Harmaasävykuvien häviötön tiivistäminen

Kaisa Komulainen

9. huhtikuuta 2001

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Digitaalisessa muodossa olevien kuvien määrä kasvaa jatkuvasti ja kuvien tilantarve on suuri. Näin ollen on löydettävä tehokkaita menetelmiä, joilla kuvia voidaan tiivistää. Kuvien tiivistäminen voidaan jakaa häviölliseen (lossy) ja häviöttömään (lossless) tiivistämiseen. Tiivistämisprosessi jaetaan yleensä kahteen erilliseen osaan: mallintamiseen ja koodaukseen.

Tässä tutkielmassa luodaan katsaus kolmeen erilaiseen mallintamismenetelmään: tilastolliseen, ennustavaan ja kontekstimallintamiseen, sekä kolmeen koodausmenetelmään: Huffman-, Golomb-Rice - ja aritmeettiseen koodaukseen. Tutkielmassa esitellään lisäksi kolme häviötöntä tiivistysstandardia: häviötön JPEG, FELICS ja JPEG-LS, ja vertaillaan kokeellisesti niiden tiivistystehoja.

Avainsanat: häviötön kuvantiivistys, tilastollinen mallintaminen, kontekstimallintaminen, ennustava mallintaminen, Huffman-koodaus, aritmeettinen koodaus, Golomb-Rice -koodaus, häviötön JPEG, FELICS, JPEG-LS.

Sisältö

1 Johdanto

Kuvien tiivistäminen on tärkeää, koska digitaalisessa muodossa olevien kuvien määrä kasvaa jatkuvasti ja kuvien tilantarve on suuri. Kuvien tiivistäminen voidaan jakaa *häviölliseen* (lossy) ja *häviöttömään* (lossless) tiivistämiseen. Häviötön tiivistäminen säilyttää kuvan muuttumattomana, kun taas häviöllinen tiivistäminen hävittää tietoa. Tiivistämisprosessi jaetaan yleensä kahteen erilliseen osaan: *mallintamiseen* (modeling) ja *koodaukseen* (coding). Häviöttömässä tiivistyksessä mallintaminen on oleellinen osa tiivistysalgoritmia. Koodaamista pidetään näistä helpompana tehtävänä ja useita hyviä menetelmiä onkin jo olemassa kuten *Huffman-koodaus* (Huffman, 1952) ja *aritmeettinen koodaus* (Rissanen ja Langdon, 1979). Tässä tutkielmassa tarkastellaan kolmea häviötöntä tiivistysstandardia, jotka ovat häviötön JPEG (Pennebaker ja Mitchell, 1993), FELICS (Howard ja Vitter, 1993) ja JPEG-LS (Weinberger et al., 2000).

Jotta tutkielmassa käsiteltävien tiivistämismenetelmien ymmärtäminen olisi helpompaa, käydään tässä luvussa läpi tärkeimpiä aiheeseen liittyviä käsitteitä.

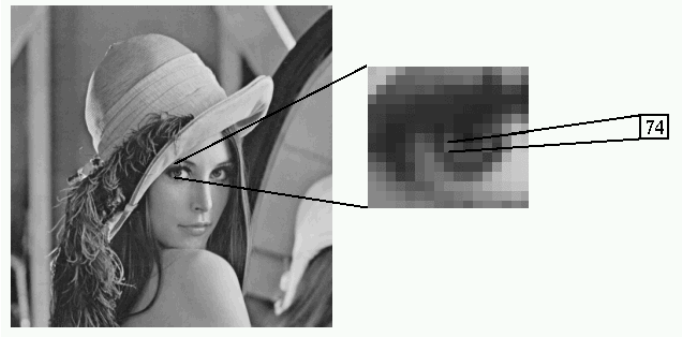
bitti: Tietokone käsittelee kuvan sisältämän tiedon binäärilukuina 0 ja 1.

bpp: Lyhenne sanoista bittiä per pikseli. Pikselin bittimäärä k on suoraan verrannollinen kuvan värimäärään. Bittien ja värimäärän suhde saadaan kaavasta $2^k = n$, missä n on värien määrä. Näin ollen esimerkiksi kuvan, jossa on vain mustia ja valkoisia pikseleitä (binäärikuva), esittämiseen riittää yksi bitti per pikseli.

pikseli: Kuva koostuu pisteistä, joita kutsutaan pikseleiksi. Pikselit esitetään tietyllä määrällä bittejä.

tiivistyssuhde: Tiivistyksen tehoa voidaan kuvata tiivistyssuhteella, joka saadaan kaavasta $tiivistyssuhde = \frac{\textit{tiivistetyn kuvan koko}}{\textit{alkuperäisen kuvan koko}}$.

Digitaalinen *bittikarttakuva* muodostuu yksittäisistä, eri väriarvoja saavista neliönmuotoisista pikseleistä. Kuvatiedoston koko riippuu kuvan koosta, resoluutiosta ja bittimäärästä, jolla kukin pikseli esitetään (Pesonen, 1998). Kuva 1 havainnollistaa pikselin osuutta kuvassa. Taulukossa 1 on esimerkkejä erilaisista kuvatyypeistä ja siitä, kuinka bittien määrä vaikuttaa värimäärään.



Kuva 1: Harmaasävykuva ja yksi sen pikseleistä.

Taulukko 1: Bittimäärän vaikutus värimäärään.

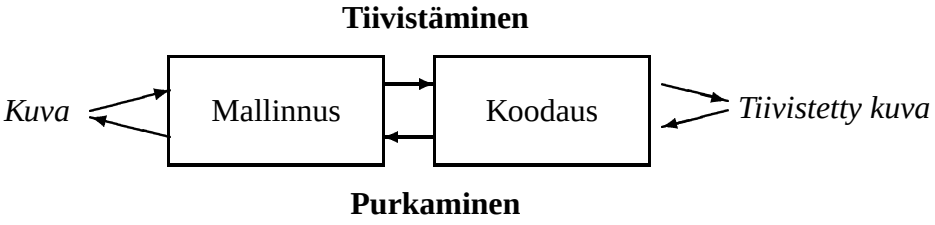
bpp	Kuvan tyyppi	Värien määrä
1	mustavalkoinen	$2^1 = 2$
4	harmaasävy tai väri	$2^4 = 16$
8	harmaasävy tai väri	$2^8 = 256$
16	harmaasävy tai väri	2^{16} =65 536
24	väri	2^{24} =16 777 216

Tässä tutkielmassa käsitellään vain harmaasävykuvien tiivistämistä. Harmaasävykuville tarkoitettuja tiivistämismenetelmiä voidaan kuitenkin soveltaa esimerkiksi RGB-kuville (R=red, G=green ja B=blue) käsittelemällä kutakin kolmesta väritasosta normaalina harmaasävykuvana.

Luvussa 2 esitellään erilaisia mallintamis- ja koodausmenetelmiä. Luku 3 käsittelee häviötöntä JPEG:iä, luku 4 FELICS:iä ja luku 5 JPEG-LS:ää. Luvussa 6 vertaillaan kokeellisesti häviöttömän JPEG:n, FELICSin ja JPEG-LS:n tiivistystehoja.

2 Häviöttömän tiivistämisen menetelmiä

Häviötön kuvantiivistys jaetaan yleensä kahteen erilliseen osaan: mallintamiseen ja koodaukseen (kuva 2).



Kuva 2: Kuvan tiivistäminen ja purkaminen.

Mallintamisen tarkoituksena on muodostaa kuvasta koodaajalle malli, jota käyttäen kuva voidaan koodata mahdollisimman tiiviisti. Tiivistämisen suurin ongelma onkin löytää mahdollisimman hyvä mallintamismenetelmä.

Jotta kuvan tiivistäminen olisi häviötöntä, on puretun kuvan oltava täsmälleen alkuperäisen kuvan mukainen. Näin ollen tiivistysvaiheessa koodaajalle menevää kuvaa muodostettaessa voidaan käyttää vain niitä pikseleitä, jotka ovat tiedossa myös kuvaa purettaessa. Toisin sanoen, jos kuvaa käydään läpi vasemmalta oikealle ja ylhäältä alas, voidaan ennusteena tai kontekstissa käyttää käsiteltävän pikselin vasemmalla ja yläpuolella olevia pikseleitä.

Tässä työssä käsiteltävät menetelmät koodaavat kuvaa pikseli kerrallaan, ja kuvaa käydään läpi riveittäin alkaen kuvan vasemmasta yläreunasta.

2.1 Tilastollinen mallintaminen

Tilastollisen mallintamisen tarkoituksena on ennustaa koodattavia symboleja niiden todennäköisyysjakauman avulla. Kaikkien symbolien joukkoa kutsutaan *aakkostoksi*. Esimerkiksi binäärikuvan aakkosto koostuu kahdesta symbolista: mustaa ja valkoista pikseliä kuvaavista symboleista. Symbolin sisältämän informaation määrää kuvaa sen *entropia*

$$H(x) = -\log_2 p(x) \tag{1}$$

missä x on symboli ja $p(x)$ sen esiintymistodennäköisyys. Mitä suurempi todennäköisyys on, sitä pienempi on entropia. Entropian ollessa pieni tulisi symbolia kuvaavan koodisanan olla lyhyt. Mallin kokonaisentropia saadaan laskemalla yksittäisten symbolien entropioiden odotusarvo kaavan

$$H = -\sum_{x=1}^{n} p(x) \cdot \log_2 p(x) \tag{2}$$

avulla, missä n on aakkoston koko. Jos symbolit esiintyvät tiivistetyssä tiedostossa mallin mukaisilla todennäköisyyksillä, on H tiivistyksen alaraja yksiköllä bittiä per symboli (Shannon, 1948). Entropia määrää symbolin x koodaukseen tarvittavan optimaalisen bittimäärän.

Tilastollinen mallintaminen voidaan jakaa kolmeen ryhmään: *staattinen* (static), *semi-adaptiivinen* (semi-adaptive) ja *adaptiivinen* (adaptive) mallintaminen (Fränti, 1999).

Staattinen malli muodostetaan aakkoston oletetun todennäköisyysjakauman mukaan, jolloin se on sama kaikille syötetiedostoille. Ongelmana on todellisen syötteen ja mallin ero. Esimerkiksi suomen kielessä kirjain *c* esiintyy harvoin, jolloin sitä kuvaa pitkä koodisana. Suomen kielelle tarkoitettu malli ei kuitenkaan sovellu englanninkieliseen tekstiin, jossa *c*:n esiintymistodennäköisyys on suurempi. Staattisen mallin etuna on se, että mallia ei tarvitse välittää dekoodaajalle, ja että syötettä ei tarvitse käydä läpi kuin kerran.

Aakkoston $\{a, b, c\}$ staattinen malli voisi olla esimerkiksi taulukon 2 mukainen. Kuten taulukosta nähdään, on entropia kääntäen verrannollinen symbolin esiintymistodennäköisyyteen.

Taulukko 2: Esimerkki staattisesta mallista.

symboli	$p(x)$	$H(x)$
a	0.75	0.42
b	0.05	4.32
c	0.20	2.32

Semi-adaptiivisessa mallissa käydään syöte ensin läpi ja lasketaan symbolien esiintymistodennäköisyydet. Koska todennäköisyysjakaumat lasketaan kullekin syötteelle erikseen, saadaan tätä mallia käyttämällä parempia tuloksia kuin staattista mallia käyttämällä. Haittapuolena on kuitenkin se, että syöte joudutaan käymään läpi kahteen kertaan, ja että tiivistetyn tiedoston lisäksi joudutaan tallettamaan myös käytetty malli. Taulukossa 3 on esimerkki semi-adaptiivisesta mallista, kun sitä sovelletaan syötteelle *abbacbc*.

Taulukko 3: Esimerkki semi-adaptiivisesta mallista. $f(x)$ on symbolin x frekvenssi, $p(x)$ sen todennäköisyys ja $H(x)$ entropia.

symboli	$f(x)$	$p(x)$	$H(x)$
a	2	0.29	1.81
b	3	0.42	1.22
c	2	0.29	1.81

Adaptiivinen (dynaaminen) malli on astetta kehittyneempi kuin edelliset mallit. Syötteen läpikäynti aloitetaan käyttämällä alkumallia, joka on jonkinlainen oletettu todennäköisyysjakauma. Alkumallin valinnalla ei yleensä ole suurta merkitystä, vaan oleellista on, että malli mukautuu syötteen todelliseen todennäköisyysjakaumaan koodauksen edetessä. Voidaan esimerkiksi olettaa, että jokainen symboli on yhtä todennäköinen. Kukin symboli koodataan jo olemassa olevalla mallilla, minkä jälkeen mallia päivitetään koodatun symbolin mukaan. Pienillä syötteillä adaptiivinen malli ei ole tehokas, koska se ei ehdi mukautua syötetietoon. Etuna on, ettei mallia tarvitse tallettaa ja syöte tarvitsee käydä läpi vain kerran.

Sovelletaan adaptiivista mallia syötteelle *abbacbc*. Oletetaan aluksi, että jokainen symboli a, b ja c esiintyy samalla todennäköisyydellä 0.33. Oletetaan lisäksi, että jokainen kirjainsymboli on esiintynyt kerran, jolloin niiden frekvenssit voidaan alustaa ykkösiksi. Kun syötteen ensimmäinen symboli, a, on luettu, on a:n frekvenssi 2 ja b:n ja c:n 1. Näin ollen symbolin a todennäköisyys on nyt 0.5 ja symbolien b ja c todennäköisyydet 0.25. Seuraavaksi luetaan b. Nyt sekä symbolien a että b frekvenssi on 2. Näin jatketaan, kunnes koko syöte on käyty läpi. Loput arvot ovat taulukossa 4.

