Textones y Clasificadores

Juliana de la Vega Fernández
código 201312387
Departamento de Ingeniería Biomédica
Universidad de los Andes
j.de10@uniandes.edu.co

Abstract

En este artículos se presentan los resultados obtenidos a partir de la implementación del código para entrenar textones en la base de datos del grupo Ponce empleando clasificadores como nearest neighbour (NN) y árboles de decisión empleando las funciones de Matlab fitcknn y TreeBagger respectivamente. Se obtuvieron resultados de 68.00% de cobertura empleando el clasificador de nearest neighbour, y 75.20% de cobertura empleando árboles de decisión.

I. Introducción

Las texturas en las imágenes consisten de elementos repetidos llamados textones. Estos textones pueden ser representados mediante elementos genéricos como puntos y barras. En una imagen estos pequeños elementos se pueden encontrar mediante el filtrado y la posterior representación de la imagen con un resumen de los patrones cercanos. De esta manera, la representación de la imagen en texturas corresponde a la respuesta de una colección de filtros, cada uno de los cuales detecta un subelemento [1]. Para este laboratorio creó un diccionario de textones empleando una base de datos de imágenes del grupo Ponce llamada *Texture Database*. Posteriormente se entrenaron dos clasificadores empleando los histogramas obtenidos en cada imagen luego de emplear el diccionario de textones. Esto se hizo con el objetivo de evaluar qué tan bien funcionaba el diccionario empleado para la base de datos seleccionada.

A. Base de Datos: Texture Database

La base de datos del grupo Ponce emplea 25 clases de texturas, con 40 muestras de cada una. Todas las imagenes están en formato JPG, con una resolución de 640x480 píxeles [2]. La base de datos se divide para tener una base de entrenamiento, sobre la cual se realizan mejoras al algoritmo, y una base de test, con la cual se evalúa el desempeño final del algoritmo desarrollado. Dentro de la base de datos se encuentran imágenes de corteza de árbol de tres tipos, tres tipos de madera, superficies de agua, cristales, cinco estilos de baldosas, dos tipos de paredes, alfombras, prendas de vestir, pelajes, entre otros.

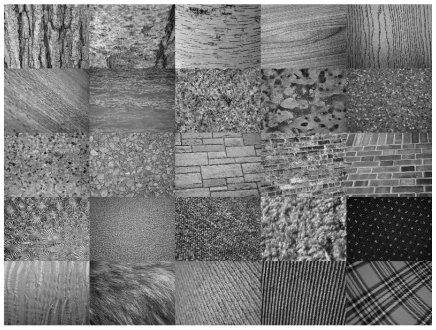

Figura 1: Imágenes de muestra representando todas las texturas en la base de datos *Texture Database*

B. Clasificación Supervisada

Mediante la clasificación supervisada se pretende emplear características de imágenes con categorías conocidas para entrenar un modelo que pueda predecir la categoría de una nueva imagen sin previamente conocerla. Para poder obtener una clasificación, primero se debe entrenar el modelo extrayendo *features* de la imagen, y los datos conocidos o *labels*. Una vez se tenga el modelo, este se puede poner a prueba extrayendo los *features* de una imagen desconocida, y aplicándolo al modelo. El modelo dará una predicción respecto al posible *label*. En este laboratorio se emplearon dos tipos de clasificadores, nearest neighbour y árboles de

decisión.

Nearest Neighbour

Corresponde a la técnica más sencilla de clasificación. Con ella se construye una partición del espacio de representación mediante un diagrama de Voronoi, y posteriormente, al evaluar el modelo con una nueva imagen, a esta se le asigna la categoría del vecino más cercano entre los datos de entrenamiento. Como tal este algoritmo es simple, y fácil de aplicar. En Matlab se puede utilizar la función *fitcknn* la cual recibe como parámetros las características extraídas o *features*, las anotaciones o *labels*, y el número de vecinos a considerar. Con la función se genera un modelo que luego se puede evaluar con las imágenes de prueba.

Árboles de Decisión

Los árboles de decisión se han empleado debido a que son clasificadores eficientes para imágenes con múltiples anotaciones. En estos, el espacio de representación de divide en regiones hiper-rectangulares que se asocian a las hojas de los árboles. Los árboles de decisión dividen la información anotada en categorías lo más homogéneas posibles, con el objetivo de que todos los datos que lleguen a una misma hoja tengan la misma anotación [1]. Como tal son más complejos que el método de nearest neighbour, sin embargo son mucho más poderosos sin sacrificar el tiempo de prueba.

