Assessing Fracture Healing with Artificial Intelligence:

Using Transfer Learning to Predict the *Radiographic Union Score for Tibial Fractures*, in the Radiography of High-Energy Trauma

Shen Zhou Hong Goldsmiths, UoL

April 4th, 2023

Contents

Chapter 1

Implementation

```
class TransferLearningModel(tf.keras.Model):
    def __init__(self, dropout_rate: float, **kwargs):
        super().__init__(**kwargs)

        self.input_layer: tf.Tensor = layers.InputLayer(input_shape=(299, 299, 3))
        self.data_augmentation: tf.keras.Sequential = tf.keras.Sequential([
            layers.RandomFlip(seed=RNG_SEED),
        ])

        self.inceptionv3: tf.keras.Model = tf.keras.applications.InceptionV3(
            include_top=False,
            weights='imagenet'
        )
        self.inceptionv3.trainable = False

        self.classifier: tf.keras.Sequential = tf.keras.Sequential([
            layers.GlobalMaxPooling2D(),
            layers.Dense(1024, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense( 512, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense( 256, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense(  18, activation='sigmoid')
        ])

        self.model: tf.keras.Sequential = tf.keras.Sequential([
            self.input_layer,
            self.data_augmentation,
            self.inceptionv3,
            self.classifier
        ])

    def call(self, inputs):
        return self.model(inputs)
```

Listing 1: Sharding dataset for K-Fold Cross Validation

```python
def cross_validate(ModelClass: tf.keras.Model, ds: tf.data.Dataset, epochs: int = 50,
↪  batch_size: int = 128, k: int = 10) -> list[tf.keras.callbacks.History]:

    history_list: list[tf.keras.callbacks.History] = []
    train_valid_pairs: list[tf.data.Dataset] = k_fold_dataset(ds, k)

    for i, (ds_train, ds_valid) in enumerate(train_valid_pairs):

        tf.keras.backend.clear_session()
        model = ModelClass()
        model.compile(
            optimizer=tf.keras.optimizers.Adam(),
            loss=tf.keras.losses.BinaryCrossentropy(),
            metrics=metrics
        )
        history = model.fit(
            ds_train,
            validation_data=ds_valid,
            epochs=epochs,
            batch_size=batch_size,
        )
        history_list.append(history.history)

    return history_list
```

Listing 2: K-Fold Cross Validation Implementation

```python
def k_fold_dataset(ds: tf.data.Dataset, k: int = 10) -> list[tuple[tf.data.Dataset,
↪  tf.data.Dataset]]:
    # First shard the given dataset into k individual folds.
    list_of_folds: list[tf.data.Dataset] = []
    for i in range(k):
        fold: tf.data.Dataset = ds.shard(num_shards=k, index=i)
        list_of_folds.append(fold)

    # Next, generate a list of train and validation dataset tuples
    list_of_ds_pairs: list[tuple[tf.data.Dataset, tf.data.Dataset]] = []
    for i, holdout_fold in enumerate(list_of_folds):
        ds_valid: tf.data.Dataset = holdout_fold

        # Select every fold except holdout_fold as the training folds
        training_folds: list[tf.data.Dataset] = list_of_folds[:i] +
        ↪  list_of_folds[i+1:]

        # ds_train size is (k-1)/k of the original dataset
        ds_train: tf.data.Dataset = training_folds[0]
        for fold in training_folds[1:]:
            ds_train = ds_train.concatenate(fold)

        ds_pair: tuple[tf.data.Dataset, tf.data.Dataset] = (ds_train, ds_valid)
        list_of_ds_pairs.append(ds_pair)

    return list_of_ds_pairs
```

