Chromagram-Only (3 secs) CNN

March 21, 2025

1 CNN for Chromagram only (3 secs)

1.1 1 - All the imports

[1]:
```
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
2025-03-21 01:32:31.211517: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.2 2 - Put the data within the model

[2]:
```
# Import a single image and save it to be read by the model

image = os.path.join('blues.00000.png')

# Load the image
image = tf.io.read_file(image)

# Convert to a numpy array
image = tf.image.decode_png(image, channels=1)
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize(image, [256, 256])
image = image.numpy()
```

2 3 - Create the model

[3]:
```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout

model = models.Sequential([
    Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[4]:
```
model.summary()
```

```
Model: "sequential"


 Layer (type)                      Output Shape                  Param #

 conv2d (Conv2D)                   (None, 34, 254, 36)               360

 max_pooling2d (MaxPooling2D)      (None, 17, 127, 36)                 0

 flatten (Flatten)                 (None, 77724)                       0

 dense (Dense)                     (None, 512)                39,795,200

 dropout (Dropout)                 (None, 512)                         0

 dense_1 (Dense)                   (None, 256)                   131,328

 dropout_1 (Dropout)               (None, 256)                         0

 dense_2 (Dense)                   (None, 128)                    32,896

 dense_3 (Dense)                   (None, 10)                      1,290
```

```
Total params: 39,961,074 (152.44 MB)

Trainable params: 39,961,074 (152.44 MB)

Non-trainable params: 0 (0.00 B)
```

3 4 - Load the images

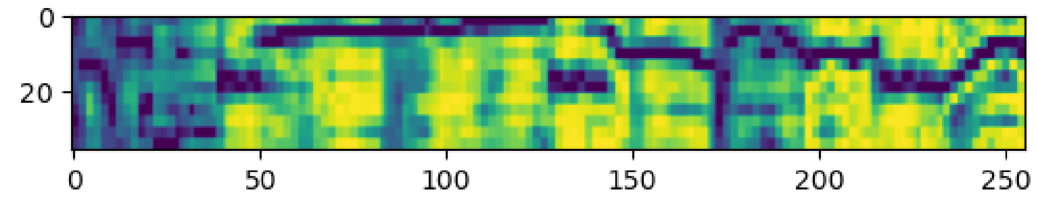
[5]:
```
# Load the images from Data/spectrograms/spectrograms_256

# Data/spectrograms/spectrogram_256

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']

# GENRES = ["classical", "hiphop", "jazz", "metal", "pop", "rock"]
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_36')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            image = tf.io.read_file(os.path.join(genre_dir, file))
            image = tf.image.decode_png(image, channels=1)
            image = tf.image.convert_image_dtype(image, tf.float32)
            image = tf.image.resize(image, [36, 256])
            image = image.numpy()
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            continue

X = np.array(X)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
Going through blues
Going through classical
```

```
2025-03-21 01:33:23.047681: W tensorflow/core/framework/op_kernel.cc:1841]
OP_REQUIRES failed at whole_file_read_ops.cc:116 : FAILED_PRECONDITION:
Data/chromagrams (3 secs)/chromagram_36/classical/.ipynb_checkpoints; Is a
directory
2025-03-21 01:33:23.049408: I tensorflow/core/framework/local_rendezvous.cc:405]
Local rendezvous is aborting with status: FAILED_PRECONDITION: Data/chromagrams
(3 secs)/chromagram_36/classical/.ipynb_checkpoints; Is a directory
```

```
Going through country
Going through disco
Going through hiphop
Going through jazz
Going through metal
Going through pop
Going through reggae
Going through rock
```

[6]:
```python
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[22].reshape(36, 256))
plt.show()
```



[7]:
```python
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[8]:
```python
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
↪batch_size=32)
```

```
Epoch 1/20
250/250               155s 592ms/step -
accuracy: 0.0959 - loss: 2.9297 - val_accuracy: 0.0950 - val_loss: 2.3025
Epoch 2/20
250/250               138s 550ms/step -
accuracy: 0.1186 - loss: 2.2965 - val_accuracy: 0.2050 - val_loss: 2.1821
Epoch 3/20
250/250               138s 554ms/step -
accuracy: 0.2096 - loss: 2.1142 - val_accuracy: 0.2855 - val_loss: 1.9025
Epoch 4/20
250/250               137s 532ms/step -
```

