Midterm 1 - Deep Learning - Spring 2025

Essay Question: Write a one paragraph essay discussing *how graph convolutional networks solve the problem of having a limited amount of labeled data samples.*

Graph Convolutional Networks (GCN) solve the problem of limited labeled data samples for datasets where different inputs are related in a way that can be characterized using a graph (directed or undirected). The model uses the graph's incidence matrix to provide side information that allows the model to transfer knowledge from labeled nodes in the dataset to unlabeled nodes. This aspect of GCNs allows them to be used for node and edge prediction problems. They have demonstrated their utility in social networking applications. GCNs therefore provide a way to transfer knowledge from labeled samples to unlabeled samples in domains where sample relationships are described by a graph.

Problem 1: Consider a learning-by-example problem that trains a model that takes a journal article whose text is one-hot encoded with respect to a dictionary of 1500 words and then outputs an estimate of the probability that the article belongs to a given class (science, sports, entertainment, etc.). Assume the article classes are one-hot encoded and that your model outputs probability estimates for 7 distinct classes.

- Let $\{(x_k, y_k)\}_{k=1}^{N}$ denote a finite dataset created by the system's generator and observer. Formally describe the input samples, x_k, and target samples, y_k.
- There are several model sets you can choose for this problem. Select one and formally describe it.
- Select a loss function for your problem, write out the equation for the loss function, and explain what the variables are in the equation.

Solution:

- The generator randomly draws binary input vectors, x_k, of length 1500 with respect to an unknown distribution in an i.i.d. manner. The observer uses these inputs to draw a binary vector of length 7 that we take to be the target vector. Let the target y_k be denoted as

$$y_k = \begin{bmatrix} y_{k0} & y_{k1} & \cdots & y_{k6} \end{bmatrix}$$

  where $y_{kj} = 1$ if x_k belongs to class j an is zero otherwise.

- The model set $h : \{01\}^{1500} \to [0, 1]^7$ mapping x_k onto a vector $\widehat{y}(x_k)$ whose jth component $h^j(x_k)$ is the probability that x_k belongs to class j.

- The loss function is the categorical crossentropy function (because we are using one-hot encoded vectors). Let the kth target y_k be the binary vector $y_k = [y_{k0}, y_{k1}, \ldots, y_{k6}]$ and let $\widehat{y}_k = h(x_k) = [\widehat{y}_{k0}, \widehat{y}_{k1}, \ldots, \widehat{y}_{k6}]$ denote the vector of predi tions made by the model. The equation for the loss function is

$$L(y_k, h(x_k)) = -\sum_{i=0}^{6} y_{ki} \log(\widehat{y}_{ki}))$$

Problem 2: Consider a model set, $\mathcal{H}$ with finite VC dimension of 3 and assume we have trained a model $h^* \in \mathcal{H}$ that minimizes the empirical risk of a training dataset having 1000 samples. You then use a separate test dataset consisting of 50 samples and find that the optimal model's empirical risk on the test data is 0.1. Determine an upper bound on the optimal model's true risk with a confidence level of $1 - \frac{2}{e^3}$.

Solution: We use the Chernoff bound for finite model sets since there is only one model, h^*, in the model set. In that case we have

$$\left| R[h^*] \leq \widehat{R}^{\text{test}}[h^* \right| + \sqrt{\frac{1}{2N} \log \frac{2}{\delta}}$$

with confidence $1 - \delta$. In our case $N = 50$, $\delta = \frac{2}{e^3}$ and $\widehat{R}^{\text{test}}[h^*] = 0.1$ so we have

$$R[h^*] \leq 0.1 + \sqrt{\frac{1}{100} \log \frac{2}{2e^{-3}}} = 0.1 + \frac{\sqrt{3}}{10} = 0.1 + 0.17 = 0.2732$$

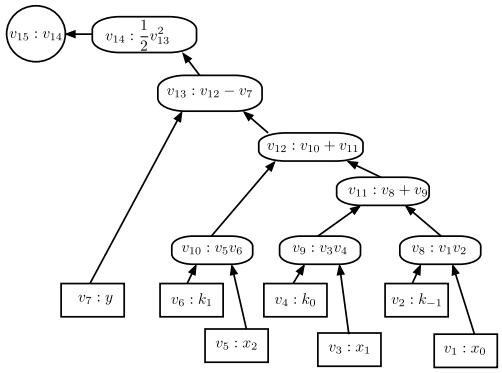
Problem 3: Consider the 1D convolutional model, $h : \mathbb{R}^3 \to \mathbb{R}$, instantiated with the following script