Taulukko 4: Esimerkki dynaamisen mallin käytöstä. Ensimmäisellä rivillä on syöte ja ensimmäisessä sarakkeessa aakkoston symbolit. Kutakin syötteen symbolia vastaavassa sarakkeessa on kunkin aakkoston symbolin senhetkinen frekvenssi ja todennäköisyys.

			a		b		b		a		c		b		c	
a	1	0.33	**2**	0.50	2	0.40	2	0.33	**3**	0.43	3	0.38	3	0.33	3	0.30
b	1	0.33	1	0.25	**2**	0.40	**3**	0.50	3	0.43	3	0.38	**4**	0.44	4	0.40
c	1	0.33	1	0.25	1	0.20	1	0.17	1	0.14	**2**	0.25	2	0.23	**3**	0.30

Tilastollinen mallintaminen ottaa huomioon vain syöteaakkoston todennäköisyysjakauman, eikä käytä hyväkseen syötteen muita ominaisuuksia kuten konteksti- ja ennustava malli.

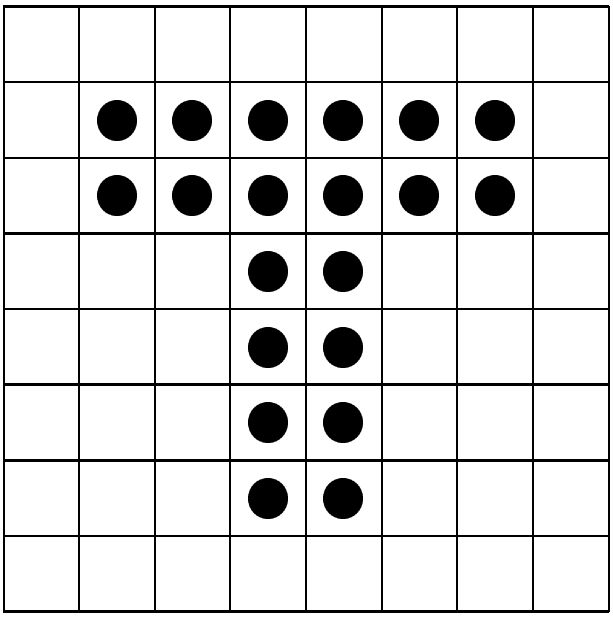
2.2 Kontekstimalli

Konteksti (context) tarkoittaa *asiayhteyttä*. Kuvissa vierekkäiset pikselit ovat tyypillisesti riippuvaisia toisistaan, jolloin tietyn pikselin arvoa voidaan mallintaa tutkimalla sen naapuripikseleitä. *Kontekstimallissa* (context model) hyödynnetään pikseleiden välisiä riippuvuussuhteita (Tischer et al., 1993).

Kuvan läpikäynti tapahtuu jossain ennaltamäärätyssä järjestyksessä, esimerkiksi rivi kerrallaan vasemmalta oikealle ja ylhäältä alas. Kontekstina käytetään jo koodattuja pikseleitä, jolloin konteksti muodostuu tässä tapauksessa käsiteltävän pikselin vasemmalla ja yläpuolella olevista pik-

seleistä tai niiden yhdistelmistä. Kontekstien lukumäärä on kontekstiin kuuluvien pikseleiden arvojen kombinaatioiden lukumäärä.

Muodostetaan kuvalle 3 todennäköisyysmalli käyttäen kontekstimallia, jossa kontekstina on kolme lähintä naapuripikseliä jo koodattujen pikseleiden joukosta. Näin saadaan kahdeksan erilaista kontekstia (taulukko 5).



Kuva 3: Binäärikuva, jonka koko on 64 bittiä.

Reunapikseleitä koodattaessa voi käydä niin, että kontekstiin kuuluva pikseli onkin kuvan ulkopuolella. Sen vuoksi kuvan ulkopuolella olevien pikseleiden oletetaan olevan valkoisia. Esimerkiksi kuvan 3 ylärivissä olevilla valkoisilla pikseleillä on kontekstina vain valkoisia pikseleitä, samoin ensimmäisellä käsiteltävällä mustalla pikselillä. Tässä kontekstissa valkoisten pikseleiden lukumäärä on kuvassa 31 ja mustien 1. Tästä saadaan todennäköisyysjakauma, jossa valkoisen pikselin todennäköisyys on $p = 31/32 = 0.97$ ja mustan $p = 1/32 = 0.03$. Kuvassa 4 on kuvan 3 pikseleiden kontekstit. Kontekstien numerot tulevat taulukosta 5 ja alaindeksi *v* tarkoittaa, että kyseessä on valkoinen pikseli ja alaindeksi *m*, että kyseessä on musta pikseli.

1_v	1_v	1_v	1_v	1_v	1_v	1_v	1_v
1_v	1_m	3_m	3_m	3_m	3_m	3_m	3_v
1_v	1_m	8_m	8_m	8_m	8_m	8_m	6_v
1_v	2_v	5_v	5_m	8_m	8_v	5_v	4_v
1_v	1_v	1_v	2_m	8_m	6_v	1_v	1_v
1_v	1_v	1_v	2_m	8_m	6_v	1_v	1_v
1_v	1_v	1_v	2_m	8_m	6_v	1_v	1_v
1_v	1_v	1_v	2_v	5_v	4_v	1_v	1_v

Kuva 4: Kuvan 3 pikseleiden kontekstit.

Pikseleiden entropiat saadaan kaavasta 1. Edellä mainitun valkoisen pikselin entropia on täten $H = -\log_2 0.97 = 0.05$ ja mustan $H = -\log_2 0.03 = 5.01$. Kontekstit, niitä vastaavien pikseleiden frekvenssit, todennäköisyydet ja entropiat ovat taulukossa 5.

Mallin kokonaisentropia lasketaan kaavan

$$H_L = -\sum_{j=1}^{L} p(c_j) \cdot [\sum_{i=1}^{n} p(x_i|c_j) \cdot \log_2(p(x_i|c_j))] \tag{3}$$

mukaan, missä L on kontekstien lukumäärä, n aakkoston koko (tässä tapauksessa värien lukumäärä) ja c konteksti. $p(c_j)$ on todennäköisyys, jolla kontekstia c_j käytetään ja $p(x_i|c_j)$ todennäköisyys, jolla symboli x_i esiintyy kontekstissa c_j. Esimerkkitapauksessa $L = 8$ (taulukko 5) ja $n = 2$. Esimerkiksi ensimmäisen kontekstin (kaksi valkoista naapuripikseliä) mallin entropia on $-(31/32 \cdot \log_2(31/32) + 1/32 \cdot \log_2(1/32)) = 0.2$. Mallia käytetään 32/64 tapauksista, joten tämän mallin suhteellinen entropia on silloin $32/64 \cdot 0.2 = 0.1$. Laskemalla vastaavasti muiden mallien entropiat, saadaan kokonaisentropiaksi $H_L = 0.37$ *bpp*. Tällaista kontekstimallia käy-

Taulukko 5: Kuvalle 3 muodostettu kontekstimalli.

nro	Konteksti	Pikseli	$f(x)$	$p(x)$	$H(x)$
1		○ ●	31 1	0.97 0.03	0.05 5.01
2		○ ●	2 4	0.33 0.67	1.60 0.60
3		○ ●	1 5	0.17 0.83	2.64 0.27
4		○ ●	2 0	1.00 0.00	0.00 ∞
5		○ ●	3 1	0.75 0.25	0.42 2.00
6		○ ●	4 0	1.00 0.00	0.00 ∞
7		○ ●	0 0	0.00 0.00	∞ ∞
8		○ ●	1 9	0.10 0.90	3.32 0.15

tettäessä kuva vie $64 \cdot 0.37 = 23.68$ bittiä, mikä on huomattavasti vähemmän kuin alkuperäisen kuvan viemä 64 bittiä.

Kontekstin koko on kompromissi ennusteen tarkkuuden ja tehokkuuden välillä. Mitä suurempi konteksti valitaan, sitä parempana ennusteena se toimii. Toisaalta suuri konteksti hidastaa mallin mukautumista syötteeseen adaptiivisessa mallissa, jolloin koodaustehokkuus on alkuvaiheessa huono. Semi-adaptiivisen mallin koko puolestaan kasvaa kontekstin koon kasvaessa. Hyvään tiivistystulokseen pääsemiseksi on löydettävä kuhunkin tilanteeseen sopiva kontekstin koko.

Kontekstien lukumäärä saadaan kaavasta

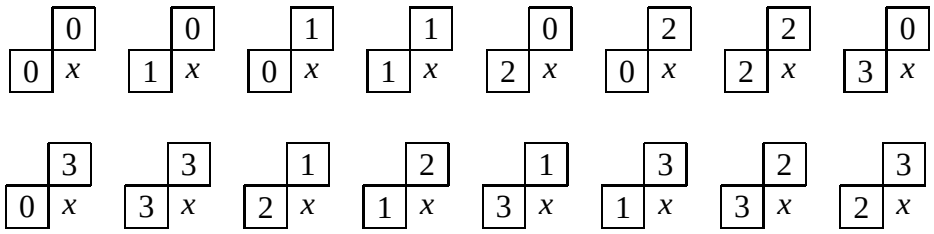
$$L = N^s \tag{4}$$

missä N on syötteen symbolien lukumäärä ja s kontekstin koko. Taulukossa 6 on esimerkkejä siitä, kuinka symbolien määrä ja kontekstin koko vaikuttavat kontekstien lukumäärään.

Taulukko 6: Symbolien lukumäärän ja kontekstin koon vaikutus kontekstien lukumäärään.

Symbolien lkm.	Kontekstin koko	Kontekstien lkm.
2	2	$2^2 = 4$
2	4	$2^4 = 16$
2	10	$2^{10} = 1024$
16	2	$16^2 = 256$
16	4	$16^4 = 65536$
256	2	$256^2 = 65536$

Taulukkoa 6 tarkasteltaessa huomataan, että värimäärän kasvaessa kontekstien lukumäärä kasvaa räjähdysmäisesti, vaikka kontekstiin kuuluisikin vain kaksi naapuripikseliä. Esimerkiksi 256 harmaasävyn kuvalle kahden pikselin kokoinen konteksti tuottaa 65536 erilaista kontekstia. Taulukossa 5 on binäärikuvan konteksit, kun kontekstiin kuuluu kolme naapuripikseliä. Kuvassa 5 on nelisymbolisen aakkoston kontekstit, kun kontekstina on käsiteltävän pikselin vasemmalla ja yläpuolella olevat pikselit.



Kuva 5: Nelisymbolisen aakkoston konteksit, kun kontekstina on kaksi naapuripikseliä.

Kontekstimalli sopii hyvin binäärikuville, sillä niiden arvojakauma on suppea. Tällöin kontekstien lukumäärä pysyy kohtuullisena. Suurempien aakkostojen tapauksessa kontekstien lu-

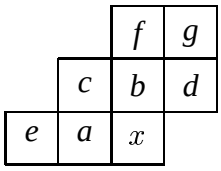
kumäärää voidaan rajoittaa kvantisoimalla, kuten esimerkiksi JPEG-LS -menetelmässä (Weinberger et al., 2000).

2.3 Ennustava mallintaminen

Ennustava mallintaminen (predictive modeling) on yksinkertainen ja tehokas menetelmä, jossa pikselin arvo pyritään ennustamaan sitä ympäröivien pikseleiden arvojen perusteella. Ennustava mallintaminen jakautuu kolmeen osaan: ennustajan valintaan, ennustevirheen laskemiseen ja ennustevirheiden koodaukseen (Habibi, 1971).

2.3.1 Ennustajan valinta

Jos kuvan läpikäynti tapahtuu vasemmalta oikealle ja ylhäältä alas, voidaan käsiteltävän pikselin vasemmalla ja yläpuolella olevia pikseleitä käyttää ennusteena. Ennustaja voi yksinkertaisimmillaan olla jokin käsiteltävän pikselin naapuripikseleistä. Häviöttömässä JPEG:ssä ja JPEG-LS:ssä ennustajana käytetään erilaisia naapuripikseleiden arvojen lineaarisia funktioita. Tässä tutkielmassa naapuripikseleihin viitataan kuvan 6 mukaisesti.



Kuva 6: Käsiteltävän pikselin x naapuripikseleihin viittaaminen.

Ennustajat voivat olla samoja kaikille kuville (*global prediction*) tai vaihdella kuvan mukaan (*local prediction*). Jos kuvan eri alueiden ominaisuudet halutaan ottaa huomioon, voidaan ennustajaa muuttaa jopa kuvan läpikäynnin aikana (*adaptive prediction*) (Rabbani ja Jones, 1991).

Adaptiivinen ennustaminen

Ennustajat ovat yleensä käsiteltävän pikselin naapuripikseleiden arvojen lineaarisia funktioita. Kuvissa on kuitenkin usein epälineaarisia kohtia kuten reunat. Eräs keino tällaisten kohtien mallintamiseksi on käyttää erilaisia lineaarisia ennustajia sen mukaan, minkälaista kohtaa kuvassa ollaan käsittelemässä. Tätä kutsutaan adaptiiviseksi ennustamiseksi.