```
accuracy: 0.2746 - loss: 1.9164 - val_accuracy: 0.3100 - val_loss: 1.8868
Epoch 5/20
250/250             107s 429ms/step -
accuracy: 0.3185 - loss: 1.8377 - val_accuracy: 0.3175 - val_loss: 1.8827
Epoch 6/20
250/250             143s 435ms/step -
accuracy: 0.3631 - loss: 1.7096 - val_accuracy: 0.3160 - val_loss: 1.8772
Epoch 7/20
250/250             107s 426ms/step -
accuracy: 0.4256 - loss: 1.5745 - val_accuracy: 0.3100 - val_loss: 1.8878
Epoch 8/20
250/250             98s 392ms/step -
accuracy: 0.4966 - loss: 1.3979 - val_accuracy: 0.3230 - val_loss: 1.9587
Epoch 9/20
250/250             138s 375ms/step -
accuracy: 0.5668 - loss: 1.2142 - val_accuracy: 0.3165 - val_loss: 2.0635
Epoch 10/20
250/250             91s 365ms/step -
accuracy: 0.6191 - loss: 1.0630 - val_accuracy: 0.3300 - val_loss: 2.1368
Epoch 11/20
250/250             89s 357ms/step -
accuracy: 0.6754 - loss: 0.9298 - val_accuracy: 0.3140 - val_loss: 2.2578
Epoch 12/20
250/250             89s 355ms/step -
accuracy: 0.7229 - loss: 0.8070 - val_accuracy: 0.3165 - val_loss: 2.3047
Epoch 13/20
250/250             89s 357ms/step -
accuracy: 0.7465 - loss: 0.7253 - val_accuracy: 0.3080 - val_loss: 2.4939
Epoch 14/20
250/250             82s 328ms/step -
accuracy: 0.7753 - loss: 0.6544 - val_accuracy: 0.3150 - val_loss: 2.4151
Epoch 15/20
250/250             89s 355ms/step -
accuracy: 0.8120 - loss: 0.5737 - val_accuracy: 0.3150 - val_loss: 2.5440
Epoch 16/20
250/250             148s 381ms/step -
accuracy: 0.8338 - loss: 0.5032 - val_accuracy: 0.3155 - val_loss: 2.7626
Epoch 17/20
250/250             96s 385ms/step -
accuracy: 0.8218 - loss: 0.5078 - val_accuracy: 0.3165 - val_loss: 2.6264
Epoch 18/20
250/250             99s 395ms/step -
accuracy: 0.8385 - loss: 0.4719 - val_accuracy: 0.3050 - val_loss: 2.6901
Epoch 19/20
250/250             97s 390ms/step -
accuracy: 0.8462 - loss: 0.4494 - val_accuracy: 0.2995 - val_loss: 2.7719
Epoch 20/20
250/250             96s 383ms/step -
```