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(3,1))
outputs = layers.Conv1D(filters = 1, kernel_size = 3, strides = 2,
     activation = "relu", use_bias = False)(inputs)
model = keras.Model(inputs = inputs, outputs = outputs)
model.compile(loss = "MSE"
```

List all nodes for a computation graph of this model along with each node's name, "expression"and node type. Use this list to sketch the computation graph for the model labeling each node with its name and expression, and drawing edges between the appropriate nodes.

Solution:

node	expression	type	derivatives
v_1	x_0	source	
v_2	k_0	source	
v_3	x_1	source	
v_4	k_1	source	
v_5	x_2	source	
v_6	k_2	source	
v_7	y	source	
v_8	$v_1 \times v_2$	hidden	$\frac{\partial v_8}{v_1} = v_2, \quad \frac{\partial v_8}{v_2} = v_1$
v_9	$v_3 \times v_4$	hidden	$\frac{\partial v_9}{\partial v_3} = v_4, \quad \frac{\partial v_9}{\partial v_4} = v_3$
v_{10}	$v_5 \times v_6$	hidden	$\frac{\partial v_{10}}{\partial v_5} = v_6, \quad \frac{\partial v_{10}}{\partial v_6} = v_5$
v_{11}	$v_8 + v_9$	hidden	$\frac{\partial v_{11}}{\partial v_8} = 1, \quad \frac{\partial v_{11}}{\partial v_9} = 1$
v_{12}	$v_{10} + v_{11}$	hidden	$\frac{\partial v_{12}}{\partial v_{10}} = 1, \quad \frac{\partial v_{12}}{\partial v_{11}} = 1$
v_{13}	$v_{12} - v_7$	hidden	$\frac{\partial v_{13}}{\partial v_7} = -1, \quad \frac{\partial v_{13}}{\partial v_{12}} = 1$
v_{14}	$\frac{1}{2} v_{13}^2$	hidden	$\frac{\partial v_{14}}{\partial v_{13}} = v_{13}$
v_{15}	v_{14}	sink	$\frac{\partial v_{15}}{\partial v_{14}} = 1$

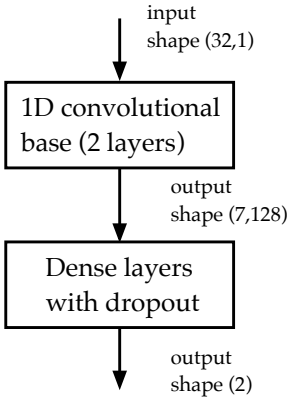


Notebook Problem: You are to train a model that detects heart arrhythmias from electrocardiogram (ECG) traces measured using a user's smartwatch. An arrhythmia is an abnormal heart rhythm that can be caused by a number of medical conditions. The dataset you will be working

with is the MIT-BIH Arrhythmia database. A pre-processed version of this database has been included as a zip file in the midterm directory. The database has 90083 normal samples and 7024 arrhythmia samples. The input samples are float64 vectors of length 32 representing the ECG time series. The associated label is a one-hot encoded vector in which `[0.0,1.0]` denotes a *normal* sample and `[1.0,0.0]` denotes an *arrhythmia* sample. Because the dataset is unbalanced, the exam's Utility file (`MD1utils.py`) declares a special loss function known as the *focal Tversky* loss function. This loss function weights the loss of the "arrhythmia" samples more heavily than the "normal" samples. You will train your arrhythmia model using this special loss function.

- **Part 1:** Load the training and test datasets from the pickle file in the `data` directory. The imported dataset is a list of length 4. The first entry in the list is the training inputs, the second list entry is the training targets, the third list entry is the testing inputs, the last list entry is the testing targets. Determine the number of normal and arrythmia training samples. Plot one of the ECG traces for a normal and arrhythmia sample.

- **Part 2:** Instantiate the model architecture in Fig. **??**. This model has a convolutional base with two 1D convolutional layers (Conv1D) with `strides` of 2 and `kernel_size` of 3. The first Conv1D layer has 64 filters and the second one has 128 filters. The output of the convolutional base is fed to a Dense layer with 256 nodes, followed by a Dense output layer with 2 nodes. The model's hidden layers use ReLu activation. The output layer uses a softmax activation. Use a dropout layer after the 256 node dense layer to help avoid overfitting. Evaluate your *untrained* model's accuracy on the test dataset. Display the confusion matrix for the untrained model on the test dataset.



Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 1)	0
conv1d (Conv1D)	(None, 15, 64)	256
conv1d_1 (Conv1D)	(None, 7, 128)	24,704
flatten (Flatten)	(None, 896)	0
dense (Dense)	(None, 256)	229,632
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 2)	514

Figure 1: MD1 Notebook Model

- **Part 3:** Fit your model for 20 epochs on the training data assuming a 20% validation split. Use the focal Tversky loss function included in the notebooks utility file. List the hyperparameters for the optimizer, regularizer, and batched training data. Adjust the hyperparameters (if necessary) until you achieve a validation accuracy greater than 92%. Plot the training curves, save the best model with respect to validation loss, and determine the best model's loss, accuracy, sensitivity, and specificity on the testing dataset. Display the confusion matrix for the trained model on the test dataset.

- **Part 4:** The original dataset is highly imbalanced. Use Synthetic Minority Oversampling

(SMOTE) [**?**] to rebalance the dataset. If your Python environment doesn't have the `imblearn` library, you will need to install it using