Ennustaja voidaan valita joko jo käsiteltyjen pikseleiden perusteella (*backward adaption*) tai vielä käsittelemättömien pikseleiden perusteella (*forward adaption*), jolloin käytetty ennustaja on välitettävä erikseen dekoodaajalle. Esimerkkinä edellisestä menetelmästä on JPEG-LS -standardin käyttämä ennustaja (kaava 8), joka esitellään luvussa 5.1.

Toinen menetelmä, joka käyttää jo käsiteltyjä pikseleitä ennustajan valinnassa on Wun (1996) kehittämä gradienttiin perustuva ennustaja (*Gradient-Adjusted Predictor, GAP*). Menetelmässä käytetään paikallisten vaaka- ja pystysuuntaisten gradienttien arvoja, joka saadaan kaavasta

$$\begin{aligned} d_h &= |a - e| + |b - c| + |d - b| \\ d_v &= |a - c| + |b - f| + |d - g| \end{aligned} \tag{5}$$

missä d_h on vaakasuora gradientti ja d_v pystysuora gradientti.

Arvojen d_h ja d_v avulla saadaan tietoa käsiteltävän pikselin läheisyydessä olevien mahdollisten reunojen jyrkkyyksistä ja suunnista. Gradienttiin perustuva ennustaja saadaan laskettua kuvan 7 algoritmilla, missä ennustajaa merkitään symbolilla $\hat{x}$.

```
if  d_v − d_h > 80
    x̂ = a
else if  d_v − d_h < −80
    x̂ = b
else
    x̂ = (a + b)/2 + (d − c)/4
    if  (d_v − d_h) > 32
        x̂ = (x̂ + a)/2
    else if  d_v − d_h > 8
        x̂ = (3x̂ + a)/4
    else if  d_v − d_h < −32
        x̂ = (x̂ + b)/2
    else if  d_v − d_h < −8
        x̂ = (3x̂ + b)/4
```

Kuva 7: Gradienttiin perustuvan ennustajan (GAP) laskeminen (Wu, 1996).

Wun kehittämä GAP-menetelmä eroaa esimerkiksi JPEG-LS:stä siinä, että ennustajaa valittaessa otetaan reunan suunnan lisäksi huomioon sen jyrkkyys. Tämän tiedon pohjalta käsiteltävän pikselin naapureille voidaan antaa eri painot. GAP:n avulla päästäänkin hieman parempiin tuloksiin kuin häviöttömällä JPEG:llä tai JPEG-LS:llä. GAP vaatii kuitenkin enemmän laskentatehoa kun JPEG:n häviöttömät versiot (Wu, 1996).

Sovelletaan GAP-algoritmia kuvan 8 laatikossa olevalle pikselille. Vaakasuoran gradientin arvo on tässä tapauksessa $d_h = |185-195|+|179-192|+|133-179| = 69$ ja pystysuoran gradientin arvo $d_v = |185 - 192| + |179 - 185| + |133 - 150| = 30$. Koska erotus $d_v - d_h = -39$ ei ole suurempi kuin 80 eikä pienempi kuin -80, hypätään kuvan 7 algoritmissa suoraan `else`-haaraan. Ensimmäisen sijoituslauseen jälkeen ennustaja $\hat{x} = (185+179)/2+(133-192)/4 = 167$. Koska erotus $d_v - d_h = -39 < -32$, tulee ennustajan lopulliseksi arvoksi $\hat{x} = (167 + 179)/2 = 173$.

	185	150
192	179	133
195 185	182	

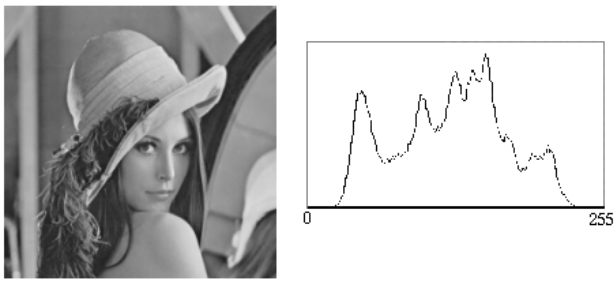
Kuva 8: Osa harmaasävykuvaa.

2.3.2 Ennustevirheen laskeminen

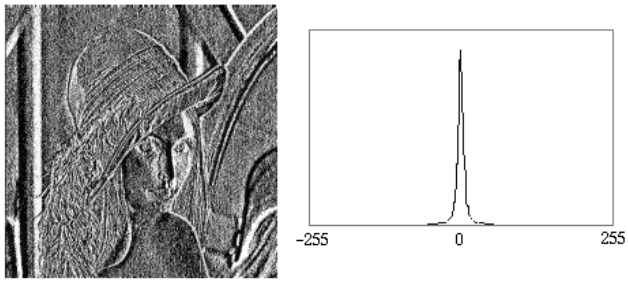
Ennustevirhe saadaan kaavasta

$$e = x - \hat{x} \tag{6}$$

missä x on käsiteltävä pikseli ja $\hat{x}$ ennustaja. Sen sijaan, että koodaajalle välitettäisiin pikseleiden arvot, välitetään ennustavassa mallintamisessa ennustevirheet. Vierekkäisten pikseleiden arvot ovat tyypillisesti lähellä toisiaan, jolloin suurin osa ennustevirheistä on lähellä nollaa. Ennustevirheiden jakauman entropia on näin ollen pieni verrattuna alkuperäisen kuvan pikseleiden arvojen entropiaan (Rabbani ja Jones, 1991). Kuvassa 9 on tyypillinen harmaasävykuva ja sen histogrammi. Kuvassa 10 on kuvan 9 harmaasävykuvasta saatu ennustevirhekuva ja sen histogrammi.

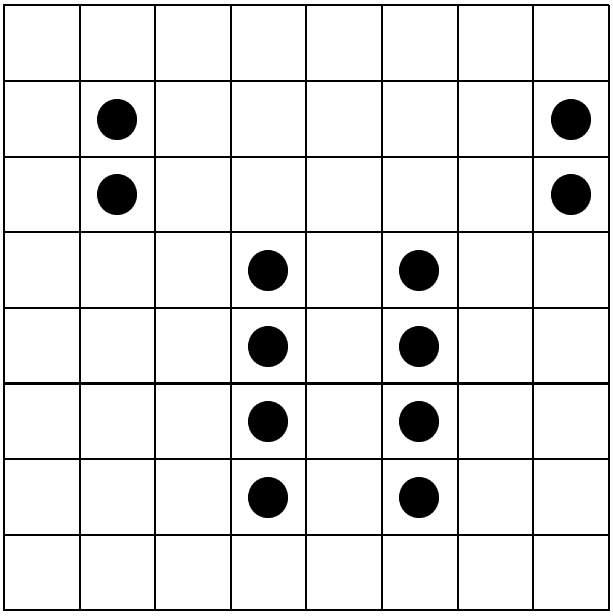


Kuva 9: Tyypillinen harmaasävykuva ja sen histogrammi.



Kuva 10: Ennustevirhekuva ja sen histogrammi.

Sovelletaan ennustavaa mallintamista kuvalle 3 käyttäen vasemmanpuoleista ennustajaa. Kaavan 6 mukaan ennustevirheet olisivat joko -1, 0 tai 1. Binäärikuvan tapauksessa riittää kuitenkin, että ennustevirhe on binäärinen, ts. ennustevirhekuvassa ennustevirheet ovat mustia pikseleitä ja loput valkoisia. Koska ennustajana on vasemmanpuoleinen pikseli, tulee ennustevirhe kohtaan, jossa käsiteltävän pikselin vasemmanpuoleinen pikseli on eri värinen. Oletetaan, että kuvan ulkopuolella olevat pikselit ovat valkoisia. Kuvassa 11 on kuvalle 3 lasketut ennustevirheet.



Kuva 11: Ennustevirheet kuvalle 3, kun ennustajana on vasemmanpuoleinen pikseli.

Ennustevirhekuvan kokonaisentropia on kaavan 2 mukaan 0.70 bpp, jolloin kuva vie 44.8 bittiä. Tässä tapauksessa kontekstimallin avulla päästiin parempaan tiivistystulokseen kuin ennustavaa mallia käytettäessä. Ennustava malli onkin tehokkaimmillaan, kun kuvan värisävymäärä on suuri, ja kun keskenään lähekkäin olevien pikseleiden arvot ovat lähellä toisiaan eli kuva ei sisällä jyrkkiä reunoja.

2.4 Ennustavan mallin ja kontekstimallin yhdistäminen

Perinteisissä tiivistysmenetelmissä, kuten häviöttömässä JPEG:ssä, ei kontekstimallia enää käytetä ennustavan mallin jälkeen. Voidaan olettaa, että sama ennustaja ja sama todennäköisyysjakauma toimivat kaikissa tapauksissa. Hyvien tulosten saavuttamiseksi tarvitaan kuitenkin parempia malleja. Esimerkiksi JPEG-LS yhdistää ennustavan mallin ja kontekstimallin (Weinberger et al.,2000).

Käytetään ennustavaa mallia ja kontekstimallia peräkkäin kuvalle 3. Kun ennusteena on vasemmanpuoleinen pikseli, on ennustevirhekuva kuvan 11 mukainen. Tässä ennustevirhekuvassa on viisi erilaista kontekstia (taulukko 7). Taulukossa 7 on kontekstien lisäksi kunkin kontekstin omaavien pikseleiden frekvenssit, todennäköisyydet ja entropiat. Kaavan 3 avulla saadaan kokonaisentropiaksi 0.463 bpp, jolloin kuva vie 29.6 bittiä. Pelkkää kontekstimallia käytettäessä kuva vie 23.68 bittiä, ja ennustavaa mallia käytettäessä 56.62 bittiä.

Kuten edellinen esimerkki osoittaa, ei ennustavan mallin ja kontekstimallin yhdistämisestä ole hyötyä binäärikuvia tiivistettäessä. Sen sijaan harmaasävykuville kontekstimallia voidaan käyttää ennustajan valinnassa tai ennustevirheen jakauman mallintamisessa kuten FELICS:ssä (Howard ja Vitter, 1993) ja JPEG-LS:ssä (Weinberger et al., 2000) on tehty.

Taulukko 7: Kuvalle 11 muodostettu kontekstimalli.

Konteksti	Pikseli	$f(x)$	$p(x)$	$H(x)$
	○	35	0.90	0.14
	●	4	0.10	0.33
	○	4	0.33	0.53
	●	8	0.67	0.39
	○	3	1.00	0.00
	●	0	0.00	∞
	○	3	1.00	0.00
	●	0	0.00	∞
	○	7	1.00	0.00
	●	0	0.00	∞

2.5 Koodaus

Koodauksessa syötteen symbolille annetaan koodisana siten, että yleisimmät symbolit saavat lyhyemmän koodin ja harvinaisimmat symbolit pidemmän. Näin koodisanan keskimääräinen pituus saadaan lyhyemmäksi kuin syötesymbolien alkuperäinen pituus. Olettaen, että käytetty malli on oikea, tulisi koodinpituuden olla mahdollisimman lähellä symbolin entropiaa (kaava 1). Tässä luvussa esitellään kolme koodausmenetelmää: *Huffman-koodaus* (Huffman, 1952), *Golomb-Rice -koodaus* (Golomb, 1966 ja Rice, 1979) ja *aritmeettinen koodaus* (Witten et al., 1987).

Huffman-koodaus

Huffman-koodi saadaan *Huffman-puusta*, joka luodaan symbolien todennäköisyysjakaumia hyväksi käyttäen. Puun rakentaminen aloitetaan muodostamalla n (aakkoston koko) erillistä puuta, jotka kukin koostuvat yhdestä solmusta (kuva 12a). Nämä solmut sisältävät symbolin ja sen esiintymistodennäköisyyden tai frekvenssin. Kussakin vaiheessa kaksi pienimmän todennäköi-

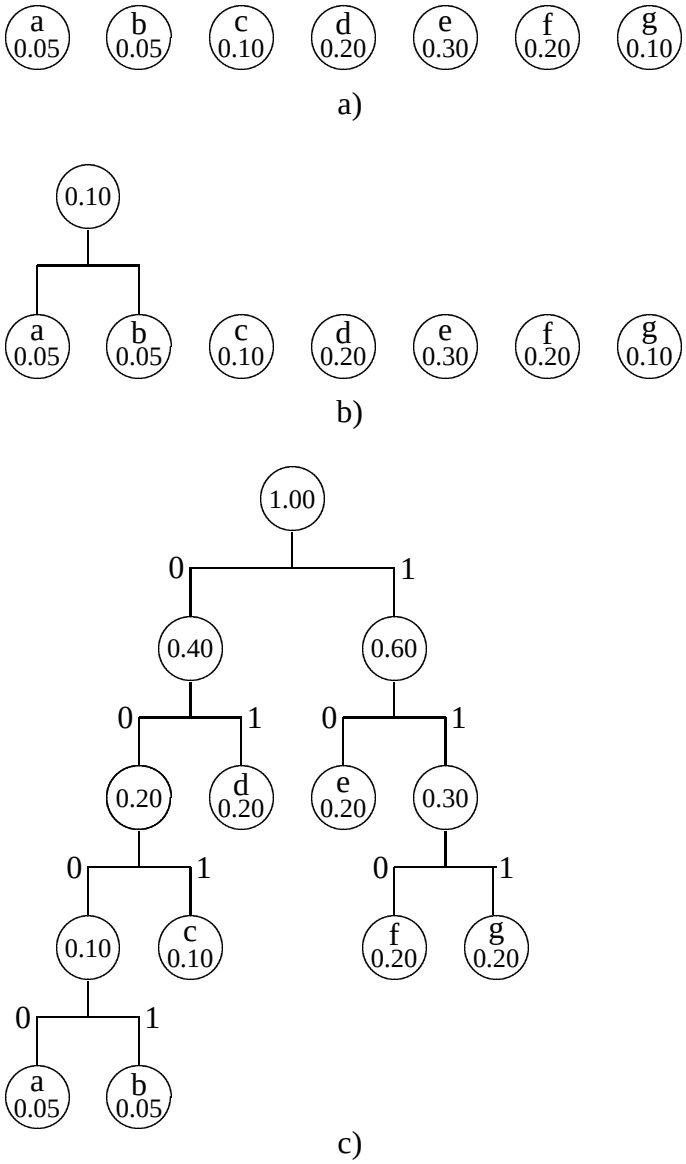
syyden tai frekvenssin sisältävää puuta liitetään yhteen (kuva 12b). Näin muodostetun puun juureen tulee sen lasten todennäköisyyksien tai frekvenssien summa. Pienimmän todennäköisyyden tai frekvenssin omaavien puiden yhdistämistä jatketaan, kunnes jäljellä on vain yksi puu. Tämän puun juuressa on kaikkien symbolien yhteenlaskettu todennäköisyys joka on yksi, tai symbolien frekvenssien summa eli syötteen koko (kuva 12c).