```
accuracy: 0.8517 - loss: 0.4369 - val_accuracy: 0.3095 - val_loss: 2.7356
```

[8]:
```
<keras.src.callbacks.history.History at 0x7fa7846536b0>
```

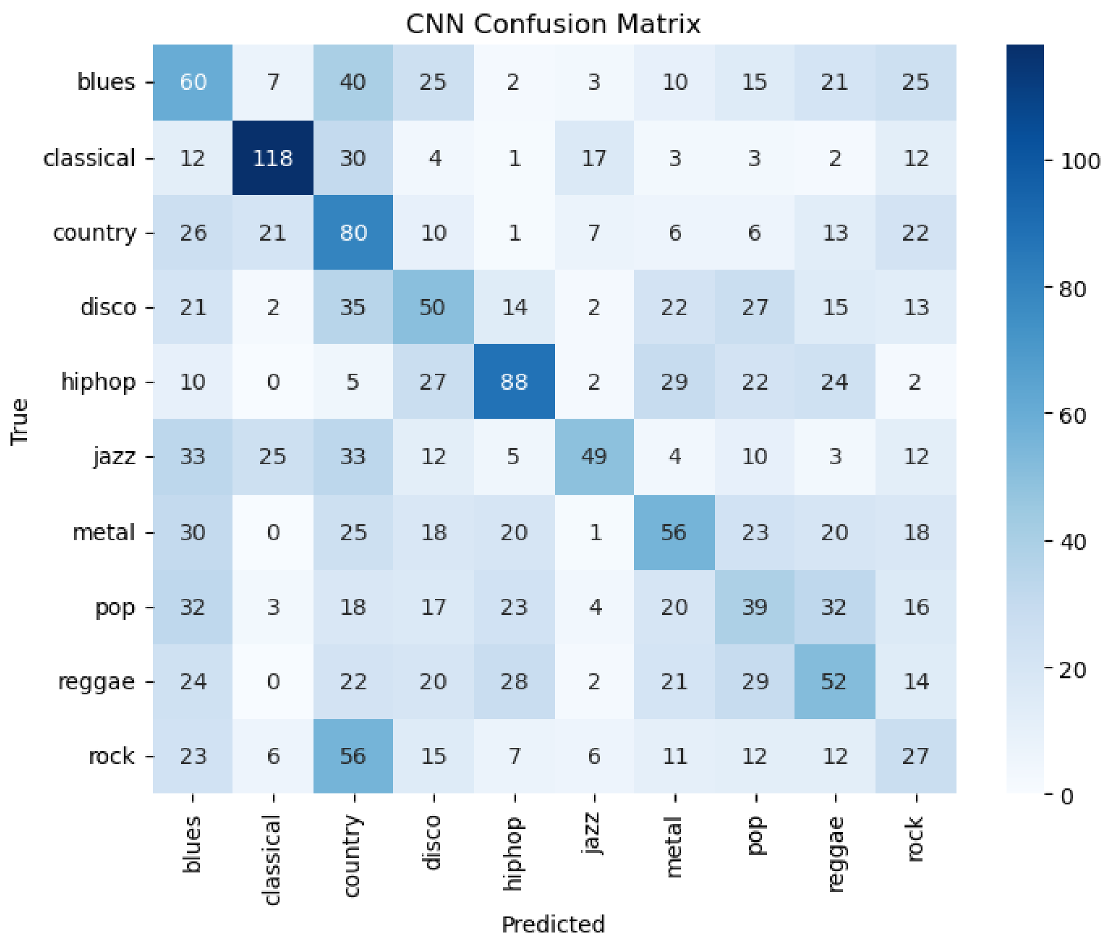
[9]:
```python
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63              3s 39ms/step -
accuracy: 0.3081 - loss: 2.7165
Test accuracy: 0.310
```

4 Apply the confusion matrix after the model

[10]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
63/63              3s 43ms/step
```



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	60	7	40	25	2	3	10	15	21	25
classical	12	118	30	4	1	17	3	3	2	12
country	26	21	80	10	1	7	6	6	13	22
disco	21	2	35	50	14	2	22	27	15	13
hiphop	10	0	5	27	88	2	29	22	24	2
jazz	33	25	33	12	5	49	4	10	3	12
metal	30	0	25	18	20	1	56	23	20	18
pop	32	3	18	17	23	4	20	39	32	16
reggae	24	0	22	20	28	2	21	29	52	14
rock	23	6	56	15	7	6	11	12	12	27

4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_36')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Define the augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            image = tf.io.read_file(os.path.join(genre_dir, file))
            image = tf.image.decode_png(image, channels=1)
            image = tf.image.convert_image_dtype(image, tf.float32)
            image = tf.image.resize(image, [36, 256])

            # Apply the augmentation
            image = augment_image(image)
            image = image.numpy()  # Convert to numpy array for further␣
 ↪processing
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            continue

X = np.array(X)
y = np.array(y)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

model = models.Sequential([
    Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
Going through classical
```

```
2025-03-21 02:14:34.543518: W tensorflow/core/framework/op_kernel.cc:1841]
OP_REQUIRES failed at whole_file_read_ops.cc:116 : FAILED_PRECONDITION:
Data/chromagrams (3 secs)/chromagram_36/classical/.ipynb_checkpoints; Is a
directory
2025-03-21 02:14:34.544324: I tensorflow/core/framework/local_rendezvous.cc:405]
Local rendezvous is aborting with status: FAILED_PRECONDITION: Data/chromagrams
(3 secs)/chromagram_36/classical/.ipynb_checkpoints; Is a directory
```