Listing 3: Sharding dataset for K-Fold Cross Validation

```
def hyperparameter_search(trials: int, kfolds: int = 6, epochs: int = 20) ->
↪  list[dict[str, Union[int, float, list[tf.keras.callbacks.History]]]]:
    search_results: list[dict[str, any]] = []

    for trial in range(trials):
        # Randomly pick hyperparameter options
        rng = np.random.default_rng()
        batch_size  : int   = rng.integers(16, 2048, endpoint=True)
        dropout_rate: float = rng.uniform(0.0, 0.5)

        # Conduct K-Fold cross-validation with given hyperparameters
        results: list[tf.keras.callbacks.History] = cross_validate(
            TransferLearningModel,
            ds_train_and_valid,
            epochs=epochs,
            batch_size=batch_size,
            dropout_rate=dropout_rate,
            k=kfolds
        )

        search_results.append({
            "max_val_auc" : calc_kfold_max(results, "val_auc"),
            "batch_size"  : batch_size,
            "dropout_rate": dropout_rate,
            "history_list": k_fold_results
        })

    return search_results
```

Listing 4: Hyperparameter Search Implementation

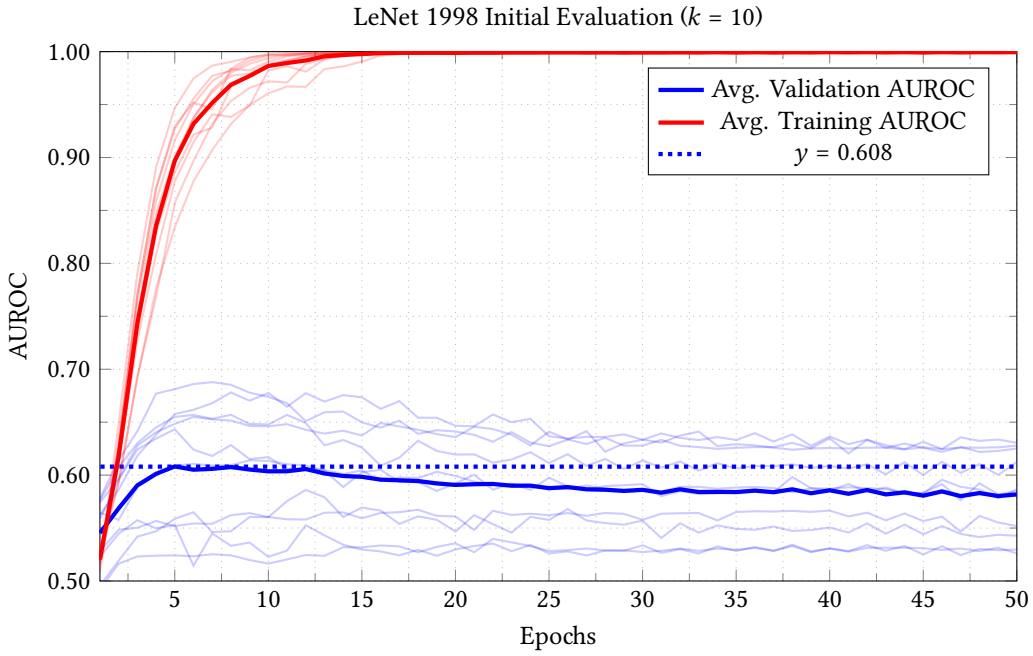

Figure 1.1: Baseline shallow CNN based on the LeNet 1998 architecture

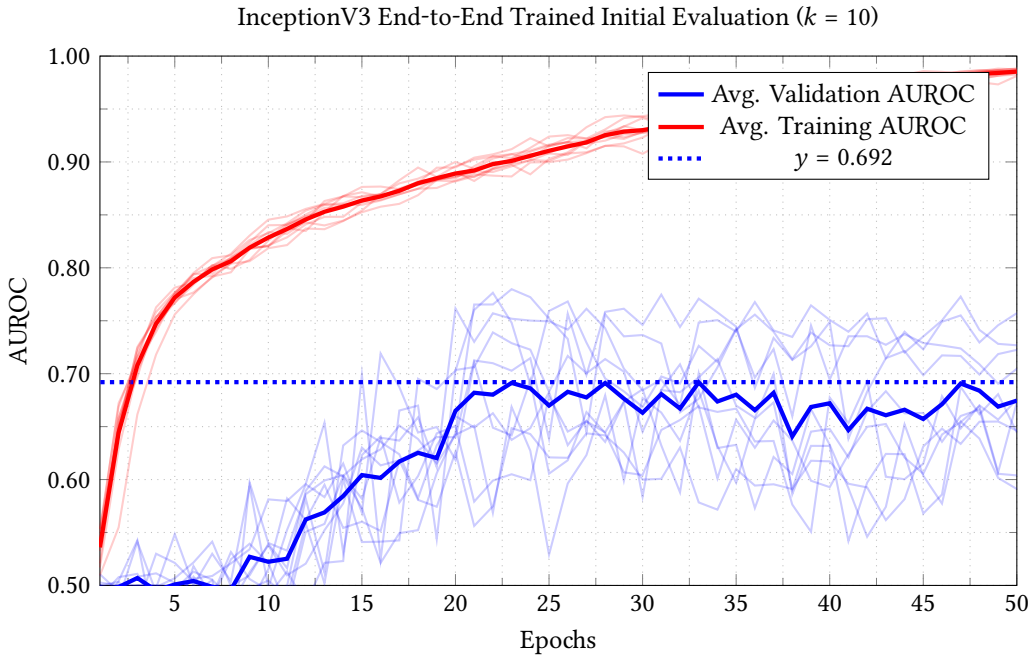

Figure 1.2: InceptionV3 Model Trained on Study Data.