II. Metodología

Con el objetivo de agilizar los métodos empleados, muchos algoritmos propuestos por David R. Martin, de la Universidad de California Berkeley, fueron utilizados en el interior de los algoritmos propios. A continuación se encuentra la descripción detallada de los pasos realizados y los algoritmos empleados en el desarrollo del laboratorio.

A. Diccionario de Textones

Para crear el diccionario de textones se debe primero crear un banco de filtros. Esto se hace mediante la función de David R. Martin llamada *fbCreate*. Con esta se crean 32 filtros, entre los cuales hay 16 de dimensión 13x13 píxeles, y 16 de 19x19 píxeles. Posteriormente, se genera la imagen de referencia, esta está compuesta por 25 imágenes, una de cada categoría, concatenadas horizontalmente. Las imágenes seleccionadas corresponden a las que se observan en la figura 1. Para este método se seleccionó emplear 60 clústeres.

El diccionario de textones se calcula al implementar la función *computeTextons*, que recibe como parámetros: la respuesta de los filtros en la imagen concatenada, y el número de clústeres a evaluar. En dicha función se realiza *k-means* para agrupar las respuestas de los textones en grupos. El resultado que se obtiene es el mapa de textones y el diccionario de textones.

El mismo algoritmo se evaluó para clústeres de tamaños 20, 30, 40 y 50, sin embargo el mejor resultado fue aquel obtenido con 60 clústeres. El diccionario de textones, y no el banco de filtros, es lo que se empleará para categorizar el resto de las imágenes. Para ello se emplea el centroide de cada grupo obtenido al realizar *k-means* sobre la imagen concatenada; estos representaran los nuevos textones.

Para distinguir qué filtro tuvo mayor y cuál tuvo menor respuesta en la evaluación del diccionario, se realiza la evaluación de la desviación estándar de la respuesta de cada filtro. Se obtiene, para el análisis de 60 clústeres, que el filtro con mayor respuesta fue el filtro 9, detectando bordes horizontales. El filtro se puede observar en la figura 2.

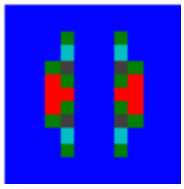

Figura 2: filtro número 9, con la mayor respuesta en la imagen de referencia

Por otro lado, el filtro que demostró la menor respuesta en la imagen de referencia corresponde al filtro 13, que detecta bordes diagonales con inclinación 45 grados. Este se puede observar en la figura 3. Es probable que la mayoría de las imágenes no presentaran una inclinación tan marcada en esa dirección, motivo por el cual su respuesta fue la más disminuida.

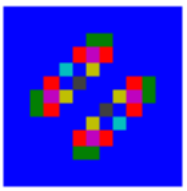

Figura 3: filtro número 13, con la menor respuesta en la imagen de referencia

B. Clasificadores y Distancias Métricas

Como se establecía en la introducción, se emplearon los clasificadores de nearest neighbour y árboles de decisión. Para ambos casos se utilizó el nombre del archivo, { *T01, T02, T03, ..., T25*} como la anotación que describe la textura a la cual pertenece, la verdad terreno. Los features que se seleccionan corresponden a los histogramas de intersección. De la base de entrenamiento se obtienen 30 histogramas por categoría de textura, es decir, 750 en total. Cada uno de estos histogramas cuenta con 60 bins, correspondientes al número de clusters evaluados.

Nearest Neighbour

Para el método de nearest neighbour se evaluó el desempeño para los casos de 8 vecinos cercanos, 4 vecinos cercanos y 1 vecino cercano, buscando determinar cuál de las tres categorías ofrecía el mejor resultado. El código para entrenar se denomina *trainClassifierTextons.m*, en el cual se emplea la función *fitcknn*. El modelo entrenado se evalúa empleando el código *testClassifierNN1Texton.m*. La distancia entre los histogramas se calculó con el kernel de intersección, el cual es una medida adecuada para las distancias entre histogramas. Se ve representado por la siguiente ecuación:

$$I(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i))$$

donde h1 corresponde al primer histograma, y h2 al segundo histograma

Árboles de Decisión

Para los árboles de decisión, la característica de variación fue la cantidad de árboles en el bosque. Se evaluó 25, 50 y 100 árboles en el bosque. El código para entrenar el bosque se denomina *trainTreeBaggerTextons.m* donde se emplea la función *TreeBagger*. El modelo entrenado se evalúa sobre la base de datos de prueba empleando el código *testClassifierTrees100Texton.m*.

III. Resultados

A continuación se presentan los resultados obtenidos con los modelos entrenados para los clasificadores de nearest neighbour y árboles de decisión.