```
pip install imbalanced-learn
```

You can then import SMOTE into your notebook and use the following script to "oversample" the original data set

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=78)
X = X_train.reshape((77685, 32))
y = y_train
X_sm, y_sm = sm.fit_resample(X,y)
```

Determine how many "normal" and "arrhythmia" samples are in the SMOTE dataset you created. Reset your model and fit it for 20 epochs on the SMOTE dataset assuming a 20% validation split. Use the same set of hyperparameters you used in the preceding part. Plot the training curves, save the best model with respect to validation loss, and determine the best model's loss, accuracy, sensitivity, and specificity on the testing dataset. Comment on the results of your new model versus the one you obtained in the preceding part.

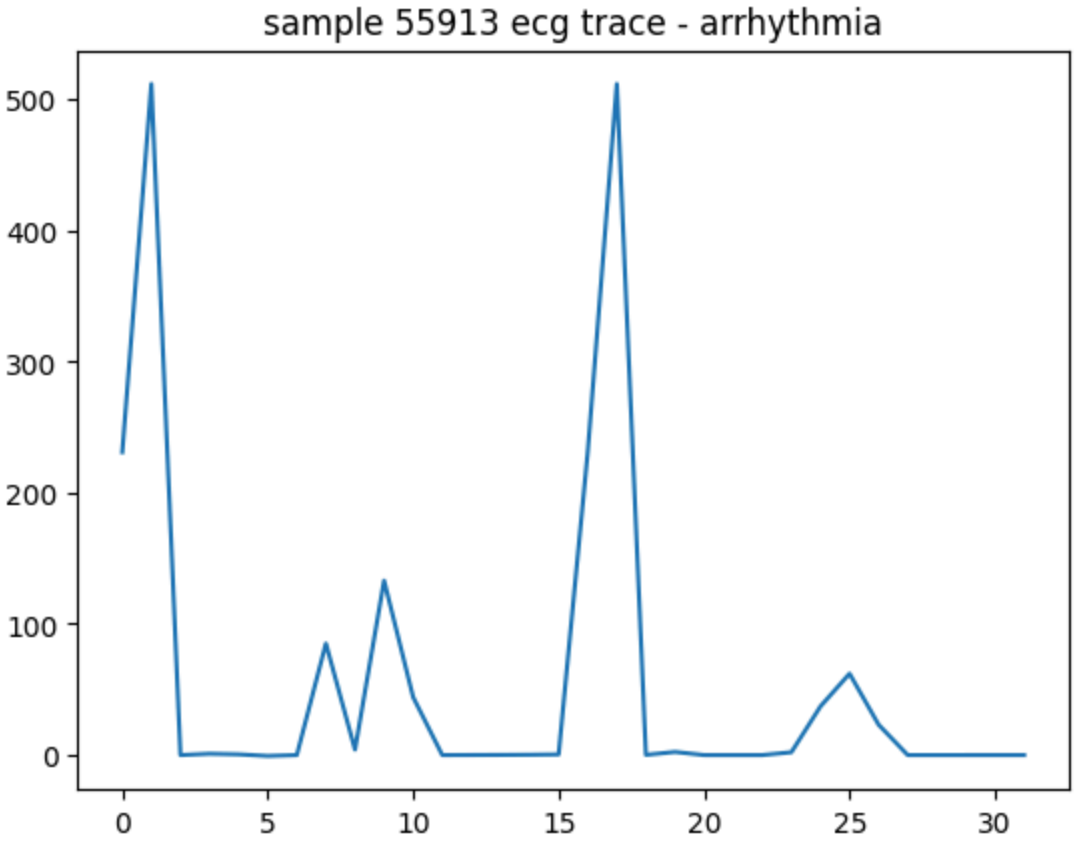
References

[1] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

In [1]:
```
import numpy as np
import pandas as pd
import pickle as pkl
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dr
from tensorflow import keras

from MD1utils import focal_tversky, fit_evaluation
```