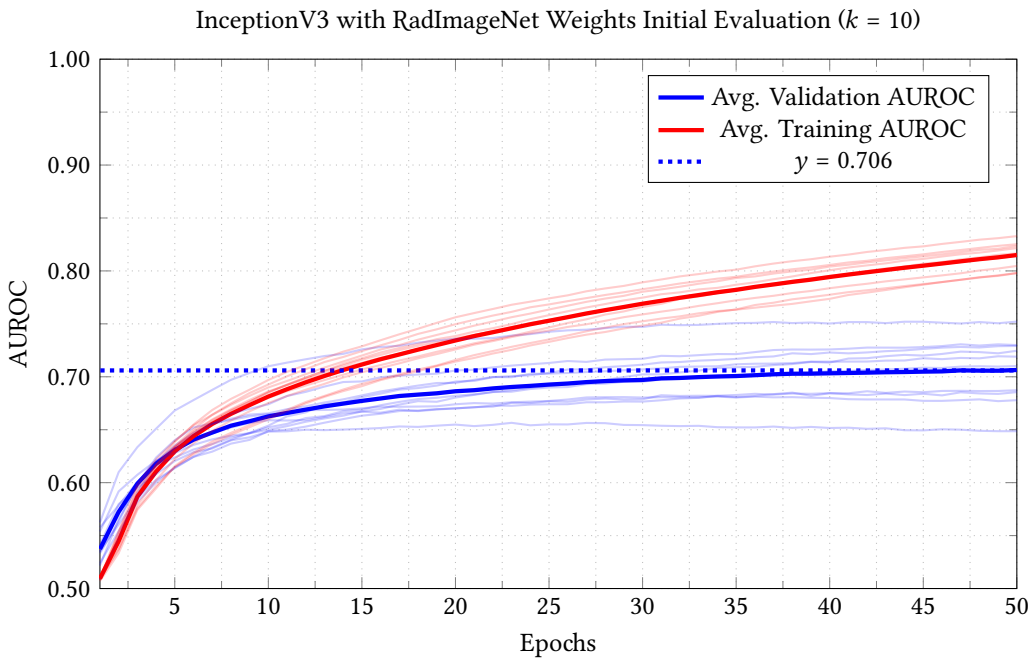

Figure 1.3: InceptionV3 with RadImageNet Weights

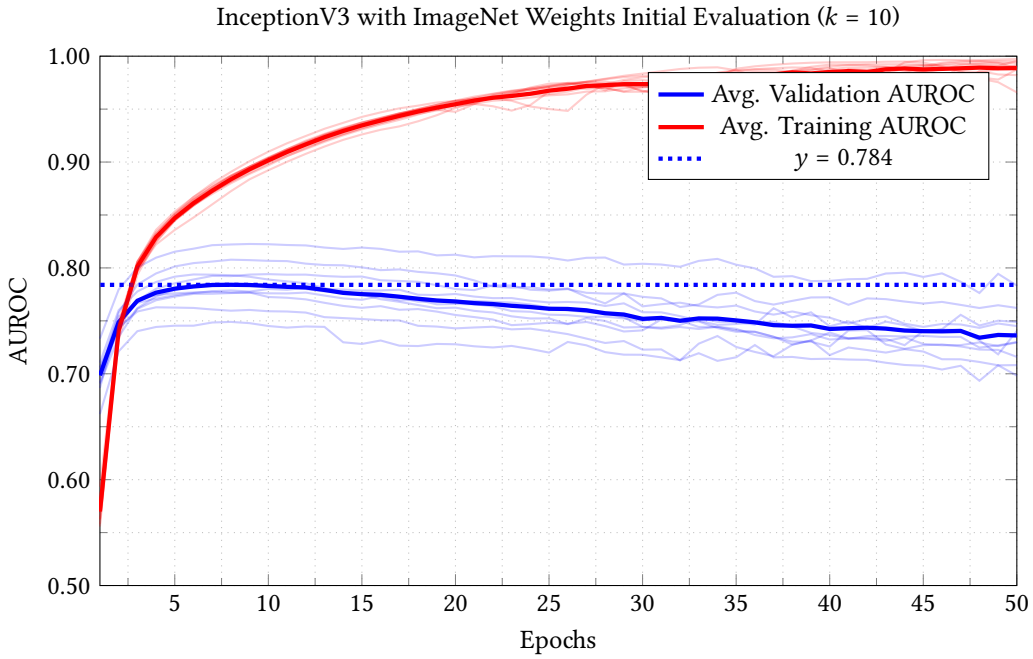