```
Going through metal
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
50/50                25s 413ms/step -
accuracy: 0.4213 - loss: 1.3227 - val_accuracy: 0.5025 - val_loss: 0.7105 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                20s 402ms/step -
accuracy: 0.5327 - loss: 0.8728 - val_accuracy: 0.8000 - val_loss: 0.6565 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                20s 390ms/step -
accuracy: 0.5543 - loss: 0.7757 - val_accuracy: 0.6150 - val_loss: 0.6156 -
learning_rate: 1.0000e-04
Epoch 4/20
```

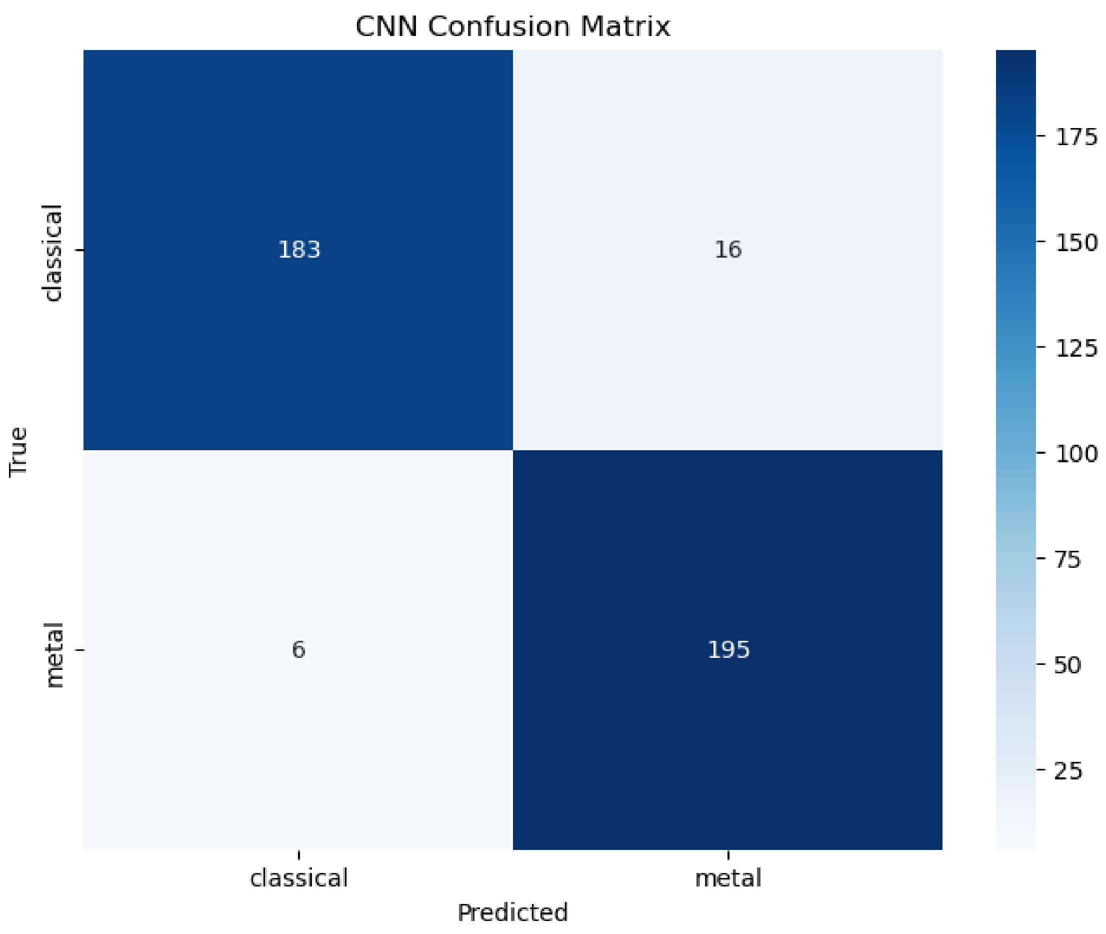
```
50/50            20s 391ms/step -
accuracy: 0.6217 - loss: 0.6781 - val_accuracy: 0.8425 - val_loss: 0.4753 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            20s 393ms/step -
accuracy: 0.7469 - loss: 0.5461 - val_accuracy: 0.8550 - val_loss: 0.3767 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            21s 398ms/step -
accuracy: 0.7789 - loss: 0.4846 - val_accuracy: 0.8975 - val_loss: 0.3276 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            20s 391ms/step -
accuracy: 0.8186 - loss: 0.4293 - val_accuracy: 0.8675 - val_loss: 0.3458 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            20s 398ms/step -
accuracy: 0.8557 - loss: 0.3809 - val_accuracy: 0.8900 - val_loss: 0.2648 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            21s 413ms/step -
accuracy: 0.8838 - loss: 0.3272 - val_accuracy: 0.8950 - val_loss: 0.2522 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            40s 396ms/step -
accuracy: 0.8713 - loss: 0.3129 - val_accuracy: 0.9075 - val_loss: 0.2462 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            20s 393ms/step -
accuracy: 0.8790 - loss: 0.3248 - val_accuracy: 0.9100 - val_loss: 0.2214 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            19s 389ms/step -
accuracy: 0.8992 - loss: 0.2880 - val_accuracy: 0.9025 - val_loss: 0.2780 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            20s 389ms/step -
accuracy: 0.9028 - loss: 0.2454 - val_accuracy: 0.9200 - val_loss: 0.2314 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            20s 393ms/step -
accuracy: 0.9179 - loss: 0.2187 - val_accuracy: 0.9300 - val_loss: 0.1956 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50            20s 394ms/step -
accuracy: 0.9249 - loss: 0.2000 - val_accuracy: 0.9300 - val_loss: 0.1905 -
learning_rate: 1.0000e-04
Epoch 16/20
```