In [2]:
```


with open("data/MD1-MIT-BIH-dataset.pkl", "rb") as fp:
    dataset = pkl.load(fp)

X_train = dataset[0]
y_train = dataset[1]
X_test = dataset[2]
y_test = dataset[3]

X_train.shape
index = np.random.uniform(len(y_train))
index = np.floor(index).astype('int')

ecg_trace = X_train[index].reshape((32,))
time = np.arange(32)
if y_train[index,0]==0.0:
    target = "normal"
else:
    target = "arrhythmia"

plt.plot(time,ecg_trace)
plt.title(f"sample {index} ecg trace - {target}")

n_arrhythmia = np.sum(y_train[:,0]).astype('int')
n_normal = np.sum(y_train[:,1]).astype('int')
print(f"training samples\n\t{n_arrhythmia} arrythmia\n\t{n_normal} normal")
```

```
training samples
        5590 arrythmia
        72095 normal
```



In [22]:
```
 tf.keras.backend.clear_session()

inputs = keras.Input(shape=(32,1))
x = Conv1D(filters=64, kernel_size=3, strides = 2,activation='relu')(inputs)
x = Conv1D(filters=128, kernel_size=3, strides = 2, activation='relu')(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(2, activation='softmax')(x)

model = keras.Model(inputs,outputs)

model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Par
input_layer (InputLayer)	(None, 32, 1)	
conv1d (Conv1D)	(None, 15, 64)	
conv1d_1 (Conv1D)	(None, 7, 128)	24
flatten (Flatten)	(None, 896)	
dense (Dense)	(None, 256)	229
dropout (Dropout)	(None, 256)	
dense_1 (Dense)	(None, 2)	

Total params: 255,106 (996.51 KB)

Trainable params: 255,106 (996.51 KB)

Non-trainable params: 0 (0.00 B)

In [23]:
```
model.compile(loss = focal_tversky, optimizer='adam', metrics=['accuracy'])

test_loss, test_acc = model.evaluate(X_test, y_test, verbose =0)
print(f"Test loss: {test_loss*100:.2f}%")
print(f"Test Accuracy: {test_acc*100: .2f}%")
```

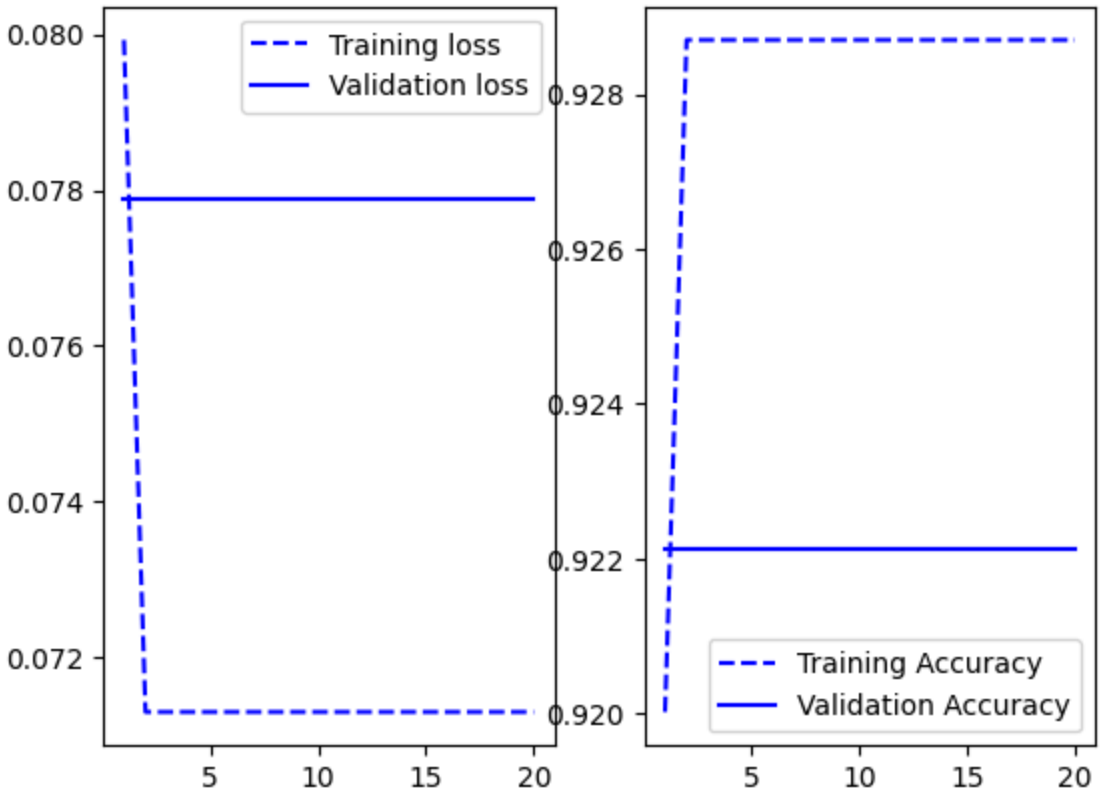
```
Test loss: 90.93%
Test Accuracy:  7.46%
```

In [24]:
```
batch_size = 1024
epochs = 20
history = model.fit(X_train,y_train, batch_size = batch_size, epochs=epochs,
```