A. Nearest Neighbour

Como se ha mencionado anteriormente, el entrenamiento del clasificador Nearest Neighbour se realizó mediante la implementación de la función *fitcknn*. La evaluación de la clasificación se realizó con la función *resubLoss*, la cual calcula el error de regresión mediante la resustitución. Adicionalmente, se emplea la función *crossval* y *kfoldLoss* para medir la pérdida de cross validación el realizar nearest neighbours. En la tabla 1 se pueden observar los resultados obtenidos para el método de nearest neighbours evaluando las pérdidas de resustitución y de cross validación empleando la base de datos de entrenamiento.

Basados en los resultados anteriores, se eligió el modelo de 1 vecino cercano para evaluar las predicciones en la base de datos de prueba. EL código se implementó en el archivo *testClassifierNN1Texton.m* en el cual a cada imagen de prueba se le pasa el diccionario de textones empleado, y se obtiene el histograma correspondiente. Posteriormente se pasa por el modelo empleando la función *predict*, la cual recibe como parámetros el modelo generado con los datos de entrenamiento, y los histogramas de test. Seguidamente, se evalúan las predicciones contra la verdad terreno, y se obtiene el recall del método. Las predicciones tomaron 0.3985 segundos en realizarse, de 250 imágenes el modelo detectó 170, por lo cual la cobertura que ofrece es de 68.00%. En la figura 4 se puede observar la matriz de confusiones generada a partir de las predicciones y la verdad terreno al emplear el modelo entrenado mediante nearest neighbours.

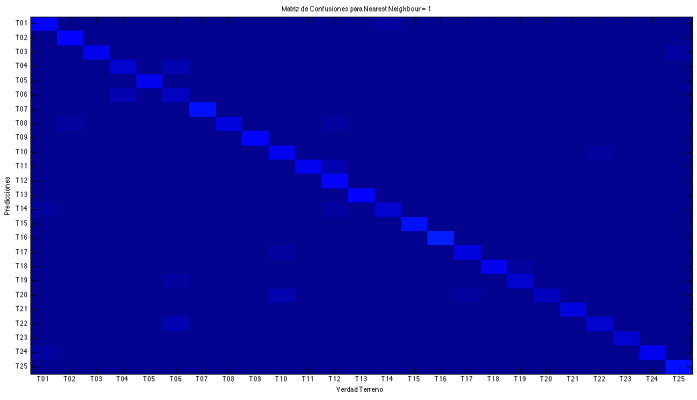

Figura 4: Matriz de confusiones para las predicciones obtenidas empleando el clasificador Nearest Neighbour

B. Árboles de Decisión

En entrenamiento del clasificador de árboles de decisión se realizó empleando la función anteriormente mencionada, *TreeBagger*. Esta se implementó en el código *trainTreeBaggerTextons.m*. Una vez entrenado el mod-

Tabla 1: Resultados para los tiempos de entrenamiento, errores de resustitución y cross validación con la base de datos de entrenamiento.

Nearest Neighbours	*Tiempo transcurrido (s)*	*Pérdidas de resustitución (%)*	*Pérdidas de cross-validación (%)*
8	0.5416	33.07%	45.47%
4	3.5753	22.40%	40.13%
2	0.5580	21.73%	41.33%
1	0.5576	0.00%	34.40%