```
50/50              19s 386ms/step -
accuracy: 0.9382 - loss: 0.1914 - val_accuracy: 0.9375 - val_loss: 0.1900 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50              19s 387ms/step -
accuracy: 0.9344 - loss: 0.1598 - val_accuracy: 0.9375 - val_loss: 0.1862 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50              19s 385ms/step -
accuracy: 0.9412 - loss: 0.1643 - val_accuracy: 0.9375 - val_loss: 0.1805 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50              20s 389ms/step -
accuracy: 0.9444 - loss: 0.1493 - val_accuracy: 0.9375 - val_loss: 0.1902 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50              20s 389ms/step -
accuracy: 0.9638 - loss: 0.1201 - val_accuracy: 0.9450 - val_loss: 0.1807 -
learning_rate: 1.0000e-04
13/13              0s 35ms/step -
accuracy: 0.9407 - loss: 0.1873
Test accuracy: 0.945
```

4.2 10 - Confusion Matrix Easy (classical and metal)

[12]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
  ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13              1s 61ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[13]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_36')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Define the augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            image = tf.io.read_file(os.path.join(genre_dir, file))
            image = tf.image.decode_png(image, channels=1)
            image = tf.image.convert_image_dtype(image, tf.float32)
            image = tf.image.resize(image, [36, 256])

            # Apply the augmentation
            image = augment_image(image)

            image = image.numpy()  # Convert to numpy array for further␣
 ↪processing
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            continue

X = np.array(X)
y = np.array(y)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

model = models.Sequential([
    Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
Going through disco
Going through pop
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
50/50                 25s 411ms/step -
accuracy: 0.4326 - loss: 1.2649 - val_accuracy: 0.5025 - val_loss: 0.7963 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                 20s 405ms/step -
accuracy: 0.4850 - loss: 0.8818 - val_accuracy: 0.5025 - val_loss: 0.7984 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                 20s 390ms/step -
accuracy: 0.4534 - loss: 0.8567 - val_accuracy: 0.5025 - val_loss: 0.7161 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50                 19s 386ms/step -
accuracy: 0.4498 - loss: 0.8016 - val_accuracy: 0.5025 - val_loss: 0.7180 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50                 19s 382ms/step -
accuracy: 0.4971 - loss: 0.7629 - val_accuracy: 0.5025 - val_loss: 0.7033 -
learning_rate: 1.0000e-04
```