Puun kaarille annetaan bitti sen mukaan, lähteekö kaari vasemmalle vai oikealle; vasemmalle lähteville kaarille annetaan arvo 0 ja oikealle lähteville 1. Koodin tehokkuuden kannalta ei ole väliä miten bitit määrätään, koska sillä ei ole vaikutusta koodin pituuteen. Symbolien Huffman-koodit saadaan kulkemalla puun juuresta symbolia vastaavaan solmuun ja lukemalla kaarien bitit. Esimerkiksi symbolin b Huffman-koodi on 0001. Loput koodit ovat taulukossa 8.

Taulukko 8: Kuvan 12 Huffman-puusta saadut koodit.

Symboli	$p(x)$	$H(x)$	Koodisana	Koodin pituus
a	0.05	4.32	0000	4
b	0.05	4.32	0001	4
c	0.1	3.32	001	3
d	0.2	2.32	01	2
e	0.3	1.74	10	2
f	0.2	2.32	110	3
g	0.1	3.32	111	3

Koska edellä mainitussa syötteessä on seitsemän symbolia, tarvitaan kunkin symbolin esittämiseen $\lceil \log_2 7 \rceil = 3$ bittiä. Olkoon symbolien lukumäärä syötteessä 100. Tällöin sen esittämiseen tarvitaan $100 \cdot 3 = 300$ bittiä. Käytettäessä Huffman-puun avulla muodostettuja koodisanoja, saadaan tiedoston koko kertomalla kunkin symbolin lukumäärä sitä vastaavan koodisanan pituudella ja summaamalla nämä tulot. Esimerkiksi symbolin a lukumäärä syötteessä on $100 \cdot 0.05 = 5$. Näin ollen tiedosto vie Huffman-koodauksen jälkeen 260 bittiä eli se on 40 bittiä pienempi kuin alkuperäinen.



Kuva 12: Huffman-puun luominen: a) aloitustilanne; b) puiden yhdistäminen; c) Huffman-puu

Koska koodinpituudet ovat kokonaislukuja, eivät Huffman-koodien pituudet aina vastaa tiivistyksen alarajaa. Tässä tapauksessa esimerkiksi symbolin f koodaamiseen riittäisi 2.32 bittiä, mutta sen Huffman-koodin pituus on 3 bittiä.

Golomb-Rice -koodaus

Kun koodattavana olevien pikseleiden arvojen jakauma on eksponentiaalinen, voidaan koodauksessa käyttää eksponentiaalista etuliitekoodia kuten Golomb-Rice -koodia.

Golomb-koodauksessa (Golomb, 1966) koodattavat symbolit asetetaan todennäköisyyden mukaan laskevaan järjestykseen, ja kullekin symbolille annetaan ei-negatiivinen kokonaisluku lähtien nollasta, joka annetaan todennäköisimmälle symbolille. Koodattava luku x jaetaan koodattaessa kahteen osaan: eniten merkitsevään osaan x_M ja vähiten merkitsevään osaan x_L käyttämällä kaavaa 7

$$\begin{aligned} x_M &= \lfloor \tfrac{x}{m} \rfloor \\ x_L &= x \ \texttt{mod} \ m \end{aligned} \tag{7}$$

missä m on Golomb-koodin parametri. Luku x esitettynä osien x_M ja x_L avulla on $x = x_M \cdot m + x_L$. Eniten merkitsevä osa x_M koodataan unaarisesti, ja vähiten merkitsevä osa x_L sellaisenaan.

Rice (1979) löysi myöhemmin Golomb-koodin erikoistapauksen, jossa $m = 2^k$ jollekin kokonaisluvulle k. Rice-koodi parametrilla k on näin ollen sama kuin Golomb-koodi parametrilla $m = 2^k$. Parametrin k valinnasta FELICS-algoritmissa kerrotaan luvussa 4.1. Howardin ja Vitterin (1993) mukaan Rice-koodi on helpompi implementoida ja sen aikavaativuus on pienempi kuin Golomb-koodin, koska valittavana olevia parametreja on vähemmän. Golomb-koodauksella päästään heidän mukaansa kuitenkin hieman parempiin tiivistystuloksiin.

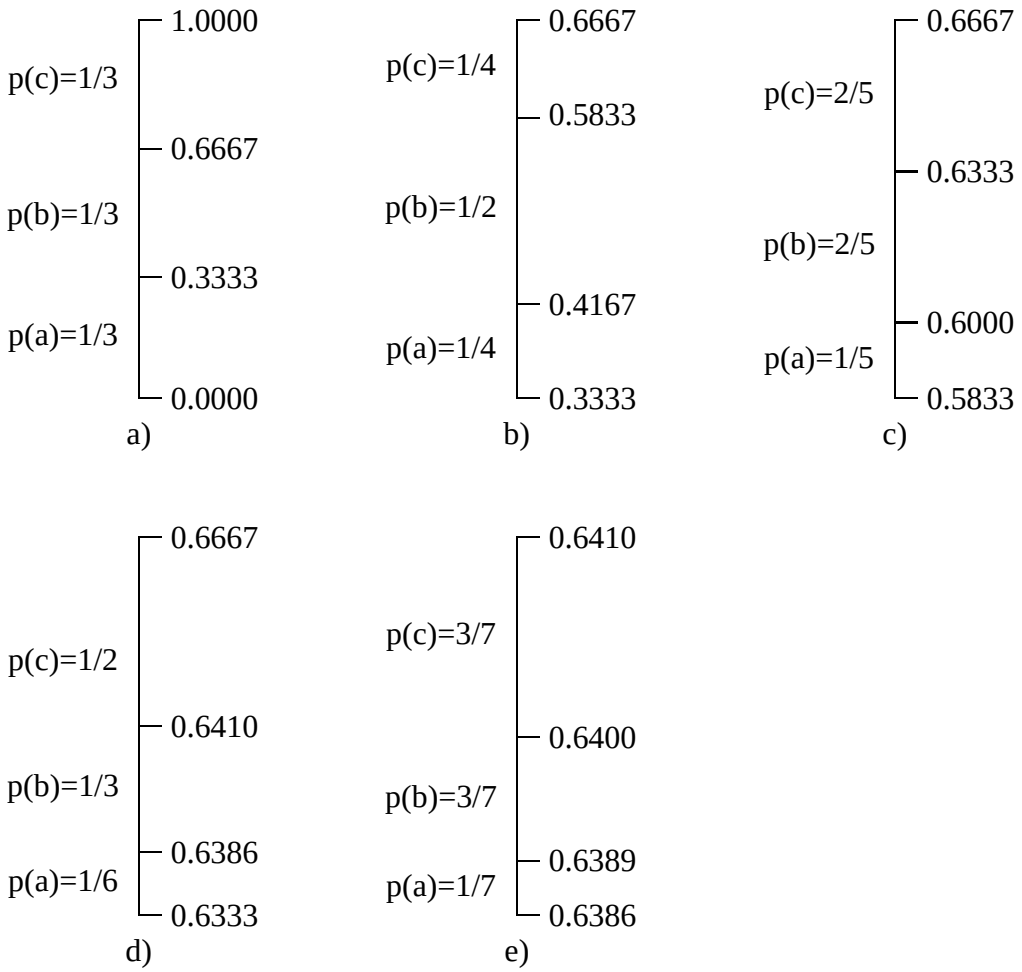
Aritmeettinen koodaus

Aritmeettinen koodaus on tilastollinen koodausmenetelmä, jossa koko syöte koodataan yhdeksi pitkäksi koodisanaksi sen sijaan, että yksittäisille symboleille annettaisiin omat koodinsa (Witten et al., 1987). Aritmeettisen koodauksen avulla voidaan päästä entropian määräämään bittimäärään yhden bitin tarkkuudella. Koska entropia on tiivistyksen alaraja (Shannon, 1948), on aritmeettinen koodaus optimaalinen. Aritmeettinen koodaus sopii hyvin dynaamiseen mallinta-

miseen, koska sillä ei ole erillistä kooditaulua päivitettävänään kuten Huffman-koodauksessa, vaan koodi muodostetaan aina lennossa. Huffman-koodauksen ja aritmeettisen koodauksen tiivistystehojen ero tulee hyvin esille binäärikuvan tapauksessa. Oletetaan, että binäärikuvassa valkoisen pikselin todennäköisyys on 0.99 ja mustan 0.01. Valkoisen pikselin entropia on tällöin $-\log_2 0.99 = 0.015$ bittiä. Koska Huffman-koodauksessa koodinpituus on vähintään 1, ei Huffman-koodauksella päästä tässä tapauksessa lähellekään tiivistyksen alarajaa.

Aritmeettisen koodauksen tuloste on bittivirta kuten muissakin koodaajissa. Aritmeettisen koodauksen periaate on helpompi ymmärtää, jos bittivirtaan lisätään etuliite 0, ja ajatellaan, että tuloste on nollan ja ykkösen välissä oleva binääriluku. Esimerkiksi tuloste 1010001111 muutetaan binääriluvuksi 0.1010001111. Kun tämä luku muutetaan kymmenjärjestelmään, saadaan sen arvoksi 0.64.

Olkoon koodattavana aakkoston $\{a, b, c\}$ symboleista muodostettu jono $bccb$. Koodataan se aritmeettisella koodaajalla käyttäen adaptiivista mallia. Alustetaan aluksi kaikkien symbolien lukumääräksi 1 ja oletetaan, että kunkin symbolin todennäköisyys on $\frac{1}{3}$. Aritmeettinen koodaaja pitää kirjaa kahdesta arvosta, jotka ova aluksi low=0 ja $high$=1. Näiden rajojen välinen alue jaetaan koodattavan todennäköisyysjakauman mukaan. Kuvassa 13a on aloitustilanteessa oleva väli, joka on jaettu kolmeen osaan aakkoston symbolien todennäköisyyksien mukaan. Aritmeettinen koodaaja kaventaa väliä vastaamaan koodattavaa symbolia. Koska ensimmäinen luettava symboli on b, saadaan rajoille uudet arvot low=0.3333 ja $high$=0.6667 (kuva 13b). Symbolien todennäköisyysjakaumat muuttuvat tässä vaiheessa siten, että p(a) =$\frac{1}{4}$, p(b) =$\frac{2}{4}$ ja p(a) = $\frac{1}{4}$. Uusi väli jaetaan näiden todennäköisyyksien mukaan kuten kuvassa 13b. Seuraavaksi luetaan c, jolloin väliksi saadaan low=0.5833 ja $high$=0.6667. Koodausta jatketaan, kunnes kaikki symbolit on luettu. Kuvassa 13c näkyy tilanne ensimmäisen c-kirjaimen lukemisen jälkeen, kuvassa 13d toisen c-kirjaimen lukemisen jälkeen ja kuvassa 13d koko jonon lukemisen ja koodaamisen jälkeen. Lopulta rajat ovat siis low=0.6386 ja $high$=0.6410.



Kuva 13: Aritmeettisen koodauksen eteneminen, kun koodattavana on jono $bccb$. a) Aloitustilanne, b) ensimmäisen, c) toisen, d) kolmannen ja e) neljännen symbolin koodaaminen.

Dekoodauksessa koodaaja välittää dekoodaajalle lopullisen low-arvon, joka on tässä tapauksessa 0.6386. Koodatun syötteen ensimmäinen symboli saadaan selville tutkimalla, mille välille low-arvo kuvan 13a tapauksessa asettuu. Tässä tapauksessa ensimmäinen symboli on b, koska $0.3333 \leq 0.6386 < 0.6667$. Tämän jälkeen koodatusta symbolista vähennetään symbolin b alkuperäinen low-arvo, jolloin saadaan $0.6386 - 0.3333 = 0.3053$. Kun näin saatu arvo vielä jaetaan symbolin b alkuperäisellä todennäköisyydellä, saadaan arvo 0.9160. Koodatun syötteen

toinen symboli saadaan selville tutkimalla, mille välille arvo 0.9160 asettuu. Dekoodaaja etenee vähentämällä saadusta arvosta symbolin low-arvon ja jakamalla symbolin alkuperäisellä todennäköisyydellä. Näin jatketaan, kunnes syöte on saatu koodattua.