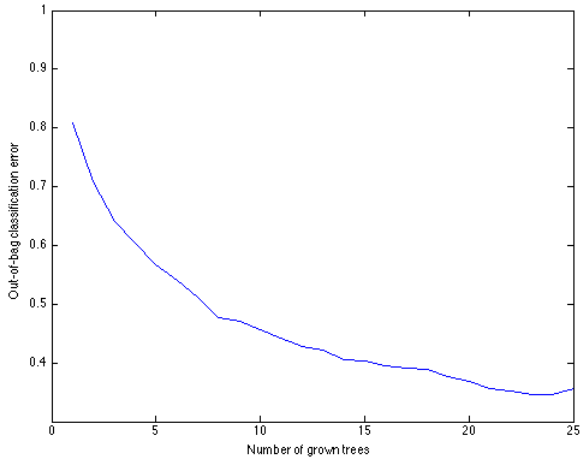

Figura 5: Error *out-of-bag* para el modelo entrenado con 25 árboles

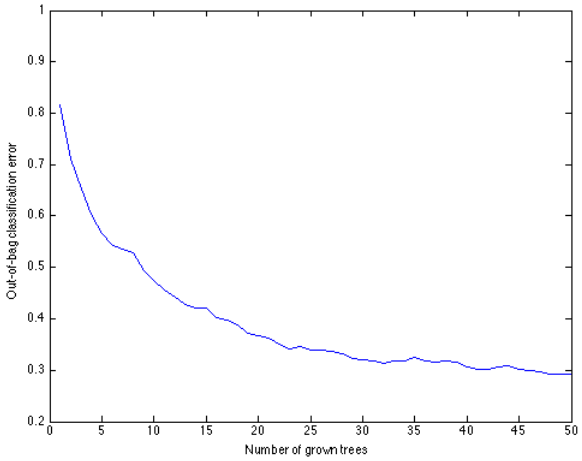

Figura 6: Error *out-of-bag* para el modelo entrenado con 50 árboles

elo, se calcula el *out of bag error* sobre el número de árboles crecidos. A continuación se presentan los errores *out of bag* para los modelos con 25, 50 y 100 árboles.

No se realizaron modelos con más árboles debido a que la curva alcanzó un mínimo local que se mantuvo hasta los 100 árboles. Adicionalmente, se descartó el modelo de 25 árboles debido a la gran varianza que presentaba en el error *out-of-bag*. Posteriormente, se evaluó el modelo de 100 árboles con la base de datos de prueba. Se implementó dicha evaluación con el código *testClassifierTrees100Texton.m*. Como en el caso de los nearest neighbours, aquí se aplicó el diccionario de textones a cada imagen de la base de pruebas obteniendo los histogramas de cada una. Luego se obtienen las predicciones aplicando los histogramas al modelo empleando la función *predict*. De 250 imágenes, el modelo detecto 188 adecuadamente, lo cual indica una cobertura de 75.20%, la cual supera el modelo anterior. Para evaluar las predicciones se elaboró una matriz de confusiones, esta se puede observar en la figura 8.

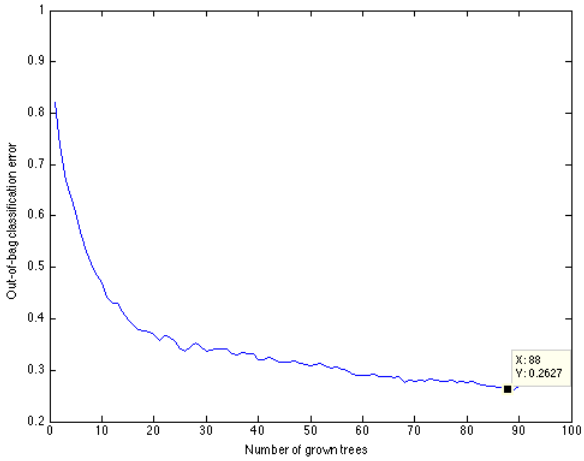

Figura 7: Error *out-of-bag* para el modelo entrenado con 100 árboles

IV. Discusión y Conclusiones

Luego de evaluar dos métodos de clasificación, es posible compararlos siguiendo los resultados obtenidos al

Tabla 2: Tiempos de entrenamiento para todos los modelos entrenados

Modelo	*Tiempo de entrenamiento (s)*	*Modelo*	*Tiempo de entrenamiento (*
Nearest Neighbour (1 vecino)	0.5576	Bosque de decisión (25 árboles)	4.7289
Nearest Neighbour (2 vecinos)	0.5580	Bosque de decisión (50 árboles)	4.4281
Nearest Neighbour (4 vecinos)	3.5753	Bosque de decisión (100 árboles)	5.9895
Nearest Neighbour (8 vecinos)	0.5576		

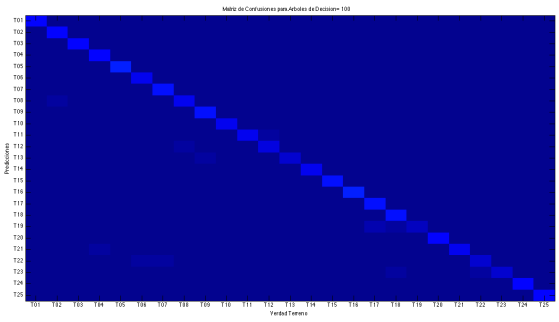

Figura 8: Matriz de confusiones para las predicciones obtenidas empleando el clasificador de árboles de decisión

Figura 9: Imagen $T06_21 delabasededatosdeprueba$

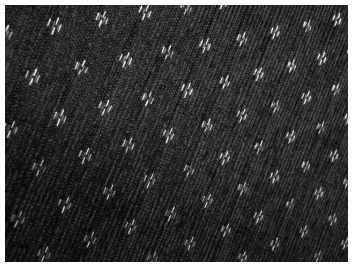

Figura 10: Imagen $T20_20 delabasededatosdeprueba$

emplear la base de datos de prueba. De acuerdo a los resultados de la cobertura, el mejor modelo de clasificación es aquel obtenido con los árboles de decisión, específicamente, el modelo que emplea un bosque de 100 árboles, con cobertura de 75.20%, superando al método de Nearest Neighbour, con un vecino, que tiene cobertura de 68.00%. Los tiempos de entrenamiento se pueden observar en la tabla 2.