Figure 1.4: InceptionV3 with ImageNet Weights

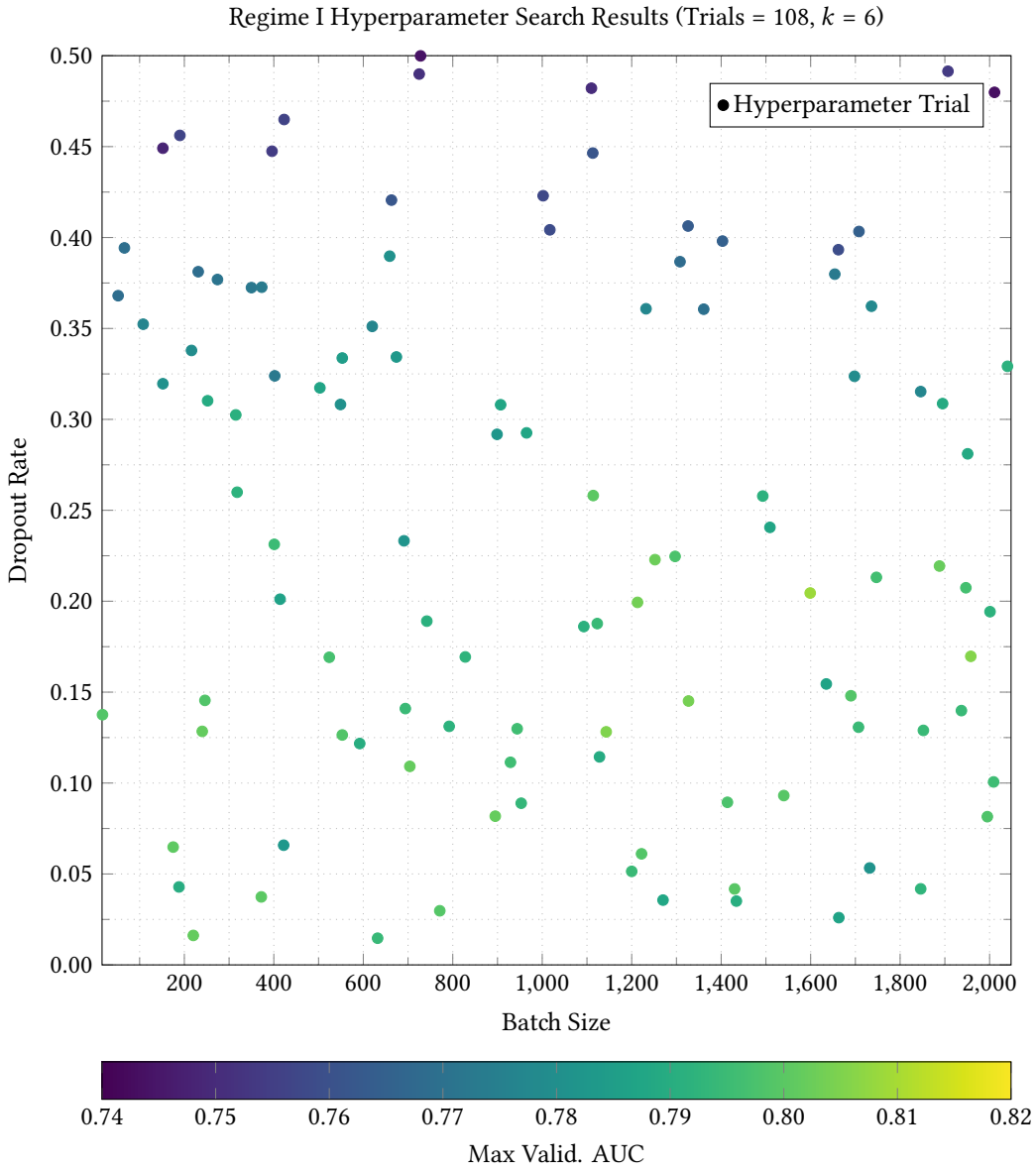

Figure 1.5: Results for the