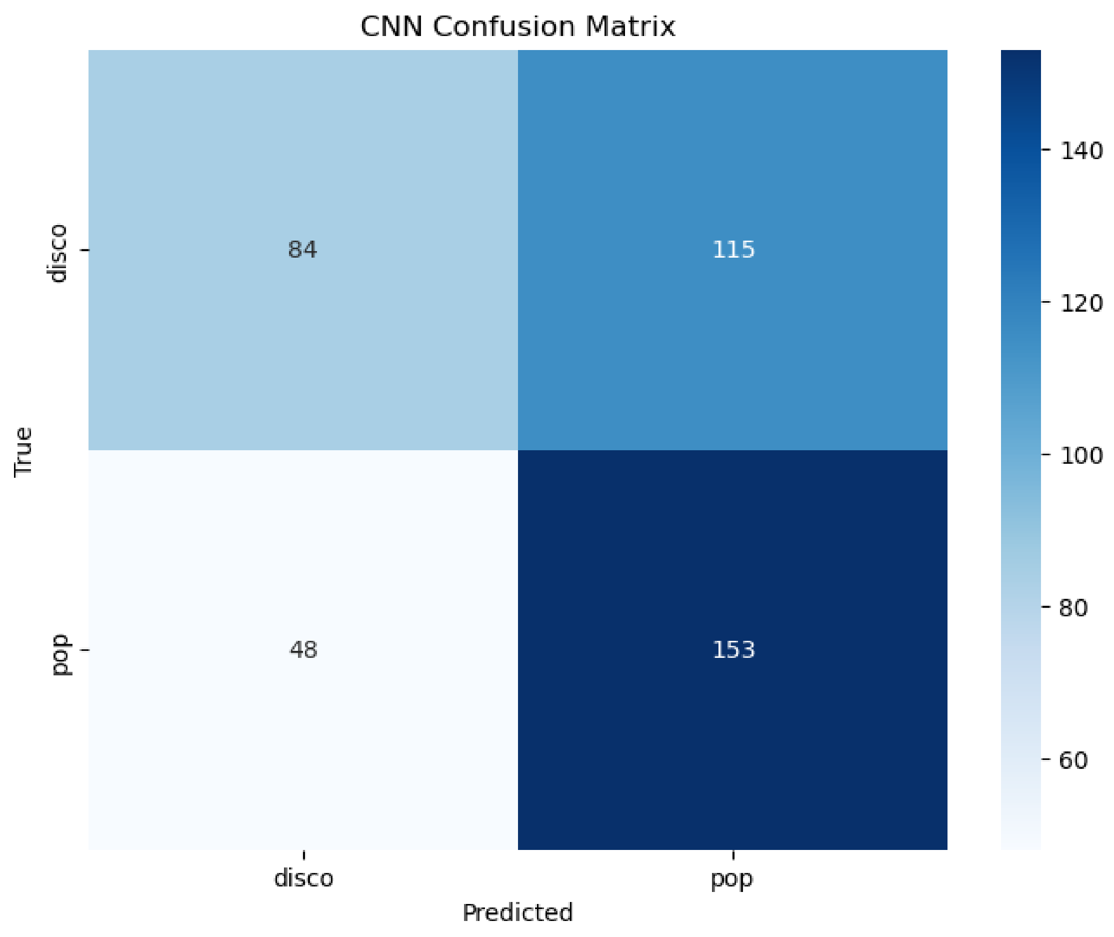
```
Epoch 6/20
50/50            19s 381ms/step -
accuracy: 0.4971 - loss: 0.7610 - val_accuracy: 0.5025 - val_loss: 0.7087 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            20s 407ms/step -
accuracy: 0.5160 - loss: 0.7472 - val_accuracy: 0.5025 - val_loss: 0.7054 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            19s 374ms/step -
accuracy: 0.5173 - loss: 0.7398 - val_accuracy: 0.5025 - val_loss: 0.7027 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            19s 388ms/step -
accuracy: 0.5358 - loss: 0.7305 - val_accuracy: 0.5025 - val_loss: 0.6996 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            20s 390ms/step -
accuracy: 0.4941 - loss: 0.7484 - val_accuracy: 0.5650 - val_loss: 0.6990 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            19s 388ms/step -
accuracy: 0.5179 - loss: 0.7162 - val_accuracy: 0.5175 - val_loss: 0.6963 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            19s 387ms/step -
accuracy: 0.5017 - loss: 0.7140 - val_accuracy: 0.5225 - val_loss: 0.6958 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            19s 387ms/step -
accuracy: 0.5446 - loss: 0.7054 - val_accuracy: 0.5100 - val_loss: 0.6922 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            19s 386ms/step -
accuracy: 0.5028 - loss: 0.7133 - val_accuracy: 0.5975 - val_loss: 0.6925 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50            19s 380ms/step -
accuracy: 0.5150 - loss: 0.7112 - val_accuracy: 0.5050 - val_loss: 0.6951 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50            19s 379ms/step -
accuracy: 0.5369 - loss: 0.7132 - val_accuracy: 0.5050 - val_loss: 0.7002 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50            19s 380ms/step -
accuracy: 0.5280 - loss: 0.7078 - val_accuracy: 0.5350 - val_loss: 0.6903 -
learning_rate: 5.0000e-05
```

```
Epoch 18/20
50/50                21s 384ms/step -
accuracy: 0.5340 - loss: 0.7086 - val_accuracy: 0.5975 - val_loss: 0.6853 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50                20s 384ms/step -
accuracy: 0.5186 - loss: 0.7116 - val_accuracy: 0.5350 - val_loss: 0.6904 -
learning_rate: 5.0000e-05
Epoch 20/20
50/50                19s 382ms/step -
accuracy: 0.5348 - loss: 0.6986 - val_accuracy: 0.5925 - val_loss: 0.6843 -
learning_rate: 5.0000e-05
13/13                1s 45ms/step -
accuracy: 0.5960 - loss: 0.6833
Test accuracy: 0.592
```