```
Epoch 1/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.8881 - loss: 0.1117 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 2/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9283 - loss: 0.0717 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 3/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9286 - loss: 0.0714 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 4/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9284 - loss: 0.0716 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 5/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9285 - loss: 0.0715 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 6/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9302 - loss: 0.0698 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 7/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9301 - loss: 0.0699 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 8/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9308 - loss: 0.0692 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 9/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9287 - loss: 0.0713 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 10/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9293 - loss: 0.0707 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 11/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9279 - loss: 0.0721 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 12/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9293 - loss: 0.0707 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 13/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9296 - loss: 0.0704 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 14/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9292 - loss: 0.0708 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 15/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9292 - loss: 0.0708 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 16/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9276 - loss: 0.0724 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 17/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9265 - loss: 0.0735 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 18/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9296 - loss: 0.0704 -
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 19/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9289 - loss: 0.0711 -
```

```
val_accuracy: 0.9221 - val_loss: 0.0779
Epoch 20/20
69/69 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9281 - loss: 0.0719 -
val_accuracy: 0.9221 - val_loss: 0.0779
```

In [25]:
```
fit_evaluation(history)
```



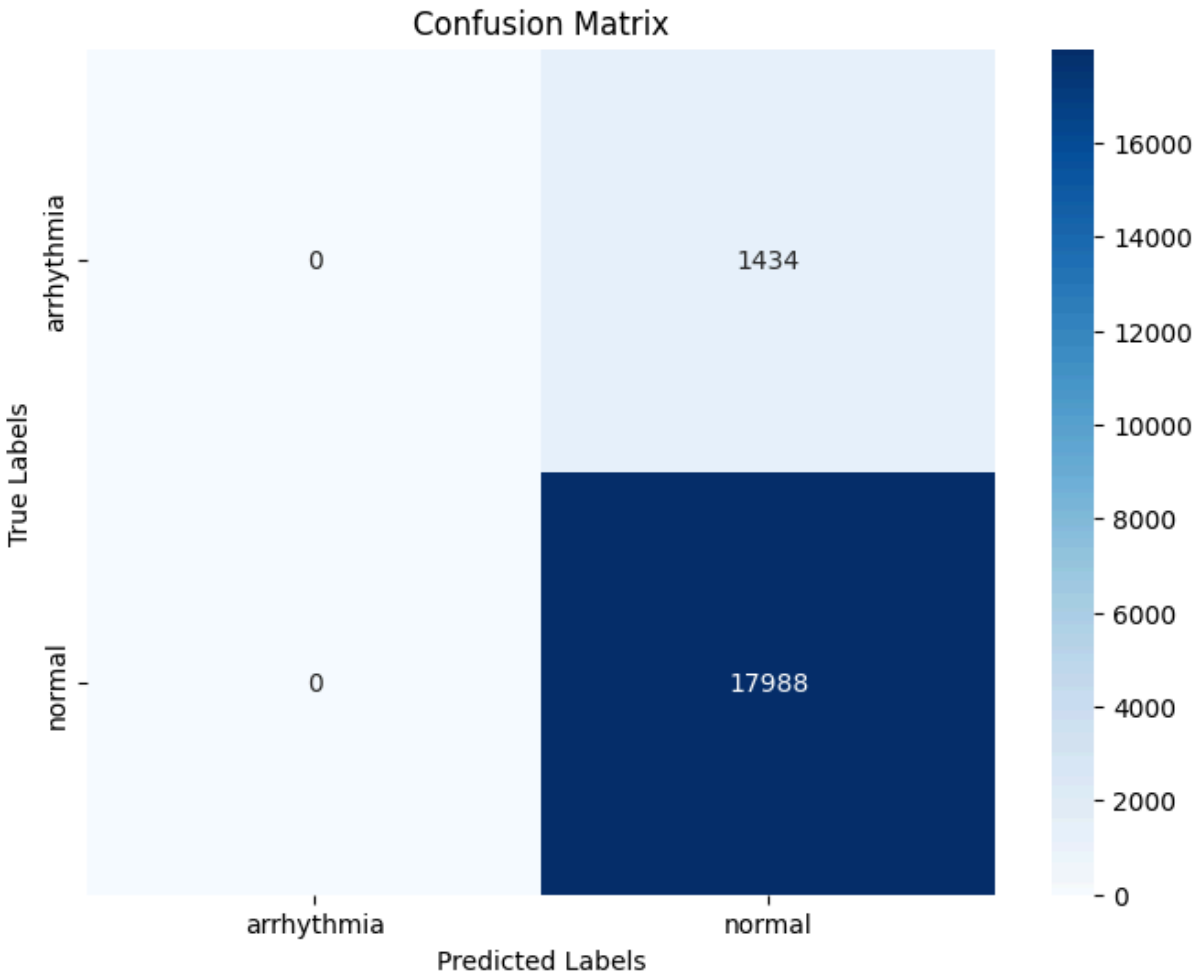
In [26]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose =0)
print(f"Test loss: {test_loss*100:.2f}%")
print(f"Test Accuracy: {test_acc*100: .2f}%")
```

```
Test loss: 7.38%
Test Accuracy:  92.62%
```