Hyperparameter Search Regime I

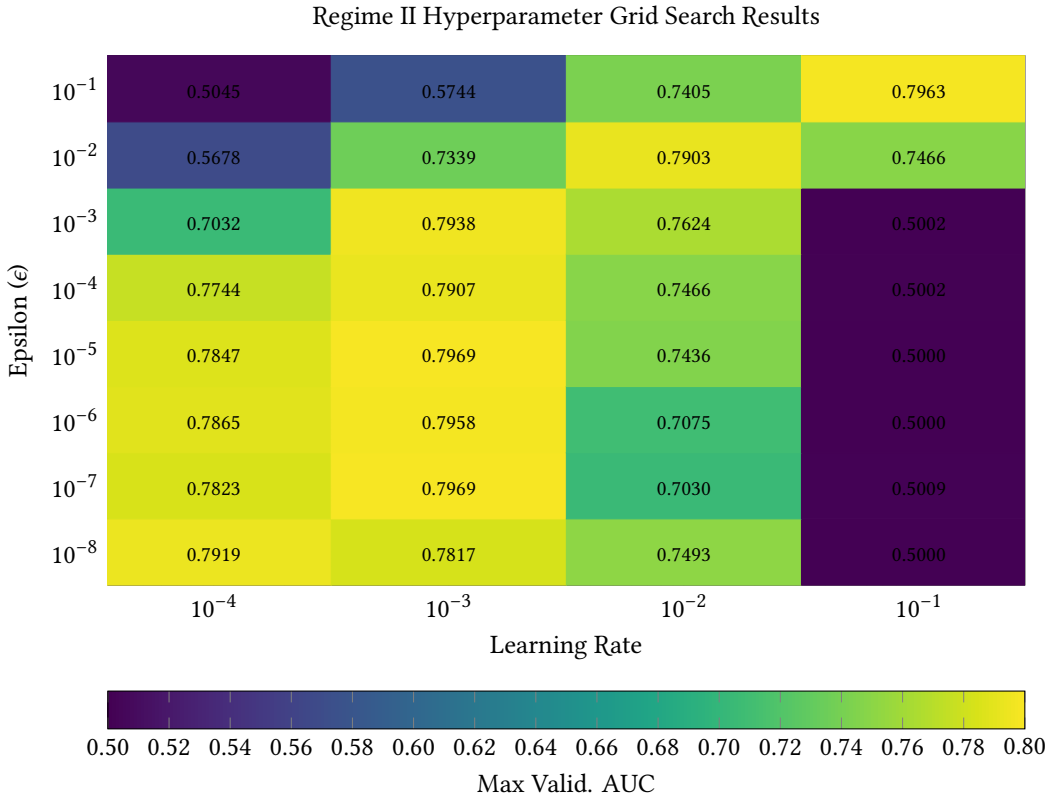

Figure 1.6: Results for the Hyperparameter Search Regime II