4.4 12 - Confusion Matrix Hard (disco and pop)

[14]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13                1s 51ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[15]:
```python
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split
import random

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_36')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}
```

```
# Define the augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            image = tf.io.read_file(os.path.join(genre_dir, file))
            image = tf.image.decode_png(image, channels=1)
            image = tf.image.convert_image_dtype(image, tf.float32)
            image = tf.image.resize(image, [36, 256])

            # Apply the augmentation
            image = augment_image(image)

            image = image.numpy()  # Convert to numpy array for further␣
 ↪processing
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            continue

X = np.array(X)
y = np.array(y)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

model = models.Sequential([
    Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
['country', 'disco', 'metal', 'blues', 'reggae']
Going through country
Going through disco
Going through metal
Going through blues
Going through reggae
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
125/125               53s 399ms/step -
accuracy: 0.1907 - loss: 1.9753 - val_accuracy: 0.1920 - val_loss: 1.6914 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125               48s 379ms/step -
accuracy: 0.2050 - loss: 1.7605 - val_accuracy: 0.2080 - val_loss: 1.6505 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125               49s 393ms/step -
accuracy: 0.2205 - loss: 1.7022 - val_accuracy: 0.2250 - val_loss: 1.6474 -
learning_rate: 1.0000e-04
```

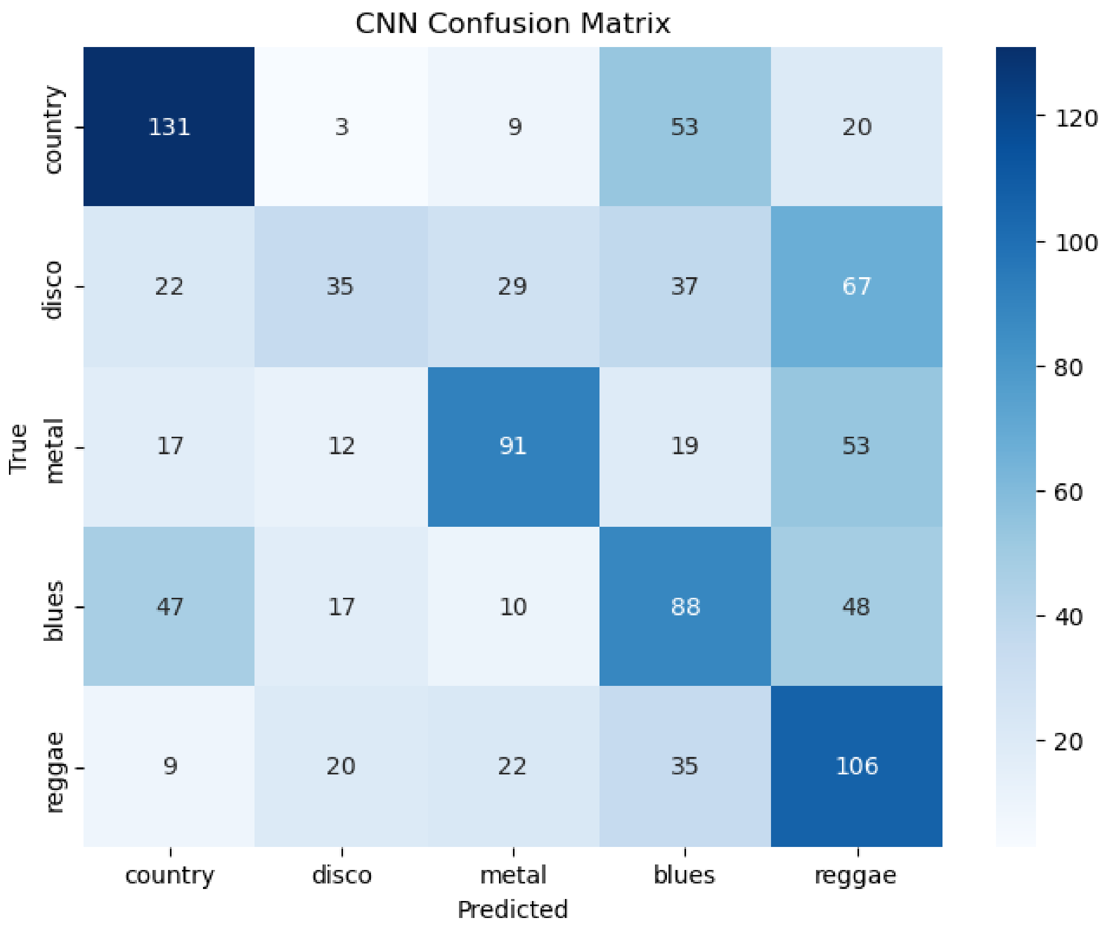
```
Epoch 4/20
125/125            48s 384ms/step -
accuracy: 0.2168 - loss: 1.6759 - val_accuracy: 0.2770 - val_loss: 1.6249 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125            47s 377ms/step -
accuracy: 0.2292 - loss: 1.6626 - val_accuracy: 0.2880 - val_loss: 1.6057 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125            49s 391ms/step -
accuracy: 0.2419 - loss: 1.6304 - val_accuracy: 0.3330 - val_loss: 1.5785 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125            80s 378ms/step -
accuracy: 0.2552 - loss: 1.6055 - val_accuracy: 0.3070 - val_loss: 1.5637 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125            48s 380ms/step -