Aunque los tiempos de entrenamiento son mayores para los modelos de árboles de decisión, los resultados de estos superan en al menos 7% la cobertura del mejor método de clasificación con nearest neighbour. Los tiempos de clasificación de la base de datos de prueba corresponden a 0.9136 segundos para el modelo de clasificación de 100 árboles de decisión, y 0.3985 segundos para el modelo de nearest neighbours con 1 vecino. Aun si en el momento de prueba del modelo los clasificadores del vecino más cercano son más veloces y si el clasificador de árboles de decisión triplican el tiempo, los resultados son mucho mas satisfactorios, con un menor error. El error del método con un vecino cercano es del 32%, mientras que el modelo con 100 árboles de decisión presenta un error del 24.80%, demostrando su superioridad.

Observando las matrices de confusión, se puede observar que la categoría con más confusión en el modelo del vecino más cercano corresponde a la T20 y la T06. Estas se pueden observar en las figuras 9 y 10. Como tal corresponden a las texturas de un tipo de madera y un patron de una prenda de vestir.

Para la matriz de confusión del modelo de 100 árboles de decisión se observa que la categoría con mayor confusión corresponde a la T19, un ejemplo de ella de observa en la figura 11, la cual corresponde a la textura de una alfombra. Para ambos modelos, las texturas con mayor confusión solo presentan 4 detecciones correctas, entre las 10 imágenes que hay por categoría. Es decir, en estas categorías hay un 60% de error, lo cual quiere decir que el diccionario de textones no la está representando adecuadamentente.

Como tal, los métodos empleados tienen varias limitaciones. El modelos de nearest neighbour es muy sencillo para describir esta base de datos, los límites de cada categoría son muy sensibles al ruido en las anotaciones y a la estructura local. El modelo de árboles de decisión también tiene varias limitaciones. Este puede hacer un *overfitting* de los datos, obteniendo una clasificación muy buena en la base de datos de entrenamiento, lo cual lleva a una clasificación de menor pre-

Figura 11: Imagen $T20_20 delabasededatosdeprueba$

cisión en la base de datos de prueba. Adicionalmente, para los árboles de decisión, es importante adecuar los parámetros de profundidad, número de árboles, la aleatoriedad de los mismos, y el *weak learner*, lo cual puede hacer que sea dispendiosa su utilización.

En cuanto a la base de datos de prueba, una de sus limitaciones corresponde a las múltiples escalas que manejan de las texturas, algunas son bastante cercanas, mientras otras son lejanas, lo cual puede cambiar la respuesta de los textones, pues la textura reconocida puede cambiar con la distancia empleada. Adicionalmente, se emplean múltiples orientaciones en las imágenes, lo cual también puede ocasionar ruido en la clasificación. En cuanto al tamaño de la base de datos, el entrenamiento podría haber sido más preciso si existieran más de 30 imágenes por categoría para entrenar. Además, las imágenes no se distribuyen de manera homogénea entre la base de prueba y la base de entrenamiento.

El método podría ser mejorado empleando un mayor número de imágenes en el entrenamiento. Podría ser útil tener mayor información sober las imágenes, como su color, lo cual ayudaría en su clasificación. También se podría mejorar los árboles de decisión al modificar los parámetros como profundidad. Para ambos métodos de clasificación podría ser útil emplear un mayor número de textones, con más orientaciones. Esto permitiría tener un diccionario de textones más completo. Sería posible también emplear diferentes tipos de clasificadores, como support vector machines.

Referencias

[1] D. Forsyth, J. Ponce, "Chapter 3: Color" in Computer Vision: A Modern Approach, Upper Saddle River: Pearson, 2003, pp. 68-83.

[2] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A Sparse Texture Representation Using Local Affine Regions. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1265-1278, August 2005.

[3] R. Szeliski, "Chapter 5: Segmentation," in Computer Vision: Algorithm and Applications, Springer, 2010, pp.235-253.