In [27]:
```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred,axis=1)
y_true_labels = np.argmax(y_test, axis=1)



conf_matrix = confusion_matrix(y_true_labels, y_pred_labels)
class_names = ['arrhythmia', 'normal']
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels = cl
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```
plt.title('Confusion Matrix')
plt.show()
```

```
607/607 ━━━━━━━━━━━━━━━━━━━━ 0s 588us/step
```



In [28]:
```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=78)
X = X_train.reshape((77685,32))
y = y_train
X_sm, y_sm = sm.fit_resample(X,y)

print(f"Shape of X before SMOTE: {X.shape} Shape of X after SMOTE: {X_sm.sha
```

```
Shape of X before SMOTE: (77685, 32) Shape of X after SMOTE: (144190, 32)
```

In [29]:
```
y_sm_x = []
for k in range(len(y_sm)):
    if y_sm[k]==0:
        y_sm_x.append([1.,0.])
    else:
        y_sm_x.append([0.,1.])
y_sm_x = np.array(y_sm_x).reshape((len(y_sm),2))
nx, ny= X_sm.shape
X_sm_x = np.array(X_sm).reshape((nx,ny,1))
```

In [34]:
```
tf.keras.backend.clear_session()

inputs = keras.Input(shape=(32,1))
x = Conv1D(filters=64, kernel_size=3, strides = 2,activation='relu')(inputs)
#x = MaxPooling1D(pool_size=2)(x)
x = Conv1D(filters=128, kernel_size=3, strides = 2, activation='relu')(x)
#x = MaxPooling1D(pool_size=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(2, activation='softmax')(x)

model = keras.Model(inputs,outputs)

model.summary()

model.compile(loss = focal_tversky, optimizer='adam', metrics=['accuracy'])

test_loss, test_acc = model.evaluate(X_test, y_test, verbose =0)
print(f"Test loss: {test_loss*100:.2f}%")
print(f"Test Accuracy: {test_acc*100: .2f}%")
```

Model: "functional"

Layer (type)	Output Shape	Par
input_layer (InputLayer)	(None, 32, 1)	
conv1d (Conv1D)	(None, 15, 64)	
conv1d_1 (Conv1D)	(None, 7, 128)	24
flatten (Flatten)	(None, 896)	
dense (Dense)	(None, 256)	229
dropout (Dropout)	(None, 256)	
dense_1 (Dense)	(None, 2)	

Total params: 255,106 (996.51 KB)

Trainable params: 255,106 (996.51 KB)