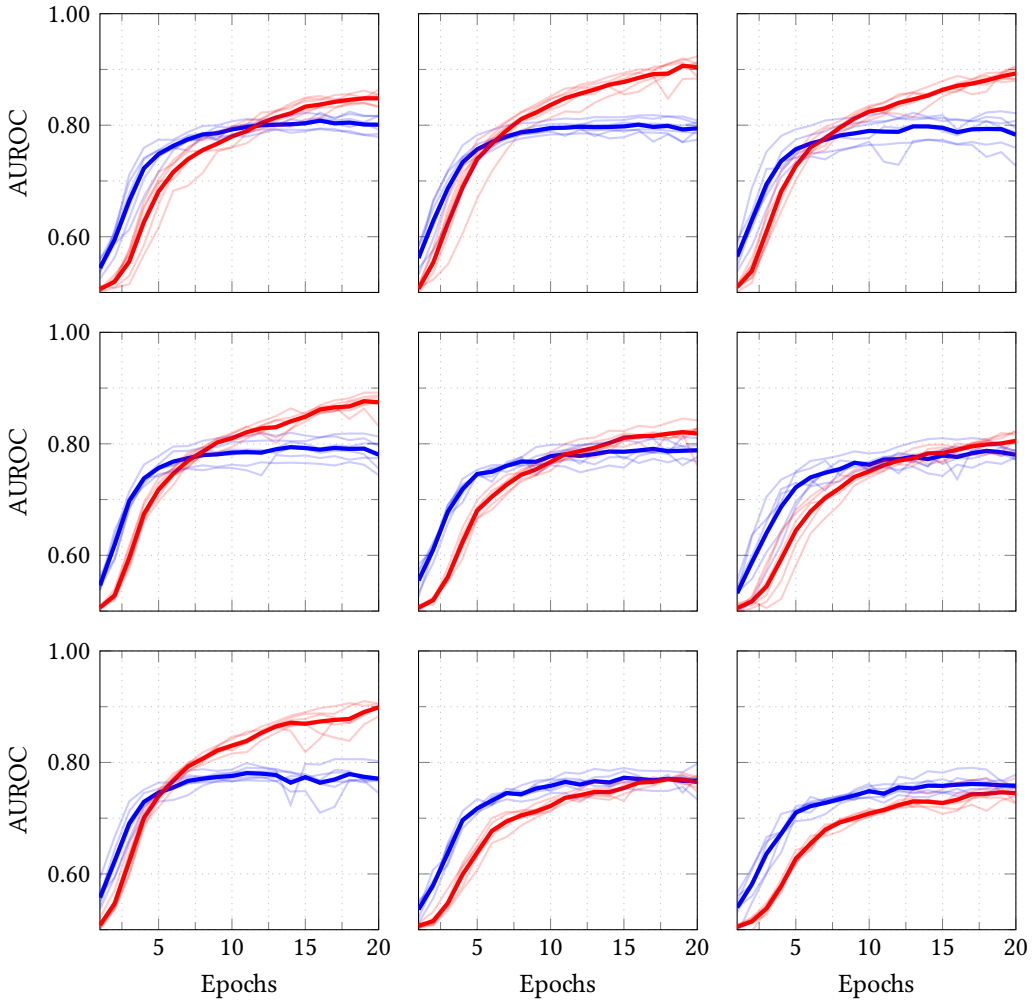

Figure 1.7: Examples of model performance from hyperparameter regime I search.

