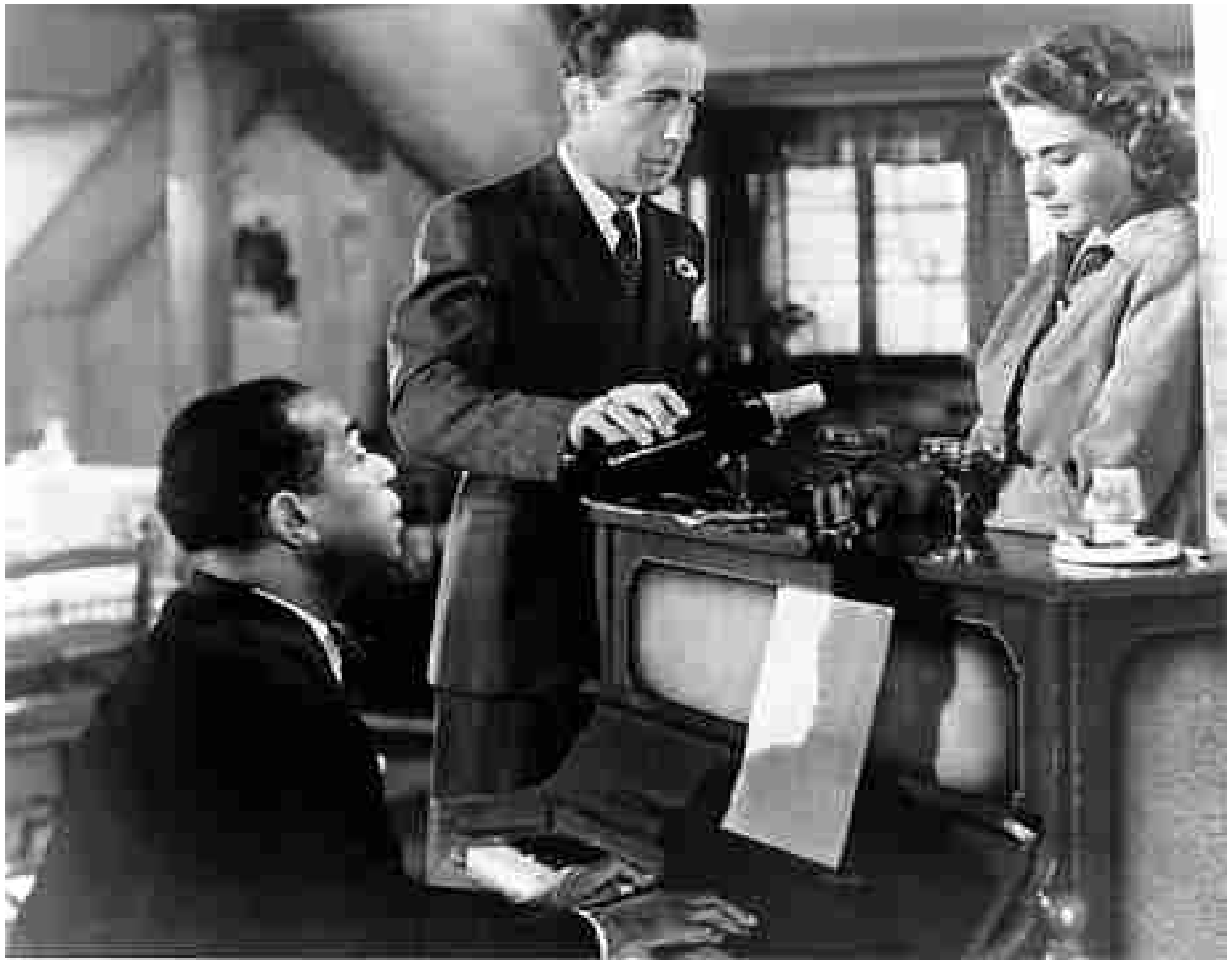
JPEG 50% Quality = 21,045 characters



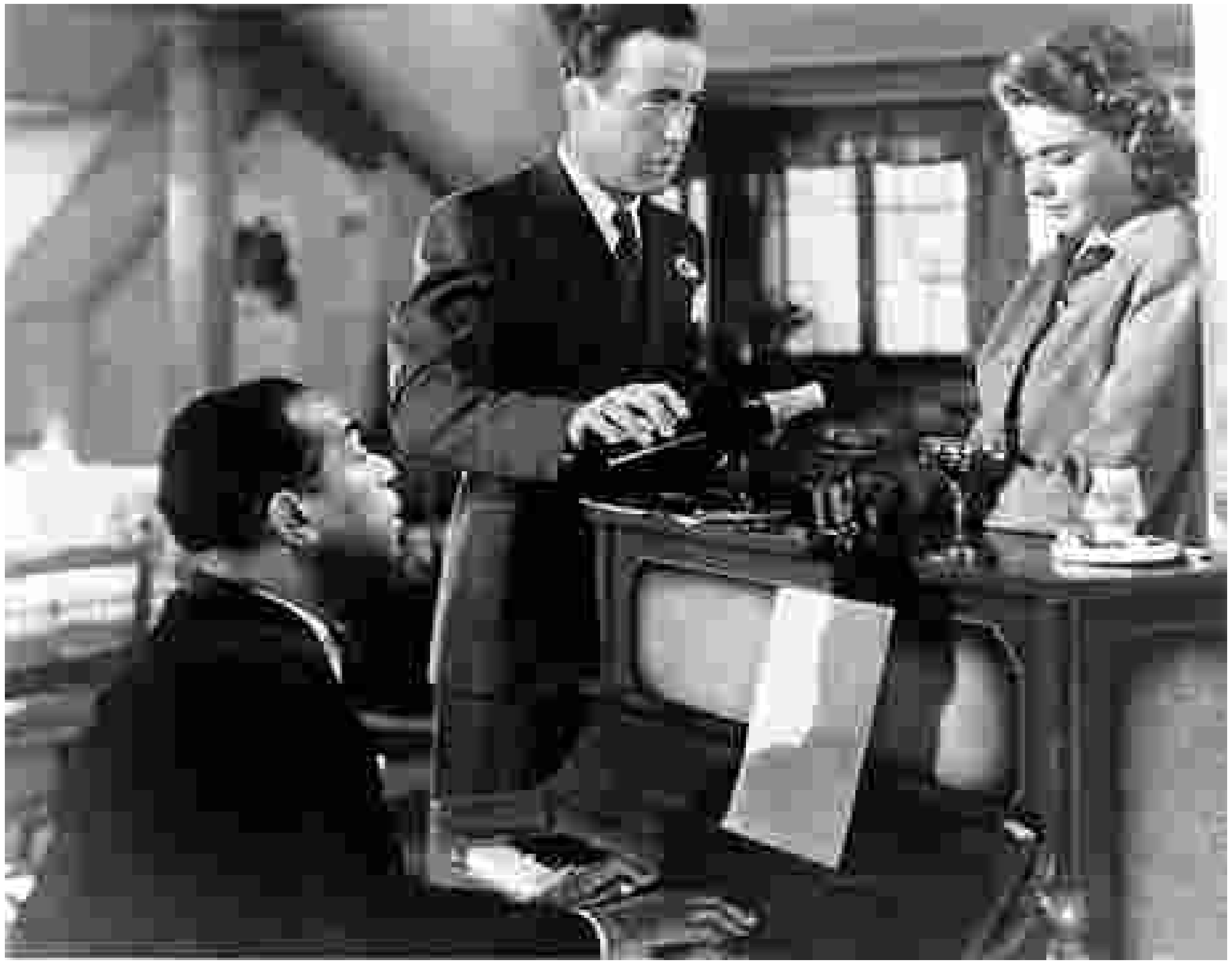
JPEG 25% Quality = 12,520 characters



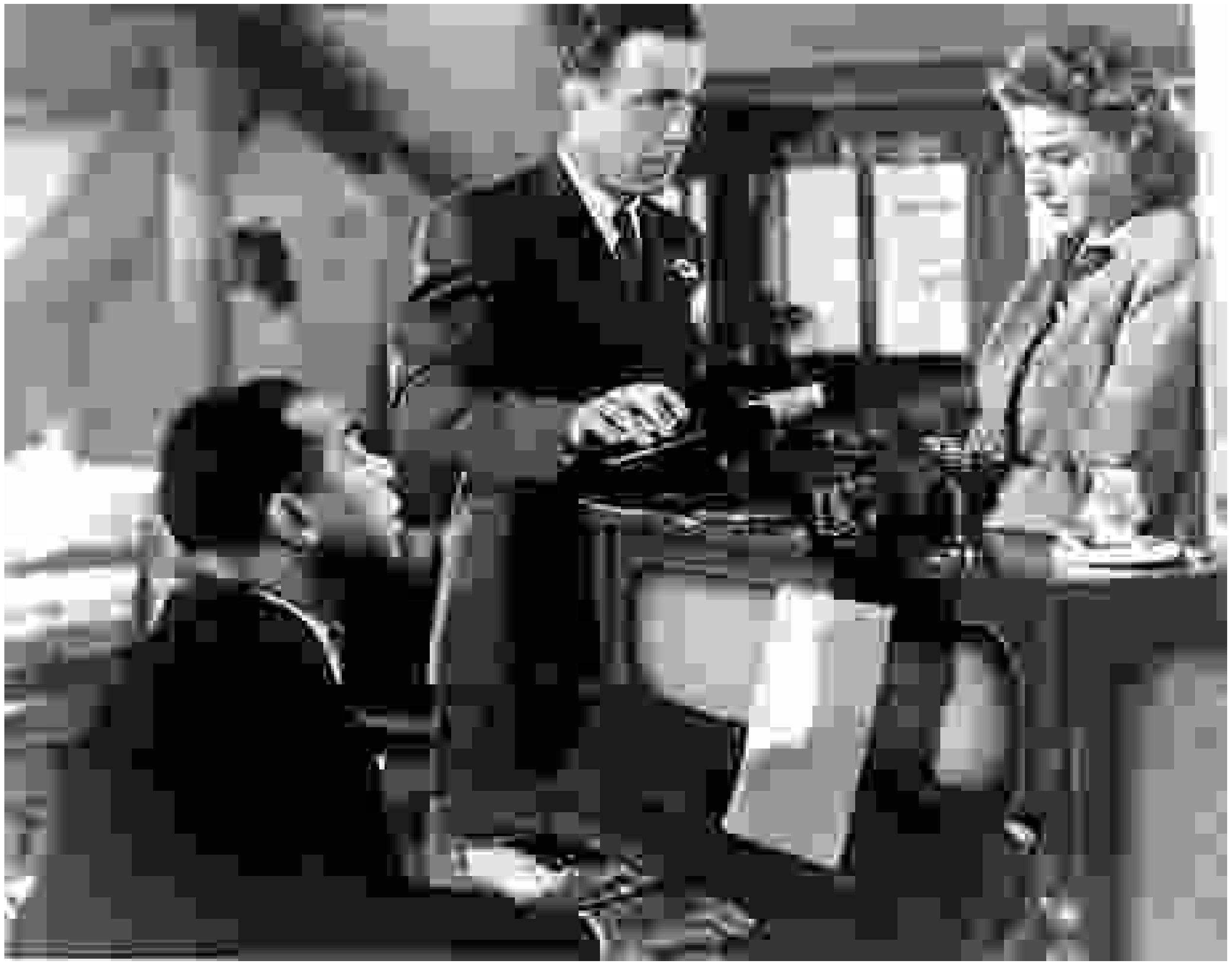
JPEG 16% Quality = 9,482 characters