3 Häviötön JPEG

JPEG-standardin häviötön versio käyttää ennustavaa mallia. Käyttäjä voi valita kahdeksasta eri ennustajasta (taulukko 9). Naapuripikseleihin viittaaminen on esitetty kuvassa 14.

c	b
a	x

Kuva 14: Käsiteltävän pikselin x naapuripikseleihin viittaaminen.

Taulukko 9: Häviöttömässä JPEG:ssä käytetyt ennustajat.

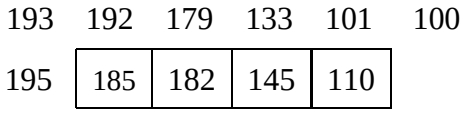
Malli	Ennustaja	Malli	Ennustaja
0	Null	4	a + b - c
1	a	5	a + (b - c)/2
2	b	6	b + (a - c)/2
3	c	7	(a + b)/2

Ennustevirheet koodataan joko aritmeettisella tai Huffman-koodauksella. Aritmeettisen koodauksen tapauksessa tilastollinen malli on adaptiivinen. Huffmankoodit ennustevirheille saadaan valmiista taulukosta (taulukko 10), eli tilastollinen malli on staattinen. Ennustevirheiden jakauma noudattaa normaalisti kuvan 10 mukaista jakaumaa. Tästä syystä taulukossa 10 olevien koodisanojen pituudet ovat suoraan verrannollisia ennustevirheen suuruuteen. Koska suurin ennustevirhe, joka taulukosta löytyy, on ± 255, voidaan kyseistä taulukkoa käyttää vain kuville, joissa on korkeintaan 256 värisävyä.

Tarkastellaan kuvaa 15 ja suoritetaan tiivistäminen laatikoiden sisällä oleville pikseleille käyttäen ennustemallia 2 (taulukko 9), jossa ennusteena käytetään koodattavan pikselin yläpuolista pikseliä. Verrattaessa vasemmanpuoleisimmassa laatikossa olevaa pikseliä sen yläpuolella olevaan pikseliin, saadaan ennustevirheeksi $185 - 192 = -7$. Tämän pikselin Huffmankoodiksi saadaan $100\ 000$ taulukosta 10. Loput koodit ovat taulukossa 11. Näin tulokseksi saadaan 6+5+7+7=25 bittiä, kun alkuperäiset pikselit vievät $4 \cdot 8 = 32$ bittiä.

Taulukko 10: Ennustevirheitä vastaavat Huffmankoodit.

Kategoria	Koodi	Ennustevirhe	Koodi
0	00	0	-
1	010	-1,1	0,1
2	011	-3,-2,2,3	00,01,10,11
3	100	-7,..,-4,4,..,7	000,..,011,100,..,111
4	101	-15,..,-8,8,..15	0000,..,0111,1000,..1111
5	110	-31,..,-16,16,..31	:
6	1110	-63,..,-32,32,..63	:
7	11110	-127,..,-64,64,..127	:
8	111110	-255,..,-128,128,..,255	:



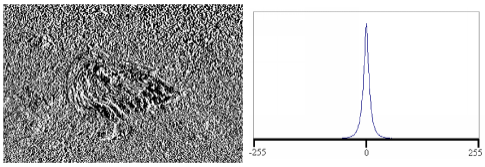
Kuva 15: Osa harmaasävykuvaa.

Taulukko 11: Eri pikselin arvoja vastaavat ennustevirheet ja Huffmankoodit, kun ennusteena on yläpuolella oleva pikseli.

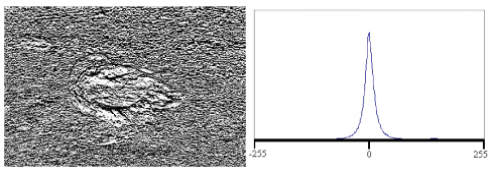
Pikseli	Ennustevirhe	Kategoria	Koodi	Koodinpituus
185	-7	3	100 000	6
182	3	2	011 11	5
145	12	4	101 1100	7
110	9	4	101 1001	7

Häviöttömässä JPEG:ssä ennustaja on koodattavan pikselin sama naapuripikseli tai niiden yhdistelmä koko kuvan läpikäynnin ajan riippumatta siitä, minkälaista kohtaa kuvassa ollaan koodaamassa. Häviötön JPEG onkin yksinkertainen toteuttaa, mutta tiivistyssuhde on huono verrattuna esimerkiksi FELICS:iin tai JPEG-LS:ään.

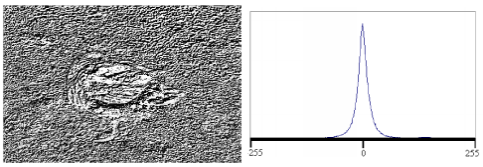
Kuva 16 havainnollistaa ennustajan valinnan vaikutusta virhekuviin ja niitä vastaaviin histogrammeihin.



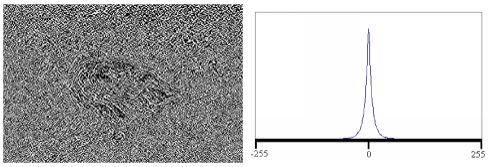
Ennustaja: *a*



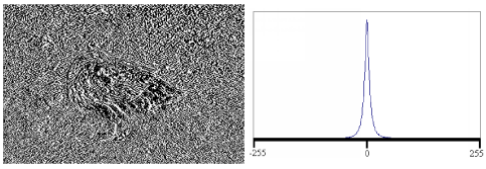
Ennustaja: *b*



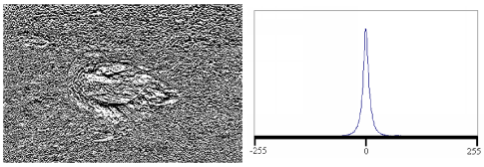
Ennustaja: *c*



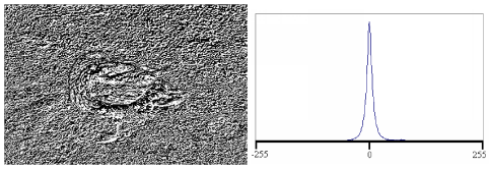
Ennustaja: *a+b-c*



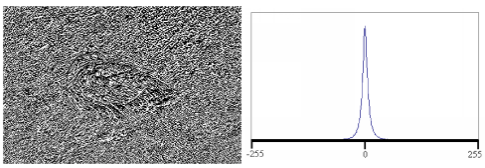
Ennustaja: *a+(b-c)/2*



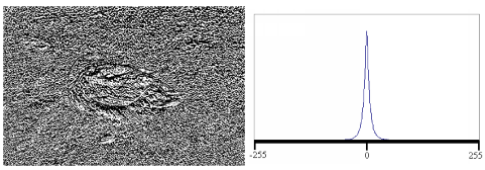
Ennustaja: *b+(a-c)/2*



Ennustaja: *(a+b)/2*



Ennustaja: *JPEG-LS*



Ennustaja: *GAP*

Kuva 16: Kuvalle *ankka.pgm* muodostetut ennustevirhekuvat ja niiden histogrammit.

4 FELICS

FELICS (Fast and Efficient Lossless Image Compression System) on yksinkertainen mutta tehokas tiivistysalgoritmi, jonka kehittäjiä ovat Howard ja Vitter (1993).

Kuvaa käydään läpi pikseli kerrallaan vasemmalta oikealle ja ylhäältä alas. FELICS käyttää apuna koodisanojen muodostuksessa kunkin koodattavan pikselin x kahta naapuripikseliä kuvan 17 mukaisesti.

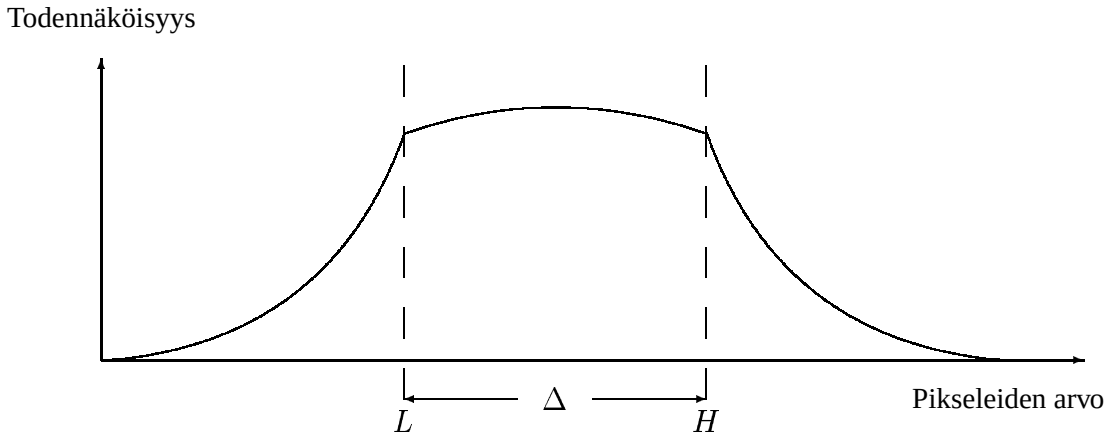
	N_2	N_1	x		
N_1	N_2			N_2	
x			N_1	x	

Kuva 17: Pikselin x koodauksessa käytettävät naapurit N_1 ja N_2.

Olkoon $L = min(N_1, N_2)$, $H = max(N_1, N_2)$ ja $\Delta = H - L$. Pikseleiden arvot noudattavat tavallisesti kuvan 18 mukaista jakaumaa.

4.1 Koodaus

Koodauksen ideana on käyttää yhtä bittiä kertomaan, onko koodattavan pikselin x arvo välillä Δ, ja mahdollisesti toista bittiä kertomaan, onko pikselin x arvo välin Δ ala- vai yläpuolella. Tämän jälkeen välillä Δ oleva pikseli koodataan käyttämällä *sovitettua binäärikoodia* (adjusted binary coding), ja välin Δ ulkopuolella oleva pikseli Golomb-Rice -koodauksella (Howard



Kuva 18: Pikseleiden arvojen todennäköisyysjakauma.

ja Vitter, 1993). Välin Δ ulkopuolella olevien pikseleiden arvojen todennäköisyysjakauma putoaa rajusti, joten eksponentiaaliset etuliitekoodit kuten Golomb-Rice -koodit soveltuvat hyvin kertomaan, kuinka kaukana pikselin x arvo on välistä Δ (Howard ja Vitter, 1993).

Sovitettu binäärikoodi

Jos koodattavana on välillä Δ oleva pikseli x, on koodattava arvo itseasiassa $x - L$ joka asettuu välille $[0, \Delta]$. Koska $\Delta = H - L$, on välillä $L...H$ olevien mahdollisten arvojen lukumäärä $\Delta + 1$. Jos $\Delta + 1$ on kahden potenssi, käytetään binäärikoodia, jonka pituus on $log_2(\Delta + 1)$ bittiä. Tällaisella koodilla päästään hyviin tuloksiin, sillä välillä Δ olevien arvojen jakauma on lähes tasainen. Muussa tapauksessa koodi sovitetaan antamalla $\lfloor log_2(\Delta+1) \rfloor$ bittiä joillekin arvoille ja $\lceil log_2(\Delta+1) \rceil$ bittiä toisille. Koska välin Δ keskellä olevat arvot ovat hieman todennäköisempiä kuin sen laidoilla olevat, annetaan keskellä oleville arvoille lyhyemmät koodisanat. Esimerkiksi, jos $\Delta = 4$, on koodattavana arvot $0, 1, 2, 3$ ja 4. $\Delta + 1 = 5$ ei ole kahden potenssi, joten koodisanojen pituudet ovat $\lfloor log_2(5) \rfloor = 2$ ja $\lceil log_2(5) \rceil = 3$. Arvoja vastaavat koodisanat ovat tällöin $00, 01, 10, 110$ ja 111. Annetaan nyt välin Δ keskellä oleville arvoille lyhyemmät koodisanat, jolloin arvoille $0...4$ saadaan taulukon 12 mukaiset koodisanat.

Taulukko 12: Arvoja $0...4$ vastaavat koodisanat.

$x - L$	koodisana
0	111
1	10
2	00
3	01
4	110

Pidempien koodisanojen lukumäärä on aina parillinen, jolloin koodisanojen pituudet asettuvat aina symmetrisesti välille $[0, \Delta]$. Näin ollen keskenään saman todennäköisyyden omaaville pikselin arvoille ei koskaan jouduta antamaan eri mittaisia koodisanoja.

Golomb-Rice -koodaus

Jos pikselin x arvo on suurempi kuin H, koodataan erotus $x - (H + 1)$. Jos taas x on pienempi kuin L, koodataan erotus $(L - 1) - x$. Koska välin Δ ulkopuolelle jäävät jakauman osat ovat keskenään saman muotoisia, voidaan ne koodata samalla Golomb-Rice -koodilla. Golomb-Rice -koodin parametri k määritetään käyttämällä väliä Δ kontekstina. Kunkin kontekstin Δ tapauksessa kullekin mahdolliselle arvolle k pidetään yllä kumulatiivista summaa sen hetkisestä koodinpituudesta, johon parametrin k arvo johtaisi, kun kontekstin kaikki siihen mennessä vastaan tulleet arvot on koodattu. Lyhimmän kumulatiivisen koodinpituuden tuottanutta parametria käytetään kontekstissa seuraavana olevan arvon koodauksessa.