accuracy: 0.2801 - loss: 1.5782 - val_accuracy: 0.3610 - val_loss: 1.5164 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125            47s 377ms/step -
accuracy: 0.3037 - loss: 1.5508 - val_accuracy: 0.3930 - val_loss: 1.4909 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125            46s 364ms/step -
accuracy: 0.3361 - loss: 1.5201 - val_accuracy: 0.4060 - val_loss: 1.4735 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125            47s 375ms/step -
accuracy: 0.3456 - loss: 1.4741 - val_accuracy: 0.3930 - val_loss: 1.4166 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125            83s 383ms/step -
accuracy: 0.3823 - loss: 1.4289 - val_accuracy: 0.4070 - val_loss: 1.4535 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125            47s 375ms/step -
accuracy: 0.4087 - loss: 1.3931 - val_accuracy: 0.4410 - val_loss: 1.3855 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125            47s 373ms/step -
accuracy: 0.4191 - loss: 1.3600 - val_accuracy: 0.4220 - val_loss: 1.3772 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125            47s 375ms/step -
accuracy: 0.4203 - loss: 1.3560 - val_accuracy: 0.4320 - val_loss: 1.3468 -
learning_rate: 1.0000e-04
```

```
Epoch 16/20
125/125              48s 385ms/step -
accuracy: 0.4393 - loss: 1.3217 - val_accuracy: 0.4250 - val_loss: 1.3481 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125              47s 376ms/step -
accuracy: 0.4688 - loss: 1.2730 - val_accuracy: 0.4300 - val_loss: 1.3369 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125              84s 389ms/step -
accuracy: 0.4782 - loss: 1.2577 - val_accuracy: 0.4370 - val_loss: 1.3250 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125              48s 380ms/step -
accuracy: 0.4893 - loss: 1.2206 - val_accuracy: 0.4420 - val_loss: 1.3261 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125              47s 379ms/step -
accuracy: 0.5065 - loss: 1.1991 - val_accuracy: 0.4510 - val_loss: 1.3250 -
learning_rate: 1.0000e-04
32/32              1s 44ms/step -
accuracy: 0.4486 - loss: 1.3139
Test accuracy: 0.451
```

4.6 14 - Confusion Matrix Medium (5 random)

[16]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
32/32              2s 47ms/step
```


CNN Confusion Matrix
True
Predicted
country
disco
metal
blues
reggae
131 3 9 53 20
22 35 29 37 67
17 12 91 19 53
47 17 10 88 48
9 20 22 35 106
120
100
80
60
40
20