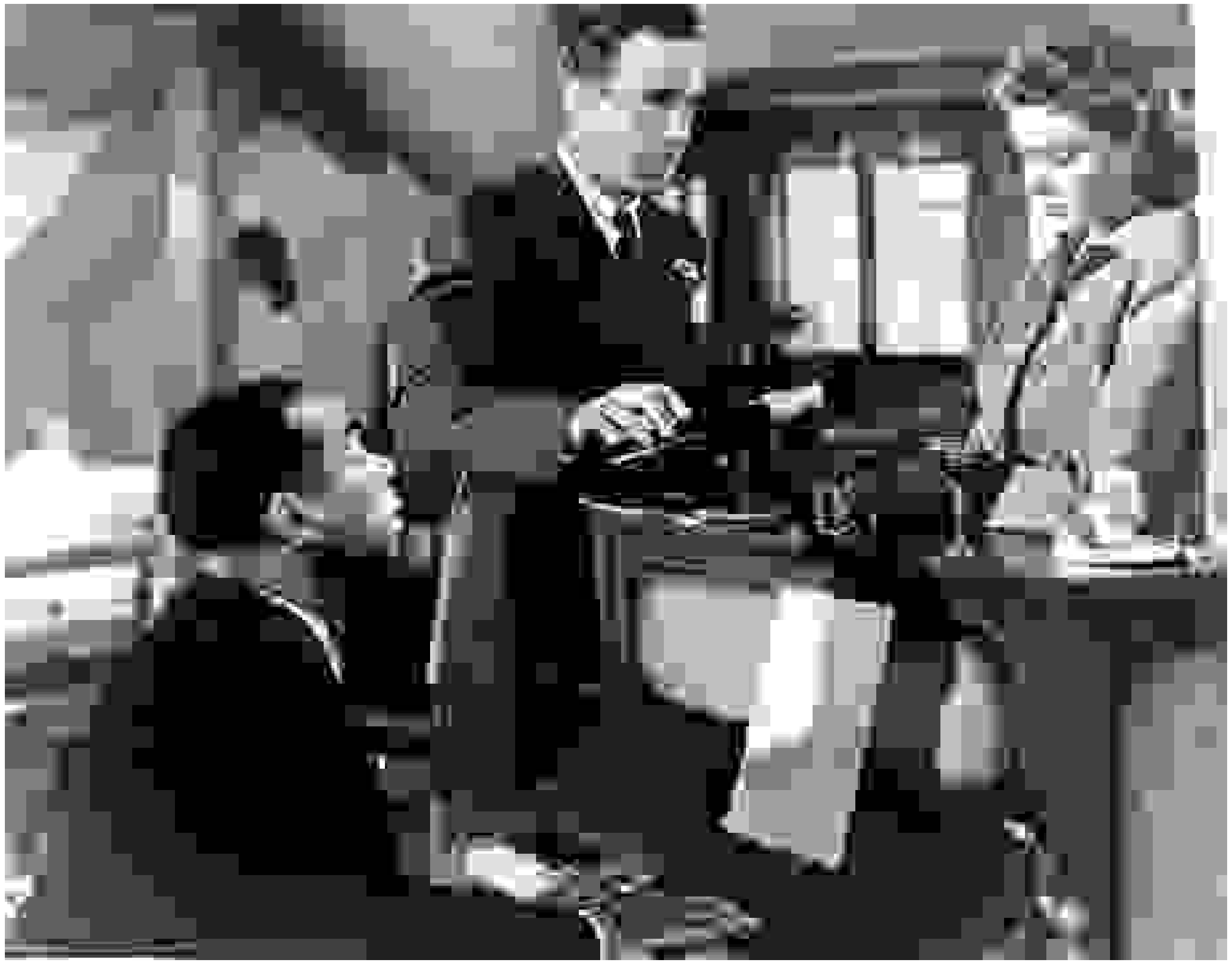
JPEG 8% Quality = 5,711 characters



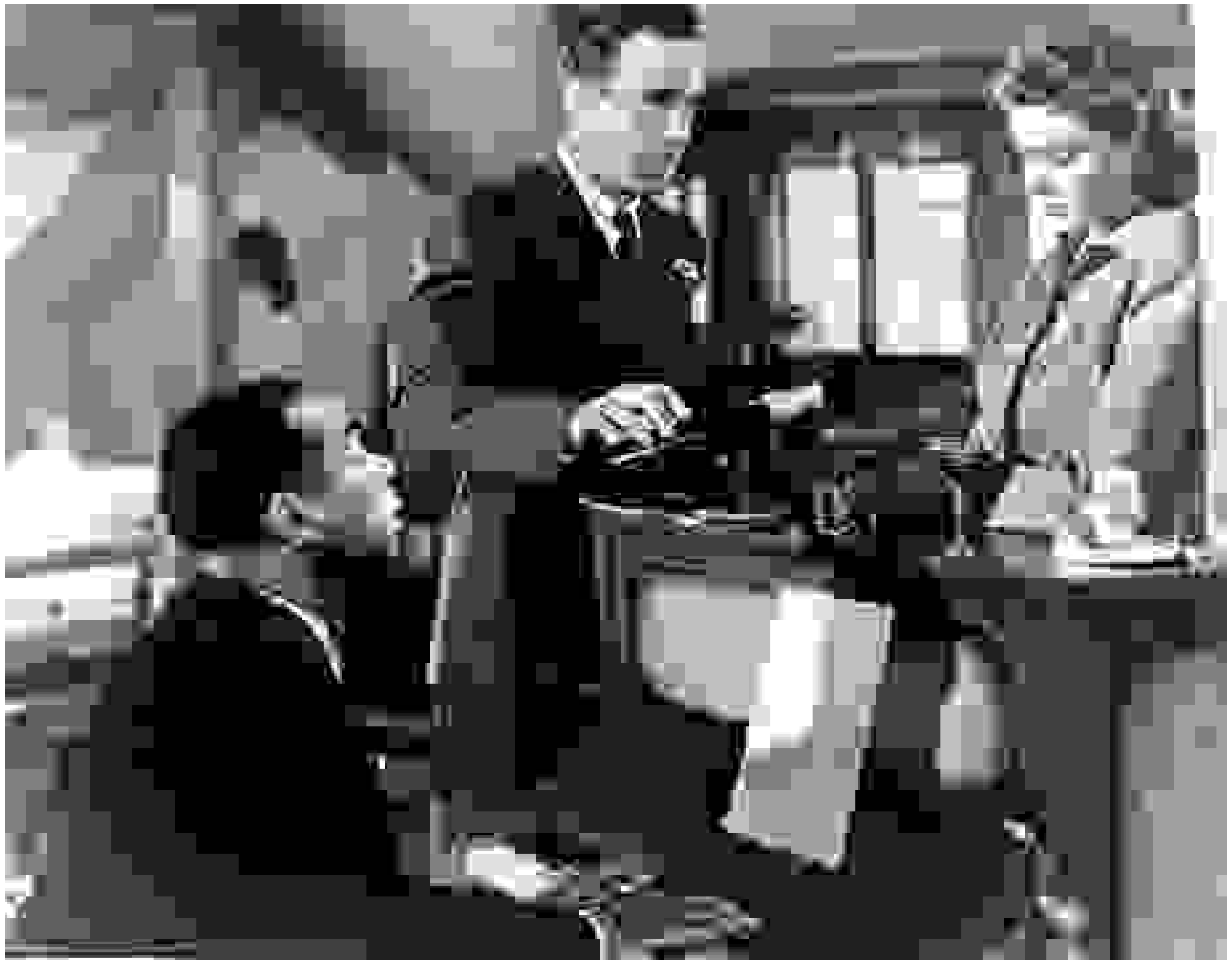
JPEG 4% Quality = 3,229 characters



JPEG 2% Quality = 2,265 characters



JPEG 1% Quality = 1,261 characters



Using JPEG, we started with our file at 100% quality and 60,231 characters and reduced it to 8% quality and 5,711 characters, for a compression factor of 12, without much noticeable loss of information.

Even going to 1% quality, 1,261 characters and a compression factor of 50, the picture is recognizable.

What is perhaps more surprising is what happens when we try to show all 9 versions of the image together. Because we reduce the size of each image, the 1% image seems about as good as the 100% image. This is because, as we said before, the eye is not good at seeing small details. When you reduce an image, only the large details remain, and we can not longer notice that the small details in the 100% picture are sharp, while the small details in the 1% picture are very blocky.