4.2 Käytännön toteutus

Kuvassa 19 on yhteenveto FELICS-algoritmista.

1. Etsitään koodattavan pikselin x naapurit N_1 ja N_2.

2. Lasketaan $L = \min(N_1, N_2)$, $H = \max(N_1, N_2)$ ja $\Delta = H - L$.

3. a) Jos $L \leq x \leq H$, tulostetaan bitti merkitsemään välillä Δ olevaa arvoa, ja koodataan erotus $x - L$ sovitetulla binäärikoodilla.
   b) Jos $x < L$, tulostetaan bitti merkitsemään välin Δ ulkopuolella olevaa arvoa, ja toinen bitti merkitsemään välin Δ alapuolella olevaa arvoa. Koodataan ei-negatiivinen kokonaisluku $(L-1) - x$ Golomb-Rice -koodilla.
   c) Jos $x > L$, tulostetaan bitti merkitsemään välin Δ ulkopuolella olevaa arvoa, ja toinen bitti merkitsemään välin Δ yläpuolella olevaa arvoa. Koodataan ei-negatiivinen kokonaisluku $x - (H+1)$ Golomb-Rice -koodilla.

Kuva 19: FELICS-algoritmi (Howard ja Vitter, 1993)

Sovelletaan FELICS-algoritmia kuvan 20 laatikoissa oleville pikseleille.

193	192	177	133
195	185	182	186

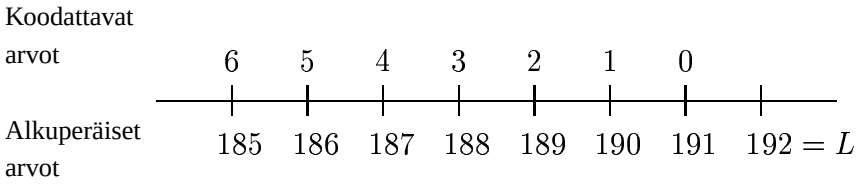
Kuva 20: Osa harmaasävykuvaa.

Vasemmanpuoleisessa laatikossa olevan pikselin $x = 185$ naapurit ovat $N_1 = 195$ ja $N_2 = 192$. Näin ollen välin Δ alaraja $L = 192$ ja yläraja $H = 195$. Koska $x = 185 < 192 = L$, siirrytään kuvan 19 algoritmissa kohtaan 3b. Koska $x = 185$ on välin alapuolella, liitetään koodisanan eteen taulukosta 13 saatavat bitit 00.

Taulukko 13: Bitit, jotka liitetään arvoon x, kun $x < L$, $L \leq x \leq H$ tai $x > H$.

Alue	Koodi
$x < L$	00
$L \leq x \leq H$	1
$x > H$	01

Oletetaan, että kontekstin $\Delta = 3$ tapauksessa parametri $k = 2$, jolloin parametri $m = 2^2 = 4$. Koska FELICS-algoritmissa koodataan pikselin arvon sijaan sen etäisyys välistä Δ, annetaan välin Δ alapuolella oleville arvoille uudet arvot kuvan 21 mukaisesti.



Kuva 21: Välin Δ alapuolella oleville arvoille annetut koodattavat arvot.

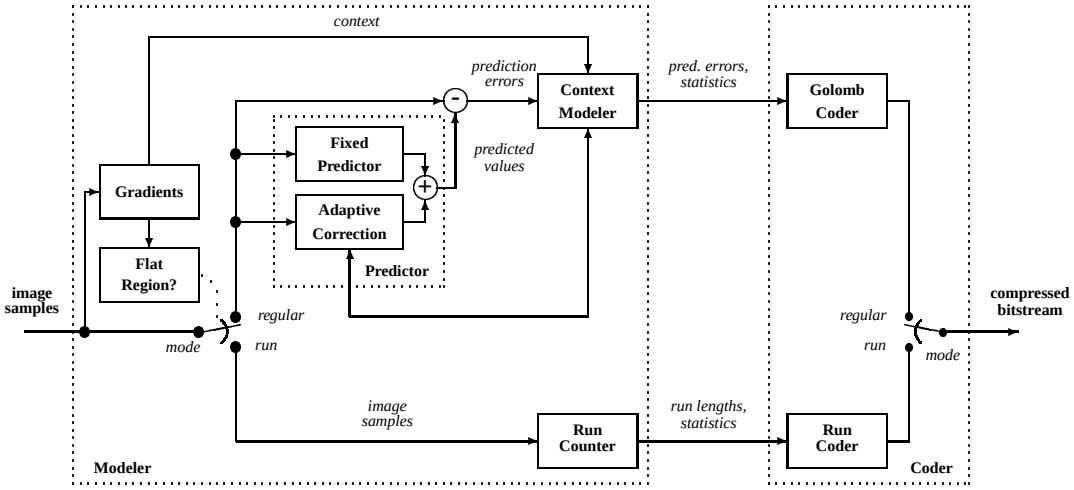
Näin ollen luvun 185 sijaan koodataan luku 6. Luku $6_{10} = 110_2$ jaetaan kahteen osaan kaavan 7 avulla. Eniten merkitsevä osa $x_M = \lfloor \frac{6}{4} \rfloor = 1_{10} = 1_2$, ja vähiten merkitsevä osa $x_L = 6 \bmod 4 = 2_{10} = 10_2$. Eniten merkitsevä osa koodataan unaarisesti, jolloin saadaan bitit 10, ja vähiten merkitsevä osa sellaisenaan. Pikselin koodisana on tällöin 001010.

Keskimmäisessä laatikossa olevan pikselin $x = 182$ naapurit ovat $N_1 = 185$ ja $N_2 = 177$, eli alaraja $L = 177$ ja yläraja $H = 185$. Tässä tapauksessa koodattava pikseli $x = 182$ kuuluu välille Δ, joten kuvan 19 algoritmissa siirrytään kohtaan 3a. Taulukosta 13 liitetään koodisanan eteen bitti 1, ja koodataan erotus $x - L = 182 - 177 = 5$ sovitetulla binäärikoodilla. Tässä tapauksessa $\Delta + 1 = 8 + 1 = 9$ on kahden potenssi, joten binäärikoodin pituus on $\log_2(8+1) = 3$. Näin ollen välin Δ kaikki arvot koodataan kolmella bitillä. Arvo 5 koodataan koodisanalla 101. Pikselin koodisanaksi tulee tällöin 1101.

Oikeanpuoleisessa laatikossa olevan pikselin $x = 186$ naapurit ovat $N_1 = 182$ ja $N_2 = 133$. Välin Δ alaraja on tällöin $L = 133$ ja yläraja $H = 182$. Koska koodattava pikseli on välin Δ yläpuolella, siirrytään kuvan 19 algoritmissa kohtaan 3c. Koodisanan eteen liitetään taulukosta 13 bitit 01, ja koodataan erotus $x - (H + 1) = 186 - (182 + 1) = 3$. Oletetaan, että kontekstin $\Delta = 49$ tapauksessa $k = 2$, jolloin $m = 4$. Kaavan 7 avulla saatavat luvun $3_{10} = 11_2$ osat ovat $x_M = \lfloor \frac{3}{4} \rfloor = 0_{10} = 0_2$ ja $x_L = 3 \bmod 4 = 3_{10} = 11_2$. Näin ollen pikselin koodisana on 01011.

5 JPEG-LS

JPEG-LS:n perustana on LOCO-I (Low Complexity Lossless Compression for Images) -algoritmi (Weinberger et al., 1996b). LOCO-I on värisävykuvien häviötön tiivistysalgoritmi, joka käyttää hyväkseen kontekstimallin tiivistystehoa. JPEG-LS:n toiminta jakautuu kolmeen osaan: ennustajan valintaan, kontekstimallin muodostamiseen ja ennustevirheiden koodaukseen. Kuvan 22 kaavio havainnollistaa JPEG-LS:n toimintaa.



Kuva 22: JPEG-LS:n toimintakaavio (Weinberger et al., 2000).

Dekoodaus tapahtuu samoja menettelytapoja käyttäen kuin koodaus, mutta päinvastaisessa järjestyksessä. Kuvan ensimmäisellä rivillä oletetaan $b = c = d = 0$. Ensimmäisessä ja viimeisessä sarakkeessa pikseleiden arvot kohdissa a ja d oletetaan samaksi kuin kohdassa b olevan pikselin arvo jos ne ovat määrittelemättömiä. Kohdassa c olevan pikselin arvoksi kopioidaan arvo, joka oli kohdassa a edellisen rivin ensimmäistä pikseliä koodattaessa.

5.1 Ennustaminen

Ennustettaessa arvoa pikselille x, tutkitaan ensin staattisen ennustajan avulla, onko pikselin x läheisyydessä pysty- tai vaakasuoraa reunaa. Ennustajan $\hat{x}_{\text{MED}}$ arvo saadaan kaavasta

$$\hat{x}_{\text{MED}} = \begin{cases} \min(a, b) & \text{jos } c \geq \max(a, b) \\ \max(a, b) & \text{jos } c \leq \min(a, b) \\ a + b - c & \text{muulloin} \end{cases} \tag{8}$$

missä a, b ja c ovat kuvassa 23 olevia pikselin arvoja.

c	*b*	*d*
a	*x*	

Kuva 23: Käsiteltävän pikselin x naapuripikseleihin viittaaminen.

Kaavassa käytettävä vaihtoehto riippuu kuvan koodattavana olevan kohdan ominaisuuksista. Kaava pyrkii valitsemaan naapuripikselin b, jos tarkasteltavan pikselin x vasemmalla puolella on pystysuora reuna. Vastaavasti ennustajaksi valitaan a, jos pikselin x yläpuolella on vaakasuora reuna. Väriliukuman tapauksessa ennustajana on $a + b - c$ (Weinberger et al., 2000). Toisin sanoen; jos c on suurin, valitaan naapuripikseleistä pienin. Jos c on pienin, valitaan naapuripikseleistä suurin. Muutoin lasketaan lineaarinen ennuste olettaen tasainen liukuma. Kuvassa 24 on esimerkit kaavan 8 kolmesta eri vaihtoehdosta.

80	40
75	*x*

a)

40	80
75	*x*

b)

75	80
40	*x*

c)

Kuva 24: Kaavan 8 kolme eri vaihtoehtoa. a) c suurin, jolloin ennuste on 40, b) c pienin, jolloin ennuste on 80, c) c ei suurin eikä pienin, jolloin ennuste on $40 + 80 - 75 = 45$.

5.2 Kontekstimalli

Konteksti määritellään laskemalla kolme naapuripikseleiden välistä erotusta: $g_1 = d - b$, $g_2 = b - c$ ja $g_3 = c - a$. Nämä erotukset edustavat paikallista gradienttia, ts. kuvaavat pikseliä x ympäröivän alueen ominaisuuksia (väriliukuma, reuna), jotka määräävät ennustevirheiden tilastollisen käyttäytymisen (Weinberger et al., 2000).

Mallin kokoa voidaan pienentää kvantisoimalla erotukset g_j, $j = 1, 2, 3$ arvosta j riippumattomalla kvantisoijalla $\kappa(\cdot)$. Periaatteessa kvantisoimalla muodostettujen kontekstien lukumäärä tulisi optimoida adaptiivisesti. Jos kuitenkin halutaan kompleksisuuden pysyvän alhaisena, on kontekstien lukumäärän oltava pieni. Jotta symmetrisyys säilyisi, indeksoidaan ryhmät arvoilla $-T, ..., -1, 0, 1, ..., T$. Kontekstien lukumäärä on tällöin $(2T + 1)^3$. Kontekstien lukumäärää voidaan edelleen vähentää olettamalla symmetrian nojalla

Prob$\{\epsilon_{t+1} = \Delta \mid C_t = [q_1, q_2, q_3]\}$ = Prob$\{\epsilon_{t+1} = -\Delta \mid C_t = [-q_1, -q_2, -q_3]\}$

missä ϵ on ennustevirhe, C_t kvantisoitu kontekstikolmikko ja $q_j = \kappa(g_j)$. Näin ollen, jos ensimmäinen kolmikon C_t nollasta eroava elementti on negatiivinen ja koodattu arvo on ϵ_{t+1}, niin konteksti on $-C_t$. Dekoodaaja ennakoi mahdolliset negatiiviset arvot ja kääntää tarvittaessa etumerkin, jolloin saadaan alkuperäinen virhearvo. Yhdistämällä vastakkaista merkkiä olevat kontekstit, saadaan lopulliseksi kontekstien lukumääräksi $((2T + 1)^3 + 1)/2$. JPEG-LS:ssä on asetettu $T = 4$, jolloin kontekstien lukumäärä on $(2 \cdot 4 + 1)^3 = 365$. Tämä lukumäärä on riittävän alhainen tilantarpeen kannalta, mutta toisaalta riittävän suuri hyvien tulosten saamiseksi (Weinberger et al., 2000). Keskikokoisten ja suurten kuvien tiivistämisessä kontekstien lukumäärää voitaisiin tästäkin vielä kasvattaa ilman, että mallin koko kasvaisi kohtuuttomasti. Tiivistysteho ei kuitenkaan parane niin paljon, että tähän kannattaisi ryhtyä (Weinberger et al., 1996a). Pienten kuvien tapauksessa kontekstien määrää voidaan rajoittaa standardin sallimissa puitteissa parametrien avulla.