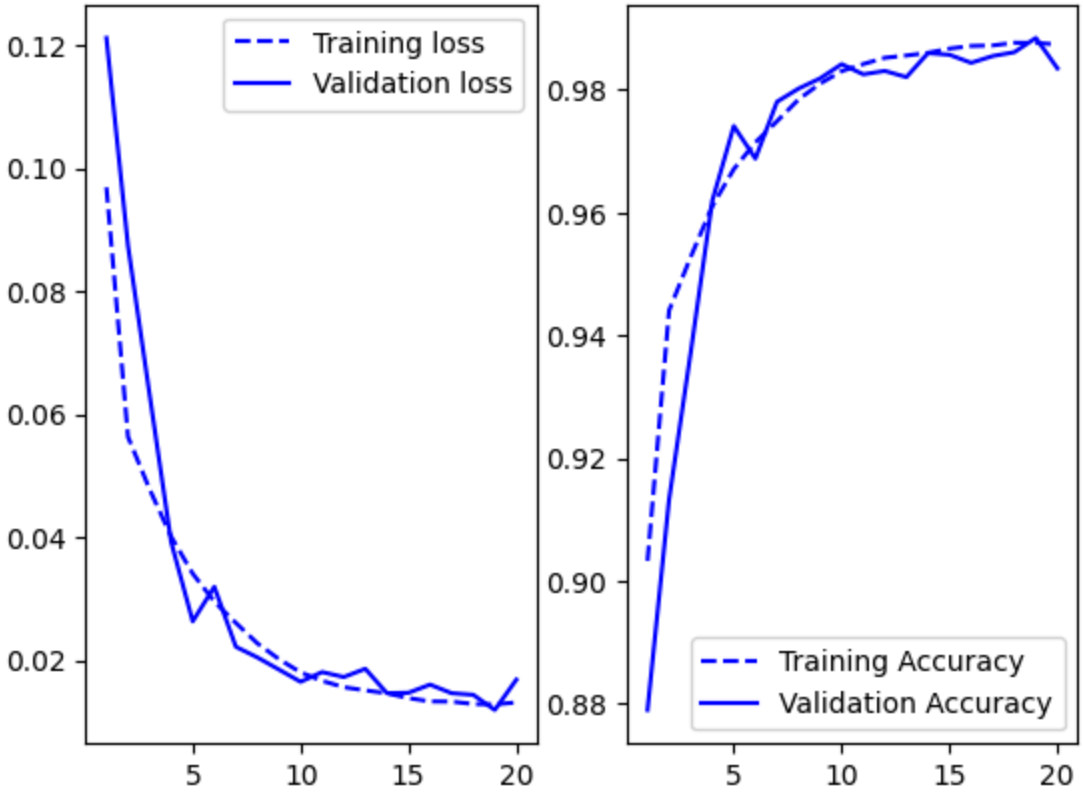
Non-trainable params: 0 (0.00 B)

```
Test loss: 92.31%
Test Accuracy:  7.38%
```

In [35]:
```
batch_size = 1024
epochs = 20
history = model.fit(X_sm_x,y_sm_x, batch_size = batch_size, epochs=epochs, v
fit_evaluation(history)
```

```
Epoch 1/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.8481 - loss: 0.1519
- val_accuracy: 0.8789 - val_loss: 0.1212
Epoch 2/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9399 - loss: 0.0603
- val_accuracy: 0.9133 - val_loss: 0.0874
Epoch 3/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9499 - loss: 0.0507
- val_accuracy: 0.9372 - val_loss: 0.0632
Epoch 4/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9590 - loss: 0.0418
- val_accuracy: 0.9619 - val_loss: 0.0392
Epoch 5/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9668 - loss: 0.0345
- val_accuracy: 0.9741 - val_loss: 0.0263
Epoch 6/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9697 - loss: 0.0312
- val_accuracy: 0.9688 - val_loss: 0.0319
Epoch 7/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9742 - loss: 0.0268
- val_accuracy: 0.9780 - val_loss: 0.0222
Epoch 8/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9778 - loss: 0.0234
- val_accuracy: 0.9802 - val_loss: 0.0204
Epoch 9/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.9805 - loss: 0.0207
- val_accuracy: 0.9819 - val_loss: 0.0185
Epoch 10/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9832 - loss: 0.0178
- val_accuracy: 0.9842 - val_loss: 0.0165
Epoch 11/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9842 - loss: 0.0170
- val_accuracy: 0.9825 - val_loss: 0.0181
Epoch 12/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9852 - loss: 0.0155
- val_accuracy: 0.9831 - val_loss: 0.0173
Epoch 13/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9859 - loss: 0.0148
- val_accuracy: 0.9821 - val_loss: 0.0186
Epoch 14/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9865 - loss: 0.0142
- val_accuracy: 0.9860 - val_loss: 0.0146
Epoch 15/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.9867 - loss: 0.0138
- val_accuracy: 0.9857 - val_loss: 0.0147
Epoch 16/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9875 - loss: 0.0130
- val_accuracy: 0.9844 - val_loss: 0.0160
Epoch 17/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9871 - loss: 0.0134
- val_accuracy: 0.9855 - val_loss: 0.0146
Epoch 18/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9874 - loss: 0.0131
- val_accuracy: 0.9861 - val_loss: 0.0144
Epoch 19/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9876 - loss: 0.0129
```

```
- val_accuracy: 0.9885 - val_loss: 0.0119
Epoch 20/20
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9872 - loss: 0.0134
- val_accuracy: 0.9835 - val_loss: 0.0168
```