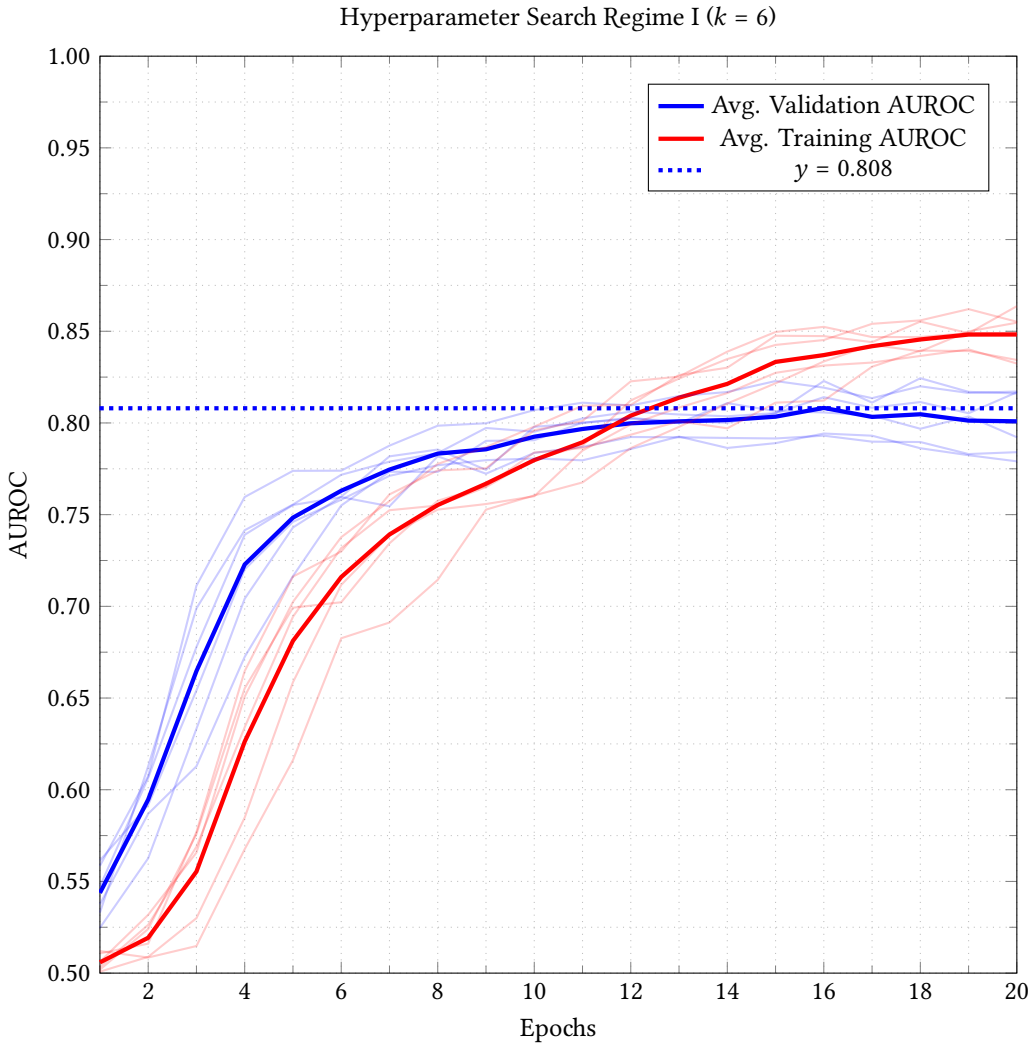

Figure 1.8: Best performing model in Regime I

Appendix A

Additional Materials