Kvantisoimalla saaduille ryhmille tulee määrittää rajat. Esimerksi 8-bittiselle syöteaakkostolle rajat on JPEG-LS:ssä asetettu $\{0\}$, $\pm\{1,2\}$, $\pm\{3,4,5,6\}$, $\pm\{7,8,...,20\}$ ja $\pm\{e|e \geq 21\}$. Rajoja voidaan muutella kunhan vain huolehditaan siitä, että keskimmäinen alue on aina $\{0\}$.

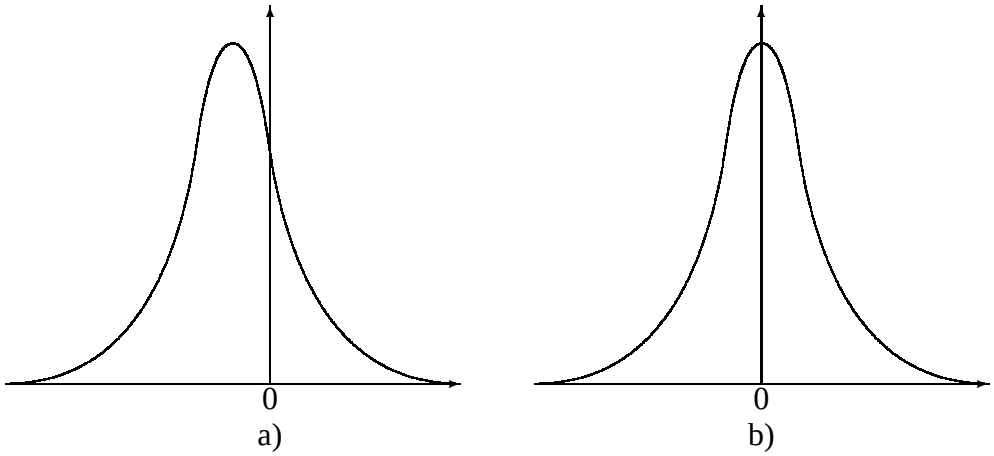
Tarkastellaan kuvaa 15 ja suoritetaan tiivistäminen laatikoiden sisällä oleville pikseleille käyttäen kaavaa 8 ja kontekstimallia. Vasemmanpuoleisessa laatikossa olevan pikselin naapurit ovat $a = 195$, $b = 192$ ja $c = 193$. Naapurin c arvo on naapureiden a ja b välissä, joten ennustajaksi valitaan kaavan 8 kolmas vaihtoehto $a + b - c$, jolloin ennustaja $\hat{x}_{\text{MED}} = 195 + 192 - 193 = 194$ ja ennustevirhe $\hat{e} = 185 - 194 = -9$. Konteksti saadaan laskemalla erotukset $g_1 = d - b = 179 - 192 = -13$, $g_2 = b - c = 192 - 193 = -1$ ja $g_3 = c - a = 193 - 195 = -2$. Loput arvot ovat taulukossa 14. Gradientit kvantisoidaan edellä mainittuihin ryhmiin, jolloin esimerkiksi kolmikkoa $(-13, -1, -2)$ vastaa ryhmät $(\{-7, -8, ..., -20\}, \{-1, -2\}, \{-1, -2\})$. Asetetaan ryhmille indeksit siten, että ryhmän $\{0\}$ indeksi on 0, ryhmien $\pm\{1,2\}$ indeksit ± 1 jne. Tällöin kolmikko $(-13, -1, -2)$ on indeksien avulla esitettynä $(-3, -1, -1)$. Loput indeksit ovat taulukossa 14.

Taulukko 14: Kuvan 15 laatikoissa oleville pikseleille lasketut ennustajat, ennustevirheet, gradientit ja niitä vastaavat kontekstit.

Pikseli	Ennustaja	Ennustevirhe	Gradientit	Konteksti
185	$a + b - c = 194$	-9	$(g_1, g_2, g_3) = (-13, -1, -2)$	$(-3, -1, -1)$
182	$min(a, b) = 179$	3	$(g_1, g_2, g_3) = (-46, -13, 7)$	$(-4, -3, 3)$
145	$a + b - c = 136$	9	$(g_1, g_2, g_3) = (-32, -46, -3)$	$(-4, -4, -2)$
110	$a + b - c = 113$	-3	$(g_1, g_2, g_3) = (-1, -32, -12)$	$(-1, -4, -3)$

5.3 Vinouman poisto

Optimaalisen ennustajan käyttäminen johtaisi ennustevirheeseen $\epsilon = 0$. Staattisen ennustajan (kaava 8) avulla saatujen ennustevirheiden todennäköisyysjakauman huippukohta ei kuitenkaan aina ole nollassa, vaan jakaumassa voi olla *vinoumaa* (bias) (kuva 25a). Todennäköisyysjakauman vinouma voi huonontaa tiivistystehoa merkittävästi, joten jakauman maksimikohta tulee siirtää nollaan (kuva 25b).



Kuva 25: a) Jakauma, jossa on vinoumaa. b) Jakauma, josta vinouma on poistettu.

Tarkastellaan, kuinka vinouma voidaan poistaa pienellä kompleksisuudella. Jotta menetelmän kompleksisuus ei kasvaisi liikaa, käytetään tarkkojen arvojen sijaan likiarvoja.

Pidetään yllä tietoa käsiteltyjen pikseleiden lukumäärästä N, ja kontekstissa esiintyneiden ennustevirheiden ϵ kumulatiivisesta summasta $D = \sum_{i=1}^{N} \epsilon_i$.
Korjausarvo C' lasketaan kaavan

$$C' = \lceil D/N \rceil \tag{9}$$

avulla, ja lisätään ennustajaan $\hat{x}_{\text{MED}}$ poistamaan mahdollinen vinouma. Toisin sanoen, C' on ennustevirheiden keskiarvo pyöristettynä ylöspäin. Jos $C' < 0$ (tai $C' > 0$), on jakauman maksimi nollan vasemmalla puolella (tai nollan oikealla puolella). Jos $C' = 0$, on jakauma keskittynyt

nollaan, jolloin korjausta ei tarvita. Tässä menetelmässä on kuitenkin kaksi ongelmaa. Ensinnäkin, se vaatii jakolaskun, jolloin kompleksisuus kasvaa. Toiseksi, suuret virheet saattavat vaikuttaa korjausarvon C' tuleviin arvoihin, kunnes se palautuu takaisin tyypilliseen arvoonsa, joka on suhteellisen vakaa. Esimerkiksi virhearvojen $\epsilon = \{0, 0, -1, +1, -20, +2\}$ joukossa oleva arvo -20 johtaa korjausarvoon $C' = \lceil -18/6 \rceil = -3$, vaikka virhearvojen jakauman maksimi on nollassa. Näiden ongelmien ratkaisemiseksi huomataan, että kaava 9 on ekvivalentti kaavan

$$C = N \cdot C' + B' \tag{10}$$

kanssa, missä $-N < B' \leq 0$ (Weinberger et al., 2000). Korjausarvo muodostetaan tallettamalla B' ja C' ja säätämällä molemmat kullekin kontekstin esiintymälle erikseen. Korjattu ennustevirhe ϵ lisätään ensin lukuun B', jonka jälkeen lisätään tai vähennetään N, kunnes lopputulos on välillä $]-N, 0]$. Näiden lisäysten ja vähennysten lukumäärä määrää, miten arvoa C' säädetään. Tätä menetelmää muunnetaan JPEG-LS:ssä edelleen rajoittamalla lisäysten ja vähennysten määrä yhteen per päivitys ja tarvittaessa pakottamalla B' halutulle välille. C' ja B' korvataan likimääräisillä arvoilla C ja B, jotka on alustettu nollaksi ja päivitetty kuvan 26 C-kielisen koodin mukaan.

Kuvan 26 menetelmä kasvattaa (tai vähentää) korjausarvoa C joka kerta, kun $B > 0$ (tai $B \leq -N$). Tässä vaiheessa B mukautetaan kuvastamaan arvon C muutoksia. Jos lisäys (tai poisto) ei riitä siirtämään arvoa B halutulle välille, asetetaan $B = 0$ (tai $B = -N+1$). Tämä menettely pyrkii tuottamaan välillä $]-1, 0]$ olevia ennustevirheiden keskiarvoja.

5.4 Ennustevirheiden koodaus

JPEG-LS käyttää ennustevirheiden koodaukseen *Golomb-koodia* (Golomb, 1966). Golomb-koodi on optimaalinen eksponentiaalisesti väheneville (geometrisille) positiivisten lukujen todennäköisyysjakaumille (Gallager ja Voorhis, 1975).

```
B = B + ε      /* accumulate prediction residual */
N = N + 1      /* update occurence counter */
/* update correction value and shift statistics */
if (B ≤ −N) {
   C = C − 1; B = B + N;
   if (B ≤ −N) B = −N + 1;
}
else if (B > 0) {
   C = C + 1; B = B − N;
   if (B > 0) B = 0
}
```

Kuva 26: Menetelmä vinouman poistamiseksi (Weinberger et al., 2000).

Ennen koodausta ennustevirheiden ϵ kaksipuolinen jakauma on muutettava positiivisista kokonaisluvuista koostuvaksi jakaumaksi kuvaamalla välillä $-\alpha/2 \leq \epsilon \leq \alpha/2 - 1$ olevat arvot välillä $0 \leq M(\epsilon) \leq \alpha - 1$ oleviksi arvoiksi kuvauksen

$$M(\epsilon) = \begin{cases} 2\epsilon & \text{jos } \epsilon \geq 0 \\ 2|\epsilon| - 1 & \text{jos } \epsilon < 0 \end{cases} \tag{11}$$

avulla. Toisin sanoen, kuvaus indeksoi arvot $0, -1, 1, -2, 2, ...$ arvoiksi $0, 1, 2, 3, 4, ...$ Jos ennustevirheet ϵ noudattavat Laplace-jakaumaa, jonka maksimi on nollassa, niin jakauma $M(\epsilon)$ on lähestulkoon geometrinen, jolloin se voidaan koodata Golomb-Rice-koodauksella.

Kuvissa 27 ja 28 on yhteenveto koodausprosessista.

0. Alustus:
   a. Laske
      $$L_{\max} = 2(\beta_{\max} + \max\{8, \beta_{\max}\}),$$
      missä
      $$\beta_{\max} = \max\{2, \lceil \log \alpha \rceil\}$$ ja α syöteaakkoston koko.
   b. Alusta kontekstilaskurien A, B, C ja N 365 joukkoa seuraavasti:
      $$B = C = 0, N = 1, A = \max\{2, \lfloor (\alpha + 32)/64 \rfloor\}.$$
   c. Aseta *run mode*-mukautustaulukkoon $I_{\text{RUN}} = 0$.
   d. Aseta x kuvan ensimmäiseksi pikseliksi.

1. Laske gradientit $g_1 = d - b$, $g_2 = b - c$ ja $g_3 = c - a$.

2. Jos $g_1 = g_2 = g_3 = 0$, siirry *run-modeen* (kuva 28).
   Muutoin jatka *regular modessa*.

3. Kvantisoi gradientit g_i, $i = 1, 2, 3$.

4. Merkitse kvantisoituja gradientteja q_i, $i = 1, 2, 3$. Jos kolmikon $[q_1, q_2, q_3]$ ensimmäinen komponentti on negatiivinen, vaihda kolmikon kaikki etumerkit ja liitä kolmikkoon miinus-merkki, muussa tapauksessa plus-merkki. Kuvaa kolmikko yksi-yhteen -kuvauksena välillä $[1, 364]$ olevaksi indeksiksi. Käytä indeksejä kontekstilaskureihin viittaamiseen.

5. Laske staattinen ennustaja $\hat{x}_{\text{MED}}$ kaavan 8 avulla.

6. Korjaa $\hat{x}_{\text{MED}}$ lisäämällä siihen (tai vähentämällä) C:n arvo, jos konteksti on positiivinen (tai negatiivinen). Kiinnitä korjattu arvo välille $[0, \alpha - 1]$ saadaksesi oikea ennuste $\hat{x}$

7. Laske ennustevirhe $\bar{\epsilon} = x - \hat{x}$ ja jos konteksti on negatiivinen, aseta $\bar{\epsilon} \leftarrow -\bar{\epsilon}$. Supista $\bar{\epsilon}$ modulo α välillä $[-\lfloor \alpha/2 \rfloor, \lceil \alpha/2 \rceil - 1]$ olevaksi arvoksi.

Kuva 27: JPEG-LS: koodausalgoritmi (Weinberger et al., 2000).

1. Lue pikseli ja siirry seuraavaan, kunnes $x \neq a$ tai ollaan rivin lopussa.

2. Olkoon $m = 2^g$ Golomb-koodin parametri.
   Tee seuraava jokaiselle pituutta m olevalle segmentille: lisää bittivirtaan $'1'$ ja kasvata indeksiä I_{RUN}.
   Tuplaa m jos taulukko niin sanoo.

3. Jos run keskeytyi rivinlopun takia, lisää bittivirtaan $'1'$ ja siirry kuvan 27 algoritmin riville 1.
   Muutoin lisää bittivirtaan $'0'$ ja jäljelle jääneen $run\ lenght$:n g-bittinen esitys sen perään, kasvata arvoa $\hat{x}_{\text{RUN}}$ ja puolita m jos taulukko niin sanoo.