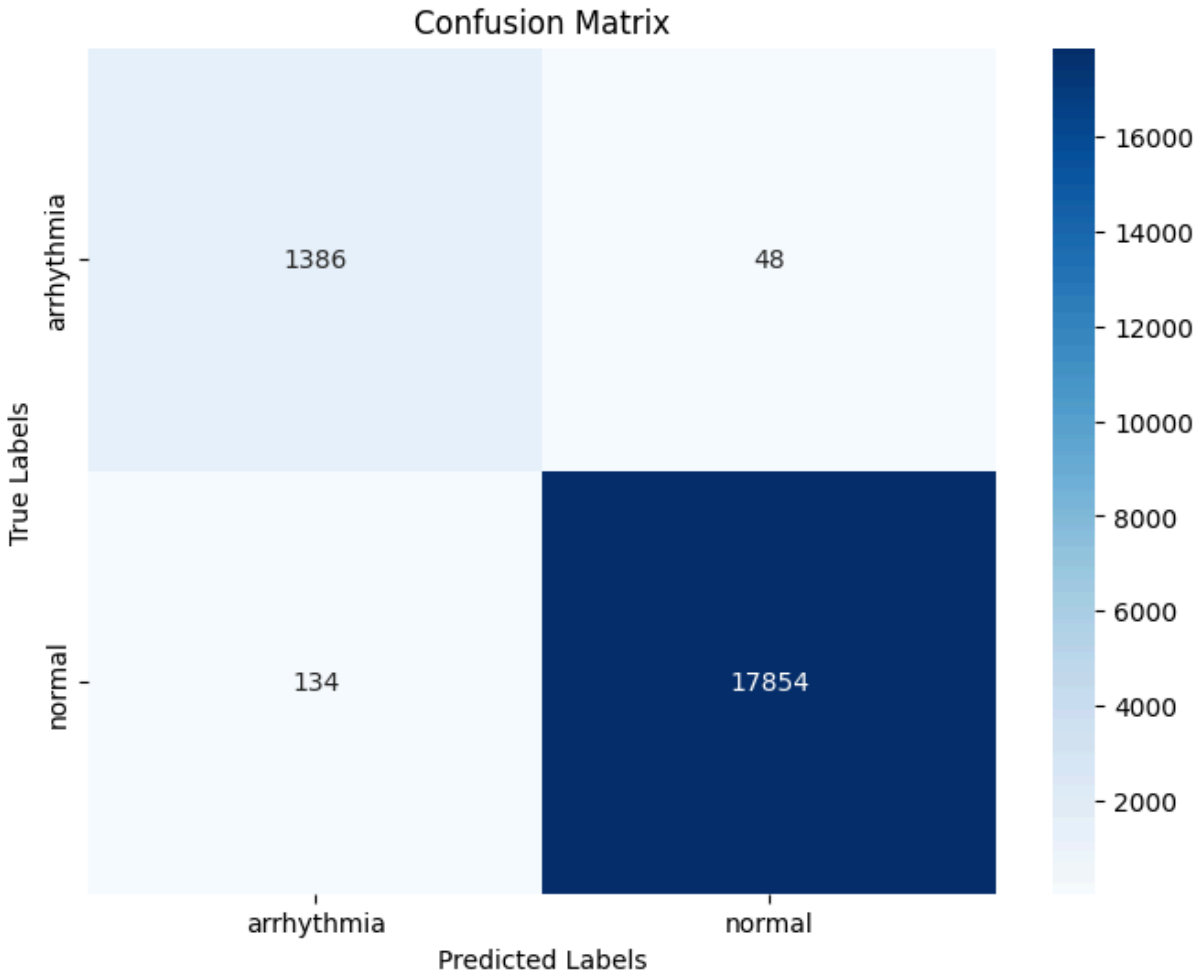


In [36]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose =0)
print(f"Test loss: {test_loss*100:.2f}%")
print(f"Test Accuracy: {test_acc*100: .2f}%")
```

```
Test loss: 0.96%
Test Accuracy:  99.06%
```

In [37]:
```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred,axis=1)
y_true_labels = np.argmax(y_test, axis=1)

conf_matrix = confusion_matrix(y_true_labels, y_pred_labels)
class_names = ['arrhythmia', 'normal']
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels = cl
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
607/607 ━━━━━━━━━━━━━━━━━━━━ 0s 603us/step
```


Confusion Matrix
arrhythmia
normal
True Labels
1386
48
134
17854
arrhythmia
normal
Predicted Labels
16000
14000
12000
10000
8000
6000
4000
2000