4. Koodaa häiriö ja siirry kuvan 27 algoritmin riville 1.

Kuva 28: JPEG-LS: $Run\ mode$ (Weinberger et al., 2000).

6 Tiivistysmenetelmien vertailua

Tässä luvussa vertaillaan erilaisilla tiivistysmenetelmillä saatuja tuloksia. Käytetyt menetelmät ovat JPEG eri ennustajineen, FELICS ja JPEG-LS. JPEG:ssä käytettiin taulukon 9 ennustajien lisäksi JPEG-LS:n käyttämää ennustajaa sekä GAP:a. Taulukossa 15 on yhteenveto käytetyistä menetelmistä.

Taulukko 15: Tiivistysmenetelmien vertailussa käytetyt menetelmät.

	Toteutus	Esitelty
JPEG	kirjoittaja	luku 3
-vakioennustajat		taulukko 9
-JPEG-LS:n ennustaja		kaava 8
-GAP-ennustaja		kuva 7
FELICS	valmis kooderi (Bell et al., 1999)	luku 4
JPEG-LS	valmis kooderi (Hewlett-Packard, 1999)	luku 5

Eri menetelmillä saadut tulokset ovat taulukossa 16. JPEG:n tapauksessa taulukkoon on listattu parhaan ja huonoimman tuloksen antaneen ennustajan tulokset. Käytetty ennustaja on annettu suluissa. Taulukossa on lisäksi tulokset, jotka on saatu käyttämällä ennustajina JPEG-LS:n ennustajaa ja GAP:ia.

Taulukko 16: Tiivistyssuhteet. JPEG:n tapauksessa on esitetty paras ja huonoin tiivistyssuhde ja niihin johtaneet ennustajat, JPEG-LS:n käyttämä ennustaja sekä GAP.

	ankka.pgm	baboon.pgm	cartman.pgm	kirkko.pgm	lena.pgm	metsa.pgm	thighs.pgm	Keskiarvo
JPEG (huonoin)	0.83 (3)	0.93 (3)	0.54 (3)	0.76 (3)	0.71 (3)	0.90 (3)	0.68 (3)	0.76
JPEG (paras)	0.72 (5)	0.84 (7)	0.41 (4)	0.62 (4)	0.64 (6 & 7)	0.79 (6)	0.58 (4)	0.66
JPEG (JPEG-LS)	0.72	0.84	0.41	0.60	0.64	0.77	0.57	0.65
JPEG (GAP)	0.72	0.84	0.44	0.63	0.63	0.77	0.58	0.66
FELICS	0.68	0.79	0.37	0.59	0.57	0.75	0.53	0.61
JPEG-LS	0.64	0.75	0.26	0.53	0.53	0.71	0.48	0.56

Taulukkoa 16 tarkasteltaessa huomataan, että JPEG:n ennustajista huonoimman tuloksen antoi kaikkien kuvien kohdalla ennustaja c eli käsiteltävän pikselin vasemmalla yläviistossa oleva pikseli. Parhaisiin tuloksiin päästiin käyttämällä kolmen naapuripikselin erilaisia yhdistelmiä. JPEG:ssä päästiin keskenään lähes samoihin tuloksiin käyttämällä standardin parasta ennustajaa, JPEG-LS:n ennustajaa ja GAP:a. Liitteen 1 testikuvia käytettäessä näyttääkin siltä, että huolellinen ennustajan valinta ei vaikuta radikaalisti testituloksiin, jos ennustevirheet koodataan sellaisenaan käyttäen valmiita Huffman-koodeja.

Kun ennustavaan malliin yhdistetään kontekstimalli kuten JPEG-LS:ssä, päästään huomattavasti parempiin tuloksiin kuin pelkkää JPEG-LS:n ennustajaa käyttämällä. FELICS tuotti parempia tuloksia kuin JPEG, mutta jäi selvästi jälkeen JPEG-LS:stä.

Taulukossa 17 on teoreettiset tiivistystulokset JPEG:n eri ennustajille. Tiivistyssuhteet on saatu laskemalla ennustevirhekuvien entropiat ja jakamalla ne kahdeksalla, joka on alkuperäisen kuvan viemä bittimäärä pikseliä kohden.

Taulukko 17: Ennustevirhekuvien entropioihin perustuvat laskennalliset tiivistyssuhteet käytettäessä JPEG-algoritmia. Taulukossa on esitetty paras ja huonoin tiivistyssuhde ja niihin johtaneet ennustajat, JPEG-LS:n käyttämä ennustaja sekä GAP.

Ennustaja	ankka.pgm	baboon.pgm	cartman.pgm	kirkko.pgm	lena.pgm	metsa.pgm	thighs.pgm	Keskiarvo
huonoin	0.77 (3)	0.86 (3)	0.51 (3)	0.74 (3)	0.66 (3)	0.83 (3)	0.64 (3)	0.71
paras	0.67 (5)	0.78 (7)	0.34 (4)	0.59 (4)	0.64 (6)	0.73 (6)	0.52 (4)	0.60
JPEG-LS	0.66	0.78	0.33	0.57	0.57	0,72	0.55	0.60
GAP	0.66	0.78	0.37	0.59	0.55	0,72	0.51	0.60

Taulukon 17 tuloksia tarkasteltaessa huomataan, että huolellinen ennustajan valinta ei tässäkään tapauksessa vaikuta tuloksiin radikaalisti. JPEG:n vakioennustajista parhaan ja huonoimman tuloksen antoivat samat ennustajat kuin kokeellisissa tuloksissa, mikä oli odotettua. Verrattaessa taulukoiden 16 ja 17 tuloksia havaitaan, että vaikka JPEG:ssä käytetty Huffman-kooditaulu on suunniteltu tyypillisille ennustevirhejakaumille, on tiivistystulos selvästi huonompi kuin esi-

merkiksi aritmeettisella koodaajalla, jolla päästään taulukon 17 tiivistyssuhteisiin.

Saatujen tulosten perusteella näyttäisi siltä, että käyttämällä esimerkiksi JPEG-LS:n ennustajaa tai GAP:a ja koodaamalla ennustevirheet aritmeettisella koodaajalla, päästään parhailla ennustajilla jopa parempiin tuloksiin kuin FELICSillä. JPEG-LS antaa kuitenkin edelleen parhaan tuloksen. Tästä voidaan päätellä, että kontekstimallin yhdistäminen ennustavaan malliin ja kontekstien kvantisointi lisäävät häviöttömissä menetelmissä tiivistystehoa huomattavasti.

Kaikilla menetelmillä päästiin parhaisiin tuloksiin, kun tiivistettävä kuva sisälsi suuria tasaisia alueita kuten *cartman.pgm*. Kuva *baboon.pgm* on hyvä esimerkki kuvasta, jossa vierekkäisten pikseleiden erot ovat suuria. Tämä huonontaa selvästi tiivistystehoa.

7 Yhteenveto

Tässä tutkielmassa luotiin katsaus staattiseen, kontekstipohjaiseen ja ennustavaan mallintamiseen, sekä Huffman-, Golomb-Rice ja aritmeettiseen koodaukseen. Tutkielmassa esiteltiin lisäksi kolme häviötöntä tiivistysmenetelmää: häviötön JPEG, FELICS ja JPEG-LS ja vertailtiin kokeellisesti niiden tiivistystehoja.

Kokeellisessa osuudessa havaittiin, että yksinkertaistakin ennustajaa käytettäessä päästään samoihin tuloksiin kuin huolella valitun ennustajan tapauksessa. Johtopäätös on sama, käytettiinpä ennustevirheiden koodaukseen sitten valmista kooditaulua tai aritmeettista koodausta.

Testitulosten perusteella häviötön JPEG johtaa huonoihin tuloksiin, kun koodauksessa käytetään valmista kooditaulua. Koodaamalla ennustevirheet valmiin kooditaulun sijaan aritmeettisella koodaajalla, päästään häviöttömän JPEG:n parhailla ennustajilla jopa parempiin tuloksiin kuin FELICSillä. JPEG-LS:n käyttämä ennustavan mallin ja kontekstimallin yhdistelmä parantaa tiivistystehoa huomattavasti. JPEG-LS:n käyttö johtikin testikuvien tapauksessa parhaisiin tiivistyssuhteisiin.

Testituloksista käy myös selvästi ilmi, että tiivistettävän kuvan ominaisuudet vaikuttavat tiivistyssuhteeseen. Kuvat, joissa on paljon lähes yksivärisiä alueita tiivistyvät paremmin kuin kuvat, joissa vierekkäisten pikseleiden väliset erot ovat suuria.

Viitteet

Bell, T.C., Moffat¸ A., Witten, I., Zobel, J. (1999) *mg-1.2.1*
`http://www.cs.mu.oz.au/mg/mg-1.2.1.tar.gz` (20.2.2001)

Fränti, P. (1999) *Image Compression*, Luentomoniste, Tietojenkäsittelytieteen laitos, Joensuun yliopisto.

Gallager, R., Voorhis, D.V. (1975) Optimal source codes for geometrically distributed integer alphabets. *IEEE Trans. Inform. Theory* **21**(2), 228-230.

Golomb, S.W. (1966) Run-length encodings. *IEEE Trans. Inform. Theory* **12**(3), 399-401.

Habibi, A. (1971) Comparison of nth order DPCM encoder with linear transformations and block quantization techniques. *IEEE Trans. Commun.* **19**(6), 948-956.

Hewlett-Packard Company (1999) *JPEG-LS Reference Encoder - V.1.00*
`http://www.hpl.hp.com/loco/jlsrefV100.linux.tar.gz` (14.1.2001)

Howard, P., Vitter, J. (1993) Fast and efficient lossless image compression. *Proc. Data compression conference*, Snowbird, Utah USA, 351-360.

Huffman, D.A. (1952) A method for the construction of minimum-redundancy codes. *Proc. Inst. Radio Engineers* **40**(9), 1098-1101.

Pennabaker, W.B., Mitchell, J.L. (1993) *JPEG still image data compression standard*. Van Nostrand Reinhold, New York.

Pesonen, S. (1998) *Kuvittele! Kuvankäsittelyn oppikirja*. Teknolit Oy, Jyväskylä.

Rabbani, M., Jones, P.W. (1991) *Digital Image Compression Techniques*. SPIE - The International Society for Optical Engineering, Bellingham, Washington, USA.

Rice, R.F. (1979) *Some Practical Universal Noiseless Coding Techniques*. Jet Propulsion Laboratory, JPL Publication **79-22**, Pasadena, California.

Rissanen, J., Langdon, G.G. (1979) Arithmetic Coding. *IBM Journal of Research and Development* **23**(2), 149-162.

Shannon, C.E. (1948) A mathematical theory of communication. *Bell Systems Technical Journal* **27**(7,10), 379-423, 623-656.

Tischer, P.E., Worley, R.T., Maeder, A.J., Goodwin, M. (1993) Context-based Lossless Image Compression. *The Computer Journal*, **36**(1), 68-77.

Weinberger, M.J., Seroussi, G., Sapiro, G. (1996a) Effects of resets and number of contexts on the baseline. *ISO/IEC JTC1/SC29/WG1 document N538*.

Weinberger, M.J., Seroussi, G., Sapiro, G. (1996b) LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm. *Proc. Data Compression Conference*, Snowbird, Utah, USA, 140-149.

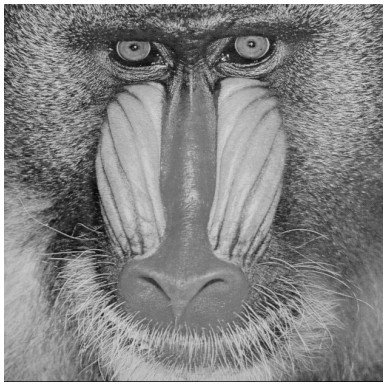
Weinberger, M., Seroussi, G., Sapiro, G. (2000) The LOCO-I Lossless Image Compression Algorithm: Principles and Stantardization into JPEG-LS. *IEEE Trans. Image Processing* **9**(8), 1309-1324.

Witten, I.H., Neal, R.M., Cleary, J.G. (1987) Arithmetic coding for data compression. *Communications of the ACM* **30**(6), 520-540.

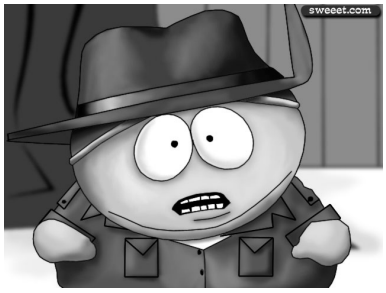
Wu, X. (1996) An algorithmic study on lossless image compression. *Proc. Data Compression Conference*, Snowbird, Utah USA, 150-159.

Liite 1: Testikuvat

ankka.pgm
438 x 289



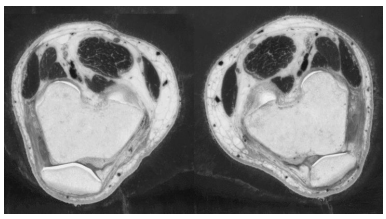
baboon.pgm
512 x 512



cartman.pgm
640 x 480

lena.pgm
512 x 512

kirkko.pgm
236 x 684



thighs.pgm
730 x 399

metsa.pgm
440 x 259